

# Applying Occam’s Razor to Transformer-Based Dependency Parsing: What Works, What Doesn’t, and What is Really Necessary

Stefan Grünewald<sup>★♦</sup>

Annemarie Friedrich<sup>♦</sup>

Jonas Kuhn<sup>★</sup>

<sup>★</sup>Institut für Maschinelle Sprachverarbeitung, University of Stuttgart

<sup>♦</sup>Bosch Center for Artificial Intelligence, Renningen, Germany

stefan.gruenewald|annemarie.friedrich@de.bosch.com

jonas.kuhn@ims.uni-stuttgart.de

## Abstract

The introduction of pre-trained transformer-based contextualized word embeddings has led to considerable improvements in the accuracy of graph-based parsers for frameworks such as Universal Dependencies (UD). However, previous works differ in various dimensions, including their choice of pre-trained language models and whether they use LSTM layers. With the aims of disentangling the effects of these choices and identifying a simple yet widely applicable architecture, we introduce STEPS, a new modular graph-based dependency parser. Using STEPS, we perform a series of analyses on the UD corpora of a diverse set of languages. We find that the choice of pre-trained embeddings has by far the greatest impact on parser performance and identify XLM-R as a robust choice across the languages in our study. Adding LSTM layers provides no benefits when using transformer-based embeddings. A multi-task training setup outputting additional UD features may contort results. Taking these insights together, we propose a simple but widely applicable parser architecture and configuration, achieving new state-of-the-art results (in terms of LAS) for 10 out of 12 diverse languages.<sup>1</sup>

## 1 Introduction

Recent years have seen considerable improvements in the performance of syntactic dependency parsers for frameworks such as Universal Dependencies (UD; de Marneffe et al., 2014). For graph-based parsers, these improvements can in large part be attributed to two developments: (1) the introduction of deep biaffine classifiers (Dozat and Manning, 2017), which now constitute the de-facto standard approach for graph-based dependency parsing, and

<sup>1</sup>We release our code and pre-trained models on [github.com/boschresearch/steps-parser](https://github.com/boschresearch/steps-parser).

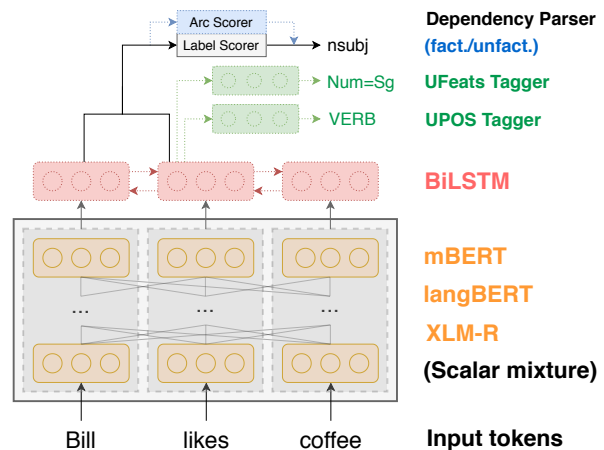


Figure 1: Modular architecture of the STEPS parser. Dotted lines denote optional components.

(2) the rise of pre-trained distributed word representations, particularly transformer-based contextualized embeddings such as BERT (Devlin et al., 2019) or RoBERTa (Liu et al., 2019). Both characteristics are present in recent top-performing systems (Che et al., 2018; Straka et al., 2019; Konratyuk and Straka, 2019; Kanerva et al., 2018, 2020; Che et al., 2018).

However, there remain a considerable number of implementation and configuration choices whose impact on parser performance is less well understood. This is evidenced by the many different model configurations (see Table 1) present in parsers that have achieved top results in recent shared tasks addressing UD parsing (Zeman et al., 2017, 2018; Bouma et al., 2020). The choices include (a) the particular pre-trained word embeddings or language model to use, (b) whether to utilize an LSTM in addition to (fine-tuned) contextualized word embeddings; and (c) whether to use a multi-task training setup simultaneously predicting additional UD features (such as morphology or parts of speech) during parsing.

The aim of this paper is to disentangle the effects of the above factors and determine their impact on parser performance. We appeal to the concept of Occam’s razor by ways of avoiding architectural elements that do not bring about a testable advantage. With this idea in mind, we introduce STEPS (the Stuttgart Transformer-based Extensible Parsing System), a modular graph-based dependency parser which implements commonly used modules such as biaffine scorers (Dozat et al., 2017; Kondratyuk and Straka, 2019) or LSTM layers (Straka, 2018) (see Figure 1). Using STEPS, we perform a series of experiments on the UD treebanks of a diverse set of languages. Our setup facilitates estimating the impact of the various architectures and configuration decisions in a comparable way.

Our most important insight is that a relatively simple architecture using biaffine heads on top of fine-tuned XLM-R (Conneau et al., 2020) leads to the highest parsing accuracy for almost all languages in our study, outperforming prior systems on most languages. Our analysis indicates that LSTM layers do not lead to benefits. Simplifying the architecture even further by using a single scorer for edge and label prediction results in similar performance but on average leads to longer training times. Our contributions are as follows:

- (1) We introduce STEPS, a new implementation of a graph-based dependency parser designed to be modular and easily extensible. STEPS achieves new state-of-the-art UD parsing performance (in terms of LAS) for 10 out of the 12 typologically diverse languages in our study. We will make our code and pre-trained models for 12 languages publicly available.
- (2) We conduct a detailed experimental study, identifying components of parser architecture that are really necessary to obtain a strongly performing system that is applicable across a wide range of languages. The final system uses XLM-R, no LSTM layer, and a factorized edge and label scoring architecture.
- (3) We show that multi-task setups predicting additional features as commonly employed in UD parsing may confound results for parsing for individual languages; we hence propose to compare parsing accuracy in unified evaluation settings in future work.
- (4) We show that our parser can be easily adapted

Parser	Pre-trained embed.	LSTM	MTL
StanfordNLP	word2vec, fastText	yes	no
UDPipe 2.0	word2vec, fastText	yes	yes
HIT-SCIR	fastText, ELMo	yes	no
UDify	mBERT	no	yes
Trankit	XLM-R	no	yes

Table 1: Settings for a number of previously state-of-the-art graph-based dependency parsers. “LSTM” states whether the parser makes use of an LSTM, and “MTL” states whether the parser is also trained to simultaneously predict other UD properties such as POS tags or morphological features.

to Enhanced UD parsing, also resulting in state-of-the-art performance (in terms of ELAS) for 5 out of 7 evaluated languages.

This paper is structured as follows. Sec. 2 gives the necessary background on relevant state-of-the-art neural graph-based dependency parsers as well as related work on analysing and comparing parsers. Sec. 3 describes the architecture and configuration options for our new STEPS parser, at the same time introducing the various factors studied in our experiments (Sec. 4). Sec. 5 presents the adaption of our system to Enhanced UD. Finally, we discuss implications for parser choice and future parser design (Sec. 6).

