Graph-to-Graph Transformer for Transition-based Dependency Parsing

Alireza Mohammadshahi
Idiap Research Inst. and EPFL / Switzerland

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

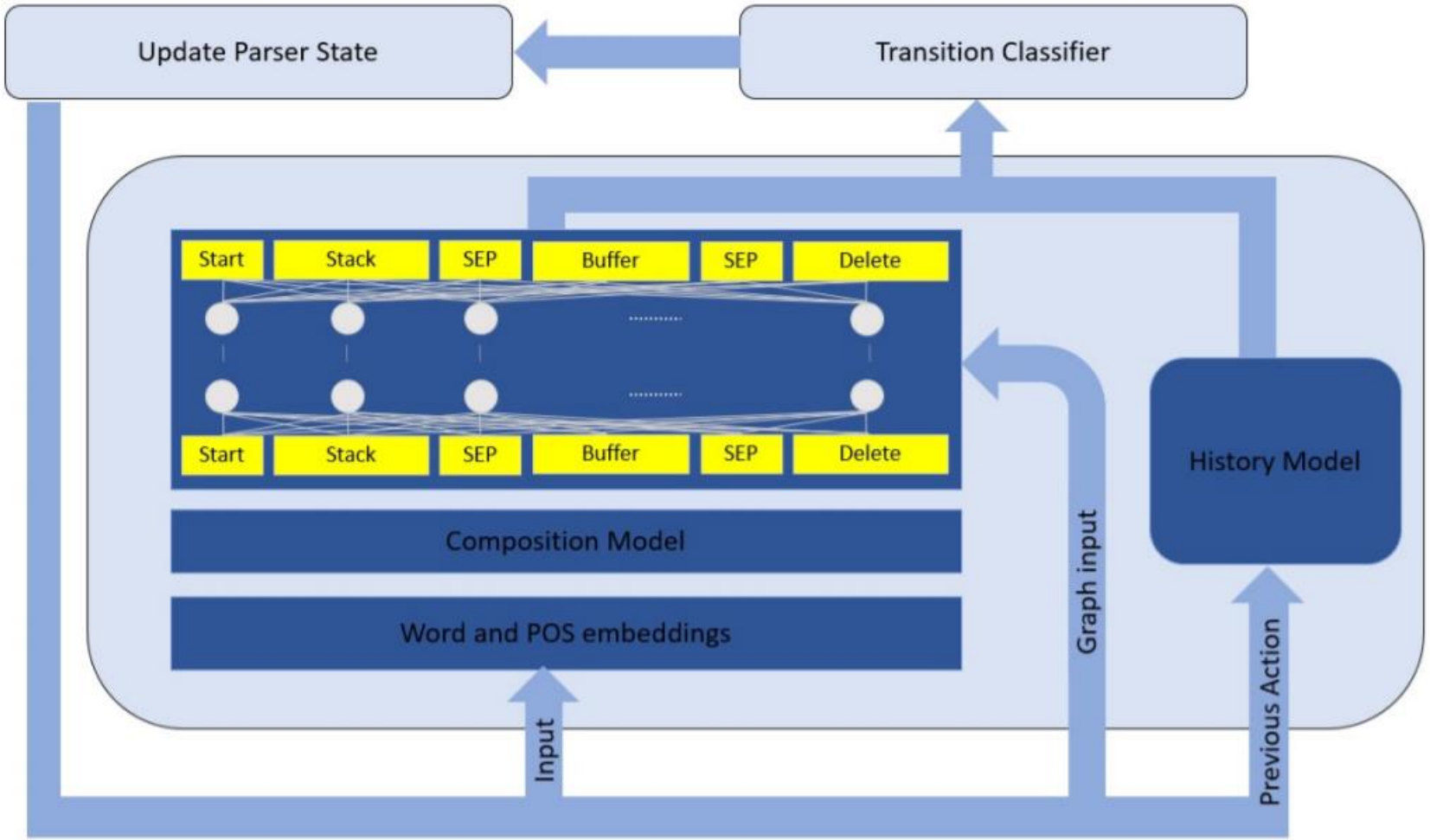

Figure 1: The Graph-to-Graph Transformer parsing model.

