A Coarse-to-Fine Labeling Framework for Joint Word Segmentation, POS Tagging, and Constituent Parsing

Yang Hou¹, Houquan Zhou¹, Zhenghua Li¹^{*}, Yu Zhang¹, Min Zhang¹,

Zhefeng Wang², Baoxing Huai², Nicholas Jing Yuan²

¹School of Computer Science and Technology,

Soochow University, China ² Huawei Cloud, China

¹{yhou1,hqzhou}@stu.suda.edu.cn

1{zhli13,minzhang}@suda.edu.cn ¹yzhang.cs@outlook.com ²{wangzhefeng, huaibaoxing, nicholas.yuan}@huawei.com

Abstract

The most straightforward approach to joint word segmentation (WS), part-of-speech (POS) tagging, and constituent parsing (PAR) is converting a word-level tree into a char-level tree, which, however, leads to two severe challenges. First, a larger label set (e.g., ≥ 600) and longer inputs both increase computational cost. Second, it is difficult to rule out illegal trees containing conflicting production rules, which is important for reliable model evaluation. If a POS tag (like VV) is above a phrase tag (like VP) in the output tree, it becomes quite complex to decide word boundaries. To deal with both challenges, this work proposes a two-stage coarse-to-fine labeling framework for joint WS-POS-PAR. In the coarse labeling stage, the joint model outputs a bracketed tree, in which each node corresponds to one of four labels (i.e., phrase, The tree is subphrase, word, subword). guaranteed to be legal via constrained CKY decoding. In the fine labeling stage, the model expands each coarse label into a final label (such as VP, VP*, VV, VV*). Experiments on Chinese Penn Treebank 5.1 and 7.0 show that our joint model consistently outperforms the pipeline approach on both settings of without and with BERT, and achieves new state-of-the-art performance.

1 Introduction

As shown in Figure 1(a), the results of word segmentation (WS), part-of-speech (POS) tagging, and constituent parsing (PAR) can be organized in a unified hierarchical tree, where leaf nodes include words and their POS tags, and non-terminal nodes correspond to phrases (or constituents) with their syntactic label. Before the deep learning (DL) era, there had been intensive research interest in jointly modeling of the three tasks, i.e., WS-POS-PAR (Luo, 2003; Qian and Liu, 2012; Zhang et al.,



Figure 1: Example WS-POS-PAR trees. The English translation is "The chocolate is very delicious."

2013; Wang et al., 2013). The motivations are three-fold: 1) alleviating error propagation, 2) promoting knowledge sharing and interaction during inference, 3) simplifying system architecture by training a single model.

In contrast, there have been few work on joint WS-POS-PAR in the DL era, with the exception of Zheng et al. (2015). Similar to this work, they treated the joint problem as a char-level parsing task, which is obviously the most straightforward way for Chinese, as illustrated in Figure 1(b). However, they showed that the joint model fails to outperform the pipeline approach.

In the past few years, constituent parsing has achieved significant progress. Compared with pre-DL parsers based on discrete features (Collins, 1997; Charniak and Johnson, 2005; Petrov and Klein, 2007), the major characteristic of graphbased parsers (Stern et al., 2017; Gaddy et al., 2018; Kitaev and Klein, 2018; Zhang et al., 2020) is that the score of a constituent tree is decomposed into scores of labeled spans (i.e., (2, 3, VP) or $VP_{2,3}$), without any consideration on production rules (i.e., $VP_{2,3} \rightarrow ADVP_{2,2} VP_{3,3}$).

For char-level graph-based WS-POS-PAR, there exist two severe challenges. (1) The first challenge is high model complexity. On the one hand, a char sequence is almost twice longer than its word sequence. On the other hand, the size of the la-

^{*}Corresponding author

bel set is very large (e.g., ≥ 600) after transforming char-level trees into Chomsky normal form (CNF), as illustrated in Figure 2(a). Both factors greatly increase time and memory complexity. (2) The second challenge is rule conflict. Since neural graph-based parsers do not consider production rules, there is no way to guarantee the output tree is legal. Conflicting rules, such as "VP \rightarrow VA* VA*" (two subwords into a phrase) and "VA \rightarrow ADVP+AD VP+VA" (POS tag above phrases), make it very difficult to decide word boundaries and thus unfeasible to evaluate model outputs.

In this work, we propose a two-stage coarse-tofine labeling framework for joint WS-POS-PAR, which can deal with both challenges. In the coarse labeling stage, the joint model outputs a bracketed tree with coarse labels (i.e., phrase, subphrase, word, subword). The constrained CKY algorithm is used to guarantee that the predicted tree contains no illegal production rules. In the fine labeling stage, the model expands each coarse label into a final fine-grained label (such as VP, VP*, VV, VV*). Experiments on three Chinese Treebank (CTB) benchmark datasets show the joint framework is superior to the pipeline framework on both settings of without and with BERT (Devlin et al., 2019), and achieves new state-of-the-art performance. We will release our code at https://github.com/ ironsword666/JointParser.

