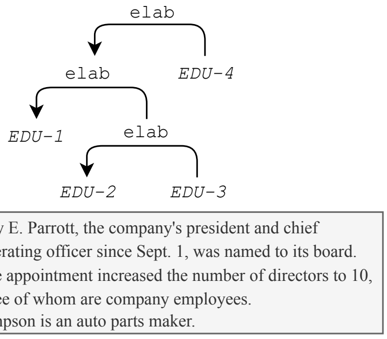
Top-down Discourse Parsing via Sequence Labelling

Fajri Koto **Jey Han Lau** **Timothy Baldwin**
School of Computing and Information Systems
The University of Melbourne
ffajri@student.unimelb.edu.au, jeyhan.lau@gmail.com, tbaldwin@unimelb.edu.au

Abstract

We introduce a top-down approach to discourse parsing that is conceptually simpler than its predecessors (Kobayashi et al., 2020; Zhang et al., 2020). By framing the task as a sequence labelling problem where the goal is to iteratively segment a document into individual discourse units, we are able to eliminate the decoder and reduce the search space for splitting points. We explore both traditional recurrent models and modern pre-trained transformer models for the task, and additionally introduce a novel dynamic oracle for top-down parsing. Based on the `Full` metric, our proposed LSTM model sets a new state-of-the-art for RST parsing.[1]



EDU-1: Roy E. Parrott, the company's president and chief operating officer since Sept. 1, was named to its board.
EDU-2: The appointment increased the number of directors to 10,
EDU-3: three of whom are company employees.
EDU-4: Simpson is an auto parts maker.

Figure 1: An example discourse tree, from the RST Discourse Treebank (`elab` = elaboration).

1 Introduction

Discourse analysis involves the modelling of the structure of text in a document. It provides a systematic way to understand how texts are segmented hierarchically into discourse units, and the relationships between them. Unlike syntax parsing which models the relationship of words in a sentence, discourse parsing operates at the document-level, and aims to explain the flow of writing. Studies have found that discourse parsing is beneficial for downstream NLP tasks including document-level sentiment analysis (Bhatia et al., 2015) and abstractive summarization (Koto et al., 2019).

Rhetorical Structure Theory (RST; Mann and Thompson (1988)) is one of the most widely used discourse theories in NLP (Hernault et al., 2010; Feng and Hirst, 2014; Ji and Eisenstein, 2014; Li et al., 2016; Yu et al., 2018). RST organizes text spans into a tree, where the leaves represent the basic unit of discourse, known as elementary discourse units (EDUs). EDUs are typically clauses of a sentence. Non-terminal nodes in the tree represent discourse unit relations.

In Figure 1, we present an example RST tree with four EDUs spanning two sentences. In this discourse tree, EDUs are hierarchically connected with arrows and the discourse label `elab`. The direction of arrows indicates the nuclearity of relations, wherein a "satellite" points to its "nucleus". The satellite unit is a supporting sentence for the nucleus unit and contains less prominent information. It is standard practice that the RST tree is trained and evaluated in a right-heavy binarized manner, resulting in three forms of binary nuclearity relationships between EDUs: Nucleus–Satellite, Satellite–Nucleus, and Nucleus–Nucleus. In this work, eighteen coarse-grained relations are considered as discourse labels, consistent with earlier work (Yu et al., 2018).[2]

Work on RST parsing has been dominated by the bottom-up paradigm (Hernault et al., 2010; Feng and Hirst, 2014; Ji and Eisenstein, 2014; Braud et al., 2017; Morey et al., 2017; Yu et al., 2018). These methods produce very competitive benchmarks, but in practice it is not a straightforward

[1] Code and trained models: https://github.com/fajri91/NeuralRST-TopDown

[2] Details of individual relations can be found at: http://www.sfu.ca/rst/index.html

Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics, pages 715–726
April 19 - 23, 2021. ©2021 Association for Computational Linguistics

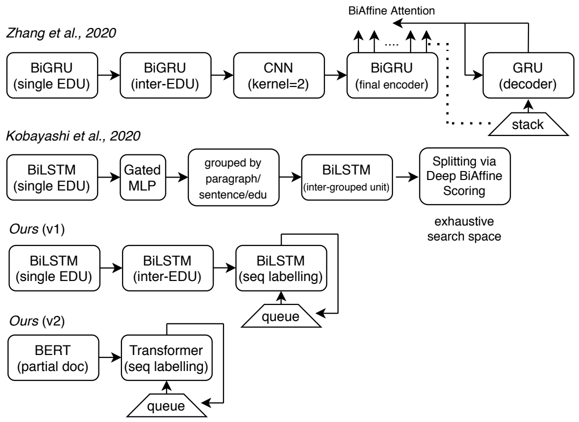


Figure 2: Comparison of our top-down models with Zhang et al. (2020) and Kobayashi et al. (2020).

approach (e.g. transition-based parser with actions prediction steps). Furthermore, bottom-up parsing limits the tree construction to local information, and macro context such as global structure/topic is prone to be under-utilized. As a result, there has recently been a move towards top-down approaches (Kobayashi et al., 2020; Zhang et al., 2020).

The general idea behind top-down parsing is to find splitting points in each iteration of tree construction. In Figure 2, we illustrate how our architecture differs from Zhang et al. (2020) and Kobayashi et al. (2020). First, Zhang et al. (2020) utilize four levels of encoder that comprise 3 Bi-GRUs and 1 CNN layer. The splitting mechanism is applied through a decoder, a stack, and bi-affine attention mechanisms. Kobayashi et al. (2020) use the gold paragraph and sentence boundaries to aggregate a representation for each unit, and generate the tree based on these granularities. Two Bi-LSTMs are used, with splitting points determined by exhaustively calculating the bi-affine score of each possible split. The use of paragraph boundaries can explicitly lower the difficulty of the task, as 77% of paragraphs in the English RST Discourse Treebank ("RST-DT") are actually text spans (Carlson et al., 2001). These boundaries are closely related to gold span boundaries in evaluation.

In this paper, we propose a conceptually simpler top-down approach for RST parsing. The core idea is to frame the problem as a sequence labelling task, where the goal is to iteratively find a segmentation boundary to split a sequence of discourse units into two sub-sequences of discourse units. This way, we are able to simplify the architecture, in eliminating the decoder as well as reducing the search space for splitting points. Specifically, we use an LSTM (Hochreiter and Schmidhuber, 1997) or pre-trained BERT (Devlin et al., 2019) as the segmenter, enhanced in a number of key ways.

Our primary contributions are as follows: (1) we propose a novel top-down approach to RST parsing based on sequence labelling; (2) we explore both traditional sequence models such as LSTMs and also modern pre-trained encoders such as BERT; (3) we demonstrate that adding a weighting mechanism during the splitting of EDU sequences improves performance; and (4) we propose a novel dynamic oracle for training top-down discourse parsers.