## 2 Related Work

This section provides a brief outline of the use of contextualized word embeddings in syntactic parsers, recently developed graph-based dependency parsers, and related work on dependency parser analysis.

**Contextualized Word Embeddings in Dependency Parsing.** Like in other sub-fields of natural language processing, using contextualized word embeddings has become the de-facto standard when building syntactic parsers. Dyer et al. (2015) use LSTM-based contextual representations for the stack and buffer in transition-based parsing, while Kiperwasser and Goldberg (2016) use BiLSTM-based feature representations for individual tokens in both graph-based and transition-based parsing. In both of these cases, the underlying LSTM is trained simultaneously with the target task. In contrast, recently the predominant approach towards contextualized word representations has been to pre-train systems on large-scale language modeling objectives, then taking their representations as

input for a target task, optionally while continuing to fine-tune them. This approach was initially proposed using an LSTM-based system (ELMo; Peters et al., 2018) and has since been transferred to transformers (e.g., BERT; Devlin et al., 2019). Transformer-based pre-trained language models have proven wildly successful and have become a standard method for a wide range of NLP tasks, including syntactic dependency parsing.

**Recent Graph-based Parsers.** Table 1 shows the configurations of three parsers that were among the best-performing systems in the CoNLL 2017 and CoNLL 2018 Shared Tasks on UD parsing, as well as the more recent UDify and Trankit parsers.

**StanfordNLP** (Dozat et al., 2017) was one of the first systems to apply the biaffine graph-based parser architecture to Universal Dependency parsing. Its token representations make use of pre-trained word2vec (Mikolov et al., 2013) embeddings that are contextualized using a BiLSTM. **UDPipe 2.0** (Straka, 2018) uses a multi-task setup in which POS and feature tagging, lemmatization, and dependency parsing share layers. The system was later extended (Straka et al., 2019, henceforth **UDPipe+**) by incorporating multilingual BERT (mBERT; Devlin et al., 2019) in its token representations. **HIT-SCIR** (Che et al., 2018) was one of the first UD parsers to make use of contextualized pre-trained word embeddings (in the form of ELMo; Peters et al., 2018). The model does not make use of a multi-task training setup. **UDify** (Kondratyuk and Straka, 2019) differs from previous UD parsers in two ways. First, it does not use an LSTM layer for token representation, instead using a learned scalar mixture of mBERT layers and fine-tuning mBERT during training. This is in contrast to the three aforementioned parsers, which do not fine-tune their pre-trained token embeddings. Second, UDify learns a single model for all languages, concatenating all UD 2.5 training sets. **Trankit** (Nguyen et al., 2021) is a recently released end-to-end UD parsing system built on the XLM-R language model. In contrast to UDify and our own STEPS parser, it does not fine-tune the entire language model, but instead inserts Adapter layers (Pfeiffer et al., 2020a,b) to efficiently create language-specific models for 56 languages.

**Multi-Purpose Parsers.** Other parsers with modular or extensible architectures include **Alto** (Gontrum et al., 2017), a prototyping tool for new

grammar formalisms based on Interpreted Regular Tree Grammars (IRTGs), and **PanParser** (Aufrant and Wisniewski, 2018), a modular framework for transition-based dependency parsing. In contrast to these two, STEPS is a graph-based dependency parser that focuses on easy configuration of different transformer-based language models and neural architecture variants.

**Parser Analyses and Comparisons.** Recent years have seen a wide range of studies comparing different language models for dependency parsing (e.g., Kanerva et al., 2018; Pyysalo et al., 2020; Smith et al., 2018). Additionally, several studies have investigated the amount of implicit syntactic information captured in pre-trained LMs such as ELMo and BERT (Tenney et al., 2019a,b; Hewitt and Manning, 2019). Conversely, several studies have investigated the utility of structural features for dependency parsing in the presence of LSTMs and/or contextualized word embeddings, generally finding that their impact is diminished in the presence of contextual information (Falenska and Kuhn, 2019; Fonseca and Martins, 2020).

Kulmizev et al. (2019) compare the effect of deep contextualized word embeddings on transition-based and graph-based dependency parsers, showing that their inclusion makes the two approaches virtually equivalent in terms of parsing accuracy. Our work is similar to theirs in the sense that we also evaluate several very different dimensions of parser architecture at the same time, utilizing the same underlying backbone and thus ensuring comparability across experiments.

### 3 STEPS: A Modular Graph-Based Dependency Parser

In this section, we describe our modular dependency parser STEPS (Stuttgart Transformer-based Extensible Parsing System). Each subsection focuses on a particular aspect of the parser setup, providing background on its usage and its potential impact on parser performance.

#### 3.1 Input Token Representation

STEPS provides a number of different options for input token representation. As Table 1 shows, parsers have made use of a variety of pre-trained embeddings, with transformer-based language models having become the predominant current approach. We hence focus on the latter and compare multilingual BERT (mBERT; Devlin et al.,

2019), language-specific BERTs (**langBERT**), and the multilingual **XLM-R-large** model (Conneau et al., 2020). XLM-R utilizes the pre-training optimizations first proposed for RoBERTa (Liu et al., 2019), which includes training on a considerably larger amount of data. A detailed overview of all transformer models used in our experiments is provided in the second column of Table 2.

STEPS represents each token  $i$  using a vector  $\mathbf{r}_i$  corresponding to the embedding of its first word-piece token. Following Kondratyuk and Straka (2019), we compute token embeddings as weighted sums of the representations of the respective tokens given by the internal transformer encoder layers, resulting in either 768- or 1024-dimensional embeddings depending on the transformer model used. Coefficients for this sum are learned during training, and layer dropout is applied in order to prevent the model from focusing on particular layers. Our model learns a different set of these coefficients for each output task (see Sec. 3.2 and Sec. 3.3 below). In addition to the above described **transformer-only** setting, we also compute another version of token embeddings by feeding the embeddings computed by the sum operations into a multi-layer bidirectional LSTM (BiLSTM), whose per-token output then constitutes  $\mathbf{r}_i$ .

### 3.2 Biaffine Classifier Architecture

STEPS makes use of biaffine classifiers as proposed by Dozat and Manning (2017), which have become the de-facto standard method for graph-based dependency parsing. In a first step, a head representation  $\mathbf{h}_i^{head}$  and a dependent representation  $\mathbf{h}_i^{dep}$  are created for each input token  $i$  represented as embedding vector  $\mathbf{r}_i$  via two single-layer feedforward neural networks:

$$\mathbf{h}_i^{head} = \text{FNN}^{head}(\mathbf{r}_i) \quad (1)$$

$$\mathbf{h}_i^{dep} = \text{FNN}^{dep}(\mathbf{r}_i) \quad (2)$$

These representations are then fed into the biaffine function, which maps head-dependent pairs  $(i, j)$  onto vectors  $\mathbf{s}_{i,j}$  of arbitrary size:

$$\mathbf{s}_{i,j} = \text{Biaff}(\mathbf{h}_i^{head}, \mathbf{h}_j^{dep}) \quad (3)$$

$$\text{Biaff}(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{x}_1^\top \mathbf{U} \mathbf{x}_2 + W(\mathbf{x}_1 \oplus \mathbf{x}_2) + \mathbf{b} \quad (4)$$

$\mathbf{U}$ ,  $W$  and  $\mathbf{b}$  are learned parameters;  $\oplus$  denotes the concatenation operation. The scores  $s_{i,j}$  can now be leveraged in different ways to construct an output tree or graph; this will be described next.