2 Joint WS-POS-PAR as Char-Level Tree Parsing

2.1 From Word-level Tree to Char-level Tree

As illustrated in Figure 1, a word-level constituent tree with POS tags in 1(a) is converted into a charlevel tree in 1(b). In a word-level tree, a leaf node corresponds to a word with its POS tag. In contrast, in a char-level tree, a leaf node is always a character, and POS tags become non-terminal nodes. We use flat structures for multi-char words, such as "NN_{1,3} \rightarrow 巧 克 力".

Chomsky normal form (CNF). To facilitate graph-based model training and inference, we use the NLTK tool¹ to transform original char-level trees into Chomsky normal form (CNF). Following the recent work of Zhang et al. (2020) on constituent parsing, we adopt left binarization.



Figure 2: Char-level tree in CNF and two typical kinds of subtrees containing illegal production rules.

Figure 2(a) gives an example of char-level tree in CNF. The non-terminal labels can be divided into four categories. (1) NN^{*}_{1,1} means that "巧" is a subword. (2) NP+NN_{1,3} means that "巧克力" corresponds to a complete word with NN as its POS tag, and the word is also a constituent labeled as NP. (3) IP^{*}_{1,6} spans a subphrase. (4) IP_{1,7} means that the span $x_1...x_7$ corresponds to a complete phrase with IP as its label. Non-terminal labels corresponding to subwords and subphrases, such as NN* and IP*, are created during breaking nary (n > 2) production into binary ones. Complex labels such as NP+NN are created during collapsing unary chains such as "NP \rightarrow NN \rightarrow 巧克力".

2.2 Two Challenges

Directly performing char-level constituent parsing confronts two severe challenges, i.e., high complexity, and illegal trees containing word-vs-phrase conflicts.

(1) The high complexity challenge. This challenge concerns two aspects. (i) *large number of non-terminal labels*. A mass of new labels is introduced by CNF transformation, most of which are produced from the collapsing process of consecutive unary chains. Taking CTB7 as an example, the number of labels grows from 63 to 638 after CNF transformation for joint WS-POS-PAR. In contrast, for word-level constituent parsing, the number of labels grows from 28 to 265. (ii) *longer inputs*. Moreover, since words are broken into characters in char-level parsing, the same sentence is almost

¹https://www.nltk.org

twice the length of that in word-level parsing, posing further challenge on model efficiency.

In summary, longer inputs and a larger label set both increase computational cost from the aspects of time and GPU memory size.

This work handles the high complexity issue in two ways. First, we adopt the two-stage parsing framework of Zhang et al. (2020). The first stage produces a bracketed tree without labeling nonterminal nodes; the second stage independently predicts the label for each node. In this way, with the CKY decoding algorithm, the total time complexity decreases to $O(n^3 + n|\mathcal{L}|)$, versus $O(n^3 + n^2|\mathcal{L}|)$ for one-stage parsing. In terms of space complexity, two-stage parsing needs to compute only $O(n^2 + n|\mathcal{L}|)$ scores, versus $O(n^2|\mathcal{L}|)$ for onestage parsing.

Second, we prune low-frequency labels by replacing them with most-similar high-frequency labels in the training data. On CTB7, less-than-10 pruning decreases the label number from 638 to 293.

(2) The word-vs-phrase conflict challenge. In the DL era, mainstream graph-based constituent parsers (Stern et al., 2017; Kitaev and Klein, 2018; Zhang et al., 2020) determine the label for each nonterminal node in a local manner. As a result, there is no guarantee that productions in 1-best parse trees are all legal. For example, " $IP \rightarrow VP^* VP$ " is an illegal production, where " VP^* " should probably be changed to " IP^* ". In the case of word-level constituent parsing, a popular way to handle such conflicts is ignoring such intermediate nodes during evaluation, since they are created to meet CNF.

However, in the case of char-level parsing, such conflicts are very difficult to handle. Figure 2 presents two typical kinds of conflicts. In Figure 2(b), the top POS tag "VA" tells that "很美味" is a word; "ADVP+AD" tells that "很" is a word and a phrase; "VP+VA" tells that "美味" is word and a phrase. However, there is only one way to segment a sentence into a word sequence.

In Figure 2(c), the marked "VP" tells that "美味" is a phrase; the two "VA*" tell that "美" and "味" are subwords. Obviously there is a gap in between.