2 Related Work

Previous work on RST parsing has been dominated by bottom-up approaches (Hernault et al., 2010; Joty et al., 2013; Li et al., 2016; Braud et al., 2017; Wang et al., 2017). For example, Ji and Eisenstein (2014) introduce DPLP, a transition-based parser based on an SVM with representation learning, combined with some heuristic features. Braud et al. (2016) propose joint text segment representation learning for predicting RST discourse trees using a hierarchical Bi-LSTM. Elsewhere, Yu et al. (2018) showed that implicit syntax features extracted from a dependency parser (Dozat and Manning, 2017) are highly effective for discourse parsing.

Top-down parsing is well established for constituency parsing and language modelling (Johnson, 1995; Roark and Johnson, 1999; Roark, 2001; Frost et al., 2007), but relatively new to discourse parsing. Lin et al. (2019) propose a unified framework based on pointer networks for sentence-level discourse parsing, while Liu et al. (2019) employ hierarchical pointer network parsers.

Morey et al. (2017) found that most previous studies on parsing RST discourse tree were incorrectly benchmarked, e.g. one study uses macro-averaging while another use micro-averaging.[3] They also advocate for evaluation based on micro-averaged F-1 scores over labelled attachment decisions (a la the original Parseval).

Pre-trained language models (Radford et al., 2018; Devlin et al., 2019) have been shown to benefit a multitude of NLP tasks, including discourse analysis. For example, BERT models have been used for classifying discourse markers (Sileo et al.,

[3] After standardizing evaluation (based on micro-averaged F-1), they found that DPLP achieves the best Full performance, outperforming the deep learning models.

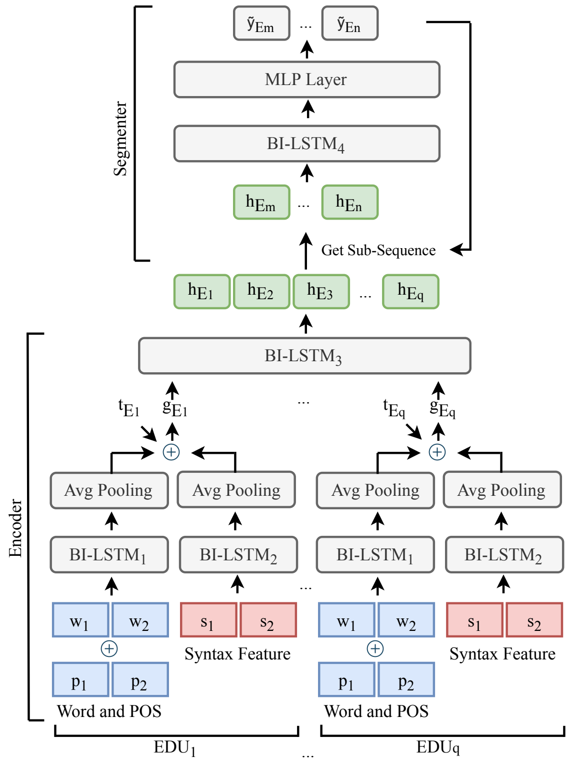


Figure 3: Architecture of the LSTM model.

2019) and discourse relations (Nie et al., 2019; Shi and Demberg, 2019). To the best of our knowledge, however, pre-trained models have not been applied in the generation of full discourse trees, which we address here by experimenting with BERT for top-down RST parsing.

3 Top-down RST Parsing

We frame RST parsing as a sequence labelling task, where given a sequence of input EDUs, the goal is to find a segmentation boundary to split the sequence into two sub-sequences. This is realized by training a sequence labelling model to predict a binary label for each EDU, and select the EDU with the highest probability to be the segmentation point. After the sequence is segmented, we repeat the same process for the two sub-sequences in a divide-and-conquer fashion, until all sequences are segmented into individual units, producing the binary RST tree (e.g. Figure 1).

3.1 LSTM Model

As illustrated in Figure 3, our LSTM parser consists of two main blocks: an encoder and a segmenter. For the encoder, we follow Yu et al. (2018) in using two LSTMs (Bi-LSTM$_1$ and Bi-LSTM$_2$) to produce EDU encodings by processing: (1) x_i, the concatenation of word embedding w_i and POS tag embedding p_i; and (2) syntax embedding s_i, the output of the MLP layer of the bi-affine dependency parser (Dozat and Manning, 2017). Similar to Yu et al. (2018), we then take the average of the output states for both LSTMs over the EDU, and concatenate it with an EDU type embedding t_{E_j} (which distinguishes the last EDU in a paragraph from other EDUs) to produce the final encoding:

$$\begin{aligned} x_i &= w_i \oplus p_i \\ \{a_1^w, .., a_p^w\} &= \text{Bi-LSTM}_1(\{x_1, .., x_p\}) \\ \{a_1^s, ..., a_p^s\} &= \text{Bi-LSTM}_2(\{s_1, .., s_p\}) \\ g_{E_j} &= \text{Avg-Pool}(\{a_1^w, .., a_p^w\}) \oplus \\ &\quad \text{Avg-Pool}(\{a_1^s, .., a_p^s\}) \oplus t_{E_j} \qquad (1) \end{aligned}$$

where E_j is an EDU, p is the number of words in E_j, and $\oplus$ denotes the concatenate operation. t_{E_j} is generally an implicit paragraph boundary feature, and provides a fair benchmark with previous models. In Section 4.3, we also show results without paragraph boundary features.

As each EDU is processed independently, we use another LSTM (Bi-LSTM$_3$) to capture the inter-EDU relationship to obtain a contextualized representation h_{E_j}:

$$\{h_{E_1}, ..., h_{E_q}\} = \text{Bi-LSTM}_3(\{g_{E_1}, ..., g_{E_q}\})$$

where q is the number of EDUs in the document. Note that h_{E_j} is the final encoder output (see Figure 3) and is only computed once for each document.

The second part is the segmenter. We frame segmentation as a sequence labelling problem with $y_{E_j} \in \{0, 1\}$, where 1 denotes the splitting point, and 0 a non-splitting point. For each EDU sequence there is exactly one EDU that is labeled 1, and we start from the full EDU sequence (whole document) and iteratively perform segmentation until we are left with individual EDUs. We use a queue to store the two EDU sub-sequences as the result of the segmentation process. In total, there are $q - 1$ iterations of segmentation (recall that q is the total number of EDUs in the document).

As segmentation is done iteratively in a divide-and-conquer fashion, h_{E_j} serves as the input to the segmenter, which takes a (sub)sequence of EDUs to predict the segmentation position:

$$\begin{aligned} \{h'_{E_m}, .., h'_{E_n}\} &= \text{Bi-LSTM}_4(\{h_{E_m}, .., h_{E_n}\}) \\ \tilde{y}_{E_j} &= \sigma(\text{MLP}(h'_{E_j})) \end{aligned}$$

where m/n are the starting/ending index of the EDU sequence,[4] and $\tilde{y}_{E_j}$ gives the probability of a segmentation. From preliminary experiments we found that it's important to have this additional Bi-LSTM$_4$ to perform the EDU sub-sequence segmentation point prediction.

