Recursive Non-Autoregressive Graph-to-Graph Transformer for Dependency Parsing with Iterative Refinement

Alireza Mohammadshahi
Idiap Research Institute and EPFL / Switzerland

James Henderson
Idiap Research Institute / Switzerland
{alireza.mohammadshahi, james.henderson}@idiap.ch

Abstract

We propose the Recursive Non-autoregressive Graph-to-graph Transformer architecture (RNG-Tr) for the iterative refinement of arbitrary graphs through the recursive application of a non-autoregressive Graph-to-Graph Transformer and apply it to syntactic dependency parsing. The Graph-to-Graph Transformer architecture of Mohammadshahi and Henderson (2019) has previously been used for autoregressive graph prediction, but here we use it to predict all edges of the graph independently, conditioned on a previous prediction of the same graph. We demonstrate the power and effectiveness of RNG-Tr on several dependency corpora, using a refinement model pre-trained with BERT (Devlin et al., 2018). We also introduce Dependency BERT (DepBERT), a non-recursive parser similar to our refinement model. RNG-Tr is able to improve the accuracy of a variety of initial parsers on 13 languages from the Universal Dependencies Treebanks and the English and Chinese Penn Treebanks, even improving over the new state-of-the-art results achieved by DepBERT, significantly improving the state-of-the-art for all corpora tested.

1 Introduction

Self-attention models, such as Transformer (Vaswani et al., 2017), have been hugely successful in a wide range of natural language processing (NLP) tasks, especially when combined with language-model pre-training, such as BERT (Devlin et al., 2018). These architectures contain a stack of self-attention layers which can capture long-range dependencies over the input sequence, while still representing its sequential order using absolute position encodings. Alternatively, Shaw et al. (2018) proposes to represent sequential order with relative position encodings, which are input to the self-attention functions.

Recently Mohammadshahi and Henderson (2019) extended this sequence input method to the input of arbitrary graph relations via the self-attention mechanism, and combined it with an attention-like function for graph relation prediction, resulting in their proposed Graph-to-Graph Transformer architecture (G2G-Tr). They demonstrated the effectiveness of G2G-Tr for transition-based dependency parsing and its compatibility with pre-trained BERT (Devlin et al., 2018). This parsing model predicts one edge of the parse graph at a time, conditioning on the graph of previous edges, so it is autoregressive in nature.

In this paper, we propose to take advantage of the graph-to-graph functionality of G2G-Tr to define a model that both predicts all edges of the graph in parallel, and is therefore non-autoregressive, and still captures between-edge dependencies, like an auto-regressive model. This is done by recursively applying a G2G-Tr model to correct the errors in a previous prediction of the output graph. At the point when no more corrections are made, all edges are predicted conditioned on all other edges in the output graph.

Our proposed Recursive Non-autoregressive Graph2graph Transformer (RNG-Tr) architecture first computes an initial graph with any given parsing model, even a trivial one. It then iteratively refines this graph by combining the G2G-Tr edge prediction step with a decoding step which finds the best graph given these predictions. The prediction of the model is the graph output by the final refinement step.

The RNG Transformer architecture can be applied to any task with a sequence or graph as input and a graph over the same set of nodes as output. We evaluate RNG-Tr on syntactic dependency parsing because it is a difficult structured prediction task, state-of-the-art initial parsers are extremely competitive, and there is little previous evidence that non-autoregressive models (as in graph-based dependency parsers) are not sufficient for this task. We aim to show that capturing correlations between dependencies with recursive non-autoregressive refinement results in improvements over state-of-the-art dependency parsers.

We propose the RNG Transformer model of graph-based dependency parsing. At each step of iterative refinement, the RNG-Tr model uses a G2G-Tr model to compute vector representation of nodes, then predict the scores for every possible dependency edge. These scores are then used to find a complete dependency graph, which is then inputted to the next step of the iterative refinement. The model stops when no changes are made to the graph, or a maximum number of iterations is reached.

The evaluation demonstrates improvements with several initial parsers, including previous state-of-the-art dependency parsers, and the empty parse. We also introduce a strong Transformer-based dependency parser which is initialised with a pre-trained BERT model (Devlin et al., 2018), called Dependency BERT (DepBERT), using it both for our initial parser and as the basis of our refinement model. Results on 13 languages from the Universal Dependencies Treebanks (Nivre et al., 2018) and the English and Chinese Penn Treebanks (Marcus et al., 1993; Xue et al., 2002) show significant improvements over all initial parsers and over the state-of-the-art.

In this paper, we make the following contributions:

- We propose a novel architecture for the iterative refinement of arbitrary graphs, called Recursive Non-autoregressive Graph-to-graph Transformer (RNG-Tr), which combines non-autoregressive edge prediction with conditioning on the complete graph.
- We propose a Transformer model for graph-based dependency parsing (DepBERT) with BERT pre-training.
- We propose the RNG-Tr model of dependency parsing, using initial parses from a variety of previous models and our DepBERT model.
- We demonstrate significant improvements over the previous state-of-the-art results on Universal Dependency Treebanks and Penn Treebanks.

2 Graph-based Dependency Parsing

Syntactic dependency parsing is a critical component in a variety of natural language understanding tasks, such as semantic role labeling (Marcheggiani and Titov, 2017), machine translation (Chen et al., 2017), relation extraction (Zhang et al., 2018), and natural language interface (Pang et al., 2019). There are two main approaches to compute the dependency tree, *transition-based* (Yamada and Matsumoto, 2003; Nivre and Scholz, 2004; Titov and Henderson, 2007; Zhang and Nivre, 2011) and *graph-based* (Eisner, 1996; McDonald et al., 2005; Koo and Collins, 2010) models. Transition-based parsers predict the dependency graph one edge at a time through a sequence of parsing actions. Graph-based models compute scores for every possible dependency edge and then apply a decoding algorithm to find the highest scoring total tree, such as the maximum spanning tree (or greedy) algorithm (Chi, 1999; Edmonds, 1967) (MST). Typically these models consist of two components: an *encoder* learns context-dependent vector representations for the nodes of the dependency graph, and a *decoder* then computes the dependency scores for each pair of nodes, then a decoder algorithm is used to find the highest-scoring dependency tree.

Graph-based models should have trouble capturing correlations between dependency edges since the score for an edge must be chosen without being sure what other edges MST will choose. MST itself only imposes the discrete tree constraint between edges. And yet, graph-based models are the state-of-the-art in dependency parsing. In this paper, we show that it is possible to improve over the state-of-the-art by recursively conditioning each edge score prediction on a previous prediction of the complete dependency graph.

3 RNG Transformer

The RNG Transformer architecture is illustrated in Figure 1, in this case, applied to dependency parsing. The input to the RNG-Tr model is an initial graph G^0 (e.g. a parse tree) over the input nodes $W=(w_1,w_2,...,w_N)$ (e.g. a sentence), and the output is a final graph G^T over the same set of nodes. Each iteration takes the previous graph G^{t-1} as input and predicts a new graph G^t.

The RNG-Tr model predicts G^t with a novel version of a Graph-to-Graph Transformer Mohammadshahi and Henderson (2019). Unlike in the work of (Mohammadshahi and Henderson, 2019), this G2G-Tr model predicts every edge of the graph in a single non-autoregressive step. As previously, the G2G-Tr first encodes the input graph G^{t-1} in a set of contextualised vector representations $Z=(z_1,z_2,...,z_N)$, with one vector for each node

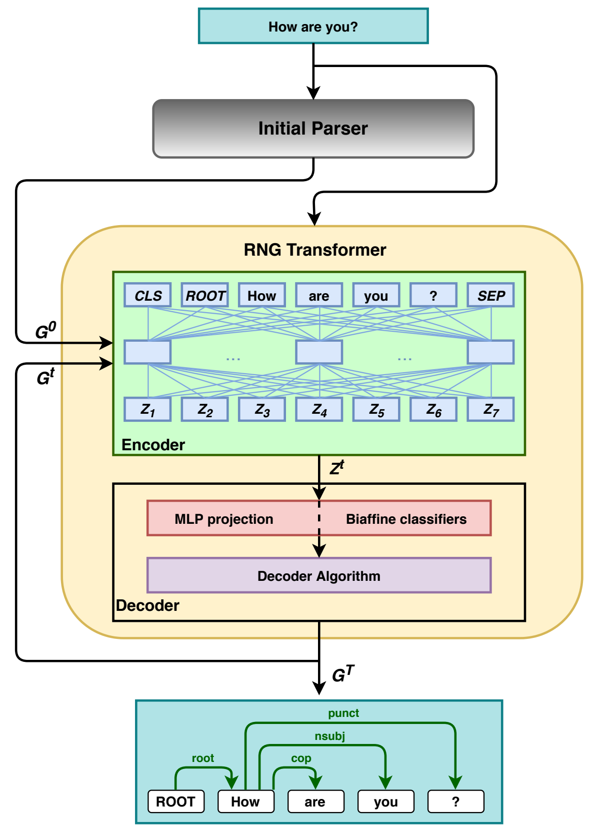

Figure 1: Recursive Non-autoregressive Graph-to-Graph Transformer architecture.

of the graph. The decoder component then predicts the output graph G^t by first computing scores for each possible edge between each pair of nodes and then applying a decoding algorithm to output the highest-scoring complete graph.