In summary, we can see that word-vs-phrase conflicts pose great challenge on reliable evaluation of joint WS-POS-PAR results. One possible way to handle this is to post-process illegal trees into legal ones based on heuristic rules, which, however, is a non-trivial task itself.



Figure 3: The architecture of our proposed joint frame-work.

As an extension to the two-stage parsing framework of Zhang et al. (2020), this work proposes a coarse-to-fine labeling framework to elegantly deal with the word-vs-phrase conflict issue. In the first stage, the joint model not only produces a bracketed tree, but also assigns a coarse label to each node. We ensure the coarsely-labeled bracketed tree is legal via constrained CKY decoding. In the fine labeling stage, the model expands each coarse label into a final fine-grained label via local classification.

3 Proposed Two-stage Coarse-to-fine Labeling Framework

As shown in Figure 3, our proposed framework consists of two independent decoders and a shared encoder. During training, two parts of losses are directly added, i.e., the weights of two tasks are both one², and the whole model is trained via standard multi-task learning (MTL).

Given a sentence consisting of n characters $x = x_1, \ldots, x_n$, we use (i, j, c) to denote a candidate span $x_i \ldots x_j$ with a coarse label c, and (i, j, f) to denote one with a fine-grained label f. Section 3.3 illustrates how to obtain their corresponding scores, i.e., s(i, j, c) and s(i, j, f).

3.1 Stage 1: Produce Tree w/ Coarse Labels

While Zhang et al. (2020) produce a completely unlabeled bracketed tree in the first stage, we obtain a bracketed tree with coarse-grained labels, as

²We try to adjust the weights of two tasks, but the performance gains no significant improvement.

Coarse labels	Examples	#Labels	#Pruned
P (Phrase)	VP VP+NP	213	82
P* (Subphrase)	$VP^* NP^*$	27	27
W (Word)	AD QP+CD	368	154
W* (Subword)	$NR^* CD^*$	30	30

Table 1: Four categories of coarse-grained labels. "#Labels" means the number of corresponding finegrained labels after CNF transformation on CTB7, and "#Pruned" refers to label pruning with $< \alpha = 10$.

illustrated by the left tree on top of Figure 3.

As illustrated in Table 1, all fine-grained nonterminal labels are categorized into four types of coarse-grained labels.

We denote a bracketed tree with coarse labels as $y = \{(i, j, c)\}$. As a mainstream practice adopted by most neural graph-based parsers, the tree score is decomposed into scores of spans.

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,j,c) \in \boldsymbol{y}} s(i,j,c)$$
(1)

Constrained CKY decoding. During evaluation, given an input sentence, the model aims to produce a highest-scoring tree.

$$\hat{\boldsymbol{y}} = \arg \max_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} s(\boldsymbol{x}, \boldsymbol{y})$$
 (2)

where $\mathcal{Y}(\boldsymbol{x})$ denotes the set of all possible legal trees for \boldsymbol{x} .

Using the standard CKY algorithm, we can efficiently find the highest-scoring tree. However, the legality of the tree, i.e., free from word-vs-phrase conflicts, cannot be guaranteed, since the scores of spans are mutually independent and the model does not consider production rules at all.

In this work, we propose an efficient constrained CKY algorithm to elegantly handle this issue, as illustrated in Algorithm 1. The basic idea is introducing two sets of production scores into decoding, i.e., binary productions $s(A \to BC)$, and unary leaf productions $s(A \to x_i)$, where A/B/C are non-terminal labels in coarse label set \mathcal{N} and x_i is a character. For illegal productions such as " $P \to W^* W^*$ ", we set their scores into $-\infty$. Scores of all legal productions are set to 0. By ruling out illegal productions, we can exclude all illegal trees eventually.

Algorithm 1 Constrained CKY Algorithm.

- input: labeled span scores s(i, j, A); production scores s(A → a) and s(A → BC)
- 2: **initialize** all $\pi(\cdots)$ **to** 0
- 3: **for** i := 1 **to** n **do**
- 4: for $A \in \mathcal{N}$ do
- 5: $\pi(i, i, A) = s(i, i, A) + s(A \to x_i)$
- 6: for width := 2 to n do
- 7: **for** i := 1 **to** n width + 1 **do**
- 8: j := i + width 1

9: for
$$A \in \mathcal{N}$$
 do

10:

$$\pi(i, j, A) = \max_{\substack{i \le k < j, \\ \{B, C\} \in \mathcal{N}^2}} \frac{\pi(i, k, B) + \pi(k+1, j, C)}{+s(i, j, A) + s(A \to BC)}$$

11: return \hat{y}

TreeCRF training loss. We follow (Zhang et al., 2020) and adopt a global TreeCRF loss. As a probabilistic model, TreeCRF defines the conditional probability of a bracketed tree as:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{e^{s(\boldsymbol{x},\boldsymbol{y})}}{\boldsymbol{Z}(\boldsymbol{x}) \equiv \sum_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} e^{s(\boldsymbol{x},\boldsymbol{y}')}} \qquad (3)$$

where Z(x) is known as normalization term and can be efficiently computed via the inside algorithm. Analogously to CKY decoding, we employ **a constrained inside algorithm** to exclude illegal trees from $\mathcal{Y}(x)$, which leads to improved performance. Please see Subsection 5.2 for more details.