First, the **factorized** approach (Dozat and Manning, 2017) uses two instances of biaffine classifiers. The first classifier (the ‘‘arc scorer’’) is responsible for predicting which (unlabeled) edges exist in the output structure. It predicts, for each token, a probability distribution over potential syntactic heads (i.e., all other tokens in the sentence). We then feed the log-probabilities to the Chu-Liu/Edmonds maximum spanning tree algorithm (Chu and Liu, 1965; Edmonds, 1967) and label the resulting tree using the label scorer. The second classifier (the ‘‘label scorer’’) then assigns dependency labels to edges predicted in the first step.

The **unfactorized** approach, proposed by Dozat and Manning (2018) for semantic graph parsing, uses only a single biaffine classifier (namely the label scorer). Non-existence of dependencies is encoded using simply another label ( $\emptyset$ ). We adapt this approach to tree parsing by discarding the arc scorer and computing the edge weights for the Chu-Liu/Edmonds MST algorithm as  $\log(1 - P(\emptyset))$  in order to extract a labeled dependency tree directly. To the best of our knowledge, this is the first time that the unfactorized architecture has been applied to the parsing of dependency tree structures.

### 3.3 Multi-Task Training

We study the effects of a multi-task training setup by implementing two approaches to training our parser: (a) **dep-only**, in which the model is trained only on syntactic dependencies; and (b) multi-task learning (**MTL**), in which the model additionally predicts universal part-of-speech tags (UPOS) and morphological features (UFeats). We follow Kondratyuk and Straka (2019) by learning different coefficients for the transformer layers for these tagging tasks (see Sec. 3.1) and then using a single-layer feed-forward neural network to extract logit vectors over the respective label vocabularies. By default, the loss for the entire system is computed as the sum of losses for the individual output modules (UPOS tagger, UFeats tagger, and dependency parser). However, we also add the option of scaling the loss of the individual output modules in order to prevent individual tasks from overwhelming the system as a whole (see Sec. 4.6).

## 4 Experiments

This section describes our experimental setup and reports the results of our experiments on pre-trained embeddings (Sec. 4.3), factorized vs. unfactor-



Language	Transformer LM	UD Treebank
Arabic	ArabicBERT-large (Safaya et al., 2020)	PADT (Smrž et al., 2008)
Chinese	Chinese BERT (Devlin et al., 2019)	GSD
Czech	Slavic-BERT (Arhipov et al., 2019)	PDT (Bejček et al., 2012)
English	RoBERTa-large (Liu et al., 2019)	EWT (Silveira et al., 2014)
Finnish	FinBERT (Virtanen et al., 2019)	TDT
German	German BERT ( <a href="https://github.com/dbmdz/berts">github.com/dbmdz/berts</a> )	GSD (McDonald et al., 2013)
Hindi	WikiBERT-Hindi (Pyysalo et al., 2020)	HDTB (Bhat et al.; Palmer et al., 2009)
Italian	Italian BERT-XXL ( <a href="https://github.com/dbmdz/berts">github.com/dbmdz/berts</a> )	ISDT
Japanese	WikiBERT-Japanese (Pyysalo et al., 2020)	GSD
Korean	KR-BERT (Lee et al., 2020)	Kaist (Chun et al., 2018)
Latvian	WikiBERT-Latvian (Pyysalo et al., 2020)	LVTB
Russian	RuBERT (Kuratov and Arhipov, 2019)	SynTagRus (Droganova et al., 2018)
Multilingual	mBERT (Devlin et al., 2019)	–
Multilingual	XLNet (Conneau et al., 2020)	–

Table 2: Language models and UD treebanks used in our experiments. Citations for treebanks are given where provided in treebank repository documentation.

ized parser architecture (Sec. 4.5), LSTM usage (Sec. 4.4), and multi-task training (Sec. 4.6).

#### 4.1 Experimental Setup

**Languages and treebanks.** We select 12 languages, covering a diverse range of language families and writing systems, by applying linguistic criteria similar to those outlined by de Lhoneux et al. (2017). For each language, we select the largest available treebank from UD 2.6 for which token data is freely available. These treebanks are listed in the third column of Table 2. In all of our experiments, we use gold tokens and train language-specific models, testing on the test set of the respective treebank.

**Evaluation metrics.** We compute UAS and LAS using the official evaluation script for the CoNLL 2018 Shared Task.<sup>2</sup> UAS (Unlabeled Attachment Score) computes the fraction of tokens that have been assigned the correct syntactic head. LAS (Labeled Attachment Score) records the fraction of tokens that have been assigned the correct syntactic head with the correct edge label.

#### 4.2 Implementation

Our parser is implemented in Python, using PyTorch (Paszke et al., 2019) and the Huggingface Transformers library (Wolf et al., 2019). Training is performed on a single nVidia Tesla V100 GPU.

**Hyperparameters.** We aim to obtain a simple yet high-performing hyperparameter configuration. To do so, we start out with the configuration of

<sup>2</sup>[https://universaldependencies.org/conll118/conll118\\_ud\\_eval.py](https://universaldependencies.org/conll118/conll118_ud_eval.py)

UDify, which is architecturally quite similar to STEPS, and tune parameters using grid search in ca. 40 runs on a small development set (consisting of English, Arabic, and Korean data), aiming at a simplified setup that achieves good results across these diverse languages. The hyperparameters examined by us were

- Hidden size of the biaffine classifier (256 / 512 / 768 / 1024)
- Batch size (16 / 32)
- Base learning rate ( $7e^{-6}$  to  $5e^{-5}$ )
- Early stopping patience (10 / 15 / 20 epochs)
- Learning rate schedule (constant LR / warmup only / cosine annealing / Noam)

In large part, our final settings are identical to UDify’s values with the following differences. We use the AdamW optimizer (Loshchilov and Hutter, 2019) instead of Adam; we perform neither label smoothing nor gradient clipping; and we do not use differential learning rates. In addition, we do not train for a fixed number of epochs, but instead stop once performance on the validation set does not increase for 15 epochs, or after at most 24 hours.

For model variants involving LSTMs, we tuned the hyperparameters involved in these layers (number of layers; hidden size; dropout; learning rate) in a second round of optimization consisting of 15 trials of random search on the English data. We then picked the two best-performing models and ran them on the other languages, finding that one of them performed best on all languages.