- `LEFT-ARC` (l): Add an arc $s_1 \rightarrow s_2$ with label l to G, remove s_2 from the stack, and insert s_2 in the delete list. Precondition: stack length must be greater than 2.
- `RIGHT-ARC` (l): Add an arc $s_2 \rightarrow s_1$ with label l to G, remove s_1 from the stack, and insert s_1 to the delete list. Precondition: stack length must be greater than 2.
- `SHIFT`: moves b_1 from the buffer to the top of the stack. Precondition: buffer length must be greater than 1.

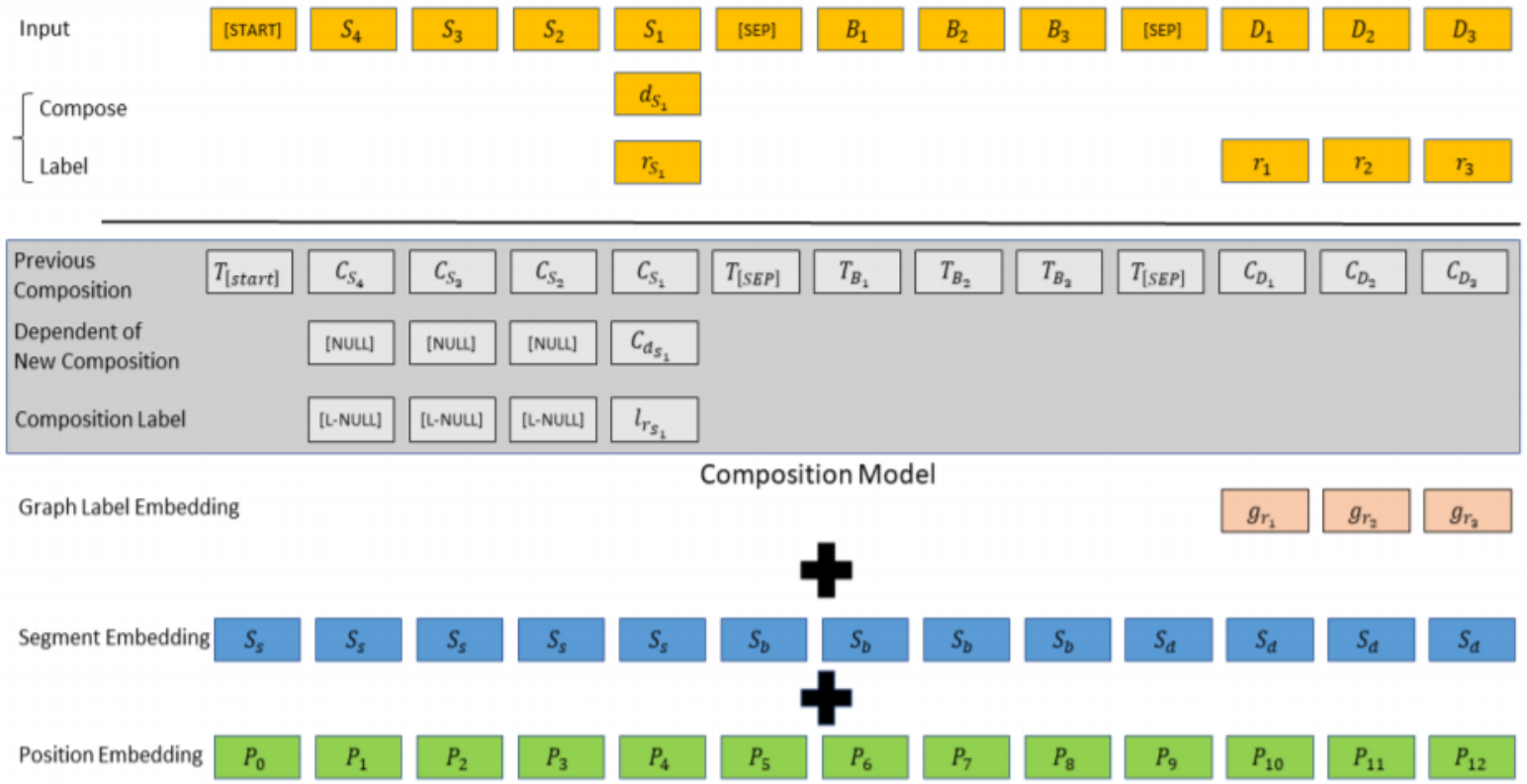

Figure 2: Input embeddings of self-attention model in G2G Transformer at a specific time step. The input embeddings are the summation of output embeddings of composition model (described in section 3.1.2), segment embeddings, position embeddings, and graph label embeddings

$$C_i^{t+1} = Comp((\psi_i^t, \omega_i^t, r_i^t)) + C_i^t$$

