Enhancing Discourse Dependency Parsing with Sentence Dependency Parsing: A Unified Generative Method Based on Code Representation

Zizhuo Shen and **Yanqiu Shao**[*] and **Wei Li**
Beijing Language and Culture University, China
blcushzz@gmail.com, yqshao163@163.com, liweitj47@blcu.edu.cn

Abstract

Due to the high complexity of Discourse Dependency Parsing (DDP) tasks, their existing annotation resources are relatively scarce compared to other NLP tasks, and different DDP tasks also have significant differences in annotation schema. These issues have led to the dilemma of low resources for DDP tasks. Thanks to the powerful capabilities of Large Language Models (LLMs) in cross-task learning, we can use LLMs to model dependency parsing under different annotation schema in an unified manner, in order to alleviate the dilemma of low resources for DDP tasks. However, enabling LLMs to deeply comprehend dependency parsing tasks is a challenge that remains underexplored. Inspired by the application of code-based methods in complex tasks, we propose a code-based unified dependency parsing method. We treat the process of dependency parsing as a search process of dependency paths and use code to represent this search process. Furthermore, we use a curriculum-learning based instruction tuning strategy for joint training of multiple dependency parsing tasks. The experimental results show that our proposed code-based DDP system has achieved good performance on two Chinese DDP tasks (especially significant improvement on the DDP task with relatively less training data).

1 Introduction

Discourse Dependency Parsing (DDP) is a structured prediction task. The input of this task is a sequence of Elementary Discourse Units (EDUs), and the output of this task is a dependency structure (a dependency tree labeled with discourse relation labels). As one of the fundamental tasks in the field of Natural Language Processing (NLP), DDP can assist downstream tasks such as emotion recognition (Ong et al., 2022), pronoun resolution (Yang et al., 2021), and event relation extraction (Tang et al., 2021).

Although the neural network-based NLP technology has greatly promoted the development of DDP tasks (Zhou and Feng, 2022; Liu et al., 2023). However, there are still some unresolved issues with DDP tasks. Firstly, the cost of annotating DDP data is relatively high, compared to Sentence-level Dependency Parsing (SDP) data, the existing scale of DDP data is smaller. Secondly, there are large differences in the annotation schema of DDP tasks, and heterogeneous data under different annotation schema are difficult to be effectively utilized (Nishida and Matsumoto, 2022). These problems have led to **the dilemma of low resources** in DDP tasks, limiting the application and development of DDP tasks.

Recently, Large Language Models (LLMs) has shown significant advantages in few-shot learning (Brown et al., 2020) and cross-task learning scenarios (Mishra et al., 2022) due to its powerful semantic understanding ability and transfer learning ability. Therefore, using LLM to design an unified dependency parsing method helps to break through the gap between dependency parsing tasks under different annotation granularity and different annotation schema, thereby overcoming the the dilemma of low resources in DDP tasks.

However, how to more effectively use LLMs for dependency parsing tasks remains an issue that requires more attention. Specifically, we need to face challenges from the following two aspects.

1. How to linearize dependency structures to suit the modeling method of language models?

2. How to design unified and LLM-friendly prompts for dependency parsing tasks?

For the first challenge, we propose a **path decomposition-based strategy for linearizing dependency structures**. Inspired by related work

[*]Corresponding Author

on Chain of Thought technology (Wei et al., 2022; Zhou et al., 2023) in complex tasks, we treat the dependency parsing task as a step-by-step search process of dependency paths. The model needs to gradually search out the entire dependency structure from the root node according to some search order, such as depth-first search (DFS) or breadth-first search (BFS). This strategy not only adapts to the modeling method of language models but also retains the structural information in the dependency structure.

For the second challenge, inspired by LLM-related works in the field of information extraction (Li et al., 2023; Sainz et al., 2023). We proposed a **code-based representation method for dependency parsing tasks**. Firstly, we use code language to represent the basic elements in dependency parsing tasks. Specifically, we use variables to represent the sequence of semantic units and semantic relation labels; classes to represent dependency arcs and their constituent head nodes, dependent nodes, and dependency relations; functions to represent some specific operations in the process of dependency analysis. Then, based on these basic elements, we can transform the process of dependency parsing into the process of code execution. This method can not only use code language to describe the process of dependency parsing clearly and concisely but also leverage the powerful underlying code generation capability of LLMs.