We can formalise the RNG-Tr model as:

$$\begin{cases} Z^t = \mathtt{E}^{\mathtt{RNG}}(W,P,G^{t-1}) \\ G^t = \mathtt{D}^{\mathtt{RNG}}(Z^t) \\ t = 1,...,T \end{cases} \tag{1}$$

where $\mathtt{E}^{\mathtt{RNG}}$ is the encoder component, $\mathtt{D}^{\mathtt{RNG}}$ is the decoder component, $P = (p_1, p_2, , \ldots, p_N)$ is the part-of-speech tags of the associated words and symbols $W = (w_1, w_2, , \ldots, w_N)$, and T is the number of recursion steps. The predicted dependency graph at iteration t is specified as:

$$\begin{cases} G^t = \{(i,j,l)_j, j = 3,...,N-1\} \\ \text{where } 2 \le i \le N-1, l \in L \end{cases} \tag{2}$$

Each word w_j has one head (parent) w_i with dependency label l from the label set L, where the parent may be the `ROOT` symbol w_2 (see Section 3.1.1).

In the following sections, we will describe each element of our RNG-Tr dependency parsing model in detail.

3.1 Encoder

To compute the embeddings Z^t for the nodes of the graph, we use the Graph-to-Graph Transformer architecture proposed by Mohammadshahi and Henderson (2019), including similar mechanism to input the previously predicted dependency graph G^{t-1} to the attention mechanism. This graph input allows the node embeddings to include both token-level and dependency-level information.

3.1.1 Input Embeddings

The RNG-Tr model receives a sequence of input tokens (W) with their associated Part-of-Speech tags (P) and builds a sequence of input embeddings (X). The sequence of input tokens starts with `CLS` and ends with `SEP` symbols (the same as BERT (Devlin et al., 2018)'s input token representation). It also adds the `ROOT` symbol to the front of the sentence to represent the root of the dependency tree.

To build token representation for a sequence of input tokens, we sum several vectors. For the input words and symbols, we sum the pre-trained word embeddings of BERT $\mathtt{EMB}(w_i)$, and learned representations $\mathtt{EMB}(p_i)$ of POS tags. To keep the order information of the initial sequence, we add the pre-trained position embeddings of BERT F_i to our token embeddings. The final input representations are the sum of the position embeddings and the token embeddings:

$$x_i = F_i + \mathtt{EMB}(w_i) + \mathtt{EMB}(p_i),\ i = 1,2,...,N \tag{3}$$

3.1.2 Self-Attention Mechanism

Conditioning on the previously predicted output graph G^{t-1} is made possible by inputting relation embeddings to the self-attention mechanism. This edge input method was originally proposed by Shaw et al. (2018) for relative position encoding, and extending to unlabelled dependency graphs in the Graph-to-Graph Transformer architecture of Mohammadshahi and Henderson (2019). We use it to input labelled dependency graphs. A relation embedding is added both to the value function and to the attention weight function wherever the two related tokens are involved.

Transformers have multiple layers of self-attention, each with multiple heads. The RNG-Tr architecture uses the same architecture as

BERT (Devlin et al., 2018) but changes the functions used by each attention head. Given the token embeddings X at the previous layer and the input graph G^{t-1}, the values $A=(a_1,\dots,a_N)$ computed by an attention head are:

$$a_i = \sum_j \alpha_{ij}(x_j \boldsymbol{W^V} + r_{ij}^{t-1} \boldsymbol{W_2^L}) \qquad (4)$$

where r_{ij}^{t-1} is a one-hot vector that represents the labelled dependency relationship between i and j in the graph G^{t-1}. These relationships include both the label and the direction (head<dependent versus dependent<head), or specify NONE. $\boldsymbol{W_2^L} \in R^{(2n+1)\times d}$ are learned relation embeddings (where n is the number of dependency labels). The attention weights α_{ij} are a softmax applied to the attention function:

$$\alpha_{ij} = \frac{\texttt{exp}(e_{ij})}{\sum \texttt{exp}(e_{ij})}$$
$$e_{ij} = \frac{(x_i \boldsymbol{W^Q})(x_j \boldsymbol{W^K} + \texttt{LN}(r_{ij}^{t-1} \boldsymbol{W_1^L}))}{\sqrt{d}} \qquad (5)$$

where $\boldsymbol{W_1^L} \in R^{(2n+1)\times d}$ are learned relation embeddings. LN() is the layer normalisation function, for better convergence.

Equations (4) and (5) constitute the mechanism by which each iteration of refinement can condition on the previous graph.

3.2 Decoder

The decoder uses the token embeddings Z^t produced by the encoder to predict the new dependency graph G^t. It consists of two components, a scoring function, and a decoding algorithm. The dependency graph found by the decoding algorithm is the output graph G^t of the decoder.

There are differences between the segmentation of the input string used by BERT and the segmentation used by the dependency treebanks. To compensate for this, the encoder uses the segmentation of BERT, but only a subset of the resulting token embeddings are considered by the decoder. For each word in the dependency annotation, only the first segment of that word from the BERT segmentation is used in the decoder. See Section 5.3 for more details.

3.2.1 Scoring Function

We first produce four distinct vectors for each token embedding z_i^t from the encoder by passing it through four feed-forward layers.

$$\begin{aligned}
z_i^{t,(arc-dep)} &= \texttt{MLP}^{(arc-dep)}(z_i^t) \\
z_i^{t,(arc-head)} &= \texttt{MLP}^{(arc-head)}(z_i^t) \\
z_i^{t,(rel-dep)} &= \texttt{MLP}^{(rel-dep)}(z_i^t) \\
z_i^{t,(rel-head)} &= \texttt{MLP}^{(rel-head)}(z_i^t)
\end{aligned} \qquad (6)$$

where the MLP's are all one-layer feed-forward networks with LeakyReLU activation functions.

These token embeddings are used to compute probabilities for every possible dependency relation, both unlabelled and labelled, similarly to Dozat and Manning (2016). The distribution of the unlabelled dependency graph is estimated using, for each token i, a biaffine classifier over possible heads j applied to $z_i^{t,(arc-dep)}$ and $z_j^{t,(arc-head)}$. Then for each pair i,j, the distribution over labels given an unlabelled dependency relation is estimated using a biaffine classifier applied to $z_i^{t,(rel-dep)}$ and $z_j^{t,(rel-head)}$.

3.2.2 Decoding Algorithms

The scoring function estimates a distribution over graphs, but the RNG-Tr architecture requires the decoder to output a single graph G^t. Choosing this graph is complicated by the fact that the scoring function is non-autoregressive, and thus the estimate consists of multiple independent components, and thus there is no guarantee that every graph in this distribution is a valid dependency graph.

We take two approaches to this problem, one for intermediate parses G^t and one for the final dependency parse G^T. To speed up each refinement iteration, we ignore this problem for intermediate dependency graphs. We build these graphs by simply applying `argmax` independently to find the head of each node. This may result in graphs with loops, which are not trees, but this does not seem to cause problems for later refinement iterations.[1] For the final output dependency tree, we use the maximum spanning tree algorithm, specifically the Chu-Liu/Edmonds algorithm (Chi, 1999; Edmonds, 1967), to find the highest scoring valid dependency tree. This is necessary to avoid problems when running the evaluation scripts.

3.3 Training

The RNG Transformer model is trained separately on each refinement iteration. Standard gradient descent techniques are used, with cross-entropy loss

[1]We will investigate different decoding strategies to keep both the speed and well-formedness of the intermediate predicted graphs in future work.

for each edge prediction. Error is not backpropagated across iterations of refinement, because there are no continuous values being passed from one iteration to another, only a discrete dependency tree.

Stopping Criterion In the RNG Transformer architecture, the refinement of the predicted graph can be done an arbitrary number of times, since the same encoder and decoder parameters are used at each iteration. In the experiments below, we place a limit on the maximum number of iterations. But sometimes the model converges to an output graph before this limit is reached, simply copying this graph during later iterations. To avoid multiple iterations where the model is trained to simply copy the input graph, during training the refinement iterations are stopped if the new predicted dependency graph is the same as the input graph. At test time we also stop computation in this case, but the output of the model is obviously not affected.

4 Initial Parsers

The RNG-Tr architecture assumes that there is an initial graph G^0 for the RNG-Tr model to refine. We consider several initial parsers to produce this graph. In our experiments, we find that the initial graph has little impact on the quality of the graph output by the RNG-Tr model.

To leverage previous work on dependency parsing, we use parsing models from the recent literature as initial parsers. To evaluate the importance of the initial parse, we also consider a setting where the initial parse is empty, so the first complete dependency tree is predicted by the RNG-Tr model itself. Finally, the success of our RNG-Tr dependency parsing model leads us to propose an initial parsing model with the same design, so that we can control for the parser design in measuring the importance of the RNG Transformer's iterative refinement.

DepBERT model We call this initial parser the Dependency BERT (DepBERT) model. It is the same as one iteration of the RNG-Tr model shown in Figure 1 and defined in Section 3, except that there is no graph input to the encoder. Analogously to (1), G^0 is computed as:

$$\begin{cases} Z^0 = \mathtt{E}^{\mathtt{DBERT}}(W,P) \\ G^0 = \mathtt{D}^{\mathtt{DBERT}}(Z^0) \end{cases} \tag{7}$$

where $\mathtt{E}^{\mathtt{DBERT}}$ and $\mathtt{D}^{\mathtt{DBERT}}$ are the DepBERT encoder and decoder respectively. For the encoder, we use the Transformer architecture of BERT (Devlin et al., 2018) and initialise with BERT's pre-trained parameters. The token embeddings of the final layer are used for Z^0. For the decoder, we use the same segmentation strategy and scoring function as described in Section 3.2, and apply Chu-Liu/Edmonds decoding algorithm (Chi, 1999; Edmonds, 1967) to find the highest scoring tree. The DepBERT parsing model is very similar to the UDify parsing model proposed by Kondratyuk and Straka (2019), but there are significant differences in the way token segmentation is handled, which result in significant differences in performance, as shown in Section 6.2.