Given a sentence x and its gold-standard coarsely-labeled tree y^* , the TreeCRF training loss is:

$$loss^{\text{coarse}}(\boldsymbol{x}, \boldsymbol{y}^*) = -\log p(\boldsymbol{y}^* | \boldsymbol{x}) \qquad (4)$$

3.2 Stage 2: Expand into Fine-grained Labels

In the second stage, we predict a fine-grained label for each non-terminal node in the 1-best tree \hat{y} obtained in the first stage.

$$\hat{f} = \arg \max_{f \in \mathcal{F}(\hat{c})} s(i, j, f)$$
(5)

where \hat{c} is the predicted coarse-grained label for span (i, j), and $\mathcal{F}(\hat{c})$ is the corresponding finegrained label set for \hat{c} .

Cross-entropy training loss. As a typical multiclass classification task, we adopt cross-entropy

loss independently in the second stage.

$$loss^{\text{fine}}(\boldsymbol{x}, \boldsymbol{z}^*) = \sum_{(i,j,f) \in \boldsymbol{z}^*} -\log \frac{e^{s(i,j,f)}}{\sum_{f'} e^{s(i,j,f')}}$$
(6)

where z^* is the gold-standard tree with fine-grained labels.

3.3 Model Details for Span Scoring

This subsection introduces model details for obtaining span scores, i.e., s(i, j, c) and s(i, j, f), corresponding to the lower part of Figure 3, most of which are borrowed from Zhang et al. (2020).

Input vectors. The input vector for the *i*-th position is composed of a char embedding and a bichar embedding.

$$\mathbf{e}_i = \mathbf{emb}(x_i) \oplus \mathbf{emb}(x_i x_{i+1}) \tag{7}$$

Encoder. We employ three BiLSTM (Hochreiter and Schmidhuber, 1997) layers over the input vectors for context encoding. The final representation of *i*-th position is:

$$\mathbf{h}_i = \mathbf{f}_i \oplus \mathbf{b}_{i+1} \tag{8}$$

where f_i and b_i are the output vectors of the toplayer forward and backward LSTMs for the *i*-th position.

Span boundary representation. Two MLPs are used to obtain two representation vectors for each position.

$$\mathbf{r}_{i}^{l};\mathbf{r}_{i}^{r}=\mathbf{MLP}^{l}\left(\mathbf{h}_{i}\right);\mathbf{MLP}^{r}\left(\mathbf{h}_{i}\right)$$
 (9)

where l/r represent the situation that the *i*-th position is the left/right boundary of some spans.

Biaffine scoring. We compute scores of coarsely labeled spans (i, j, c) using biaffine operations.

$$s(i,j,c) = \begin{bmatrix} \mathbf{r}_i^l \\ 1 \end{bmatrix}^{\mathrm{T}} \mathbf{W}_c^{\mathrm{coarse}} \begin{bmatrix} \mathbf{r}_i^r \\ 1 \end{bmatrix}$$
(10)

where $\mathbf{W}_c^{\text{coarse}} \in \mathbb{R}^{501 \times 501}$ is the biaffine parameter matrix for the label c.

Analogously, for scoring fine-grained labeled spans (i, j, f), two extra MLPs and an extra set of Biaffines are used. Since the number of fine-grained labels are very large, we use a smaller biaffine matrix dimension $\mathbf{W}_{f}^{\text{fine}} \in \mathbb{R}^{101 \times 101}$ to reduce computational cost.

3.4 Label Pruning

As mentioned above, CNF transformation introduces a large number of labels, largely due to collapsing of unary rules, such as "VP+NP". Taking CTB7 as an example, the number of fine-grained labels in CTB7 increases from 63 to 638, among which 355 labels occur less than 10 times in the training data and only 172 labels simultaneously appear in all train/dev/test datasets.

To boost computational efficiency, we propose to reduce the number of fine-grained labels by projecting a low-frequency label into a most similar one. For example, both "CP+VV" and "NP+CP+IP+VP+VV" can be used to replace "NP+CP+CP+IP+VP+VV", and the latter is chosen to recover more constituents.

Table 1 and 2 show the results after pruning low-frequency labels ($< \alpha = 10$). Effect of label pruning on model performance is investigated in Figure 4.