We attempt to model the SDP task and DDP task in an unified way, in order to achieve the goal of enhancing the DDP task using the SDP task. Specifically, we set the SDP task (we chose a syntactic dependency parsing task and a semantic dependency parsing task) as the source task, and set the DDP task as the target task. Leveraging our proposed dependency structure linearization strategy and the code-based dependency parsing task representation method, we are able to craft unified dependency parsing prompts for these tasks, and employ instruction tuning to equip LLMs with dependency parsing capabilities.

To achieve better performance on DDP tasks, we further adopt a **Curriculum Learning-based (CL-based) instruction tuning strategy**. Specifically, we first calculate a difficulty value for each sample in the dependency parsing task based on prior features such as the type of dependency parsing task, the number of dependency paths, and the average depth of dependency paths. Then, we dynamically adjust the loss weight of each sample during the entire training process according to the difficulty value of the sample and the current training steps, so as to achieve a training mode that learns from simple samples first, and then learns from difficult samples.

Our contributions can be summarized in the following three aspects: 1) we first propose to treat the dependency parsing process as a search process of dependency paths and design a code-based unified dependency parsing task prompt to represent this process. 2) We use a curriculum learning-based instruction tuning strategy to jointly train SDP tasks and DDP tasks, aiming to enhance the performance of DDP tasks with SDP tasks. 3) Our experimental results show that our code-based DDP system has achieved good performance in two Chinese DDP tasks, confirming the effectiveness of our proposed method.

2 Approach

Figure 1 illustrates code-based unified dependency parsing method we proposed. First, we give a definition of the unified dependency pasing task. Then, based on the path decomposition-based linearization strategy, we treat different types of dependency parsing tasks as the search process of dependency paths and use code-based prompt to represent this process. Finally, we propose a curriculum learning-based instruction tuning strategy to more effectively achieve the goal of using SDP tasks to improve DDP tasks.

2.1 Formal Definition of Unified Dependency Parsing Tasks

Given a sequence of semantic units U and a set of semantic relation labels L, the dependency parsing model needs to predict a set of dependency arcs A based on U and L. Each element $a_i = (h_i, d_i, r_i)$ in the dependency arc set A is a triplet. In the triplet, h_i represents the head node of the dependency arc, d_i represents the dependent node of the dependency arc, and r_i represents the dependency relation label. Among them, $h_i \in U$, $d_i \in U$, $r_i \in L$. The triples in the dependency arc set A can form the entire dependency structure D according to the linking relation.