5 Experimental Setup

5.1 Datasets

To evaluate our models, we apply them on two kinds of datasets, Universal Dependency (UD) Treebanks (Nivre et al., 2018), and Penn Treebanks. For evaluation, following Kulmizev et al. (2019); Nivre et al. (2018), we keep punctuation for UD Treebanks and remove it for Penn Treebanks (Nilsson and Nivre, 2008).

Universal Dependency Treebanks: We evaluate our models on Universal Dependency Treebanks (UD v2.3) (Nivre et al., 2018). We select languages based on the criteria proposed in de Lhoneux et al. (2017b), and adapted by Smith et al. (2018b). This set contains several languages with different language families, scripts, character set sizes, morphological complexity, and training sizes and domains. The description of the selected Treebanks is in Appendix A.

Penn Treebanks: We also evaluate our models on English and Chinese Penn Treebanks (Marcus et al., 1993; Xue et al., 2002). For English, we use sections 2-21 for training, section 22 for development, and section 23 for testing. We add section 24 to our development set to mitigate over-fitting. We use the Stanford PoS tagger (Toutanova et al., 2003) to produce PoS tags. We convert constituency trees to Stanford dependencies using version 3.3.0 of the converter (De Marneffe et al., 2006). For Chinese, we apply the same set-up as described in Chen and Manning (2014), and use gold PoS tags during training and evaluation.

5.2 Baseline Models

For UD Treebanks, we consider several parsers both as baselines and to produce initial parses for the RNG-Tr model. We use the monolingual parser proposed by Kulmizev et al. (2019), which uses and extends the UUParser[2] (de Lhoneux et al., 2017a; Smith et al., 2018a), and applies BERT (Devlin et al., 2018) and ELMo (Peters et al., 2018) embeddings as additional input features. In addition, we also compare our models with multilingual models proposed by Kondratyuk and Straka (2019) and Straka (2018). UDify (Kondratyuk and Straka, 2019) is a multilingual multi-task model for predicting universal part-of-speech, morphological features, lemmas, and dependency graphs at the same time for all UD Treebanks. UDPipe (Straka, 2018) is one of the winners of CoNLL 2018 Shared Task (Zeman et al., 2018). It's a multi-task model that performs sentence segmentation, tokenization, POS tagging, lemmatization, and dependency parsing. For a fair comparison, we use reported scores of Kondratyuk and Straka (2019) for the UDPipe model which they retrained using gold segmentation. UDify outperforms the UDPipe model on average, so we use both UDify and DepBERT models as our initial parsers to integrate with the RNG Transformer model. We also train the RNG-Tr model without any initial dependency graph, called Empty+RNG-Tr, to further analyse the impact of the initial graph.

For Penn Treebanks, we compare our models with previous state-of-the-art transition-based, and graph-based models. The Biaffine parser (Dozat and Manning, 2016) includes the same decoder as our model, with an LSTM-based encoder. Ji et al. (2019) also integrate graph neural network models with the Biaffine parser, to find a better representation for the nodes of the graph. For these datasets, we use the Biaffine and DepBERT parsers as the initial parsers for our RNG Transformer model.

5.3 Implementation Details

All hyper-parameters are provided in Appendix B.

For the self-attention model, we use the pre-trained "bert-multilingual-cased"[3] with 12 self-attention layers.[4] For tokenization, we apply the wordpiece tokenizer of BERT (Wu et al., 2016). Since dependency relations are between the tokens of a dependency corpus, we apply the BERT tokenizer to each corpus token and run the encoder on all the resulting sub-words. Each dependency between two words is inputted as a relation between their first sub-words. We also input a new relationship with each non-first sub-word as the dependent and the associated first sub-word as its head. In the decoder, we only consider candidate dependencies between the first sub-words for each word.[5] Finally, we map the predicted heads and dependents to their original positions in the corpus for proper evaluation.

Model	UAS	LAS
DepBERT	75.62	70.04
DepBERT+RNG-Tr(T=1)	**76.37**	70.67
DepBERT+RNG-Tr(T=3) w/o *stop*	**76.33**	70.61
DepBERT+RNG-Tr(T=3)	**76.29**	**70.84**
UDify (Kondratyuk and Straka, 2019)	72.78	65.48
UDify+RNG-Tr(T=1)	74.13	68.60
UDify+RNG-Tr(T=3) w/o *stop*	75.68	70.32
UDify+RNG-Tr(T=3)	**75.91**	**70.66**

Table 1: Dependency parsing scores for different variations of RNG Transformer model on the development set of UD Turkish Treebank (IMST).

6 Results and Discussion

After some initial experiments to determine the number of refinement iterations, we report the performance of the RNG Transformer model on UD treebanks and Penn treebanks. The RNG-Tr models perform substantially better than models without refinement on almost every dataset. We also perform various analyses of the models to better understand these results.

6.1 The Number of Refinement Iterations

To select the best number of refinement iterations allowed by RNG Transformer model, we evaluate different variations of our model on the Turkish Treebank (Table 1).[6] We use both DepBERT and UDify as initial parsers. The DepBERT model significantly outperforms the UDify model, so adding the RNG-Tr model to the UDify model results in more relative error reduction in LAS compared to its integration with DepBERT (17.65% vs. 2.67%). In both cases, using three refinement iterations achieves the best result, and excluding the

[2]https://github.com/UppsalaNLP/uuparser

[3]https://github.com/google-research/bert

[4]For Chinese and Japanese, we use pre-trained "bert-base-chinese" and "bert-base-japanese" models (Wolf et al., 2019) respectively.

[5]In preliminary experiments, we found that using the dependency predictions of the first sub-words achieves better or similar results compared to using the last sub-word or all sub-words of each word.

[6]We choose the Turkish Treebank because it is a low-resource Treebank and there are more errors in the initial parse for RNG-Tr to correct.

Language	Mono [1]	Multi UDPipe	Multi UDify	Multi+Mono UDify+RNG-Tr	Mono DepBERT	Mono DepBERT+RNG-Tr	Mono Empty+RNG-Tr
Arabic	81.8	82.94	82.88	85.93 (+17.81%)	**86.23**	86.10 (-0.93%)	86.05
Basque	79.8	82.86	80.97	87.55 (+34.57%)	87.49	**88.2** (+5.68%)	87.96
Chinese	83.4	80.5	83.75	89.05 (+32.62%)	89.53	**90.48** (+9.08%)	89.82
English	87.6	86.97	88.5	91.23 (+23.74%)	91.41	**91.52** (+1.28%)	91.23
Finnish	83.9	87.46	82.03	91.87 (+54.76%)	91.34	**91.92** (+6.7%)	91.78
Hebrew	85.9	86.86	88.11	90.80 (+22.62%)	91.07	**91.29** (+2.46%)	90.56
Hindi	90.8	91.83	91.46	93.94 (+29.04%)	93.95	**94.21** (+4.3%)	93.97
Italian	91.7	91.54	93.69	94.65 (+15.21%)	**95.08**	**95.08** (0.0%)	94.96
Japanese	92.1	93.73	92.08	95.41 (+42.06%)	95.66	**95.71** (+1.16%)	95.56
Korean	84.2	84.24	74.26	89.12 (+57.73%)	89.29	**89.45** (+1.5%)	89.1
Russian	91.0	92.32	93.13	94.51 (+20.09%)	**94.60**	94.47 (-2.4%)	94.31
Swedish	86.9	86.61	89.03	92.02 (+27.26%)	92.03	**92.46** (+5.4%)	92.40
Turkish	64.9	67.56	67.44	72.07 (+14.22%)	72.52	**73.08** (+2.04%)	71.99
Average	84.9	85.81	85.18	89.86	90.02	**90.31**	89.98

Table 2: Labelled attachment scores on UD Treebanks for monolingual ([1] (Kulmizev et al., 2019) and DepBERT) and multilingual (UDPipe (Straka, 2018) and UDify (Kondratyuk and Straka, 2019)) baselines, and the refined models (+RNG-Tr).

stopping strategy described in Section 3.3 decreases the performance. In subsequent experiments, we use three refinement iterations with the stopping strategy, unless mentioned otherwise.

6.2 UD Treebanks Results

Results for the UD treebanks are reported in Table 2. We compare our models with previous state-of-the-art results (both trained mono-lingually and multi-lingually), based on labelled attachment score.[7] The UDify+RNG-Tr model achieves significantly better performance than the UDify model, which demonstrates the effectiveness of the RNG-Tr model at refining an initial dependency graph. The DepBERT model significantly outperforms previous state-of-the-art models on all UD Treebanks. But despite this good performance, the DepBERT+RNG-Tr model achieves further improvement over DepBERT in almost all languages and on average. As expected, we get more improvement when combining the RNG-Tr model with UDify, because UDify's initial dependency graph contains more incorrect dependency relations for RNG-Tr to correct.

Although generally better, there is surprisingly little difference between the performance after refinement of the UDify+RNG-Tr and DepBERT+RNG-Tr models. To investigate the power of the RNG-Tr architecture to correct any initial parse, we also show results for a model with an empty initial parse, Empty+RNG-Tr. For this model, we run four iterations of refinement (T=4), so that the amount of computation is the same as for UDify+RNG-Tr and DepBERT+RNG-Tr. The Empty+RNG-Tr model achieves competitive results with the UDify+RNG-Tr model (i.e. above the previous state-of-the-art), indicating that the RNG-Tr architecture is a very powerful method for graph refinement. We will discuss this conclusion further in Section 6.4.