3.2 Transformer Model

Adapting BERT to discourse parsing is not trivial due to the limited number of input tokens it takes (typically 512 tokens), which is often too short for documents. Moreover, BERT is designed to encode sentences (and only two at maximum), where in our case we want to encode sequences of EDUs that span multiple sentences.

In our case, EDU truncation is not an option (since that would produce an incomplete RST tree), and the average number of words per document in our data is 521 (741 word pieces after BERT tokenization), which is much larger than the 512 limit. We therefore break the document into a number of partial documents, each consisting of multiple sentences that fit into the 512 token limit. This way, we allow the model to capture the fine-grained word-to-word relationships across (most) EDUs. Each partial document is then processed based on Liu and Lapata (2019) trick where we use an alternating even/odd segmentation embedding to encode all the EDUs in a document.

We illustrate this approach in Figure 4. First, all EDUs are formatted to start with [CLS] and end with [SEP], and words are tokenized using WordPiece. If the document has more than 512 tokens, we break it into multiple partial documents based on EDU boundaries, and pad accordingly (e.g. in Figure 4 we break the example document of 3 EDUs into 2 partial documents), and process each partial document independently with BERT.

We also experimented with the second alternative by encoding each EDU independently first with BERT, and use a second inter-EDU transformer to capture the relationships between EDUs. Preliminary experiments, however, suggest that this approach produces sub-optimal performance.

In Figure 4 each token is assigned three kinds of embeddings: (1) word, (2) segment, and (3) position. The input vector is computed by summing these three embeddings, and fed into BERT (initialized with `bert-base`). The output of BERT gives us a contextualized embedding for each token, and we use the [CLS] embedding as the encoding for each EDU (g_{E_j}).

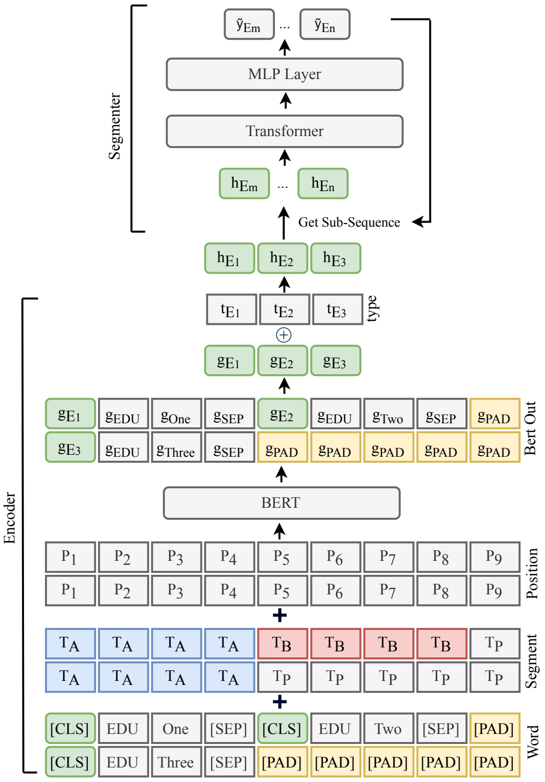

Segmenter
ỹEm ... ỹEn
MLP Layer
Transformer
hEm ... hEn
Get Sub-Sequence
hE1 hE2 hE3
tE1 tE2 tE3
type
gE1 gE2 gE3
gE1 gEDU gOne gSEP gE2 gEDU gTwo gSEP gPAD
gE3 gEDU gThree gSEP gPAD gPAD gPAD gPAD gPAD
Bert Out
BERT
Encoder
P1 P2 P3 P4 P5 P6 P7 P8 P9
P1 P2 P3 P4 P5 P6 P7 P8 P9
Position
TA TA TA TA TB TB TB TB TP
TA TA TA TA TP TP TP TP TP
Segment
[CLS] EDU One [SEP] [CLS] EDU Two [SEP] [PAD]
[CLS] EDU Three [SEP] [PAD] [PAD] [PAD] [PAD] [PAD]
Word

Figure 4: Architecture of the transformer model. In practice, 1 row of input can have more than two EDUs.

Unlike the LSTM model, we do not incorporate syntax embeddings into the transformer model as we found no empirical benefit (see Section 4.3). This observation is in line with other studies (e.g. Jawahar et al. (2019)) that have found BERT to implicit encode syntactic knowledge.

For the segmenter we use a second transformer (initialized with random weights) to capture the inter-EDU relationships for sub-sequences of EDUs during iterative segmentation:

$$\{h'_{E_m}, .., h'_{E_n}\} = \text{transformer}(\{h_{E_m}, .., h_{E_n}\})$$
$$\tilde{y}_{E_j} = \sigma(\text{MLP}(h'_{E_j}))$$

where $\tilde{y}_{E_j}$ gives the probability of a segmentation, and h_{E_j} is the concatenation of the output of BERT (g_{E_j}) and the EDU type embedding (t_{E_j}).

3.3 Nuclearity and Discourse Relation Prediction

In Figure 5, we give an example of the iterative segmentation process to construct the RST tree. In each iteration, we pop a sequence from the queue (initialized with the original sequence of EDUs in

[4] In the first iteration, $m = 1$ and $n = q$ (number of EDUs in the document).

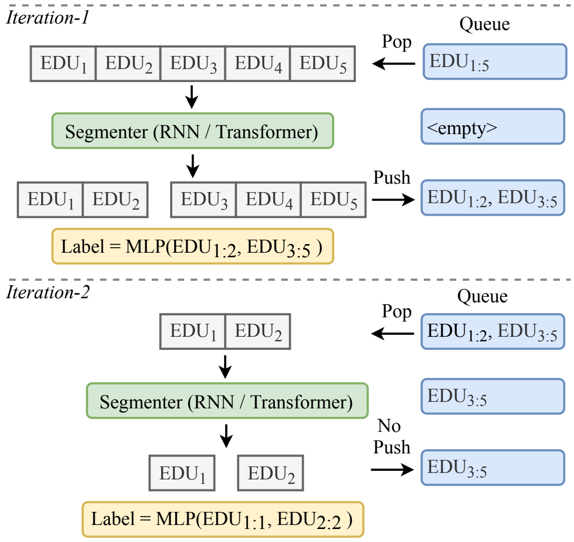


Figure 5: Nuclearity and relation prediction.

the document) and compute the segmentation label for each EDU using an LSTM (Section 3.1) or transformer (Section 3.2). After the sequence is segmented (using the ground truth label during training, or the highest-probability label at test time), we push to the queue the two sub-sequences (if they contain at least two EDUs) and repeat this process until the queue is empty.