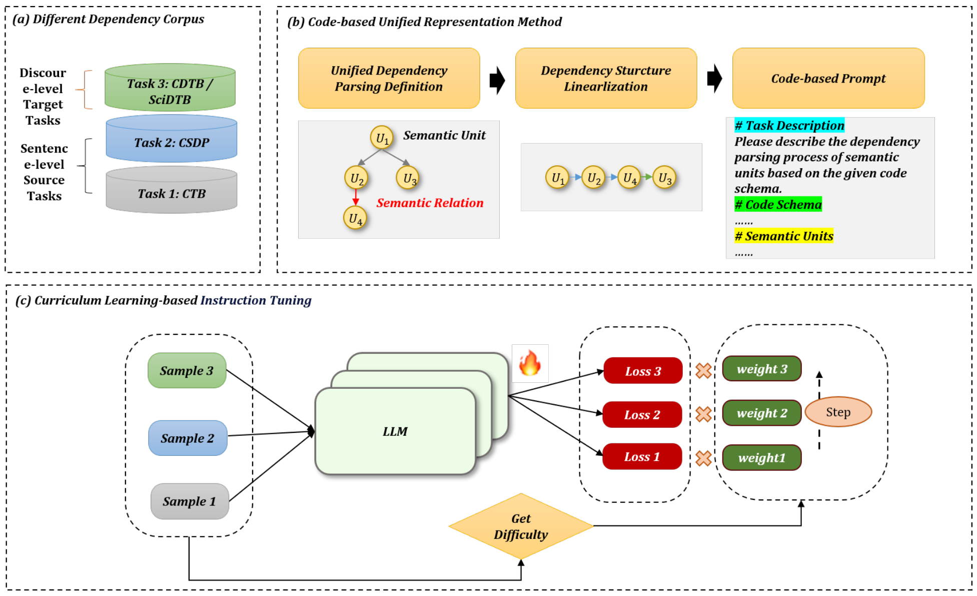

(a) Different Dependency Corpus
Discourse-level Target Tasks
Task 3: CDTB / SciDTB
Sentence-level Source Tasks
Task 2: CSDP
Task 1: CTB
(b) Code-based Unified Representation Method
Unified Dependency Parsing Definition
Dependency Sturcture Linearization
Code-based Prompt
Semantic Unit
Semantic Relation
# Task Description
Please describe the dependency parsing process of semantic units based on the given code schema.
# Code Schema
# Semantic Units
(c) Curriculum Learning-based Instruction Tuning
Sample 3
Sample 2
Sample 1
LLM
Loss 3
Loss 2
Loss 1
weight 3
weight 2
weight1
Step
Get Difficulty

Figure 1: Illustration of the code-based unified dependency parsing method. Figure (a) shows different types of dependency parsing tasks. Figure (b) shows the construction idea of code-based dependency parsing task prompt. Figure (c) shows the process of curriculum learning-based instruction tuning strategy.

2.2 Path Decomposition-based Linearization Strategy

We regard the process of dependency parsing as a search process for dependency paths. For a given dependency structure, taking the root node as the starting node and through depth-first search or breadth-first search, we can transform it into a sequence $\mathcal{P} = \{P_i\}$ composed of several dependency paths. For any two adjacent dependency arcs $a_i = (h_i, d_i, r_i)$ and $a_j = (h_j, d_j, r_j)$ in this sequence, they all satisfy the condition $d_i = h_j$.

Furthermore, through the path decomposition-based linearization strategy, we can model the task of dependency parsing as a structure-aware conditional language modeling task. For a given sequence of semantic units U and a set of semantic relation labels L, we can infer the dependency structure D through the following formula:

$$D = argmax \sum_{i}^{n} logp(a_{i+1}|a_i, U, L) \quad (1)$$

2.3 Code-based Representation Method

We use the concepts of variables, classes, and functions in the code language to design the code schema that can be used to represent dependency parsing tasks. Figure 2 shows the code schema we designed.

Variables: We use the concept of variables to represent the semantic unit sequence and the set of semantic relation labels. The semantic unit sequence contains two parts of information, one part is the positional information of the semantic unit in the sequence, and the other part is the textual information of the semantic unit itself. For positional information, we use the variable name ***index*** to represent the positional index of the semantic unit, and the value range of this variable is $0 \sim N$, where N is the length of the semantic unit sequence. For textual information, we use the variable name ***text*** to represent it, and the value range of this variable is all texts of the semantic units in the sample. We use the variable name ***label*** to represent the label

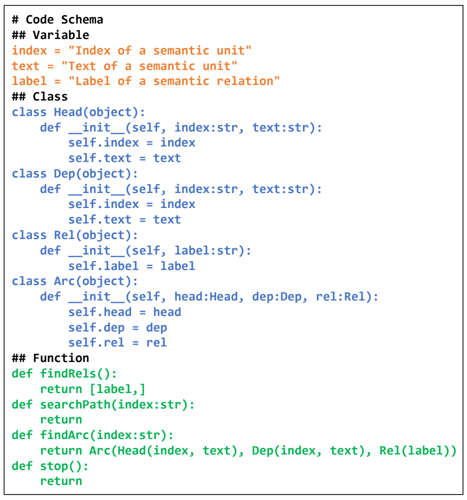

```
# Code Schema
## Variable
index = "Index of a semantic unit"
text = "Text of a semantic unit"
label = "Label of a semantic relation"
## Class
class Head(object):
    def __init__(self, index:str, text:str):
        self.index = index
        self.text = text
class Dep(object):
    def __init__(self, index:str, text:str):
        self.index = index
        self.text = text
class Rel(object):
    def __init__(self, label:str):
        self.label = label
class Arc(object):
    def __init__(self, head:Head, dep:Dep, rel:Rel):
        self.head = head
        self.dep = dep
        self.rel = rel
## Function
def findRels():
    return [label,]
def searchPath(index:str):
    return
def findArc(index:str):
    return Arc(Head(index, text), Dep(index, text), Rel(label))
def stop():
    return
```