6.3 Penn Treebanks Results

UAS and LAS results for the Penn Treebanks are reported in Table 3. We compare to the results of previous state-of-the-art models and DepBERT, and we use the RNG-Tr model to refine both the Biaffine parser (Dozat and Manning, 2016) and DepBERT, on the English and Chinese Penn Treebanks [8].

Again, the DepBERT model significantly outperforms previous state-of-the-art models, with a 5.78% and 9.15% LAS relative error reduction in English and Chinese, respectively. Despite this level of accuracy, adding RNG-Tr refinement improves accuracy further under both measures in both languages, although in English the differences are not significant. For the Chinese Treebank, RNG-Tr refinement achieves a 4.7% LAS relative error reduction. When RNG-Tr refinement is applied to the output of the Biaffine parser (Dozat and Manning, 2016), it achieves LAS relative error

[7]Unlabelled attachment scores are provided in Appendix C. All results are computed with the official CoNLL 2018 shared task evaluation script (`https://universaldependencies.org/conll18/evaluation.html`).

[8]Results are calculated with the official evaluation script: (`https://depparse.uvt.nl/`).

Model		English		Chinese	
	Type	UAS	LAS	UAS	LAS
Chen and Manning (2014)	T	91.8	89.6	83.9	82.4
Dyer et al. (2015)	T	93.1	90.9	87.2	85.7
Ballesteros et al. (2016)	T	93.56	91.42	87.65	86.21
Weiss et al. (2015)	T	94.26	92.41	-	-
Andor et al. (2016)	T	94.61	92.79	-	-
Mohammadshahi and Henderson (2019)	T	95.64	93.81	-	-
Ma et al. (2018)	T	95.87	94.19	90.59	89.29
Fernndez-Gonzlez and Gmez-Rodrguez (2019)	T	96.04	94.43	-	-
Kiperwasser and Goldberg (2016)	G	93.1	91.0	86.6	85.1
Wang and Chang (2016)	G	94.08	91.82	87.55	86.23
Cheng et al. (2016)	G	94.10	91.49	88.1	85.7
Kuncoro et al. (2016)	G	94.26	92.06	88.87	87.30
Ma and Hovy (2017)	G	94.88	92.98	89.05	87.74
Ji et al. (2019)	G	95.97	94.31	-	-
Li et al. (2019)+ELMo	G	96.37	94.57	90.51	89.45
Li et al. (2019)+BERT	G	96.44	94.63	90.89	89.73
Biaffine (Dozat and Manning, 2016)	G	95.74	94.08	89.30	88.23
Biaffine+RNG-Tr	G	96.44	94.71	91.85	90.12
DepBERT	G	**96.60**	**94.94**	92.42	90.67
DepBERT+RNG-Tr	G	**96.66**	**95.01**	**92.86**	**91.11**

Table 3: Comparison of our models to previous SOTA models on English (PTB) and Chinese (CTB5.1) Penn Treebanks. ”T” and ”G” stand for ”transition-based” and ”graph-based” models.

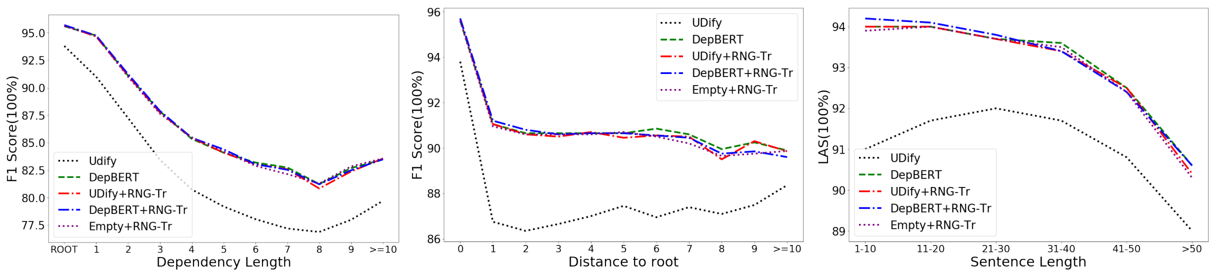

Figure 2: Error analysis of initial parsers (UDify and DepBERT), their integration with the RNG-Tr model, alongside with Empty+RNG-Tr model on the concatenation of UD Treebanks.

reduction of 10.64% for the English Treebank and 16.5% for the Chinese Treebank. These improvements, even over such strong initial parsers, again demonstrate the effectiveness of the RNG-Tr architecture for graph refinement.

6.4 Error Analysis

To better understand the distribution of errors for our models, we follow McDonald and Nivre (2011) and plot labelled attachment scores as a function of dependency length, sentence length and distance to root [9]. We compare the distributions of errors made by UDify (Kondratyuk and Straka, 2019), DepBERT, and refined models (UDify+RNG-Tr, DepBERT+RNG-Tr, and Empty+RNG-Tr). Figure 2 shows the accuracies of the different models on the concatenation of all development sets of UD Treebanks.[10] Results show that applying RNG-Tr refinement to the UDify model results in a great improvement in accuracy across all cases. They also show little difference in the error profile between the better performing models.

Dependency Length: The leftmost plot compares the accuracy of models on different dependency lengths. Adding RNG-Tr refinement to UDify results in better performance both on short and long dependencies, with particular gains for the longer and more difficult cases.

[9]We use MaltEval tool (Nilsson and Nivre, 2008) for calculating accuracies in all cases.

[10]Tables for the error analysis section, and graphs for each language are provided in Appendix D.

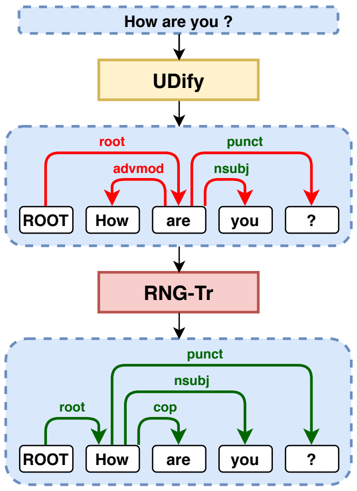

Figure 3: Example of UDify+RNG-Tr on English UD Treebank.

Distance to Root: The middle plot illustrates the accuracy of models as a function of the distance to the root of the dependency tree, which is calculated as the number of dependency relations from the dependent to the root. Again, when we add RNG-Tr refinement to the UDify parser we get significant improvement for all distances, with particular gains for the difficult middle distances, where the dependent is neither near the root nor a leaf of the tree.

Sentence Length: The rightmost plot shows the accuracy of models on different sentence lengths. Again, adding RNG-Tr refinement to UDify achieves significantly better results on all sentence lengths. But in this case, the larger improvements are for the shorter, presumably easier, sentences.

6.5 Refinement Analysis

To better understand how the RNG Transformer model is doing refinement, we perform several analyses of the trained UDify+RNG-Tr model.[11] An example of this refinement is shown in Figure 3, where the UDify model predicts an incorrect dependency graph, but the RNG-Tr model modifies it to build the gold dependency tree.

Refinements by Iteration: To measure the accuracy gained from refinement at different iterations, we define the following metric:

$$\mathtt{REL}^t = \mathtt{RER}(\mathtt{LAS}^{t-1}, \mathtt{LAS}^t) \qquad (8)$$

where RER is relative error reduction, and t is the refinement iteration. $\mathtt{LAS}^0$ is the accuracy of the initial parser, UDify in this case. To illustrate the refinement procedure for different dataset types, we split UD Treebanks based on their training size to ”Low-Resource” and ”High-Resource” datasets.[12]

[11] We choose UDify as the initial parser because the RNG-Tr model makes more changes to the parses of UDfiy than DepBERT, so we can more easily analyse these changes.

Model Type	$t=1$	$t=2$	$t=3$
Low-Resource	+13.62%	+17.74%	+0.16%
High-Resource	+29.38%	+0.81%	+0.41%

Table 4: Refinement Analysis (LAS relative error reduction) of the UDify+RNG-TR model for different refinement steps on the UD Treebanks.

Table 4 shows this refinement metric ($\mathtt{REL}^t$) after each refinement iteration of the UDify+RNG-Tr model on the UD Treebanks.[13] Every refinement step achieves an increase in accuracy, on both low and high resource languages. But the amount of improvement decreases for higher refinement iterations. Interestingly, for languages with less training data, the model cannot learn to make all corrections in a single step but can learn to make the remaining corrections in a second step, resulting in approximately the same total percentage of errors corrected as for high resource languages.

Dependency Type Refinement: Table 5 shows the accuracy of a selection of different dependency types for the UDify model as the initial parser, and the refined model (+RNG-Tr).[14] On average, we achieve 29.78% LAS error reduction compared to UDify by adding the RNG-Tr model to UDify. Significant improvements are achieved for all dependency types, especially in hard cases, such as copula (cop).[15]

Non-Projective Trees: Predicting non-projective trees is a challenging issue for dependency parsers. Figure 4 shows precision and recall for the UDify and UDify+RNG-Tr models on non-projective trees of the UD Treebanks. Adding the RNG-Tr model to the initial parser results in a significant improvement in both precision and recall, which again demonstrates the effectiveness of the RNG-Tr model on hard cases.

[12] We consider languages that have training data more than 10k sentences as ”High-Resource”.

[13] For these results we apply MST decoding after every iteration, to allow proper evaluation of the intermediate graphs.

[14] The accuracy of UDify, DepBERT, and their integration with RNG-Tr on all dependency types are provided in Appendix E.

[15] A cop (copula) is the relation of a function word used to link a subject to a nonverbal predicate.