4 The Pipeline Framework

We adopt the typical cascaded pipeline framework as our baseline. In the training phase, three separate models (WS/POS/PAR) are separately trained. In the evaluation phase, the first step is word segmentation; then the word sequence is fed into the POS tagger; finally the word sequence is fed into the constituent parser.

Constituent parsing. We directly adopt the competitive two-stage CRF parser of Zhang et al. (2020), which is also backbone of our joint WS-POS-PAR model. As a word-level parser, its input is composed of two parts: 1) word embedding and 2) CharLSTM word representation (Lample et al., 2016).

$$\mathbf{e}_{i}^{(w)} = \mathbf{emb}(w_{i}) \oplus \mathbf{CharLSTM}(w_{i})$$
 (11)

Other components, such as encoding and span scoring, are the same with those in Section 3.3.

POS tagging. The inputs and encoder of the POS tagging model are the same with the above word-level constituent parser. We then feed h_i into a MLP layer to directly calculate the scores of different POS tags. We directly use the local word-level cross-entropy loss in the training phase, and greed-ily select the highest-scoring tag for each word during evaluation. Our preliminary experiments show that applying a global CRF layer does not lead to better performance.

	#Train	#Dev	#Test	d Orig.	#Labe CNF	ls CNF*
CTB5	18,104	352	348	57	323	183
CTB5-big	16,091	803	1,910	57	313	179
CTB5 CTB5-big CTB7	46,572	2,079	2,796	63	638	293

Table 2: Numbers of sentences and constituent labels. "Orig." indicates the number of labels before CNF. "CNF*" represents the number of remained labels after pruning labels with frequency $< \alpha = 10$.

Word Segmentation. The inputs and encoder of the WS model are the same with our joint WS-POS-PAR model. Similar to POS tagging, we feed h_i into a MLP layer to directly calculate the scores of "BMES" tags, and use local cross-entropy training loss. During evaluation, we apply constrained Viterbi decoding to avoid illegal tag transitions like "B \rightarrow S".

5 Experiments

5.1 Settings

Data. We conduct experiments on Chinese Penn Treebank 5.1 (CTB5) and 7 (CTB7). For CTB5, we follow previous works (Zhang and Clark, 2009; Liu and Zhang, 2017) to split train/dev/test datasets. Because both dev and test datasets of CTB5 contain only about 350 sentences, which is not enough to conduct robust investigations, we adopt another partition of Chinese Penn Treebank 5.1 (CTB5big), which is proposed by Duan et al. (2007). For CTB7, we follow the data split suggested in official guidelines. Table 2 shows the data statistics.

Evaluation metrics. Before the evaluation, we convert predicted character-level CNF trees into word-level *n*-ary trees. Since our joint framework is based on characters, we follow Zhang et al. (2013) to redefine the span by the index of its beginning and ending characters. We adopt the standard constituent-level labeled precision, recall, F-score (Par P/R/F) as the evaluation metrics for constituent parsing, where POS tags are discarded. Metrics of labeled precision, recall, and F-score (Tag P/R/F) of POS tags are used to evaluate the POS tagging task. For word segmentation, the unlabeled precision, recall, F-score (Seg P/R/F) of POS tagging are served as the evaluation metrics.

Parameter settings. We adopt most of hyperparameter settings from Zhang et al. (2020) directly,



Figure 4: Effect of label pruning threshold (< α) on CTB7-dev.

with the exception that we increase the decay steps of learning rate from 5000 to 7500/6500/15000 (CTB5/CTB5-big/CTB7) in favour of more training steps. In our joint framework, the dimensions of character and bichar embedding are 100, which are identical to the dimensions of word embedding and output vector of CharLSTM in the pipeline framework. All embeddings are randomly initialized. For experiments with BERT, we adopt "bertbase-chinese" to get contextual char/word representations. Both pipeline and joint frameworks use 3 BiLSTM layers with the hidden size of 400 as their encoder. The dimensions of coarse-grained and fine-grained label MLPs are 500 and 100 respectively. We use the Adam optimizer (Kingma and Ba, 2014) for all models. All dropout ratios are 0.33. The mini-batch size is 5000 words (or characters). We stop the training process when the peak performance on dev data does not increase in 100 consecutive epochs.