Figure 2: The code schema that can be used to represent dependency parsing tasks.

of semantic relation.

Classes: We use the concept of classes to represent the dependency arc and its components: head node, dependent node, and dependency relation. The head node is represented by the ***Head*** class, and the dependent node is represented by the ***Dep*** class. Both classes need to be instantiated with the ***index*** variable and ***text*** variable. The dependency relation is represented by the ***Rel*** class, which is instantiated by passing the ***label*** variable. The ***Arc*** class is used to represent the dependency arc, which is instantiated by passing the ***Head*** class, ***Dep*** class, and ***Arc*** class.

Functions: We use the concept of functions to represent specific operations in the process of dependency parsing. The function ***findLabels*** is used to find all possible semantic relation labels between current semantic units, which takes no input and returns a set of semantic relation labels. The function ***searchPath*** is used to start searching for a new dependency path, which takes the position index of the starting semantic unit of the dependency path as input and has no return value. The function ***findArc*** is used to find a dependency arc, which takes the position index of the head node of the dependency arc as input and returns an instance of a dependency arc class. The function ***stop*** is used to indicate that all dependency paths in the dependency structure have been obtained, which takes neither input nor return value.

2.4 Code-based Unified Dependency Parsing Task Prompt

Figure 3 presents an example of the code-based unified dependency parsing task prompt.

The input part of the prompt mainly includes the following sections of information: Task Description, Code Schema, Task Name, and Semantic Units. The Task Description section is a natural language explanation of the dependency parsing task. The Code Schema section contains all code languages needed in the process of dependency parsing. The Task Name indicates the type of dependency parsing task. The Semantic Units section contains position index information and text information of the semantic unit sequence, which we store in a dictionary data structure with the index as the key and the text information as the value.

The output part of the prompt is the whole process of dependency parsing based on the current semantic unit. According to the example in Figure 3, we can see that firstly, LLM needs to call the ***findLabels*** function to predict all possible semantic relation labels in the current sample. Then, LLM will call ***searchPath*** and ***findArc*** functions to predict all possible dependency paths in the current sample until the output of the ***stop*** function ends.

2.5 Curriculum Learning-based Instruction Tuning

To better utilize SDP tasks to enhance the performance of DDP tasks, we adopt a CL-based instruction tuning strategy for LLMs. Inspired by the work of (Elgaar and Amiri, 2023; Wang et al., 2022a), we implement a simple and effective instance-level weighting-based curriculum learning algorithm.

Firstly, for each sample, we count several features: the type of dependency parsing task, the number of semantic units, the number of semantic relation labels, the number of dependency paths, and the average depth of dependency paths. For the type of dependency parsing task, we set the weight of syntactic dependency parsing task to 0.2, the weight of sentence-level semantic dependency parsing task to 0.3, and the weight of discourse-level dependency parsing task to 0.5. For the other features, we calculate the normalized value of each feature using formula 2, where X_i represents the value of a feature, X_{max} represents the maximum value of that feature class, and X_{min} represents the minimum value of that feature class.