In addition to segmentation, we also need to predict the nuclearity/satellite relationship (3 classes) and the discourse label (18 classes) for the segmented pairs. To that end, we average the EDU encodings for the segments, and feed them to a MLP layer to predict the nuclearity and discourse labels:

$$
\begin{aligned}
u_l &= \text{Avg-Pool}(h'_{E_m}, ..., h'_{E_{m+ind}}) \\
u_r &= \text{Avg-Pool}(h'_{E_{m+ind+1}}, ..., h'_{E_n}) \\
z_{nuc+dis} &= \text{softmax}(\text{MLP}(u_l, u_r))
\end{aligned}
$$

where ind is the index of the segmentation point (given by the ground truth during training, or argmax of the segmentation probabilities $\tilde{y}_{E_j}$ at test time), and $z_{nuc+dis}$ gives the joint probability distribution over the nuclearity and discourse classes.[5]

3.4 Segmentation Loss with Penalty

One drawback of the top-down approach is that segmentation errors incurred closer to the root can be detrimental, as the error will propagate to the rest of the sub-trees. To address this, we explore scaling the segmentation loss based on the current tree depth and the number of EDUs in the input sequence. Preliminary experiments found that both approaches work, but that the latter is marginally better, and so we present results using the latter.

Formally, the modified segmentation loss of an example (document) is given as follows:

$$
\begin{aligned}
L(E_{m:n}) = -\sum_{i=m}^{n} \Big(& y_{E_i} \log(\tilde{y}_{E_i}) + \\
& (1 - y_{E_i}) \log(1 - \tilde{y}_{E_i}) \Big) \\
\mathcal{L}_{seg} = \frac{1}{|S|} \sum_{(m,n) \in S} & (1 + (n - m)^{\beta}) L(E_{m:n})
\end{aligned}
$$

where $y_{E_i} \in \{0, 1\}$ is the ground truth segmentation label, $L(E_{m:n})$ is the cross-entropy loss for an EDU sequence, S is the set of all EDU sequences (based on ground truth segmentation), and β is a scaling hyper-parameter.

To summarize, the total training loss of our model is a (weighted) combination of segmentation loss ($\mathcal{L}_{seg}$) and nuclearity-discourse prediction loss ($\mathcal{L}_{nuc+dis}$):

$$
\mathcal{L} = \lambda_1 \mathcal{L}_{seg} + \lambda_2 \mathcal{L}_{nuc+dis} \qquad (2)
$$

3.5 Dynamic Oracle

The training regimen for discourse parsing creates an exposure bias, where the parser may struggle to recover when it makes a mistake at test time. Goldberg and Nivre (2012) propose a dynamic oracle for transition-based dependency parsing to tackle this. The idea is to allow the model during training to use its predictions (instead of ground truth actions), and introduce a dynamic oracle to find the next best/optimal action sequences. It does so by comparing the current state of the constructed tree and the gold-standard tree, and aims to minimize the deviation. As the model is exposed to prediction errors during training time, it has a better chance of recovering from them at test time.

We explore a similar idea, and propose a dynamic oracle for our top-down discourse parser. A crucial question to ask when designing a dynamic oracle is: *how can we compare the current state to the gold tree to obtain the next best series of actions when an error occurs during training?* In transition-based parsing, Goldberg and Nivre (2012) compute a cost/loss of each transition by

[5] We also experimented with predicting the nuclearity and discourse labels separately, but found joint prediction to work better in preliminary experiments.

Algorithm 1 Top-down Dynamic Oracle

```
function DYNORACLE($E, O, R$)
    # For training only
    # E is list of EDUs
    # O is gold order for segmentation
    # R is list of gold discourse labels based on O
    $q$ = length($E$); $queue = [E_{1:q}]$
    while $queue$ is not empty do
        $E_{m:n} = queue$.pop()
        $id_{gold}, r_{gold}$ = match($E_{m:n}, O, R$)
        $id_{pred}$ = predictSplit($E_{m:n}$)
        $r_{pred1}$ = predictLabel($E_{m:n}, id_{gold}$) # for loss
        $r_{pred2}$ = predictLabel($E_{m:n}, id_{pred}$) # ignored
        if random() $> \alpha$ then
            $L, R$ = separate($E_{m:n}, id_{gold}$)
        else
            $L, R$ = separate($E_{m:n}, id_{pred}$)
        end if
        $queue$.push($L$) if len($L$) $> 1$
        $queue$.push($R$) if len($R$) $> 1$
    end while
end function
```

counting the gold arcs that are no longer reachable based on the action taken (e.g. SHIFT, REDUCE). We apply similar reasoning when finding the next best segmentation sequence in our dynamic oracle, which we illustrate below with an example.

Say we have a document with 4 EDUs ($E_{1:4}$), and the gold tree given in Figure 6 (left). The correct sequence of segmentation is given by $O_{1:4} = [2, 1, 3, -]$, which means we should first split at E_2 (creating $E_{1:2}$ and $E_{3:4}$), and then at E_1 (creating $E_1, E_2, E_{3:4}$), and lastly at E_3, producing E_1, E_2, E_3, E_4 as the leaves with the gold tree structure. We give the last EDU E_4 a "$-$" label (i.e. O_4 ='$-$') because no segmentation is needed for the last EDU.

Suppose the model predicts to do the first segmentation at E_3. This produces $E_{1:3}$ and E_4. What is the best way to segment $E_{1:3}$ to produce a tree that is as close as possible to the gold tree? The canonical segmentation order $O_{1:3}$ is $[2, 1, -]$ (the label of the last EDU is replaced by '$-$'), from which we can see the next best segmentation is to segment at E_2 to create $E_{1:2}$ and E_3. Creating the canonical segmentation order O, and following it as much as possible, ensures the sub-tree that we're creating for $E_{1:3}$ mimics the structure of the gold tree.

The dynamic oracle labels nuclearity-discourse relations following the same idea. We introduce R, a list of gold nuclearity-discourse relations. For our example $R_{1:4} = [r_2, r_1, r_3, -]$ (based on the gold tree; see Figure 6 (left)). If the model decides to first segment at E_3 and creates $E_{1:3}$ and E_4, when we segment at E_2 (next best choice of segmentation), we will follow R and label the nuclearity-discourse relation with r_1. As before, following the original label list R ensures we keep the nuclearity-discourse relation as faithful as possible (Figure 6 (right bottom)).