$$\begin{cases} \texttt{RIGHT-ARC}(r): \\ \quad \psi_1^t = C_2^t,\ \omega_1^t = C_1^t,\ r_1^t = r, \\ \quad \psi_{i\neq 1}^t = C_{i+1}^t,\ \omega_{i\neq 1}^t = [\texttt{NULL}],\ r_{i\neq 1}^t = [\texttt{L-NULL}] \\ \texttt{LEFT-ARC}(r): \\ \quad \psi_1^t = C_1^t,\ \omega_1^t = C_2^t,\ r_1^t = r, \\ \quad \psi_{i\neq 1}^t = C_{i+1}^t,\ \omega_{i\neq 1}^t = [\texttt{NULL}],\ r_{i\neq 1}^t = [\texttt{L-NULL}] \\ \texttt{SHIFT}: \\ \quad \psi_1^t = T_{B_1}^t,\ \omega_1^t = [\texttt{NULL}],\ r_1^t = [\texttt{L-NULL}]. \\ \quad \psi_{i\neq 1}^t = C_{i-1}^t,\ \omega_{i\neq 1}^t = [\texttt{NULL}],\ r_{i\neq 1}^t = [\texttt{L-NULL}] \end{cases}$$

$$\theta^t = [z^t_{s_1}, z^t_{s_2}]$$

Relation	assigned dimension
None	0
(head $<$ dependent)	1
(dependent $<$ head)	2

Table 1: Index of the type of dependency relation. In this work we only input the unlabelled directionality of the relation, if any.

- No Relation: Do `SHIFT`
- Right Relation: Do `RIGHT-ARC`
- Left Relation: Do `LEFT-ARC`