Type	UDify	UDify+RNG-Tr
`acl`	75.06	83.60 (+34.2%)
`advcl`	71.68	80.34 (+30.6%)
`advmod`	83.45	89.10 (+34.1%)
`amod`	92.10	95.80 (+46.8%)
`aux`	95.14	98.40 (+67.1%)
`case`	97.10	98.20 (+37.9%)
`ccomp`	74.29	85.63 (+44.1%)
`compound`	83.65	90.40 (+41.3%)
`cop`	86.88	93.55 (+50.8%)

Table 5: F-score (and relative difference) of UDify and its refinement with RNG-Tr for a subset of dependency types on the concatenation of UD Treebanks.

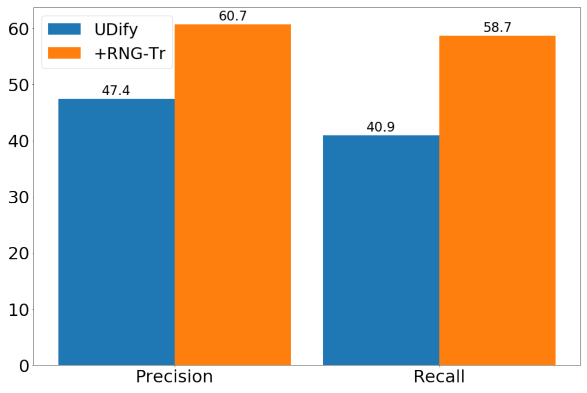

Figure 4: Precision and Recall of Udify and UDify+RNG-Tr on non-projective dependency trees of the UD Treebanks.

6.6 Time complexity

In this section, we compute the time complexity of both proposed models, DepBERT and RNG-Tr.

DepBERT: The time complexity of the original Transformer (Vaswani et al., 2017) is $O(n^3)$, where n is the sequence length, so the time complexity of the encoder is $O(n^3)$. The time complexity of the decoder is determined by the Chu-Liu/Edmonds algorithm (i Chu and Liu, 1965; Edmonds, 1967), which is $O(n^3)$. So, the total time complexity of the DepBERT model is $O(n^3)$, the same as other graph-based models.

RNG-Tr: In each refinement step, the time complexity of Graph-to-Graph Transformer (Mohammadshahi and Henderson, 2019) is $O(n^3)$. Since we use the `argmax` function in the intermediate steps, the decoding time complexity is determined by the last decoding step, which is $O(n^3)$. So, the total time complexity is $O(n^3)$.

7 Conclusion

We proposed a novel architecture for structured prediction, Recursive Non-autoregressive Graph-to-graph Transformer (RNG-Tr) to iteratively refine arbitrary graphs. Given an initial graph, RNG Transformer learns to predict a corrected graph over the same set of nodes. Each iteration of refinement predicts the edges of the graph in a non-autoregressive fashion, but conditions these predictions on the entire graph from the previous iteration. This graph conditioning and prediction are done with the Graph-to-Graph Transformer architecture (Mohammadshahi and Henderson, 2019), which makes it capable of capturing complex patterns of interdependencies between graph edges. Graph-to-Graph Transformer also benefits from initialisation with a pre-trained BERT (Devlin et al., 2018) model. We also propose a graph-based dependency parser called DepBERT, which is the same as our refinement model but without graph inputs.

We evaluate the RNG Transformer architecture on syntactic dependency parsing. We run experiments with a variety of initial parsers, including DepBERT, on 13 languages of Universal Dependencies Treebanks, and on English and Chinese Penn Treebanks. Our DepBERT model already significantly outperformed previous state-of-the-art models on both types of Treebanks. Even with this very strong initial parser, RNG-Tr refinement almost always improves accuracies, setting new state-of-the-art accuracies for all treebanks. Regardless of the initial parser (e.g. UDify (Kondratyuk and Straka, 2019) on UD Treebanks, and Biaffine parser (Dozat and Manning, 2016) on Penn Treebanks), RNG-Tr reaching around the same level of accuracy, even when it is given an empty initial parse, demonstrating the power of this iterative refinement method. Finally, we provided error analyses of the proposed model to illustrate its advantages and understand how refinements are made across iterations.

The RNG Transformer architecture is a very general and powerful method for structured prediction, which could easily be applied to other NLP tasks. It would especially benefit tasks that require capturing complex structured interdependencies between graph edges, even with the computational benefits of a non-autoregressive model.

8 Acknowledgement

We are grateful to the Swiss NSF, grant CRSII5_180320, for funding this work. We also thank

Lesly Miculicich and other members of the Idiap NLU group for helpful discussions.

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany. Association for Computational Linguistics.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. 2016. Training with exploration improves a greedy stack LSTM parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2005–2010, Austin, Texas. Association for Computational Linguistics.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. 2017. Improved neural machine translation with a syntax-aware encoder and decoder. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. 2016. Bi-directional attention with agreement for dependency parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2204–2214, Austin, Texas. Association for Computational Linguistics.

Zhiyi Chi. 1999. Statistical properties of probabilistic context-free grammars. *Computational Linguistics*, 25(1):131–160.

Yau i Chu and T. Liu. 1965. On the shortest arborescence of a directed graph.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Lrec*, volume 6, pages 449–454.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding.

Timothy Dozat and Christopher D. Manning. 2016. Deep biaffine attention for neural dependency parsing.

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.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*.

Daniel Fernndez-Gonzlez and Carlos Gmez-Rodrguez. 2019. Left-to-right dependency parsing with pointer networks. *Proceedings of the 2019 Conference of the North*.

Tao Ji, Yuanbin Wu, and Man Lan. 2019. Graph-based dependency parsing with graph neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2475–2485, Florence, Italy. 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. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11. 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. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Distilling an ensemble of greedy dependency parsers into one MST parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1744–1753, Austin, Texas. Association for Computational Linguistics.

Miryam de Lhoneux, Yan Shao, Ali Basirat, Eliyahu Kiperwasser, Sara Stymne, Yoav Goldberg, and Joakim Nivre. 2017a. From raw text to universal dependencies - look, no tags! In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 207–217, Vancouver, Canada. Association for Computational Linguistics.

Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2017b. Old school vs. new school: Comparing transition-based parsers with and without neural network enhancement. In *Proceedings of the 15th International Workshop on Treebanks and Linguistic Theories (TLT15)*, pages 99–110.

Zuchao Li, Hai Zhao, and Kevin Parnow. 2019. Global greedy dependency parsing.

Xuezhe Ma and Eduard Hovy. 2017. Neural probabilistic model for non-projective mst parsing.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1403–1414, Melbourne, Australia. Association for Computational Linguistics.

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*, pages 1506–1515, Copenhagen, Denmark. Association for Computational Linguistics.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98, Ann Arbor, Michigan. Association for Computational Linguistics.

Ryan McDonald and Joakim Nivre. 2011. Analyzing and integrating dependency parsers. *Computational Linguistics*, 37(1):197–230.

Alireza Mohammadshahi and James Henderson. 2019. Graph-to-graph transformer for transition-based dependency parsing.

Jens Nilsson and Joakim Nivre. 2008. MaltEval: an evaluation and visualization tool for dependency parsing. In *LREC 2008*.

Joakim Nivre, Mitchell Abrams, Željko Agić, Lars Ahrenberg, Lene Antonsen, Katya Aplonova, Maria Jesus Aranzabe, Gashaw Arutie, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, Victoria Basmov, John Bauer, Sandra Bellato, Kepa Bengoetxea, Yevgeni Berzak, Irshad Ahmad Bhat, Riyaz Ahmad Bhat, Erica Biagetti, Eckhard Bick, Rogier Blokland, Victoria Bobicev, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Adriane Boyd, Aljoscha Burchardt, Marie Candito, Bernard Caron, Gauthier Caron, Gülşen Cebiroğlu Eryiğit, Flavio Massimiliano Cecchini, Giuseppe G. A. Celano, Slavomír Čéplö, Savas Cetin, Fabricio Chalub, Jinho Choi, Yongseok Cho, Jayeol Chun, Silvie Cinková, Aurélie Collomb, Çağrı Çöltekin, Miriam Connor, Marine Courtin, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Carly Dickerson, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Ali Elkahky, Binyam Ephrem, Tomaž Erjavec, Aline Etienne, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Sebastian Garza, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Céline Guillot-Barbance, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỹ, Na-Rae Han, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Jena Hwang, Radu Ion, Elena Irimia, Ọlájídé Ishola, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Sylvain Kahane, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Tolga Kayadelen, Jessica Kenney, Václava Kettnerová, Jesse Kirchner, Kamil Kopacewicz, Natalia Kotsyba, Simon Krek, Sookyoung Kwak, Veronika Laippala, Lorenzo Lambertino, Lucia Lam, Tatiana Lando, Septina Dian Larasati, Alexei Lavrentiev, John Lee, Phng Lê H`ông, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, KyungTae Lim, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Ruli Manurung, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Gustavo Mendonça, Niko Miekka, Margarita Misirpashayeva, Anna Missilä, Cătălin Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Keiko Sophie Mori, Shinsuke Mori, Bjartur Mortensen, Bohdan Moskalevskyi, Kadri Muischnek, Yugo Murawaki, Kaili Müürisep, Pinkey Nainwani, Juan Ignacio Navarro Horñiacek, Anna Nedoluzhko, Gunta Nešpore-Bērzkalne, Lng Nguy˜ên Thị, Huy`ên Nguy˜ên Thị Minh, Vitaly Nikolaev, Rattima Nitisaroj, Hanna Nurmi, Stina Ojala, Adédayọ Olúòkun, Mai Omura, Petya Osenova, Robert Östling, Lilja Øvrelid, Niko Partanen, Elena Pascual, Marco Passarotti, Agnieszka Patejuk, Guilherme Paulino-Passos, Siyao Peng, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Emily Pitler, Barbara Plank, Thierry Poibeau, Martin Popel, Lauma Pretkalniņa, Sophie Prévost, Prokopis Prokopidis, Adam Przepiórkowski, Tiina Puolakainen, Sampo Pyysalo, Andriela Rääbis, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Carlos Ramisch, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Michael Rießler, Larissa Rinaldi, Laura Rituma, Luisa Rocha, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Valentin Roca, Olga Rudina, Jack Rueter, Shoval Sadde, Benoît Sagot, Shadi Saleh, Tanja Samardžić, Stephanie Samson, Manuela Sanguinetti, Baiba Saulīte, Yanin Sawanakunanon, Nathan Schneider, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Muh Shohibussirri, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Isabela Soares-Bastos, Carolyn Spadine, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Yuta Takahashi, Takaaki Tanaka, Isabelle Tellier, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Lars Wallin, Jing Xian Wang, Jonathan North Washington, Seyi Williams, Mats Wirén, Tsegay Woldemariam, Taksum Wong, Chunxiao Yan, Marat M. Yavrumyan, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, Manying Zhang, and Hanzhi Zhu. 2018. Universal dependencies 2.3. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Joakim Nivre and Mario Scholz. 2004. Deterministic dependency parsing of English text. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 64–70, Geneva, Switzerland. COLING.