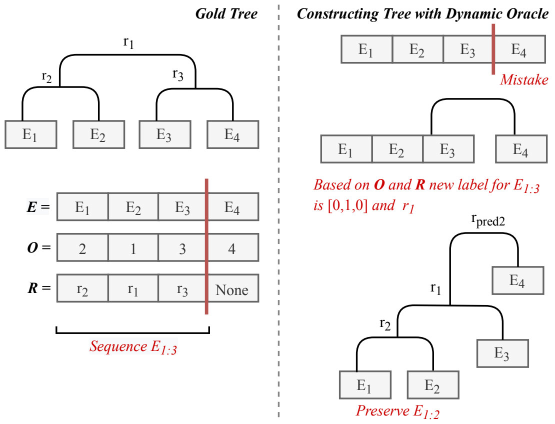

Gold Tree
Constructing Tree with Dynamic Oracle
r1
r2
r3
E1
E2
E3
E4
E =
O =
R =
2
1
3
4
None
Sequence E1:3
Mistake
Based on O and R new label for E1:3 is [0,1,0] and r1
rpred2
Preserve E1:2

Figure 6: Dynamic oracle for top-down approach.

The dynamic oracle of our top-down parser is arguably quicker than that of a transition-based parser, as we do not need to accumulate cost for every transition taken. Instead, the dynamic oracle simply follows the gold segmentation order O to preserve as many subtrees as possible when an error occurs. We present pseudocode for the proposed dynamic oracle in Algorithm 1.

The probability of using the ground truth segmentation or predicted segmentation during training is controlled by the hyper-parameter $\alpha \in [0, 1]$ (see Algorithm 1). Intuitively, this hyper-parameter allows the model to alternate between exploring its (possibly erroneous) segmentation or learning from the ground truth segmentation. The oracle reverts to its static variant when $\alpha = 0$.

4 Experiments

4.1 Data

We use the English RST Discourse Treebank (Carlson et al., 2001) for our experiments, consistent with recent studies (Ji and Eisenstein, 2014; Li et al., 2014; Feng and Hirst, 2014; Yu et al., 2018). The dataset is based on the Wall Street Journal portion of the Penn Treebank (Marcus et al., 1993), with 347 documents for training, and the remaining 38 documents for testing. We use the same development set as Yu et al. (2018), which consists of 35 documents selected from the training set. We also use the same 18 discourse labels. Stanford

Variant	LSTM	Transformer
Vanilla	48.4±0.5	51.3±0.2
+Syntax	50.0±0.7	51.9±0.4
+Penalty	49.6±0.5	**52.1±0.4**
+Syntax+Penalty	**51.6±0.1**	51.8±0.8

Table 1: Feature addition study over the development set to find the best configuration for our models. Presented results are the mean and standard deviation of the `Full` metric (micro-averaged F-score on labelled attachment decisions) over three runs.

CoreNLP (Manning et al., 2014) is used for POS tagging.[6]

4.2 Model Configurations

We experiment with two segmentation models — LSTM (Section 3.1) and transformer (Section 3.2) — both implemented in PyTorch framework.[7] As EDUs are provided in the dataset, no automatic segmentation of EDU is required in our experiments.

For the LSTM model, the dimensionality of the Bi-LSTMs in the encoder is 256, while the segmenter (Bi-LSTM$_4$) is 128 (Figure 3). The embedding dimensions of words, POS tags, EDU type, and syntax features are 200, 200, 100, and 1,200, respectively, and we initialize words in EDU with GloVe embedding (Pennington et al., 2014).[8] For hyper-parameters, we use the following: batch size $= 4$, gradient accumulation $= 2$, learning rate $= 0.001$, dropout probability $= 0.5$, and optimizer = Adam (with epsilon of 1e-6). The loss scaling hyper-parameters (Equation (2)), are tuned based on the development set, and set to $\lambda_1 = 1.0$, and $\lambda_2 = 1.0$.

For the transformer model, the document length limit is set to 512 tokens, and longer documents are broken into smaller partial documents. As before, we truncate each EDU to the first 50 words. We initialize the transformer in the encoder with `bert-base`, and the transformer in the segmenter with random weights (Figure 4). The transformer segmenter has 2 layers with 8 heads and 2048 feed-forward hidden size. The training hyper-parameters are: initial learning rate $=$ 5e-5, maximum epochs $= 250$, warm up $= 2000$ steps, and drop out $= 0.2$. For the λ hyper-parameters, we use the same configuration as for the LSTM model.

We tuned the segmentation loss penalty hyper-parameter β (Section 3.4) and the dynamic oracle hyper-parameter α (Section 3.5) based on the development set. Both the LSTM and transformer models use the same $\beta = 0.35$ and $\alpha = 0.65$. We activate the dynamic oracle after training for 50 epochs for both models.

In terms of evaluation, we use the standard metrics introduced by Marcu (2000): `Span`, `Nuclearity`, `Relation`, and `Full`. We report micro-averaged F-1 scores on labelled attachment decisions (original Parseval), following the recommendation of Morey et al. (2017). Additionally, we also present the evaluation with RST-Parseval procedure in Appendix A.

4.3 Results

We first perform a feature addition study over our models to find the best model configuration; results are presented in Table 1. Note that these results are computed over the *development set*, based on a static oracle.

For the vanilla models, the transformer model performs much better than the LSTM model. Adding syntax features (+Syntax) improves both models, although it's more beneficial for the LSTM. A similar trend is observed when we modify the segmentation loss to penalize the model if a segmentation error is made with more EDUs in the input sequence (+Penalty; Section 3.4): the transformer model sees an improvement of +0.8 while the LSTM model improves by +1.2. Lastly, when we combine both syntax features and the segmentation penalty, the LSTM model again shows an appreciable improvement, while the transformer model drops in performance marginally.[9] Given these results, we use both syntax features and the segmentation penalty for the LSTM model, but only the segmentation penalty for the transformer model in the remainder of our experiments.

We next benchmark our models against state-of-the-art RST parsers over the *test set*, as presented in Table 2 (original Parseval) and Table 5 (RST-Parseval as additional result). Except Yu et al. (2018), all bottom-up results are from Morey et al. (2017). We present the labelled attachment decision performance for Yu et al. (2018) by running the code of the authors for three runs and taking

[6] `https://stanfordnlp.github.io/CoreNLP`

[7] We use the Huggingface framework for the transformer models.