<b>Transformer LM</b>	
Token mask probability	0.15
Layer dropout	0.1
Hidden dropout	0.2
Attention dropout	0.2
Output dropout	0.5
<b>Biaffine classifier</b>	
Arc scorer dimension	768 or 1024 <sup>a</sup>
Label scorer dimension	256 or 768/1024 <sup>b</sup>
Dropout	0.33
<b>LSTM</b>	
Hidden size	330
Number of layers	3
Dropout	0.5
LSTM learning rate	$5e^{-4}$
<b>Optimization</b>	
Optimizer	AdamW
$\beta_1, \beta_2$	0.9, 0.999
Weight decay	0
Batch size	32
Base learning rate	$4e^{-5}$
LR schedule	Noam
LR warmup	1 epoch

Table 3: Hyperparameter values. <sup>a</sup>Identical to hidden size of the transformer encoder. <sup>b</sup>256 in factorized models, hidden size of transformer encoder in unfactorized models.

All of our final hyperparameter settings can be found in Table 3.

### 4.3 Impact of Pre-Trained Word Embeddings

We first evaluate how parsing performance differs when varying the underlying pre-trained language model. We here do not include an LSTM layer and perform only dependency parsing. Table 4 shows results for all 12 treebanks used in this study. UD-Pipe+ refers to the version of UD-Pipe enhanced with BERT and Flair embeddings proposed by Straka et al. (2019) and described in Sec. 2. UDify refers to the original system trained on all UD languages without treebank-specific fine-tuning. As multilingual training usually results in improved performance for low-resource languages at the cost of lowering scores for high-resource languages (Üstün et al., 2020), for meaningful comparison, we train UDify<sub>mono</sub> on single treebanks. Trankit<sub>large</sub> refers to the version of Trankit which uses XLM-R-large as the underlying language model, same as STEPS<sub>XLM-R</sub>.

STEPS<sub>mBERT</sub> roughly corresponds to UDify<sub>mono</sub>, and indeed the models overall perform similarly. We attribute differences to slightly different training setups. While UDify is trained for 80 epochs, STEPS employs early stopping after 15 epochs without improvement. Moreover, we did not dis-

able multi-task learning for parallel UD feature prediction in UDify<sub>mono</sub>, and this may be an explanation why STEPS<sub>mBERT</sub> does much better on Finnish, Czech and Russian, where morphological features may be harder to predict. (For a principled comparison of multi-task setups, see Sec. 4.6.) By contrast, UD-Pipe+ often outperforms UDify, UDify<sub>mono</sub>, and STEPS<sub>mBERT</sub>, which is likely due to the fact that it trains its own word embeddings in addition to mBERT and additionally makes use of character-level representations via GRUs.

Parsing accuracy of STEPS is very high across the board, with new state-of-the-art results being achieved on all languages except Japanese and German. For most languages, the best results are achieved using STEPS<sub>XLM-R</sub>, with STEPS<sub>langBERT</sub> coming in second. In contrast, using mBERT is not the best option on any treebank. In fact, the only languages for which mBERT achieves better results than langBERT in our experiments are Latvian and Hindi.<sup>3</sup> While using langBERT usually yields worse parsing accuracy than XLM-R, results are roughly on par for Arabic and English. We note that the language-specific models we chose for these treebanks (ArabicBERT-large and RoBERTa-large, respectively) are the only ones with a number of trainable parameters similar to XLM-R, while all others have a considerably smaller number of parameters. This highlights the importance of model size in pre-trained word embeddings.

STEPS<sub>XLM-R</sub> and Trankit<sub>large</sub> show rather similar performance overall, which is to be expected given the fact that both are built on the same underlying language model (XLM-R-large). The slight advantage for STEPS<sub>XLM-R</sub> observed on most languages may stem from the fact that it fine-tunes the entire transformer model instead of merely adding Adapter layers, and that it does not use a multi-task training setup (cf. Sec. 4.6). Interestingly, on Finnish and Latvian, both systems outperform other existing parsers by very large margins (around 4.9 and 6.5 LAS, respectively). We assume that there are two main reasons for this. First, XLM-R is pre-trained on CommonCrawl data (Conneau et al., 2020; Wenzek et al., 2020) as opposed to Wikipedia dumps, which results not only in several orders

<sup>3</sup>In a similar study comparing mBERT- and langBERT-based parsers, Kanerva et al. (2020) also found Latvian to be one of the few languages for which mBERT outperformed the language-specific (WikiBERT) version. Both the Latvian and the Hindi Wikipedias are rather small, consisting of only 21M and 35M tokens, respectively (Pyysalo et al., 2020).

	ar PADT	cs PDT	de GSD	en EWT	fi TDT	hi HDTB	it ISDT	ja GSD	ko Kaist	lv LVTB	ru STR	zh GSD
UDPipe+	84.62	92.56	84.06	90.40	89.49	92.50	93.38	<b>94.27</b>	87.54	84.50	93.68	86.74
UDify	82.88	92.88	83.59	88.50	82.03	91.46	93.69	92.08	84.52	85.09	93.13	83.75
UDify <sub>mono</sub>	83.34	91.58	84.28	89.52	86.74	91.44	93.14	92.14	86.45	85.45	92.32	82.95
Trankit <sub>large</sub>	86.51	93.11	<b>86.27</b>	91.64	94.31	93.17	94.63	78.14	40.76	91.76	95.16	87.38
STEPS <sub>mBERT</sub>	83.80	92.69	84.08	89.16	88.91	91.30	93.13	92.22	24.49	85.05	93.74	84.94
STEPS <sub>langBERT</sub>	<b>86.60</b>	92.99	85.87	<b>91.98</b>	93.57	91.16	94.25	92.98	84.56	82.61	94.44	86.20
STEPS <sub>XLM-R</sub>	86.55	<b>94.58</b>	86.07	91.91	<b>94.36</b>	<b>93.34</b>	<b>94.86</b>	94.10	<b>89.93</b>	<b>91.93</b>	<b>95.30</b>	<b>87.75</b>
STEPS <sub>XLM-R-LSTM</sub>	86.41	94.52	86.20	91.58	93.92	93.28	94.57	94.01	89.91	91.61	95.27	86.96
STEPS <sub>XLM-R-unfact</sub>	86.32	94.38	86.19	91.57	94.11	93.21	94.58	93.58	89.86	91.76	95.17	87.47

Table 4: Labeled Attachment Score (LAS) for **basic dependency parsing** varying input embeddings and architecture. STEPS scores are averages of three runs.

of magnitude more training data (over 1 billion tokens for both languages), but also presumably more heterogenous data, which may provide better generalizations for the domains in our test data.<sup>4</sup> Second, XLM-R has a much larger vocabulary size than mBERT (250k vs. 100k), which means it may account better for the rich morphology of these languages. On average, a Finnish (Latvian) token is split up into 2.4 (2.1) word pieces when using mBERT, but only 1.9 (1.8) word pieces when using XLM-R.