$$\sigma_i = \frac{X_i - X_{min}}{X_{max} - X_{min}} \tag{2}$$

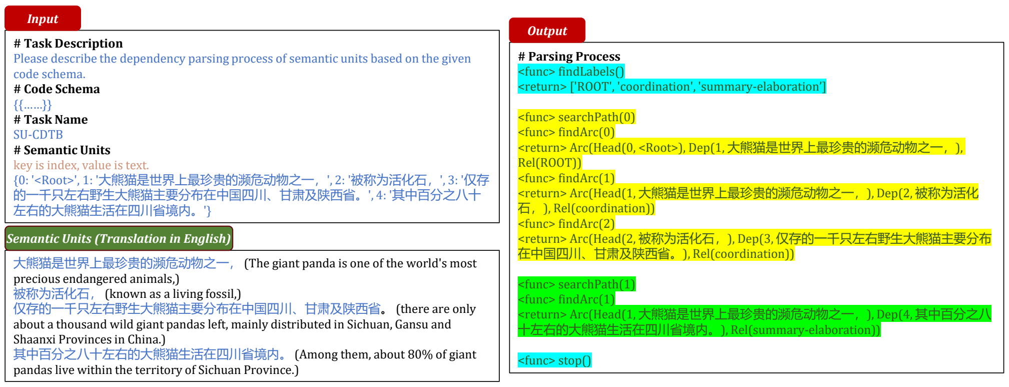
Secondly, We calculate the average of all features for each sample using formula 3. This value can be regarded as the difficulty value of each sample.

$$\phi_j = \frac{\sum_i^n \sigma_i}{n} \tag{3}$$

Finally, we use the negative-sigmoid formula to calculate the weight of each sample throughout the entire training process. The input value of formula 4 is the difficulty value ϕ_j, the training progress $p \in [0, 1]$, and a hyperparameter $\beta \in [1, \infty)$ that controls the rate of weight growth.

$$w_j = \frac{1}{1 + exp(\phi_j - p \times \beta)} \tag{4}$$

For a certain batch of samples, we can calculate the weighted loss through formula 5 for model



Input

```
# Task Description
Please describe the dependency parsing process of semantic units based on the given
code schema.
# Code Schema
{{......}}
# Task Name
SU-CDTB
# Semantic Units
key is index, value is text.
{0: '<Root>', 1: '大熊猫是世界上最珍贵的濒危动物之一，', 2: '被称为活化石，', 3: '仅存
的一千只左右野生大熊猫主要分布在中国四川、甘肃及陕西省。', 4: '其中百分之八十
左右的大熊猫生活在四川省境内。'}
```

Semantic Units (Translation in English)

大熊猫是世界上最珍贵的濒危动物之一， (The giant panda is one of the world's most precious endangered animals,)
被称为活化石， (known as a living fossil,)
仅存的一千只左右野生大熊猫主要分布在中国四川、甘肃及陕西省。 (there are only about a thousand wild giant pandas left, mainly distributed in Sichuan, Gansu and Shaanxi Provinces in China.)
其中百分之八十左右的大熊猫生活在四川省境内。 (Among them, about 80% of giant pandas live within the territory of Sichuan Province.)

Output

```
# Parsing Process
<func> findLabels()
<return> ['ROOT', 'coordination', 'summary-elaboration’]

<func> searchPath(0)
<func> findArc(0)
<return> Arc(Head(0, <Root>), Dep(1, 大熊猫是世界上最珍贵的濒危动物之一，),
Rel(ROOT))
<func> findArc(1)
<return> Arc(Head(1, 大熊猫是世界上最珍贵的濒危动物之一，), Dep(2, 被称为活化
石，), Rel(coordination))
<func> findArc(2)
<return> Arc(Head(2, 被称为活化石，), Dep(3, 仅存的一千只左右野生大熊猫主要分布
在中国四川、甘肃及陕西省。), Rel(coordination))

<func> searchPath(1)
<func> findArc(1)
<return> Arc(Head(1, 大熊猫是世界上最珍贵的濒危动物之一，), Dep(4, 其中百分之八
十左右的大熊猫生活在四川省境内。), Rel(summary-elaboration))

<func> stop()
```

Figure 3: An example of the unified dependency parsing task prompt. In the lower left of the figure, we offer the Chinese and English translations of the semantic unit text (this is not part of the prompt).