[8] `https://nlp.stanford.edu/projects/glove`

[9] The result is consistent with the test set (see Appendix B)

Method	S	N	R	F
Bottom Up:				
Feng and Hirst (2014)*†	68.6	55.9	45.8	44.6
Ji and Eisenstein (2014)*†	64.1	54.2	46.8	46.3
Surdeanu et al. (2015)*†	65.3	54.2	45.1	44.2
Joty et al. (2015)*	65.1	55.5	45.1	44.3
Hayashi et al. (2016)*	65.1	54.6	44.7	44.1
Li et al. (2016)*	64.5	54.0	38.1	36.6
Braud et al. (2017)*	62.7	54.5	45.5	45.1
Yu et al. (2018) (static)‡	71.1	59.7	48.4	47.4
Yu et al. (2018) (dynamic)‡	71.4	60.3	49.2	48.1
Top Down:				
Zhang et al. (2020)*	67.2	55.5	45.3	44.3
Our model				
Transformer (static)‡	70.6	59.9	50.6	49.0
Transformer (dynamic)‡	70.2	60.1	50.6	49.2
LSTM (static)‡	72.7	61.7	50.5	49.4
LSTM (dynamic)‡	**73.1**	**62.3**	**51.5**	**50.3**
Our best model without paragraph boundary feature				
LSTM (static)	66.3	56.6	47.1	46.1
LSTM (dynamic)	67.3	57.4	48.5	47.4
Human	78.7	66.8	57.1	55.0

Table 2: Results over the test set calculated using micro-averaged F-1 on labelled attachment decisions (original Parseval). All metrics (S: `Span`, N: `Nuclearity`, R: `Relation`, F: `Full`) are averaged over three runs. "*" denotes reported performance. "†" and "‡" denote that the model uses sentence and paragraph boundary features, respectively. In this evaluation, Kobayashi et al. (2020) does not report the original Parseval result.

the average.[10] We also present the reported scores for the other top-down RST parsers (Zhang et al., 2020; Kobayashi et al., 2020).[11] Human performance in Table 2 and Table 5 is the score of human agreement reported by Joty et al. (2015) ad Morey et al. (2017).

Overall, in Table 2 our top-down models (LSTM and transformer) outperform all bottom-up and top-down baselines across all metrics. As we saw in the feature addition study, the LSTM model outperforms the transformer model, even though the transformer uses pre-trained BERT. We hypothesize that this may be because BERT is trained over shorter texts (paragraphs or sentence pairs), while our documents are considerably longer. Also, due to memory constraints, we break long documents into partial documents (Section 3.2), limiting fine-grained word-to-word attention to only nearby EDUs.

#EDUs	#Docs	#Spans	Type	S	N	R
$(0, 50]$	21	404	Static	81.0	72.0	58.9
			Dynamic	79.3	71.6	**59.1**
$(50, 100]$	9	639	Static	76.8	66.4	56.2
			Dynamic	**79.3**	**69.2**	**59.2**
$(100, 150]$	5	604	Static	69.6	58.4	49.1
			Dynamic	**70.3**	57.4	49.1
$(150, \infty)$	3	661	Static	66.6	53.9	41.0
			Dynamic	66.0	**54.4**	**41.8**

Table 3: Impact of the dynamic oracle over documents of differing length. Scores (micro-averaged F-1 on labelled attachment decisions) are averaged over three runs on the test set.

In Table 2, we also present results for our model without paragraph features, and compare against other models which don't use paragraph features (each marked with "‡").[12] First, we observe that our best model substantially outperforms all models with paragraph boundary features in terms of the `Full` metric. Compared to Zhang et al. (2020), our models (without this feature) achieve an improvement of +0.1, +1.9, +3.2, and +3.1 for `Span`, `Nuclearity`, `Relation`, and `Full` respectively.

5 Analysis

In Table 3 we present the impact of the dynamic oracle over documents of differing length for the LSTM model. Generally, we found that the static model performs better for shorter documents, and the dynamic oracle is more effective for longer documents. For instance, for documents with 50–100 EDUs, the dynamic oracle improves the `Span`, `Nuclearity`, and `Relation` metrics substantially. We also observe that the longer the document, the more difficult the tree prediction is. It is confirmed by the decreasing trends of all metrics for longer documents in Table 3.

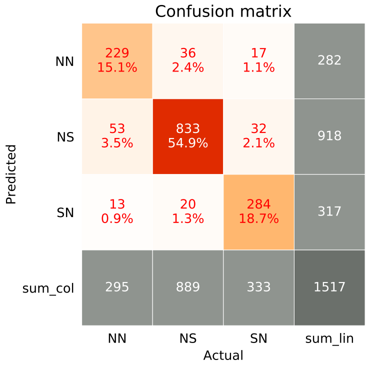
In total, our best model obtains 1,698 out of 2,308 spans of original Parseval trees, and correctly predict 1,517 segmentation points (pairs). We further analyze these pairs by presenting the confusion matrices of nuclearity and relation prediction in Figure 7 and Figure 8, respectively.

First, the model tends to have more errors on `NN` (Nucleus–Nucleus) prediction where 53 span

[10] `https://github.com/yunan4nlp/NNDisParser`.

[11] Neither Zhang et al. (2020) nor Kobayashi et al. (2020) released their code, so we were unable to rerun their models.

[12] Yu et al. (2018) use paragraph boundary features in their original code, but do not report it in the paper.



	NN	NS	SN	sum_lin
NN	229 15.1%	36 2.4%	17 1.1%	282
NS	53 3.5%	833 54.9%	32 2.1%	918
SN	13 0.9%	20 1.3%	284 18.7%	317
sum_col	295	889	333	1517

Figure 7: Confusion matrix of nuclearity prediction over the test set (NS = Nucleus-Satellite, NN = Nucleus-Nucleus, SN = Satellite-Nucleus).

EDU-1	EDU-2	Actual	Pred
senator sasser of tennessee is chairman of the appropriations subcommittee on military construction;	mr. bush 's $ 87 million request for tennessee increased to $ 109 million.	back	elab
a law went in the books in january that let him smoke bacon without breeding pigs.	he chased in	cause	elab
that 's the rule.	that 's the market.	list	elab

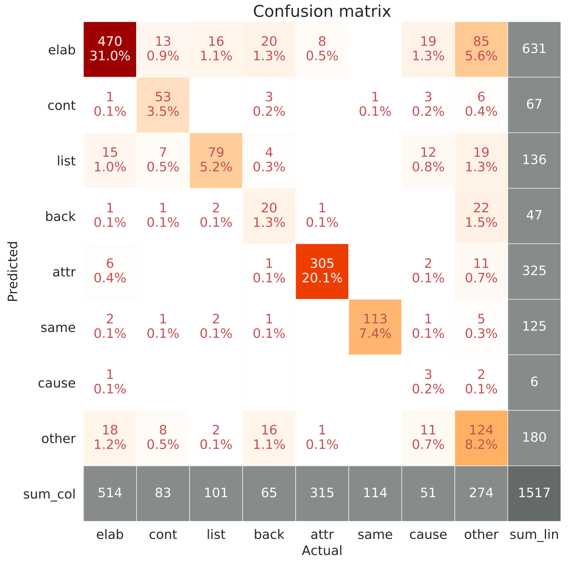
Table 4: Examples of misclassified relations.

pairs (18% of NN) are classified as NS (Nucleus–Satellite). Class imbalance in the training set (NN:NS:SN = 23:61:16) is the main factor that drives the model to favor NS over the other classes.

In Figure 8 we present analysis over top-7 relations and a relation other that represents the rest of 11 classes. Similar to the nuclearity prediction, the relation class distribution is also imbalance where elab accounts for 37% of the examples. Some relations are related to elab (see Table 4 for examples), such as back, cause, and list which we see some false positives. This produces the low precision of elab (74%). Unlike elab, relation attr is also a major class (represents 14% of the training data) but its precision and recall is substantially higher, at 94% and 96% respectively, suggesting it is less ambiguous. For other, its recall is 45%, and most of the errors are classified as elab (31%).