$$a^t = Exist([\theta^t, h^t])$$
$$r^t = Relation([\theta^t, h^t])$$

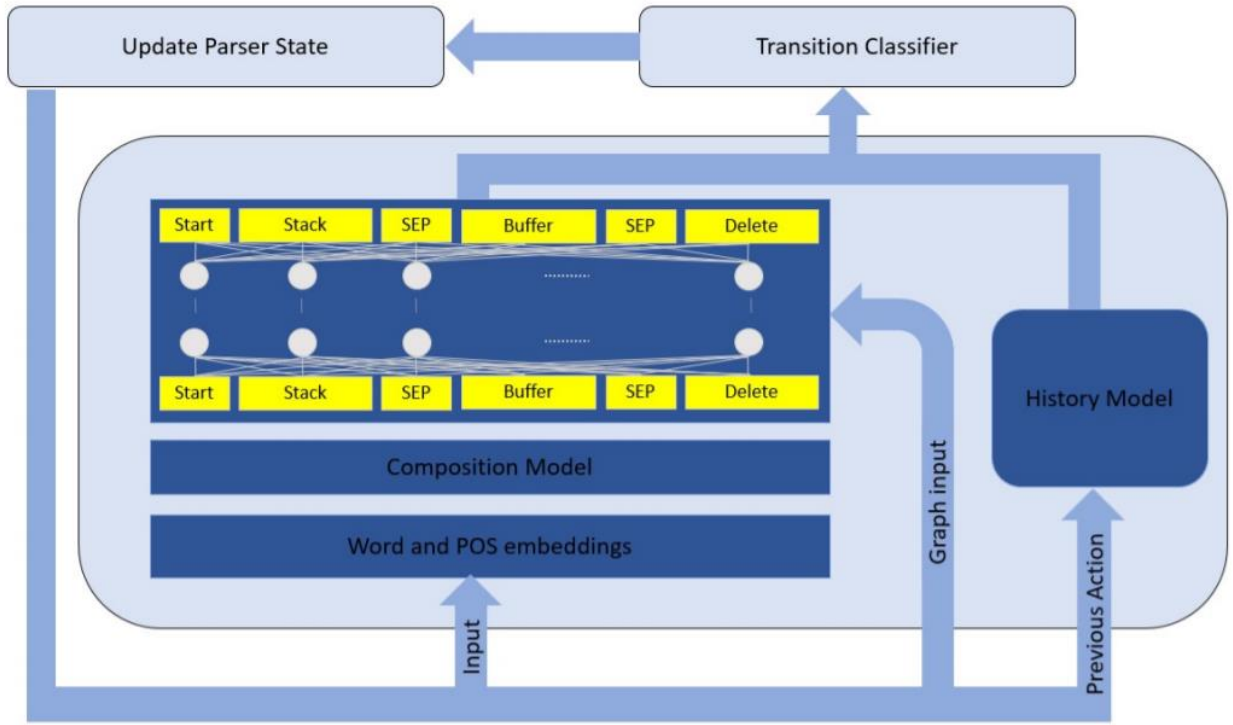

Figure 1: The Graph-to-Graph Transformer parsing model.

BERT-Pre-training

- But unlike the previous work we are aware of which has used BERT pre-training, our version of Transformer has novel inputs which were not present when BERT was trained. These novel inputs are the graph inputs to the attention mechanism, and the composition embeddings. Also, the input sequence has a novel structure, which is only partially similar to the input sentences which BERT was trained on. So it is not clear that BERT pre-training will even work with this novel architecture.

- To evaluate whether BERT pre-training works for our proposed architecture, we initialise the weights of the Graph2Graph Transformer model with the first n layers of BERT, where n is the number of layers in our model

To demonstrate the usefulness of each part of the proposed G2G Transformer model, we compare the full model (G2G Tr) with four different reduced versions of the model. We define the Dependency Transformer (DepTr) model as the same model described in Section 3 but without the composition and history models, and without graph inputs to attention (i.e. attention Equation 8). This model does, however, include the same graph output mechanism, conditioning on the two tokens on the top of the stack. Then we add the history model (DepTr+H), and both composition and history models (DepTr+CH), to the Dependency Transformer baseline. Finally, we also consider a version of the full model with the graph output mechanism removed (G2CLS Tr), where we predict the next parser action from the `START` symbol's token embedding (referred to as `CLS` in BERT). All five of these models are evaluated both with and without initialisation with the first n layers of a pre-trained BERT model.

	Test Set	
	UAS	LAS
Chen et al. (2014)	91.80	89.60
Dyer et al. (2015)	93.10	90.90
Ballesteros et al. (2016)	93.56	92.41
Weiss et al. (2015)	94.26	91.42
Andor et al. (2016)	**94.61**	**92.79**
DepTr	88.40	84.23
DepTr+H	90.44	86.91
DepTr+CH	92.35	89.51
G2CLS Tr	92.73	90.65
G2G Tr	93.21	91.06
BERT DepTr	91.61	87.81
BERT DepTr+H	92.90	89.42
BERT DepTr+CH	**94.86**	92.28
BERT G2CLS Tr	94.27	92.29
BERT **G2G** Tr	**95.30**	**93.44**
BERT **G2G** Tr 7-layer	**95.64**	**93.81**

Table 2: Results on English WSJ Treebank Stanford dependencies. All models have 6 layers of self-attention, except the last line which has 7 layers.