Deric Pang, Lucy H. Lin, and Noah A. Smith. 2019. Improving natural language inference with a pretrained parser.

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.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana. Association for Computational Linguistics.

Aaron Smith, Bernd Bohnet, Miryam de Lhoneux, Joakim Nivre, Yan Shao, and Sara Stymne. 2018a.

82 treebanks, 34 models: Universal dependency parsing with multi-treebank models. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 113–123, Brussels, Belgium. Association for Computational Linguistics.

Aaron Smith, Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2018b. 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.

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.

Ivan Titov and James Henderson. 2007. A latent variable model for generative dependency parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 144–155, Prague, Czech Republic. Association for Computational Linguistics.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 252–259.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Wenhui Wang and Baobao Chang. 2016. Graph-based dependency parsing with bidirectional LSTM. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2306–2315, Berlin, Germany. Association for Computational Linguistics.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. 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 323–333, Beijing, China. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Nianwen Xue, Fu-Dong Chiou, and Martha Palmer. 2002. Building a large-scale annotated Chinese corpus. In *COLING 2002: The 19th International Conference on Computational Linguistics*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the Eighth International Conference on Parsing Technologies*, pages 195–206, Nancy, France.

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.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA. Association for Computational Linguistics.

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*, pages 2205–2215, Brussels, Belgium. Association for Computational Linguistics.

Appendix A UD Treebanks Details

Language	Treebank	Order	Family	Train size	Resource Type
Arabic	PADT	VSO	non-IE	6.1K	Low
Basque	BDT	SOV	non-IE	5.4K	Low
Chinese	GSD	SVO	non-IE	4K	Low
English	EWT	SVO	IE	12.5K	High
Finnish	TDT	SVO	non-IE	12.2K	High
Hebrew	HTB	SVO	non-IE	5.2K	Low
Hindi	HDTB	SOV	IE	13.3K	High
Italian	ISDT	SVO	IE	13.1K	High
Japanese	GSD	SOV	non-IE	7.1K	Low
Korean	GSD	SOV	non-IE	4.4K	Low
Russian	SynTagRus	SVO	IE	48.8K	High
Swedish	Talbanken	SVO	IE	4.3K	Low
Turkish	IMST	SOV	non-IE	3.7K	Low

Table 6: description of languages chosen from UD Treebanks v2.3.

Appendix B Implementation Details

For better convergence, we use two separate optimizers for pre-trained parameters and randomly initialized parameters. We apply bucketed batching, grouping sentences by their lengths into the same batch to speed up the training. Here is the list of hyper-parameters for RNG Transformer model:

Component	Specification
Optimiser[1]	BertAdam
Base Learning rate	2e-3
BERT Learning rate	1e-5
Adam Betas(b_1,b_2)	(0.9,0.999)
Adam Epsilon	1e-5
Weight Decay	0.01
Max-Grad-Norm	1
Warm-up	0.01
Self-Attention	
No. Layers	12
No. Heads	12
Embedding size	768
Max Position Embedding	512
Feed-Forward layers (arc)	
No. Layers	2
Hidden size	500
Drop-out	0.33
Activation	`LeakyReLU`
Negative Slope	0.1
Feed-Forward layers (rel)	
No. Layers	2
Hidden size	100
Drop-out	0.33
Activation	`LeakyReLU`
Negative Slope	0.1
Epoch	200
patience	100

[1] (Wolf et al., 2019)

Table 7: Hyper-parameters for training on both UD and Penn Treebanks.

Appendix C Unlabeled Attachment Scores for UD Treebanks

Language	Multi UDPipe	Multi UDify	Multi+Mono UDify+RNG-Tr	Mono DepBERT	Mono DepBERT+RNG-Tr	Mono Empty+RNG-Tr
Arabic	87.54	87.72	89.73(+16.37%)	**89.89**	89.76(-1.28%)	89.68
Basque	86.11	84.94	90.49(+36.85%)	90.46	**90.90**(+4.61%)	90.69
Chinese	84.64	87.93	91.04(+25.76%)	91.38	**92.47**(+12.64%)	91.81
English	89.63	90.96	92.81(+20.46%)	92.92	**93.08**(+2.26%)	92.77
Finnish	89.88	86.42	93.49(52.06%)	93.52	**93.55**(+0.47%)	93.36
Hebrew	89.70	91.63	93.03(+16.73%)	93.36	**93.51**(+2.26%)	92.80
Hindi	94.85	95.13	96.44(+26.9%)	96.33	**96.56**(+6.27%)	96.37
Italian	93.49	95.54	95.72(+4.04%)	96.03	**96.11**(+2.02%)	95.98
Japanese	95.06	94.37	96.25(+33.40%)	96.43	**96.54**(+3.08%)	96.37
Korean	87.70	82.74	91.32(+49.71%)	91.35	**91.49**(+1.62%)	91.28
Russian	93.80	94.83	**95.54**(+13.73%)	95.53	95.47(-1.34%)	95.38
Swedish	89.63	91.91	93.72(+22.37%)	93.79	**94.14**(+5,64%)	94.14
Turkish	74.19	74.56	77.74(+12.5%)	77.98	**78.50**(+2.37%)	77.49
Average	88.94	89.13	92.10	92.23	**92.47**	92.16

Table 8: Unlabelled attachment scores on UD Treebanks for monolingual (DepBERT) and multilingual (UDPipe (Straka, 2018) and UDify (Kondratyuk and Straka, 2019)) baselines, and refined models(+RNG-Tr).

Appendix D Error Analysis of UD Treebanks

D.A Dependency Length

Model	ROOT	1	2	3	4	5	6	7	8	9	>=10
UDify	93.80	91.00	87.25	83.45	80.75	79.20	78.05	77.20	76.89	77.99	79.72
DepBERT	95.60	94.75	91.10	87.85	85.35	84.10	83.20	82.75	81.24	82.70	83.48
UDify+RNG-Tr	95.70	94.65	91.10	87.75	85.45	84.10	83.05	82.59	80.84	82.39	83.56
DepBERT+RNG-Tr	95.70	94.75	91.25	87.90	85.45	84.40	83.05	82.54	81.22	82.49	83.50
Empty+RNG-Tr	95.60	94.69	90.99	87.65	85.49	84.19	82.89	82.14	81.28	82.84	83.56

Table 9: labelled F-Score for different dependency lengths.

D.B Distance to Root

Model	0	1	2	3	4	5	6	7	8	9	>=10
UDify	93.80	86.75	86.35	86.65	87.00	87.45	86.95	87.39	87.09	87.49	88.36
DepBERT	95.60	91.05	90.65	90.65	90.65	90.65	90.85	90.60	89.95	90.25	89.89
UDify+RNG-Tr	95.70	91.05	90.60	90.50	90.70	90.45	90.55	90.50	89.50	90.30	89.85
DepBERT+RNG-Tr	95.70	91.20	90.80	90.60	90.65	90.65	90.55	90.45	89.75	89.85	89.60
Empty+RNG-Tr	95.6	90.94	90.6	90.59	90.59	90.69	90.49	90.19	89.64	89.74	89.87

Table 10: labelled F-Score for different distances to root.

D.C Sentence Length

Model	1-10	11-20	21-30	31-40	41-50	>50
UDify	91.0	91.7	92.0	91.7	90.8	89.0
DepBERT	94.0	94.0	93.7	93.6	92.5	90.6
UDify+RNG-Tr	94.0	94.0	93.7	93.4	92.5	90.4
DepBERT+RNG-Tr	94.2	94.1	93.8	93.4	92.4	90.6
Empty+RNG-Tr	93.9	94.0	93.7	93.5	92.4	90.3

Table 11: Accuracy for different sentence lengths.

D.D Error Analysis for each Language

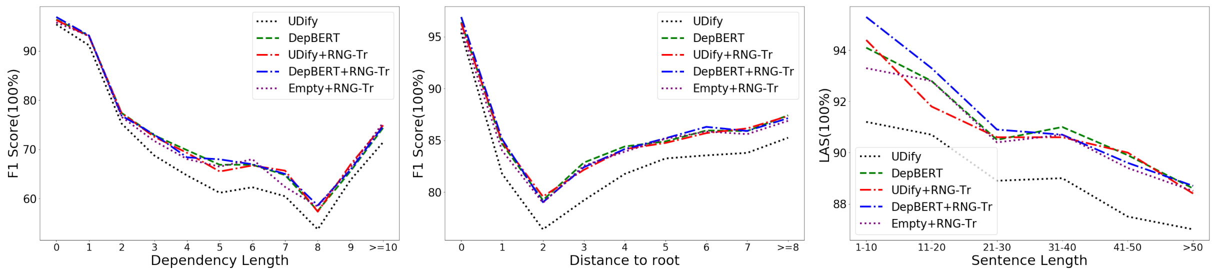

Figure 5: Error analysis of initial parsers (UDify and DepBERT), their integration with the RNG-Tr model, alongside with Empty+RNG-Tr model on Arabic-PADT Treebank.