5.2 Results on the Dev Datasets

Results of the pipeline framework. The upper part of Table 3 shows the results of pipeline framework. We conduct the study on dev datasets to understand the influence of automatic word segmentation. The "Pipeline" row presents the results of using automatic word segmentation as the input of the tagger and parser. Then, in the "w/ gold-standard WS" row, we feed the gold-standard word segmentation into the tagger and the parser. Although the performance of automatic word seg-

	CTB5			C	CTB5-big			CTB7		
	Seg	Tag	Par	Seg	Tag	Par	Seg	Tag	Par	
Pipeline w/ gold-standard WS		95.17 97.35	• / · · = =		92.23 95.75		96.42 -	92.88 96.12		
Joint w/o constrained inside		95.97 95.89			93.22 93.07	84.51 84.27	96.99 96.97	93.81 93.77	85.89 85.66	

Table 3: Results (F-score) of pipeline and joint frameworks on dev datasets.

mentation is fairly high, the tagger and parser using automatic word segmentation are still distinctly inferior to those using gold-standard segmentation. It proves that the error propagation issue has a strong impact on the performance of downstream tasks.

Results of the joint framework. First, to study the impact of constrained inside algorithm, we retain illegal trees in $\mathcal{Y}(x)$. The "Joint" row uses the constrained inside and CKY algorithms. In the "w/o constrained inside" row, we try to use a standard inside algorithm in the training phase but still perform constrained CKY decoding. It is interesting that using the constrained inside algorithm achieves a consistent improvement in POS tagging and constituent parsing. One possible reason is that the constrained inside algorithm produces a more precise normalization term Z(x) while the standard inside algorithm creates a mismatch between training and decoding.

Second, we explore the impact of threshold value α used to prune low-frequency labels for joint framework. As shown in Figure 4, we select four different threshold values (i.e., 1, 10, 100, 1000) and conduct experiments on CTB7 dev dataset. We can clearly see that (Figure 4(a)) the number of labels is halved from 638 to 293 when α is set to 10. A fairly large value 1000 should be chosen to further prune labels to a quarter of the original. In terms of model performance, different α values have very slight impact, except constituent parsing, in which $\alpha = 10$ outperforms others and higher thresholds lead to worse results.

5.3 Efficiency Comparison

Table 4 compares two frameworks from the aspects of speed and parameter size. Our models are all run on a Nvidia Tesla V100 GPU. The comparison is made on CTB7 test dataset. First, in the upper part of the table, we list the results of the three tasks (WS/POS/PAR). As illustrated in the table, WS is

	#Speed (Sents/sec)	#Params (M)
WS	353	22.8
POS	1785	12.8
PAR	444	16.7
Pipeline	217	51.3
Joint	208	27.7

Table 4: Speed and parameter size comparisons on CTB7 test. For "#Speed", we divide total sentences by total time used. "#Params" is counted in millions with Pytorch function "numel()".

the bottleneck of the pipeline framework since it is in char-level. Second, we calculate the speed of the pipeline framework and compare the two frameworks. Although using a char-level parser, the joint framework can parse 208 sentences per second, which is only about 4% slower than the pipeline counterpart. What's more, our joint framework only has half parameter size compared with pipeline framework (27.7M vs 51.3M).

5.4 Final Results on the Test Datasets

Comparisons with the pipeline framework. In this part, we compare pipeline framework and joint framework on the settings of without and with BERT. The joint framework uses the constrained inside and CKY algorithms, and threshold value α is set to 10. Without using BERT representations, our joint framework outperforms the pipeline framework by a large margin. Considering that the pipeline framework can make use of both character and word information, it is not easy for the joint framework which only has access to the former one. After incorporating BERT representations into two frameworks, our joint framework still achieves better performance except the word segmentation task on CTB5. We owe it to the overfitting phenomenon of complicated model since CTB5 is relatively small.

	CTB5		CTB5-big			CTB7			
	Seg	Tag	Par	Seg	Tag	Par	Seg	Tag	Par
Pipeline Joint		95.17 95.64 [†]			91.62 92.72 [‡]	83.15 84.28 [‡]		92.50 93.30 [‡]	
Pipeline w/ BERT Joint w/ BERT			91.39 91.51	97.36 97.39			97.27 97.49 ‡	94.70 95.11 ‡	

Table 5: Results (F-score) of pipeline and joint frameworks on different test datasets. " \dagger " marks statistical significance with p < 0.005 and " \ddagger " marks the significance with p < 0.0005 (Noreen, 1989)

	Seg	Tag	Par
Joint frameworks			
Qian and Liu (2012)	97.96	93.81	82.85
Zhang et al. (2013)	97.84	94.80	84.43
Wang et al. (2013)	97.86	94.40	83.42
Zheng et al. (2015)	_	_	84.22
Pipeline	98.05	95.17	87.47
Joint	98.41	95.64	87.57

Table 6: Comparisons with previous works on CTB5 test.

Comparisons with previous works. Finally, we list results of all recent related works of the joint framework on CTB5. Please note that our pipeline framework is already remarkably superior to the previous state-of-the-art joint framework proposed by Zhang et al. (2013). Compared with it, our joint framework achieves an absolute improvement of 0.57, 0.84, 3.14 on word segmentation, POS tagging, and constituent parsing respectively on CTB5 test.