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

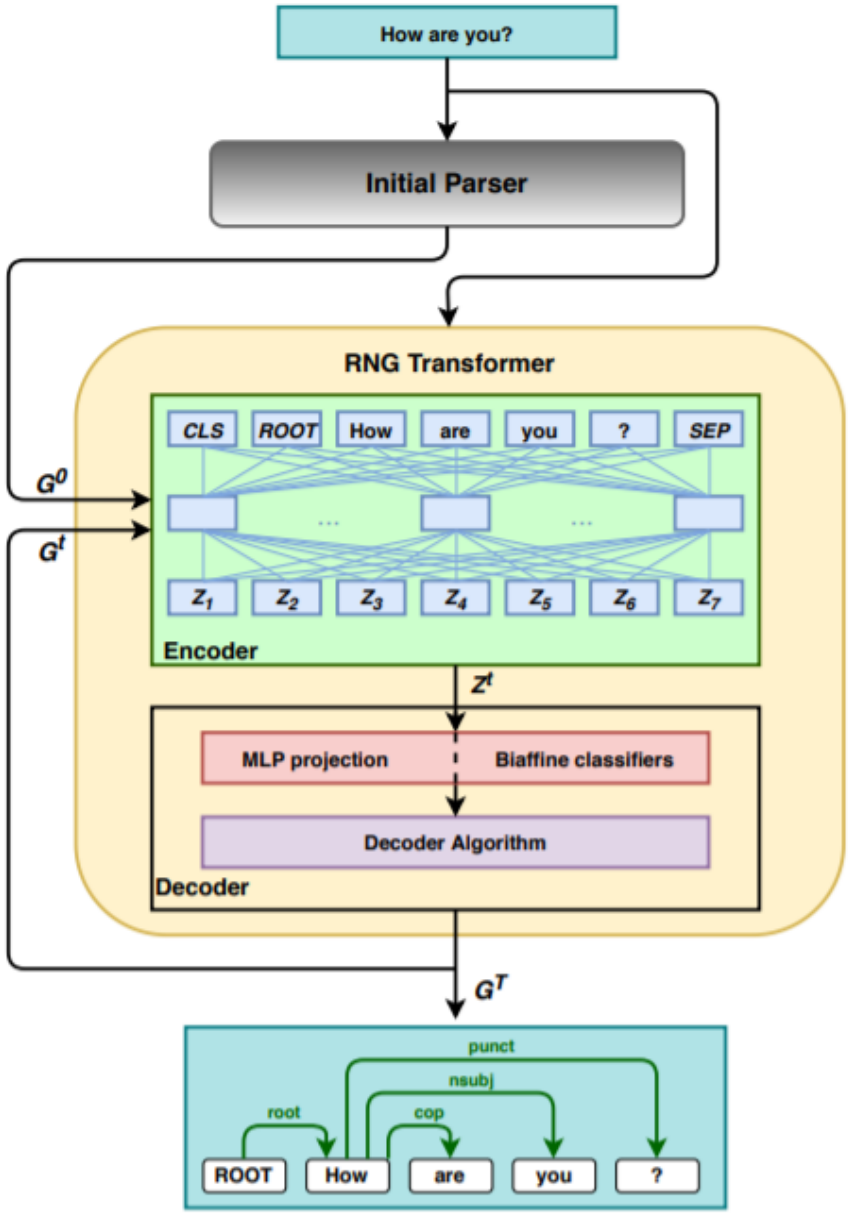

Figure 1: Recursive Non-autoregressive Graph-to-Graph Transformer architecture.