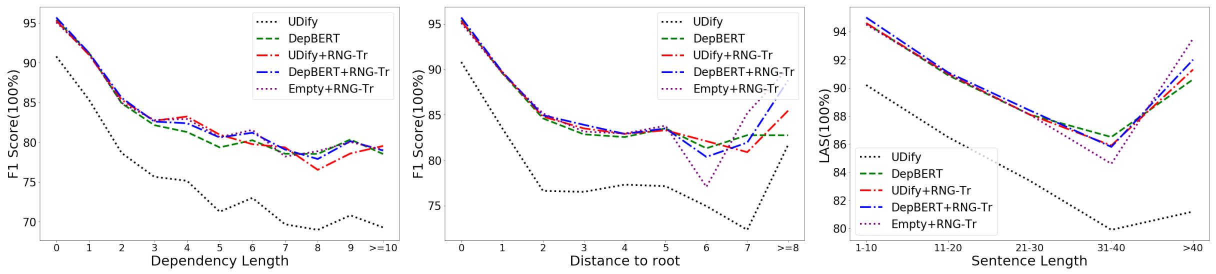

Figure 6: Error analysis of initial parsers (UDify and DepBERT), their integration with the RNG-Tr model, alongside with Empty+RNG-Tr model on Basque-BDT Treebank.

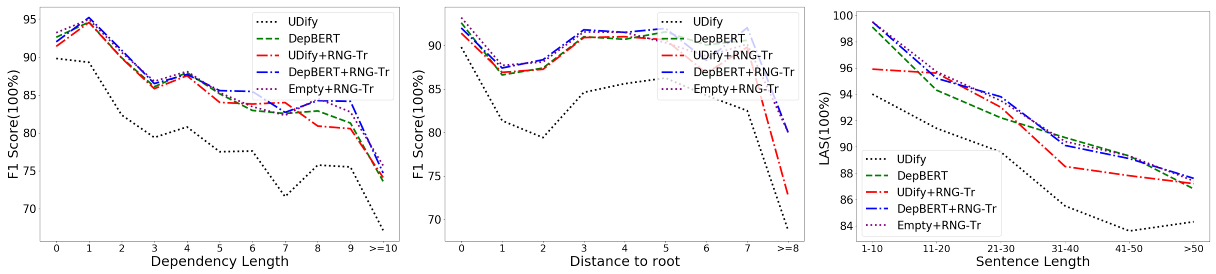

Figure 7: Error analysis of initial parsers (UDify and DepBERT), their integration with the RNG-Tr model, alongside with Empty+RNG-Tr model on Chinese-GSD Treebank.

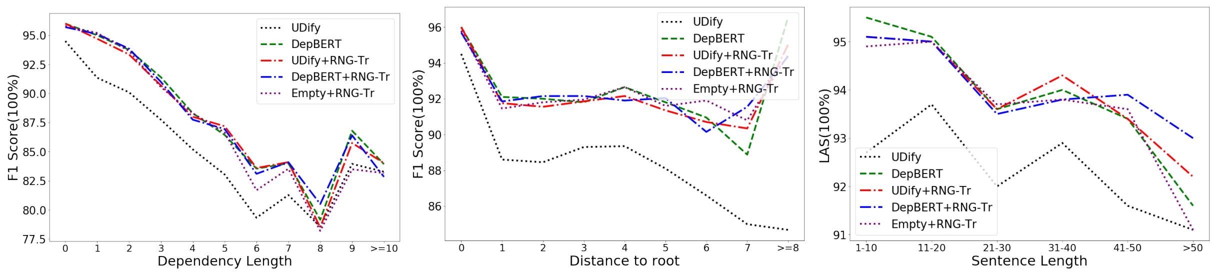

Figure 8: Error analysis of initial parsers (UDify and DepBERT), their integration with the RNG-Tr model, alongside with Empty+RNG-Tr model on English-EWT Treebank.

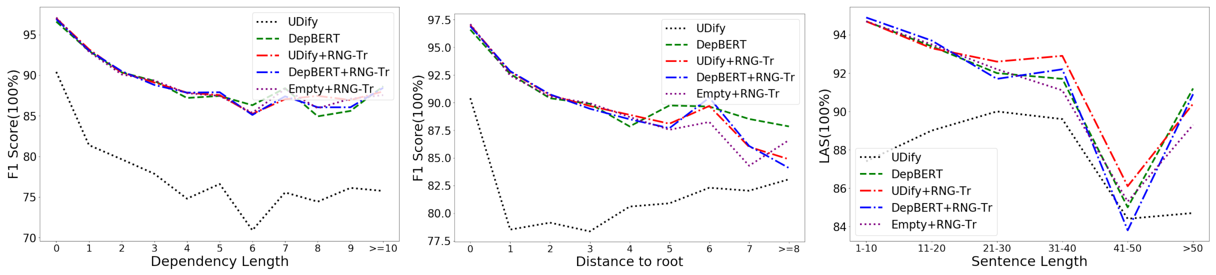

Figure 9: Error analysis of initial parsers (UDify and DepBERT), their integration with the RNG-Tr model, alongside with Empty+RNG-Tr model on Finnish-TDT Treebank.

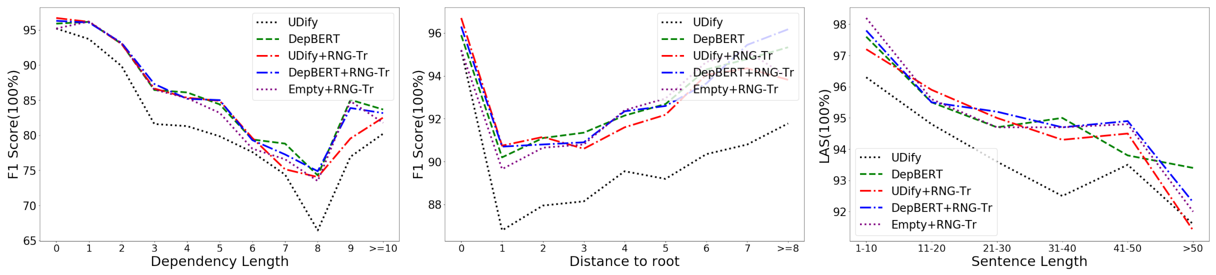

Figure 10: Error analysis of initial parsers (UDify and DepBERT), their integration with the RNG-Tr model, alongside with Empty+RNG-Tr model on Hebrew-HTB Treebank.

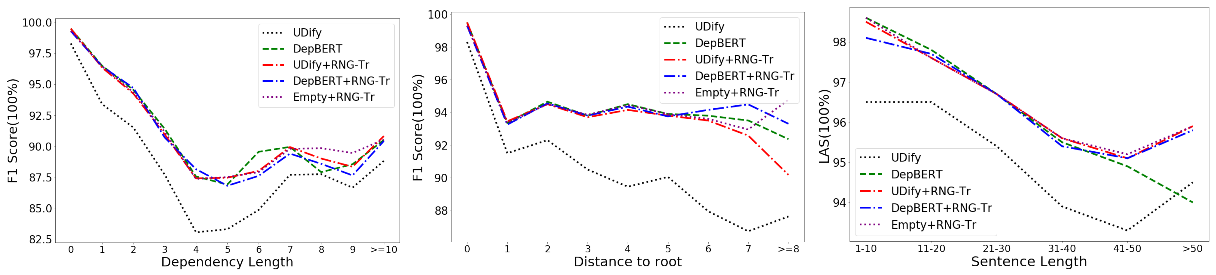

Figure 11: Error analysis of initial parsers (UDify and DepBERT), their integration with the RNG-Tr model, alongside with Empty+RNG-Tr model on Hindi-HDTB Treebank.

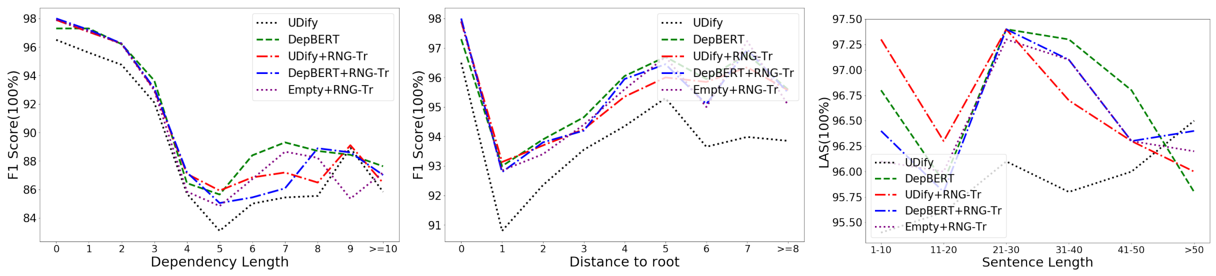

Figure 12: Error analysis of initial parsers (UDify and DepBERT), their integration with the RNG-Tr model, alongside with Empty+RNG-Tr model on Italian-ISDT Treebank.