Finally, we note that STEPS<sub>mBERT</sub> and Trankit<sub>large</sub> perform extremely poorly on Korean (24.49/40.76 LAS on average), indicating that the models do not properly learn from the data. We assume that this may be a tokenization or character encoding issue unique to the Korean-Kaist treebank.<sup>5</sup> However, a similar pattern is not observed for any of the other parser models, and we were unfortunately unable to identify the exact cause despite our best efforts.

#### 4.4 Impact of LSTM Layer

We evaluate the performance of a system identical to STEPS<sub>XLM-R</sub> described above, but with 3 additional BiLSTM layers added on top of the language model (STEPS<sub>XLM-R-LSTM</sub> in Table 4). Changes in performance are generally small. With the exception of German, including LSTM layers actually decreases parsing accuracy slightly. The LSTM model contains more trainable parameters and also

<sup>4</sup>Both fi-TDT and lv-LVTB contain, among others, “non-standard” data such as blog entries, legal texts, and spoken language (Haverinen et al., 2014; Pretkalniņa et al., 2018).

<sup>5</sup>As pointed out by an anonymous reviewer, Korean-Kaist uses a rather different tokenization strategy than other UD treebanks, with tokens corresponding to larger chunks. Relying on just the first word pieces for token embeddings may be problematic in this context.

makes use of differential learning rates, yet, we did not find any meaningful differences in convergence speed and training times. Hence, we conclude that when fine-tuning an underlying transformer-based language model, adding LSTM layers on top is not necessary. However, results may differ for systems that additionally train their own token embeddings or make use of character-based representations, both of which we do not address in our experiments.

#### 4.5 Impact of Factorization

Dozat and Manning (2018) show that for semantic dependency graph parsing, a simplified parser architecture predicting edge presence and edge labels from the same scoring matrix achieves largely identical results compared to a model using two separate classifiers for arcs and labels. We here dive into the question whether such an unfactorized approach is also able to achieve competitive results in syntactic tree parsing. We do so by implementing a version of STEPS<sub>XLM-R</sub> that makes use of the unfactorized approach as described in Sec. 3.2.

Results of our experiments can be found in the row labeled STEPS<sub>XLM-R-unfact</sub> in Table 4. Overall, performance of the unfactorized approach is very close to the factorized version, but slightly lower. While this shows that the unfactorized approach is indeed viable for tree parsing, analysis of the training times reveals an increase by ca. 30% on average when using the unfactorized model, indicating that the shared scorer takes a longer time to converge.

In light of these results, we propose to stick with the factorized version for syntactic tree parsing. At least in a research setting, shorter training times allow for a larger set of experiments and thus ultimately in using fewer resources. When applying

the parser, differences in model size and parsing time are negligible.

#### 4.6 Impact of Multi-Task Approach

Finally, we analyze how performance changes when predicting UPOS and UFeats in addition to dependencies. For these experiments, we use XLM-R as input embeddings and a factorized architecture. For UFeats, we follow UDify’s approach and consider each possible combination of morphological features a unique label. As shown in Table 5, STEPS<sub>MTL</sub> achieves very high accuracies for UPOS and UFeats, performing on par with or only slightly worse than the previous state of the art (Trankit<sub>large</sub>) for most languages. However, we find that compared to the dependency-only system, parsing accuracy drops considerably in the multi-task setting (up to over 1 LAS for Finnish).

During training of STEPS<sub>MTL</sub>, accuracy on the validation set increased very rapidly for the tagging tasks and reached levels close to the final values after only a few epochs, while accuracy for the parsing task increased much slower. This suggests that the loss for the tagging tasks might overwhelm the system as a whole, causing the parser modules to underfit. We therefore also test STEPS<sub>MTLscale</sub>, in which the loss for UPOS and UFeats is scaled down to 5% during training. STEPS<sub>MTLscale</sub> performs close to STEPS<sub>MTL</sub>, even outperforming it in the case of Hindi. In turn, however, accuracy for UPOS and particularly UFeats drops considerably.

To sum up, our experiments indicate that multi-task setups as commonly employed in UD parsing have a non-negligible effect on parsing performance. Hence, when comparing parser performance, it is crucial to take potential multi-task setups into account. If the respective setups differ, ignoring them may result in misleading interpretations of parsing performance of model architectures (unless the variable of interest is the multi-task setup itself).

#### 4.7 Summary

Our experimental findings can be summarized as follows: (a) Choice of pre-trained embeddings has the greatest impact on parser performance, with XLM-R yielding the best results in most cases; (b) adding LSTM layers is not necessary when working with a large fine-tuned language model; (c) a factorized parser architecture is preferable due to faster training; (d) when using a multi-task approach incorporating UPOS and UFeats prediction,

there is a tradeoff between tagging and parsing accuracy, and conclusions regarding architecture should be drawn by comparing experiments performed in the same setting. Crucially, one of the simplest parsers in our evaluation (STEPS<sub>XLM-R</sub>) achieves the best results overall, often surpassing more complex previous work.

### 5 Enhanced UD Parsing with STEPS

In order to determine whether our conclusions also hold for the related graph parsing task of Enhanced UD (Schuster and Manning, 2016), we run an additional batch of experiments on 7 treebanks from the IWPT 2020 Shared Task (Bouma et al., 2020).

**Modifications to STEPS.** We modify STEPS to generate dependency graphs using a factorized approach as proposed by Dozat and Manning (2018) for semantic dependency parsing, weighting the losses of the edge and label scorers:

$$\ell = \lambda_{edge}\ell_{edge} + \lambda_{label}\ell_{label}. \quad (5)$$

After tuning on English in a set of preliminary experiments, we set the hyperparameters  $\lambda_{edge}$  to 1.0 and  $\lambda_{label}$  to 0.05. For comparison, we also evaluate the unfactorized version of our parser.

While enhanced UD does not require output graphs to be trees, it imposes the constraint that every node must be reachable from the root. We use the heuristic proposed by Grünewald and Friedrich (2020) for graph post-processing, which greedily adds the highest-scoring edge from a node that is reachable from the root to a node that is unreachable from the root until the condition is fulfilled.

Furthermore, for certain relations such as *nmod* or *obl*, enhanced UD allows for the inclusion of lexical material (such as prepositions) in dependency labels. To avoid data sparsity issues resulting from the increase in the number of dependency labels, we follow the label de- and re-lexicalization strategy proposed by Grünewald and Friedrich (2020), replacing lexical materials in labels with placeholders such as *obl:[case]*. At prediction time, lexicalized parts of the labels can be retrieved from the respective child nodes in the graph. We apply this strategy for all languages in our study except Finnish and Russian (which do not have lexicalized labels) and Arabic (for which we additionally look up lemmas of the lexical material using a simple majority baseline method).