6 Related Work

Coarse-to-fine Parsing. Directly performing dynamic programming parsing with fine-grained grammar is computationally expensive from the aspects of time and memory size. Coarse-to-fine parsers introduce complexity gradually by coarseto-fine utilizing a sequence of grammars. Charniak and Johnson (2005) succeed in reranking n-best parses with a two-stage method. They predict nbest parses with a coarse-grained grammar in the first stage, then, the best parse is selected from the n-best parses with the finer-grained second-stage grammar which makes use of more important contextual information. Charniak et al. (2006) extend the basic two-stage coarse-to-fine parsing to multistage parsing by constructing multiple levels of grammars. They cluster constituent labels of the raw treebank into coarser categories. For example, in the most coarse grammar, there only exists one label "P" which corresponds to the phases. The proposed clustering idea in their work is similar to that used in our work. The work of Petrov and Klein (2007) also builds a multi-stage parser which constructs a sequence of increasingly refined grammars in a automatic fashion. However, the final grammar is finer than the raw treebank grammar.

Joint modeling in the pre-DL era. In this part, we try to briefly discuss works on joint modeling of word segmentation, POS tagging, and parsing, which were conducted in the pre-DL era, and most of which are transition-based (shift-reduce) systems instead of graph-based.

Joint POS-ConPAR. Wang and Xue (2014) integrate POS tagging and constituent parsing based on a transition-based parsing model. They modify the action to assign a POS tag when the word is shifted into the stack, making POS tagging as a part of parsing naturally.

Joint POS-DepPAR. Li et al. (2011) extend graph-based dependency parsing to handle POS tagging simultaneously based on a dynamic programming algorithm for joint decoding. Hatori et al. (2011) combine POS tagging and dependency parsing into a shift-reduce parsing system.

Joint WS-POS-ConPAR. Qian and Liu (2012) design a graph-based joint decoding algorithm to aggregate the outputs of three independently trained models for the three tasks. Zhang et al. (2013) integrate word segmentation and POS tagging information into parse trees, and then use a transition-based parser to perform joint learning and decoding. They manually annotated the intra-structures of words on the CTB5 dataset, which is proven to be very beneficial.

Joint WS-POS-DepPAR. Hatori et al. (2012) propose a transition-based parser that uses extra actions for word segmentation and POS tagging. Zhang et al. (2014) apply their previous work on intra-word structures (Zhang et al., 2013) to dependency parsing. The idea is to compose characterlevel dependency trees by combing inter- and intraword dependencies.

Joint modeling in the DL era. Most closely related to our work, Zheng et al. (2015) also focus on joint WS-POS-ConPAR. They adopt CNN with pooling layers to encode the input character sequence. They show that utilizing the intra-word structures annotated by Zhang et al. (2013) can bring considerable gains to parsing performance. Compared with their work, we simply use left binarization to decide the intra-word structures, and our model achieves much higher performance by adopting the state-of-the-art BiLSTM-based parsing model. Our main contribution is proposing an elegant way to handle word-vs-phrase conflicts, which unfortunately are not mentioned in their work.

Kurita et al. (2017) for the first time apply neural networks to jont WS-POS-DepPAR. They enhance the transition-based parser with char/word embeddings and BiLSTM which alleviate the efforts of feature engineering. Li et al. (2018) annotate a character-level dependency treebank for joint WS-POS-DepPAR. They use a neural characterlevel transition-based parsing model to reduce computational cost. Yan et al. (2020) focus on joint WS-DepPAR. They adopt the character-level graphbased parsing approach. For intra-word dependencies, a character is designed to modify its subsequent character with the label of "app". Wu and Zhang (2021) split joint WS-POS-DepPAR into joint WS-POS and joint WS-DepPAR by using a shared character-level encoder and two independent decoders. They adopt outputs from joint WS-POS as final word segmentation results.

7 Conclusions

In this work, we propose a two-stage coarse-to-fine labeling framework of joint WS-POS-PAR, which is shown to be able to handle both challenges for char-level parsing, i.e., high model complexity, and word-vs-phrase label conflicts. In particular, we believe that it is a novel and elegant way to remove illegal trees using constrained CKY decoding in the coarse-labeling stage. It is also very interesting that under a TreeCRF loss, performance is consistently improved by ruling out illegal trees in computing the normalization factor via the constrained inside algorithm. Experiments and analysis on three Chinese benchmark datasets (CTB5 and CTB7) show that 1) the joint framework is clearly superior to the pipeline framework, and achieves new state-of-theart performance; 2) our joint framework has very similar parsing speed compared with the pipeline counterpart, but only requires a half number of parameters.