We can formalise the RNG-Tr model as:

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

dependency graph at iteration t is specified as:

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

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

Self-Attention Mechanism

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 $A=(a_1,\ldots,a_N)$ computed by an attention head are:

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

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:

$$\alpha_{ij} = \frac{\texttt{exp}(e_{ij})}{\sum \texttt{exp}(e_{ij})}$$

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

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

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.

Training

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

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.

DepBERT model

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

Results and Discussion

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]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).

+x%: relative error reduction

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.

Refinement Analysis

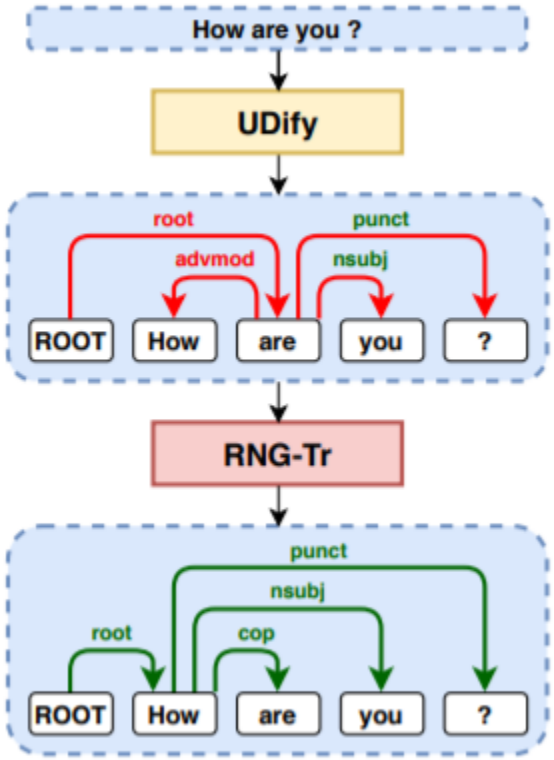

Figure 3: Example of UDify+RNG-Tr on English UD Treebank.

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.

[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.

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.

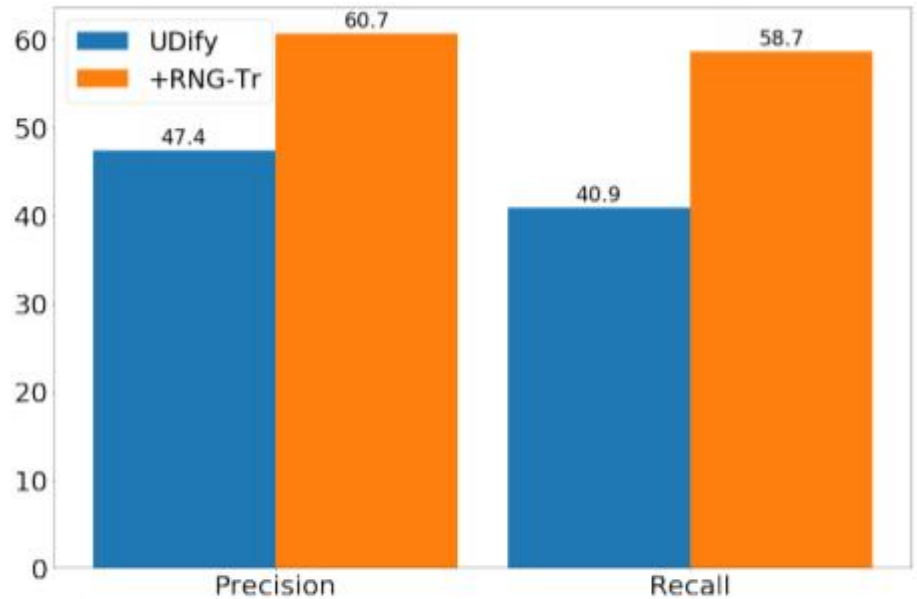

Figure 4: Precision and Recall of Udify and UDify+RNG-Tr on non-projective dependency trees of the UD Treebanks.

Time complexity

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 $O(n^3)$.