TREEBANK	MODEL	UPOS	UFEATS	UAS	LAS
Arabic (ar_padt)	Trankit <sub>large</sub>	95.47	<b>95.54</b>	90.90	86.51
	STEPS <sub>dep-only</sub>	–	–	<b>90.96</b>	<b>86.55</b>
	STEPS <sub>MTL</sub>	<b>97.24</b>	94.89	90.34	86.01
	STEPS <sub>MTLscale</sub>	96.47	87.74	90.80	86.41
Czech PDT (cs_pdt)	Trankit <sub>large</sub>	99.37	<b>98.23</b>	95.51	93.11
	STEPS <sub>dep-only</sub>	–	–	<b>95.89</b>	<b>94.58</b>
	STEPS <sub>MTL</sub>	<b>99.41</b>	98.06	95.59	94.19
	STEPS <sub>MTLscale</sub>	98.97	94.20	95.85	94.52
German (de_gsd)	Trankit <sub>large</sub>	<b>95.48</b>	<b>91.91</b>	<b>90.11</b>	<b>86.27</b>
	STEPS <sub>dep-only</sub>	–	–	90.02	86.07
	STEPS <sub>MTL</sub>	95.40	<b>91.91</b>	89.53	85.46
	STEPS <sub>MTLscale</sub>	94.65	83.33	89.74	85.80
English (en_ewt)	Trankit <sub>large</sub>	<b>97.91</b>	<b>98.04</b>	93.59	91.64
	STEPS <sub>dep-only</sub>	–	–	<b>93.90</b>	<b>91.91</b>
	STEPS <sub>MTL</sub>	97.84	98.02	93.47	91.50
	STEPS <sub>MTLscale</sub>	96.58	96.49	93.80	91.78
Finnish TDT (fi_tdt)	Trankit <sub>large</sub>	<b>98.72</b>	<b>97.07</b>	95.55	94.31
	STEPS <sub>dep-only</sub>	–	–	<b>95.69</b>	<b>94.36</b>
	STEPS <sub>MTL</sub>	98.52	96.75	94.62	93.11
	STEPS <sub>MTLscale</sub>	98.19	88.70	95.59	94.26
Hindi (hi_hdtb)	Trankit <sub>large</sub>	<b>98.12</b>	93.98	96.16	93.17
	STEPS <sub>dep-only</sub>	–	–	96.11	93.34
	STEPS <sub>MTL</sub>	98.09	<b>94.49</b>	95.96	93.03
	STEPS <sub>MTLscale</sub>	97.51	88.60	<b>96.18</b>	<b>93.39</b>
Italian (it_isdt)	Trankit <sub>large</sub>	98.80	98.43	95.93	94.63
	STEPS <sub>dep-only</sub>	–	–	<b>96.25</b>	<b>94.86</b>
	STEPS <sub>MTL</sub>	<b>98.81</b>	<b>98.58</b>	95.80	94.36
	STEPS <sub>MTLscale</sub>	98.43	94.28	96.09	94.69
Japanese (ja_gsd)	Trankit <sub>large</sub>	92.57	97.58	86.62	78.14
	STEPS <sub>dep-only</sub>	–	–	<b>95.62</b>	<b>94.10</b>
	STEPS <sub>MTL</sub>	<b>98.21</b>	<b>99.98</b>	95.38	93.78
	STEPS <sub>MTLscale</sub>	96.94	99.91	95.53	94.00
Korean (ko_kaist)	Trankit <sub>large</sub>	69.86	98.95	67.96	40.76
	STEPS <sub>dep-only</sub>	–	–	<b>91.71</b>	<b>89.93</b>
	STEPS <sub>MTL</sub>	<b>96.41</b>	<b>100.00</b>	91.48	89.73
	STEPS <sub>MTLscale</sub>	93.90	<b>100.00</b>	91.53	89.77
Latvian (lv_lvttb)	Trankit <sub>large</sub>	<b>97.83</b>	<b>95.38</b>	94.19	91.76
	STEPS <sub>dep-only</sub>	–	–	<b>94.32</b>	<b>91.93</b>
	STEPS <sub>MTL</sub>	97.73	94.79	93.44	90.99
	STEPS <sub>MTLscale</sub>	96.72	81.42	93.97	91.61
Russian (ru_syntagrus)	Trankit <sub>large</sub>	<b>99.34</b>	<b>98.47</b>	96.18	95.16
	STEPS <sub>dep-only</sub>	–	–	<b>96.32</b>	<b>95.30</b>
	STEPS <sub>MTL</sub>	99.27	98.34	95.94	94.88
	STEPS <sub>MTLscale</sub>	98.99	95.91	96.19	95.20
Chinese (zh_gsd)	Trankit <sub>large</sub>	96.83	99.48	90.03	87.38
	STEPS <sub>dep-only</sub>	–	–	<b>90.72</b>	<b>87.75</b>
	STEPS <sub>MTL</sub>	<b>97.20</b>	<b>99.50</b>	89.70	86.86
	STEPS <sub>MTLscale</sub>	95.21	98.53	90.31	87.39

Table 5: Results for **basic dependency parsing** vs. **parsing and feature prediction (multi-task)** for STEPS<sub>XLM-R</sub>. Scores are averages of three runs. For UPOS and UFeats, we report accuracy.

**Experimental Results.** We compare our results against **TurkuNLP**, a modified version of UDify which scored 1st in the official evaluation of the IWPT 2020 Shared Task, and **ShanghaiTech**, which scored 1st in the unofficial post-evaluation.

We evaluate in terms of ELAS (Enhanced LAS, i.e., F1 score over the set of enhanced dependencies in the system output and the gold standard) using the official evaluation script for the IWPT 2020 Shared Task<sup>6</sup> and report per-treebank results for TurkuNLP and ShanghaiTech as submitted.<sup>7</sup> To ensure comparability with previous work, we compute our results using raw text as input and using Stanza (Qi et al., 2020) for tokenization and sentence segmentation. Table 6 reports our results. Our parser achieves very high accuracy, outperforming TurkuNLP and ShanghaiTech on all evaluated languages except Arabic and Czech. Notably, the latter system also uses XLM-R embeddings, but with a more complex parser architecture.

Unlike in tree parsing, the unfactorized system actually slightly outperforms the factorized system on a number of languages, with the largest margins

<sup>6</sup>[https://universaldependencies.org/iwpt20/iwpt20\\_xud\\_eval.py](https://universaldependencies.org/iwpt20/iwpt20_xud_eval.py)

<sup>7</sup><https://universaldependencies.org/iwpt20/Results.html>

	Turku-NLP	Shanghai-Tech	STEPS <sub>XLM-R</sub>	
			fact	unfact
ar-PADT	<b>77.83</b>	77.73	77.42	77.68
cs-PDT	88.17	<b>90.63</b>	89.49	89.43
en-EWT	86.14	86.30	87.11	<b>87.28</b>
fi-TDT	89.24	89.97	<b>91.53</b>	91.51
it-ISDT	91.54	91.49	92.35	<b>92.39</b>
lv-LVTB	84.94	87.64	<b>89.04</b>	88.95
ru-STR	90.69	92.31	93.68	<b>93.75</b>

Table 6: Results (ELAS) for **enhanced dependency parsing**. Scores are averages of three runs.

on Arabic and English. Taken together, these results show that (a) our best approach is not only robust across languages, but also across (syntactic) parsing tasks, and (b) the unfactorized approach may be well-suited to graph parsing tasks, which is in line with the results of Dozat and Manning (2018).