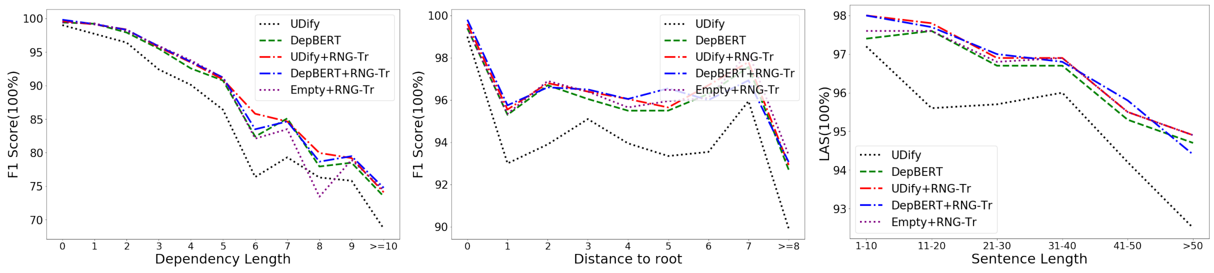

Figure 13: Error analysis of initial parsers (UDify and DepBERT), their integration with the RNG-Tr model, alongside with Empty+RNG-Tr model on Japanese-GSD Treebank.

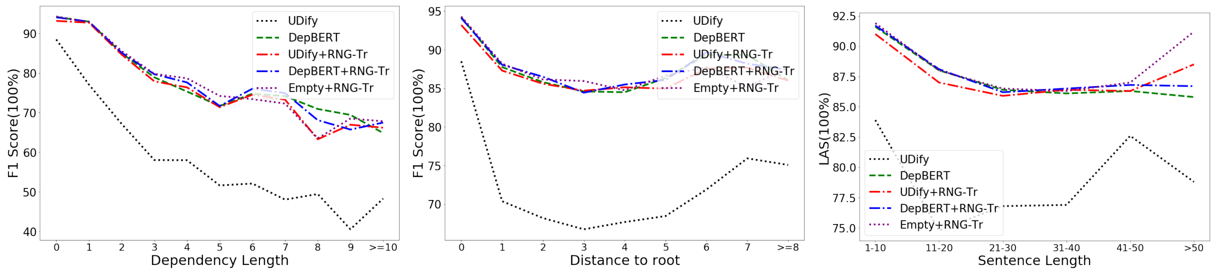

Figure 14: Error analysis of initial parsers (UDify and DepBERT), their integration with the RNG-Tr model, alongside with Empty+RNG-Tr model on Korean-GSD Treebank.

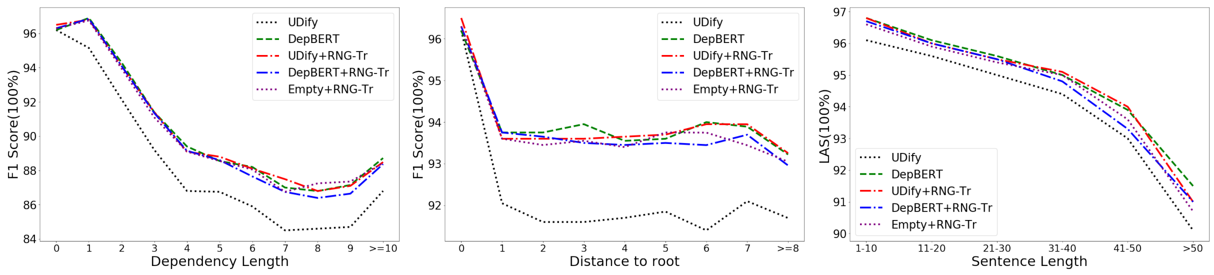

Figure 15: Error analysis of initial parsers (UDify and DepBERT), their integration with the RNG-Tr model, alongside with Empty+RNG-Tr model on Russian-SynTagRus Treebank.

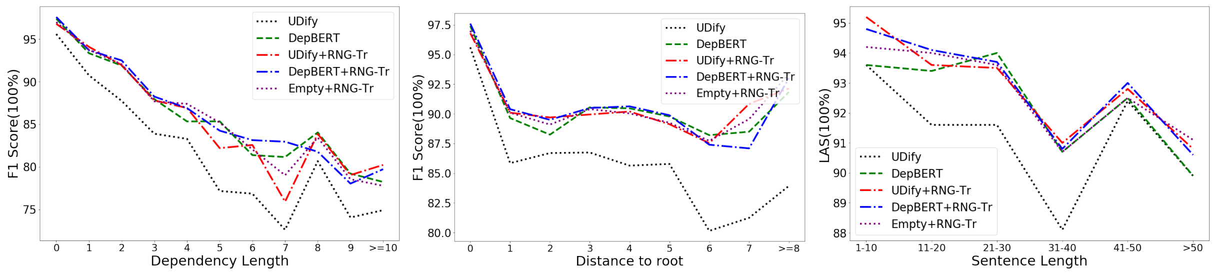

Figure 16: Error analysis of initial parsers (UDify and DepBERT), their integration with the RNG-Tr model, alongside with Empty+RNG-Tr model on Swedish-Talbanken Treebank.

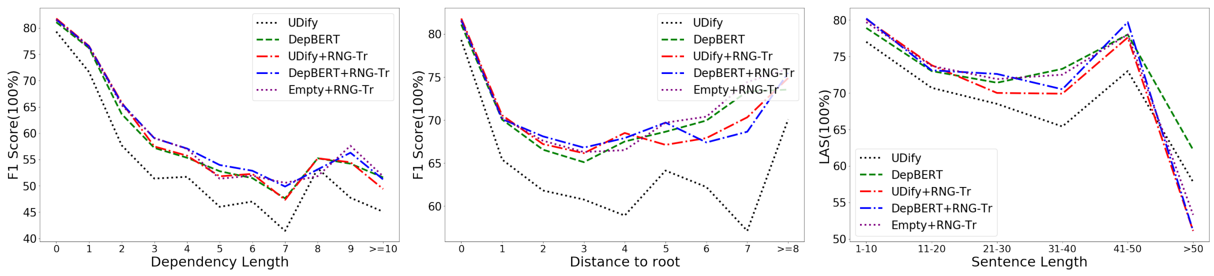

Figure 17: Error analysis of initial parsers (UDify and DepBERT), their integration with the RNG-Tr model, alongside with Empty+RNG-Tr model on Turkish-IMST Treebank.

Appendix E Refinement Analysis

Dep. Type	UDify	UDify+RNG-Tr	DepBERT	DepBERT+RNG-Tr
acl	75.06	83.60 (34.2%)	82.99	83.14 (0.9%)
advcl	71.68	80.34 (30.6%)	80.20	80.20 (0.0%)
advmod	83.45	89.10 (34.1%)	89.35	89.45 (0.9%)
amod	92.10	95.80 (46.8%)	95.90	95.90 (-0.0%)
appos	67.20	74.44 (22.1%)	75.85	75.10 (-3.1%)
aux	95.14	98.40 (67.1%)	98.55	98.65 (6.9%)
case	97.10	98.20 (37.9%)	98.20	98.30 (5.6%)
cc	91.15	93.10 (22.0%)	92.95	92.95 (0.0%)
ccomp	74.29	85.63 (44.1%)	85.43	85.18 (-1.7%)
clf	91.15	95.50 (49.1%)	95.08	96.50 (28.8%)
compound	83.65	90.40 (41.3%)	90.70	90.70 (0.0%)
conj	80.94	83.90 (15.5%)	83.75	83.95 (1.2%)
cop	86.88	93.55 (50.8%)	93.70	93.70 (-0.0%)
csubj	82.62	85.65 (17.4%)	86.58	85.82 (-5.7%)
dep	68.62	76.26 (24.3%)	75.74	76.54 (3.3%)
det	94.20	96.30 (36.2%)	96.30	96.35 (1.3%)
discourse	73.16	80.10 (25.9%)	80.74	82.12 (7.1%)
dislocated	5.72	74.75 (73.2%)	72.00	69.98 (-7.2%)
expl	83.22	86.22 (17.9%)	87.45	86.45 (-7.9%)
fixed	85.39	88.85 (23.6%)	88.75	89.29 (4.9%)
flat	77.12	88.60 (50.2%)	89.40	89.35 (-0.5%)
goeswith	20.94	70.92 (63.2%)	74.35	72.51 (-7.2%)
iobj	84.65	87.04 (15.6%)	87.80	87.04 (-6.2%)
list	22.00	34.89 (16.5%)	36.22	36.23 (0.0%)
mark	92.55	96.00 (46.3%)	96.25	95.95 (-8.0%)
nmod	83.50	87.75 (25.8%)	87.90	88.10 (1.7%)
nsubj	87.50	90.90 (27.2%)	90.85	90.75 (-1.1%)
nummod	90.20	93.09 (29.5%)	93.04	92.79 (-3.7%)
obj	87.15	89.65 (19.5%)	89.95	90.05 (1.0%)
obl	83.00	87.30 (25.3%)	87.70	87.40 (-2.4%)
orphan	31.25	39.54 (12.1%)	37.18	41.84 (7.4%)
parataxis	72.10	75.45 (12.0%)	75.35	74.90 (-1.8%)
punct	87.65	91.55 (31.6%)	91.40	91.65 (2.9%)
root	93.80	95.70 (30.6%)	95.60	95.70 (2.3%)
vocative	63.96	71.30 (20.3%)	68.03	65.95 (-6.5%)
xcomp	81.10	87.70 (34.9%)	87.39	87.19 (-1.6%)

Table 12: F-score of UDify and DepBERT models, and their combination with the RNG-Tr model (and relative difference) for all dependency types on the concatenation of UD Treebanks.