6 Conclusion

We introduce a top-down approach for RST parsing via sequence labelling. Our model is conceptually simpler than previous top-down discourse parsers and can leverage pre-trained language models such as BERT. We additionally propose a dynamic-oracle for our top-down parser, and demonstrate that our best model achieves a new state-of-the-art for RST parsing.



	elab	cont	list	back	attr	same	cause	other	sum_lin
elab	470 31.0%	13 0.9%	16 1.1%	20 1.3%	8 0.5%		19 1.3%	85 5.6%	631
cont	1 0.1%	53 3.5%		3 0.2%		1 0.1%	3 0.2%	6 0.4%	67
list	15 1.0%	7 0.5%	79 5.2%	4 0.3%			12 0.8%	19 1.3%	136
back	1 0.1%	1 0.1%	2 0.1%	20 1.3%	1 0.1%			22 1.5%	47
attr	6 0.4%			1 0.1%	305 20.1%		2 0.1%	11 0.7%	325
same	2 0.1%	1 0.1%	2 0.1%	1 0.1%		113 7.4%	1 0.1%	5 0.3%	125
cause	1 0.1%						3 0.2%	2 0.1%	6
other	18 1.2%	8 0.5%	2 0.1%	16 1.1%	1 0.1%		11 0.7%	124 8.2%	180
sum_col	514	83	101	65	315	114	51	274	1517

Figure 8: Confusion matrix of relation prediction over the test set with top-7 relations (elab = Elaboration, cont = Contrast, list = List, back = Background, same = Same, temp = Temporal, eval = Evaluation, other = Other 11 relations).

Acknowledgments

We are grateful to the anonymous reviewers for their helpful feedback and suggestions. In this research, the first author is supported by the Australia Awards Scholarship (AAS), funded by the Department of Foreign Affairs and Trade (DFAT), Australia. This research was undertaken using the LIEF HPC-GPGPU Facility hosted at The University of Melbourne. This facility was established with the assistance of LIEF Grant LE170100200.

References

Parminder Bhatia, Yangfeng Ji, and Jacob Eisenstein. 2015. Better document-level sentiment analysis from RST discourse parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2212–2218.

Chloé Braud, Maximin Coavoux, and Anders Søgaard. 2017. Cross-lingual RST discourse parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 292–304, Valencia, Spain. Association for Computational Linguistics.

Chloé Braud, Barbara Plank, and Anders Søgaard. 2016. Multi-view and multi-task training of RST discourse parsers. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1903–1913, Osaka, Japan. The COLING 2016 Organizing Committee.

Lynn Carlson, Daniel Marcu, and Mary Ellen Okurowski. 2001. Building a discourse-tagged corpus in the framework of rhetorical structure theory. In *SIGDIAL '01 Proceedings of the Second SIGdial Workshop on Discourse and Dialogue - Volume 16*, pages 1–10.

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 *Proceedings of the 2016 International Conference on Learning Representations*, pages 1–8.

Vanessa Wei Feng and Graeme Hirst. 2014. A linear-time bottom-up discourse parser with constraints and post-editing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 511–521, Baltimore, Maryland. Association for Computational Linguistics.

Richard Frost, Rahmatullah Hafiz, and Paul Callaghan. 2007. Modular and efficient top-down parsing for ambiguous left-recursive grammars. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 109–120, Prague, Czech Republic. Association for Computational Linguistics.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India. The COLING 2012 Organizing Committee.

Katsuhiko Hayashi, Tsutomu Hirao, and Masaaki Nagata. 2016. Empirical comparison of dependency conversions for RST discourse trees. In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 128–136, Los Angeles. Association for Computational Linguistics.

Hugo Hernault, Helmut Prendinger, David A. duVerle, and Mitsuru Ishizuka. 2010. Hilda: A discourse parser using support vector machine classification. *Dialogue and Discourse*, 1(3):1–33.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9:1735–1780.

Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. 2019. What does BERT learn about the structure of language? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3651–3657, Florence, Italy. Association for Computational Linguistics.

Yangfeng Ji and Jacob Eisenstein. 2014. Representation learning for text-level discourse parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13–24, Baltimore, Maryland. Association for Computational Linguistics.

Mark Johnson. 1995. Squibs and discussions: Memoization in top-down parsing. *Computational Linguistics*, 21(3):405–417.

Shafiq Joty, Giuseppe Carenini, Raymond Ng, and Yashar Mehdad. 2013. Combining intra- and multi-sentential rhetorical parsing for document-level discourse analysis. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 486–496, Sofia, Bulgaria. Association for Computational Linguistics.

Shafiq Joty, Giuseppe Carenini, and Raymond T. Ng. 2015. CODRA: A novel discriminative framework for rhetorical analysis. *Computational Linguistics*, 41(3):385–435.

Naoki Kobayashi, Tsutomu Hirao, Hidetaka Kamigaito, Manabu Okumura, and Masaaki Nagata. 2020. Top-down RST parsing utilizing granularity levels in documents. In *AAAI 2020 : The Thirty-Fourth AAAI Conference on Artificial Intelligence*.

Fajri Koto, Jey Han Lau, and Timothy Baldwin. 2019. Improved document modelling with a neural discourse parser. In *Proceedings of the Australasian Language Technology Association Workshop 2019*, Sydney, Australia.

Jiwei Li, Rumeng Li, and Eduard Hovy. 2014. Recursive deep models for discourse parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2061–2069, Doha, Qatar. Association for Computational Linguistics.

Qi Li, Tianshi Li, and Baobao Chang. 2016. Discourse parsing with attention-based hierarchical neural networks. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 362–371, Austin, Texas. Association for Computational Linguistics.

Xiang Lin, Shafiq Joty, Prathyusha Jwalapuram, and M Saiful Bari. 2019. A unified linear-time framework for sentence-level discourse parsing. In *Proceedings of the 57th Annual Meeting of the Asso-*

ciation for Computational Linguistics, pages 4190–4200, Florence, Italy. Association for Computational Linguistics.

Linlin Liu, Xiang Lin, Shafiq Joty, Simeng Han, and Lidong Bing. 2019. Hierarchical pointer net parsing. In *2019 Conference on Empirical Methods in Natural Language Processing*, pages 1007–1017.

Yang Liu and Mirella Lapata. 2019. Text summarization with pretrained encoders. In *2019 Conference on Empirical Methods in Natural Language Processing*.

William C. Mann and Sandra A. Thompson. 1988. Rhetorical structure theory: Toward a functional theory of text organization. *Text Interdisciplinary Journal for the Study of Discourse*, pages 243–281.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, Maryland. Association for Computational Linguistics.