## 6 Discussion and Conclusion

In this paper, we have performed a detailed and principled analysis on a variety of decisions arising during dependency parser design. **What works?** We have identified an architecture based on fine-tuned XLM-R embeddings and factorized scoring

that lead to new state-of-the-art performance for 11 out of 12 diverse language in our study on basic UD parsing, and for 5 out of 7 languages for enhanced UD parsing. **What doesn't?** Adding LSTM layers on top of the transformer leads to a decrease in accuracy in most cases. We have also shown that multi-task setups predicting UPOS and UFeats often degrade parsing performance. **What is really necessary?** For current state-of-the-art UD parsers, we recommend making sure that the pre-trained language model covers the intended domain well. In addition, keeping a factorized approach is a good idea for tree parsing, while in graph parsing, a single scorer module may suffice.

In this paper, we have addressed a high- to medium-resource scenario, assuming that we know the application language of a parser and thus training a single parser per language. Future work may address multilingual approaches such as the training setup used by UDify or the recently proposed UDapter (Üstün et al., 2020), which aims at boosting performance of low-resource languages while keeping performance of high-resource languages high. Furthermore, it would be interesting to see if our results about biaffine architectures also hold for non-syntactic tasks that have recently been framed as dependency parsing tasks, such as Named Entity Recognition (Yu et al., 2020), negation scope detection (Kurtz et al., 2020) or Semantic Role Labeling (Shi et al., 2020).

To sum up, in this paper we have applied “Occam’s razor” to graph-based dependency parsing. We believe that the insights from our study will foster further research on dependency parsing and on framing other tasks as dependency parsing, taking our simplified but robustly performing STEPS parser as a starting point.

## Acknowledgments

We thank Agnieszka Falenska, Heike Adel, Lukas Lange, Hendrik Schuff, Jannik Strötgen, as well as the anonymous reviewers for their valuable comments with regard to this work.

## References

- Mikhail Arhipov, Maria Trofimova, Yuri Kuratov, and Alexey Sorokin. 2019. [Tuning multilingual transformers for language-specific named entity recognition](#). In *Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing*, pages 89–93, Florence, Italy. Association for Computational Linguistics.
- Lauriane Aufrant and Guillaume Wisniewski. 2018. Panparser: a modular implementation for efficient transition-based dependency parsing. *The Prague Bulletin of Mathematical Linguistics*, 111(1):57–86.
- Eduard Bejček, Jarmila Panevová, Jan Popelka, Pavel Straňák, Magda Ševčíková, Jan Štěpánek, and Zdeněk Žabokrtský. 2012. [Prague Dependency Treebank 2.5 – a revisited version of PDT 2.0](#). In *Proceedings of COLING 2012*, pages 231–246, Mumbai, India. The COLING 2012 Organizing Committee.
- Riyaz Ahmad Bhat, Rajesh Bhatt, Annahita Farudi, Prescott Klassen, Bhuvana Narasimhan, Martha Palmer, Owen Rambow, Dipti Misra Sharma, Ashwini Vaidya, Sri Ramagurumurthy Vishnu, et al. The hindi/urdu treebank project. In *Handbook of Linguistic Annotation*. Springer Press.
- Gosse Bouma, Djamé Seddah, and Daniel Zeman. 2020. [Overview of the IWPT 2020 shared task on parsing into enhanced Universal Dependencies](#). In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 151–161, Online. Association for Computational Linguistics.
- Wanxiang Che, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu. 2018. [Towards better UD parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation](#). In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64, Brussels, Belgium. Association for Computational Linguistics.
- Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14:1396–1400.
- Jayeol Chun, Na-Rae Han, Jena D. Hwang, and Jinho D. Choi. 2018. [Building Universal Dependency treebanks in Korean](#). In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).
- Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. [Unsupervised cross-lingual representation learning at scale](#). pages 8440–8451.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. [BERT: Pre-training of deep bidirectional transformers for language understanding](#). In *Proceedings of the 2019 Conference of the North American Chapter of the Association*

- for *Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.
- Timothy Dozat and Christopher D. Manning. 2017. [Deep biaffine attention for neural dependency parsing](#). In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.
- Timothy Dozat and Christopher D. Manning. 2018. [Simpler but more accurate semantic dependency parsing](#). In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 484–490, Melbourne, Australia. Association for Computational Linguistics.
- Timothy Dozat, Peng Qi, and Christopher D. Manning. 2017. [Stanford’s graph-based neural dependency parser at the CoNLL 2017 shared task](#). In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30, Vancouver, Canada. Association for Computational Linguistics.
- Kira Drogonova, Olga Lyashevskaya, and Daniel Zeman. 2018. Data conversion and consistency of monolingual corpora: Russian ud treebanks. In *Proceedings of the 17th International Workshop on Treebanks and Linguistic Theories (TLT 2018), December 13–14, 2018, Oslo University, Norway*, 155, pages 52–65. Linköping University Electronic Press.
- Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. [Transition-based dependency parsing with stack long short-term memory](#). In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343, Beijing, China. Association for Computational Linguistics.
- Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the national Bureau of Standards B*, 71(4):233–240.
- Agnieszka Falenska and Jonas Kuhn. 2019. [The \(non-\)utility of structural features in BiLSTM-based dependency parsers](#). In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 117–128, Florence, Italy. Association for Computational Linguistics.
- Erick Fonseca and André F. T. Martins. 2020. [Revisiting higher-order dependency parsers](#). In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8795–8800, Online. Association for Computational Linguistics.
- Johannes Gontrum, Jonas Groschwitz, Alexander Koller, and Christoph Teichmann. 2017. [Alto: Rapid prototyping for parsing and translation](#). In *Proceedings of the Software Demonstrations of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 29–32, Valencia, Spain. Association for Computational Linguistics.
- Stefan Grünewald and Annemarie Friedrich. 2020. [RobertNLP at the IWPT 2020 shared task: Surprisingly simple enhanced UD parsing for English](#). In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 245–252, Online. Association for Computational Linguistics.
- Katri Haverinen, Jenna Nyblom, Timo Viljanen, Veronika Laippala, Samuel Kohonen, Anna Missilä, Stina Ojala, Tapio Salakoski, and Filip Ginter. 2014. Building the essential resources for finnish: the turku dependency treebank. *Language Resources and Evaluation*, 48(3):493–531.
- John Hewitt and Christopher D. Manning. 2019. [A structural probe for finding syntax in word representations](#). In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4129–4138, Minneapolis, Minnesota. Association for Computational Linguistics.
- Jenna Kanerva, Filip Ginter, Niko Miekka, Akseli Leino, and Tapio Salakoski. 2018. [Turku neural parser pipeline: An end-to-end system for the CoNLL 2018 shared task](#). In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 133–142, Brussels, Belgium. Association for Computational Linguistics.
- Jenna Kanerva, Filip Ginter, and Sampo Pyysalo. 2020. [Turku enhanced parser pipeline: From raw text to enhanced graphs in the IWPT 2020 shared task](#). In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 162–173, Online. Association for Computational Linguistics.
- Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.
- Dan Kondratyuk and Milan Straka. 2019. [75 languages, 1 model: Parsing universal dependencies universally](#). 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)*, pages 2779–2795, Hong Kong, China. Association for Computational Linguistics.