8 Acknowledgements

We would like to thank the anonymous reviewers for their helpful comments. This work was supported by National Natural Science Foundation of China (Grant No. 61876116 and 62176173), and a Project Funded by the Priority Academic Program Development (PAPD) of Jiangsu Higher Education Institutions.

References

- Eugene Charniak and Mark Johnson. 2005. Coarseto-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of ACL*, pages 173–180.
- Eugene Charniak, Mark Johnson, Micha Elsner, Joseph Austerweil, David Ellis, Isaac Haxton, Catherine Hill, R. Shrivaths, Jeremy Moore, Michael Pozar, and Theresa Vu. 2006. Multilevel coarse-to-fine PCFG parsing. In *Proceedings of NAACL*, pages 168–175.
- Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of EMNLP*, pages 16–23.
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL*, pages 4171– 4186.
- Xiangyu Duan, Jun Zhao, and Bo Xu. 2007. Probabilistic models for action-based chinese dependency parsing. In *Proceedings of ECML*, pages 559–566.
- David Gaddy, Mitchell Stern, and Dan Klein. 2018. What's going on in neural constituency parsers? an analysis. In *Proceedings of NAACL*, pages 999– 1010.
- Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2011. Incremental joint POS tagging and dependency parsing in Chinese. In *Proceedings* of *IJCNLP*, pages 1216–1224.

- Jun Hatori, Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. 2012. Incremental joint approach to word segmentation, POS tagging, and dependency parsing in Chinese. In *Proceedings of ACL*, pages 1045–1053.
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.
- Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Nikita Kitaev and Dan Klein. 2018. Constituency parsing with a self-attentive encoder. In *Proceedings of ACL*, pages 2676–2686.
- Shuhei Kurita, Daisuke Kawahara, and Sadao Kurohashi. 2017. Neural joint model for transition-based Chinese syntactic analysis. In *Proceedings of ACL*, pages 1204–1214.
- Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of NAACL*, pages 260–270.
- Haonan Li, Zhisong Zhang, Yuqi Ju, and Hai Zhao. 2018. Neural character-level dependency parsing for chinese. In *Proceedings of AAAI*, pages 5205– 5212.
- Zhenghua Li, Min Zhang, Wanxiang Che, Ting Liu, Wenliang Chen, and Haizhou Li. 2011. Joint models for Chinese POS tagging and dependency parsing. In *Proceedings of EMNLP*, pages 1180–1191.
- Jiangming Liu and Yue Zhang. 2017. Shift-reduce constituent parsing with neural lookahead features. *TACL*, 5:45–58.
- Xiaoqiang Luo. 2003. A maximum entropy Chinese character-based parser. In *Proceedings of EMNLP*, pages 192–199.
- Eric W Noreen. 1989. Computer-intensive methods for testing hypotheses. Wiley New York.
- Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of NAACL*, pages 404–411.
- Xian Qian and Yang Liu. 2012. Joint Chinese word segmentation, POS tagging and parsing. In *Proceedings of EMNLP*, pages 501–511.
- Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. In *Proceedings of ACL*, pages 818–827.
- Zhiguo Wang and Nianwen Xue. 2014. Joint POS tagging and transition-based constituent parsing in Chinese with non-local features. In *Proceedings of ACL*, pages 733–742.

- Zhiguo Wang, Chengqing Zong, and Nianwen Xue. 2013. A lattice-based framework for joint Chinese word segmentation, POS tagging and parsing. In *Proceedings of ACL*, pages 623–627.
- Linzhi Wu and Meishan Zhang. 2021. Deep graphbased character-level chinese dependency parsing. *TASLP*, 29:1329–1339.
- Hang Yan, Xipeng Qiu, and Xuanjing Huang. 2020. A graph-based model for joint Chinese word segmentation and dependency parsing. *TACL*, 8:78–92.
- Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. 2013. Chinese parsing exploiting characters. In *Proceedings of ACL*, pages 125–134.
- Meishan Zhang, Yue Zhang, Wanxiang Che, and Ting Liu. 2014. Character-level Chinese dependency parsing. In *Proceedings of ACL*, pages 1326–1336.
- Yu Zhang, Houquan Zhou, and Zhenghua Li. 2020. Fast and accurate neural crf constituency parsing. In *Proceedings of IJCAI*, pages 4046–4053.
- Yue Zhang and Stephen Clark. 2009. Transition-based parsing of the Chinese treebank using a global discriminative model. In *Proceedings of IWPT*, pages 162–171.
- Xiaoqing Zheng, Haoyuan Peng, Yi Chen, Pengjing Zhang, and Wenqiang Zhang. 2015. Characterbased parsing with convolutional neural network. In *Proceedings of IJCAI*, pages 1054–1060.