Daniel Marcu. 2000. *The Theory and Practice of Discourse Parsing and Summarization*. MIT Press, Cambridge, USA.

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.

Mathieu Morey, Philippe Muller, and Nicholas Asher. 2017. How much progress have we made on RST discourse parsing? a replication study of recent results on the RST-DT. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1319–1324, Copenhagen, Denmark. Association for Computational Linguistics.

Allen Nie, Erin Bennett, and Noah Goodman. 2019. DisSent: Learning sentence representations from explicit discourse relations. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4497–4510, Florence, Italy. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Building a discourse-tagged corpus in the framework of rhetorical structure theory. In *CoRR, abs/1704.01444, 2017*.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276.

Brian Roark and Mark Johnson. 1999. Efficient probabilistic top-down and left-corner parsing. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 421–428, College Park, Maryland, USA. Association for Computational Linguistics.

Wei Shi and Vera Demberg. 2019. Next sentence prediction helps implicit discourse relation classification within and across domains. In *2019 Conference on Empirical Methods in Natural Language Processing*, pages 5789–5795.

Damien Sileo, Tim Van De Cruys, Camille Pradel, and Philippe Muller. 2019. Mining discourse markers for unsupervised sentence representation learning. 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 3477–3486, Minneapolis, Minnesota. Association for Computational Linguistics.

Mihai Surdeanu, Tom Hicks, and Marco Antonio Valenzuela-Escárcega. 2015. Two practical rhetorical structure theory parsers. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 1–5, Denver, Colorado. Association for Computational Linguistics.

Yizhong Wang, Sujian Li, and Houfeng Wang. 2017. A two-stage parsing method for text-level discourse analysis. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 184–188, Vancouver, Canada. Association for Computational Linguistics.

Nan Yu, Meishan Zhang, and Guohong Fu. 2018. Transition-based neural RST parsing with implicit syntax features. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 559–570.

Nan Yu, Meishan Zhang, and Guohong Fu. 2018. Transition-based neural RST parsing with implicit syntax features. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 559–570, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Longyin Zhang, Yuqing Xing, Fang Kong, Peifeng Li, and Guodong Zhou. 2020. A top-down neural architecture towards text-level parsing of discourse rhetorical structure. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6386–6395, Online. Association for Computational Linguistics.

A Evaluation with RST-Parseval Procedure

Method	S	N	R	F
Bottom-Up				
Feng and Hirst (2014)*†	84.3	69.4	56.9	56.2
Ji and Eisenstein (2014)*†	82.0	68.2	57.8	57.6
Surdeanu et al. (2015)*†	82.6	67.1	55.4	54.9
Joty et al. (2015)*	82.6	68.3	55.8	54.4
Hayashi et al. (2016)*	82.6	66.6	54.6	54.3
Li et al. (2016)*	82.2	66.5	51.4	50.6
Braud et al. (2017)*	81.3	68.1	56.3	56.0
Yu et al. (2018) (1 run)*‡	85.5	73.1	60.2	59.9
Yu et al. (2018) (static)‡	85.8	72.6	59.5	59.0
Yu et al. (2018) (dynamic)‡	85.6	72.9	59.8	59.3
Top-Down				
Kobayashi et al. (2020)*†‡	**87.0**	**74.6**	60.0	-
Our model				
Transformer (static)‡	85.2	72.0	60.3	59.6
Transformer (dynamic)‡	85.5	72.3	60.5	59.9
LSTM (static)‡	86.4	73.4	60.8	60.3
LSTM (dynamic)‡	86.6	73.7	**61.5**	**60.9**
Our best model without boundary feature				
LSTM (static)	83.2	70.4	58.4	57.9
LSTM (dynamic)	83.6	70.4	58.8	58.2
Human	88.3	77.3	65.4	64.7

Table 5: Results over the test set calculated using micro-averaged F-1 on RST-Parseval. All metrics (S: `Span`, N: `Nuclearity`, R: `Relation`, F: `Full`) are averaged over three runs. "*" denotes reported performance. "†" and "‡" denote that the model uses sentence and paragraph boundary features, respectively. In this evaluation, Zhang et al. (2020) does not report the RST-Parseval result. Also, both Zhang et al. (2020); Kobayashi et al. (2020) do not release their code. First, We can see that our model achieves the best results in terms of `Relation` and `Full`. Compared to other models without paragraph boundary features, our proposed model also performs best on the `Full` metric by a comfortable margin.

B Feature Addition on the Test Set

Variants	LSTM	Transformer
Vanilla	46.5±0.4	48.0±0.4
+Syntax	48.2±0.7	49.3±1.0
+Penalty	46.8±0.8	**49.0±0.2**
+Syntax+Penalty	**49.4±0.4**	48.7±0.6

Table 6: Feature addition over the test set to find the best configuration for our models. Presented results are the mean and standard deviation of the `Full` metric (micro-averaged F-score on labelled attachment decisions) over three runs.

C Model Configuration for Training

Configuration	Value
LSTM1, LSTM2, LSTM3	200, **256**
LSTM4	100, **128**, 200
Word embedding	**200**
POS embedding	**200**
EDU type embedding	**100**
Syntax Feature	**1200**
λ_1	**1.0**, 1.2
λ_2	0.6, 0.8, **1.0**, 1.2
β (Loss penalty)	0, 0.2, **0.35**, 0.4, 0.6, 0.8, 1.0
α (Dynamic oracle)	0, 0.25, 0.5, **0.65**, 0.75, 1.0
Batch size	2, **4**
Gradient accumulation	**2**, 4
Learning rate	**0.001**
Dropout probability	**0.5**
Infrastructure	1 GPU V100 (16 GB)
Metrics to evaluate	`Full`

Table 7: Parameter trials of our LSTM model. Bold indicates the best value after tuning over the development set.

Configuration	Value
BERT encoder	BERT-Base
Transformer2	L=2, H=8, FF=2048
EDU type embedding	**100**
λ_1	0.5, **1.0**, 1.5
λ_2	0.6, 0.8, **1.0**, 1.5
β (Loss penalty)	0, 0.2, **0.35**, 0.4, 0.6, 0.8, 1.0
α (Dynamic oracle)	0, 0.25, 0.5, **0.65**, 0.75, 1.0
Batch size	1, **2**
Gradient accumulation	2, 3, **4**, 8
Learning rate	**5e-5**
Dropout probability	0.1, 0.3, 0.4, **0.5**
Infrastructure	4 GPU V100 (16 GB)
Metrics to evaluate	`Full`

Table 8: Parameter trials of our transformer model. Bold indicates the best value after tuning over the development set.

Method	1 Epoch	Converge at Epoch
Transition-based	5 mins	60–70
LSTM (Ours)	1 min 21 secs	60–70
Transformer (Ours)	1 min	130-150

Table 9: Running time of the static models during the training. The transition-based model is Yu et al. (2018)