- Artur Kulmizev, Miryam de Lhoneux, Johannes Gontrum, Elena Fano, and Joakim Nivre. 2019. [Deep contextualized word embeddings in transition-based and graph-based dependency parsing - a tale of two parsers revisited](#). 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)*, pages 2755–2768, Hong Kong, China. Association for Computational Linguistics.
- Yuri Kuratov and Mikhail Arkhipov. 2019. Adaptation of deep bidirectional multilingual transformers for russian language. *arXiv preprint arXiv:1905.07213*.
- Robin Kurtz, Stephan Oepen, and Marco Kuhlmann. 2020. [End-to-end negation resolution as graph parsing](#). In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 14–24, Online. Association for Computational Linguistics.
- Sangah Lee, Hansol Jang, Yunmee Baik, Suzi Park, and Hyopil Shin. 2020. Kr-bert: A small-scale korean-specific language model. *arXiv preprint arXiv:2008.03979*.
- Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2017. Old school vs. new school: Comparing transition-based parsers with and without neural network enhancement. In *The 15th Treebanks and Linguistic Theories Workshop (TLT)*, pages 99–110.
- 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*.
- Ilya Loshchilov and Frank Hutter. 2019. [Decoupled weight decay regularization](#). In *International Conference on Learning Representations*.
- Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. 2014. [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. European Language Resources Association (ELRA).
- Ryan McDonald, Joakim Nivre, Yvonne Quirnbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. 2013. [Universal Dependency annotation for multilingual parsing](#). In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 92–97, Sofia, Bulgaria. Association for Computational Linguistics.
- Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. [Efficient estimation of word representations in vector space](#). In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*.
- Minh Van Nguyen, Viet Dac Lai, Amir Poursan Ben Veyseh, and Thien Huu Nguyen. 2021. [Trankit: A light-weight transformer-based toolkit for multilingual natural language processing](#). In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 80–90, Online. Association for Computational Linguistics.
- Martha Palmer, Rajesh Bhatt, Bhuvana Narasimhan, Owen Rambow, Dipti Misra Sharma, and Fei Xia. 2009. Hindi syntax: Annotating dependency, lexical predicate-argument structure, and phrase structure. In *The 7th International Conference on Natural Language Processing*, pages 14–17.
- Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. [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.
- 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)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.
- Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020a. [AdapterHub: A framework for adapting transformers](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 46–54, Online. Association for Computational Linguistics.
- Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020b. [MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online. Association for Computational Linguistics.



- Lauma Pretkalniņa, Laura Rituma, and Baiba Saulīte. 2018. Deriving enhanced universal dependencies from a hybrid dependency-constituency treebank. In *International Conference on Text, Speech, and Dialogue*, pages 95–105. Springer.
- Sampo Pyysalo, Jenna Kanerva, Antti Virtanen, and Filip Ginter. 2020. Wikibert models: deep transfer learning for many languages. *arXiv preprint arXiv:2006.01538*.
- 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*.
- Ali Safaya, Moutasem Abdullatif, and Deniz Yuret. 2020. Kuisail at semeval-2020 task 12: Bert-cnn for offensive speech identification in social media.
- Sebastian Schuster and Christopher D. Manning. 2016. Enhanced English Universal Dependencies: An improved representation for natural language understanding tasks. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 2371–2378, Portorož, Slovenia. European Language Resources Association (ELRA).
- Tianze Shi, Igor Malioutov, and Ozan Irsoy. 2020. Semantic role labeling as syntactic dependency parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7551–7571, Online. Association for Computational Linguistics.
- Natalia Silveira, Timothy Dozat, Marie-Catherine de Marneffe, Samuel Bowman, Miriam Connor, John Bauer, and Christopher D. Manning. 2014. A gold standard dependency corpus for English. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*.
- Aaron Smith, Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2018. An investigation of the interactions between pre-trained word embeddings, character models and POS tags in dependency parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2711–2720, Brussels, Belgium. Association for Computational Linguistics.
- Otakar Smrž, Viktor Bieličický, Iveta Kouřilová, Jakub Kráčmar, Jan Hajič, and Petr Zemánek. 2008. Prague arabic dependency treebank: A word on the million words. In *Proceedings of the Workshop on Arabic and Local Languages (LREC 2008)*, pages 16–23, Marrakech, Morocco.
- Milan Straka. 2018. UDPipe 2.0 prototype at CoNLL 2018 UD shared task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 197–207, Brussels, Belgium. Association for Computational Linguistics.
- Milan Straka, Jana Straková, and Jan Hajič. 2019. Evaluating contextualized embeddings on 54 languages in pos tagging, lemmatization and dependency parsing. *arXiv preprint arXiv:1908.07448*.
- Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019a. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.
- Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Sam Bowman, Dipanjan Das, and Ellie Pavlick. 2019b. What do you learn from context? probing for sentence structure in contextualized word representations. In *International Conference on Learning Representations*.
- Ahmet Üstün, Arianna Bisazza, Gosse Bouma, and Gertjan van Noord. 2020. UDapter: Language adaptation for truly Universal Dependency parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2302–2315, Online. Association for Computational Linguistics.
- Antti Virtanen, Jenna Kanerva, Rami Ilo, Jouni Luoma, Juhani Luotolahti, Tapio Salakoski, Filip Ginter, and Sampo Pyysalo. 2019. Multilingual is not enough: Bert for finnish. *arXiv preprint arXiv:1912.07076*.
- Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, and Edouard Grave. 2020. CCNet: Extracting high quality monolingual datasets from web crawl data. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 4003–4012, Marseille, France. European Language Resources Association.
- Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2019. Huggingface’s transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.
- Juntao Yu, Bernd Bohnet, and Massimo Poesio. 2020. Named entity recognition as dependency parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6470–6476, Online. Association for Computational Linguistics.
- Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and

Slav Petrov. 2018. CoNLL 2018 shared task: Multilingual parsing from raw text to Universal Dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium. Association for Computational Linguistics.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajič, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gokirmak, Anna Nedoluzhko, Silvie Cinková, Jan Hajič jr., Jaroslava Hlaváčová, Václava Kettnerová, Zdeňka Urešová, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher D. Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Lung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria de Paiva, Kira Drohanova, Héctor Martínez Alonso, Çağrı Çöltekin, Umut Sulubacak, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadová, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonça, Tatiana Lando, Rattima Nitisaroj, and Josie Li. 2017. CoNLL 2017 shared task: Multilingual parsing from raw text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19, Vancouver, Canada. Association for Computational Linguistics.