Algorithm 1 Curriculum Learning-based Instruction Tuning

```
Require: Training Samples $\mathcal{D}_{train}$, Model $\Theta$, Loss function $f$
 1: step ← 0
 2: while step < total_steps do
 3:     batch ← GetBatch($step, \mathcal{D}_{train}$)
 4:     loss ← GetLoss(batch, $\Theta$, $f$)
 5:     $\phi$ ← GetDifficulty(batch)
 6:     $w$ ← GetWeight(step, $\phi$)
 7:     weighted_loss ← loss $\otimes$ $w$
 8:     $\Theta$ ← UpdateModel(weighted_loss, $\Theta$)
 9:     step ← step + 1
10: end while
```

training. The entire CL-based instruction tuning strategy can be described as algorithm 1.

$$loss = \sum_{j=1} -w_j logp_j(a_{i+1}|a_i, U, L) \tag{5}$$

3 Experiments

3.1 Datasets

Sentence-level Dependency Parsing (SDP) Tasks: For sentence-level dependency parsing tasks, we chose a syntactic dependency parsing task and a semantic dependency parsing task for experiments. The dataset used for syntactic dependency parsing is Chinese Treebank 5.0 (CTB5), and the dataset used for semantic dependency parsing is Chinese Semantic Dependency Graph (CSDG) (Shao et al., 2023; Che et al., 2016). The basic information of these two datasets is shown in Table 1.

Discourse-level Dependency Parsing (DDP) Tasks: For DDP tasks, we chose a DDP dataset based on news corpus (SU-CDTB) (Li et al., 2014) and a DDP dataset based on scientific paper abstracts (Sci-CDTB) (Cheng and Li, 2019) for experiments. The basic information of these two datasets is shown in Table 1.

3.2 Implementation Details

We refer to the DDP system proposed in this work as the **Code-based** (if the system uses a curriculum learning-based instruction tuning strategy, we will refer it as **Code-based + CL**). In the main experiment, we primarily select baichuan2-7b-base [1] and deepseek-llm-7b-base [2], two 7B Base LLMs, to train our Code-based DDP System. Due to computational resource constraints, we adopt a QLoRA-based approach (Dettmers et al., 2023) for instruction tuning. The hyperparameters used during instruction tuning are as follows: learning_rate is 2e-4, lora_rank is 64, lora_alpha is 16, lora_dropout is 0.05, and the maximum number of epochs is 30.

3.3 Baseline Systems

In this work, the baseline systems we employ are categorized into three types: statistical machine learning-based (SML-based) systems, pre-trained language model-based (PLM-based) systems, and LLM-based systems.

SML-based Systems: We mainly compare the SML-based methods used in previous work. **Graph-based** System (Li et al., 2014) utilizes a perceptron-based algorithm to implement the DDP system. **Vanilla Transition-based** System (Nivre, 2003) uses an SVM-based model to predict transition actions and achieve DDP. **Two-stage Transition-based** System(Wang et al., 2017) initially uses an SVM-based model to predict transi-

[1]https://huggingface.co/baichuan-inc/Baichuan2-7B-Base

[2]https://huggingface.co/deepseek-ai/deepseek-llm-7b-base

Datasets		# labels	avg.# units	Train		Dev		Test	
				# samples	# arcs	# samples	# arcs	# samples	# arcs
SDP	CTB5	12	26	16091	437991	803	-	-	-
	CSDG	56	17	19055	389226	2069	-	-	-
DDP	SU-CDTB	18	5	1599	7533	350	1186	374	1794
	Sci-CDTB	23	13	68	885	20	267	21	255

Table 1: Data statistics. "#" and "avg." indicate "number of" and "average number of", respectively.

tion actions for the prediction of dependency structures, followed by another SVM-based model to accomplish the prediction of dependency relations.

PLM-based Systems: We select some open-source PLM-based DDP systems as baselines. **BERT-based Two-stage** System (Zhou and Feng, 2022) is a two-stage transition-based DDP system. This system first uses a transition-based model (BERT (Devlin et al., 2019) is used for the extraction of transition system state features) to predict dependency structures, and then uses a BERT-based sequence labeling model to predict dependency relations. **BERT-based Biaffine** System (Nishida and Matsumoto, 2022) is a classic graph-based DDP system. This system uses BERT to extract EDU features and uses Biaffine networks (Dozat and Manning, 2017) to predict dependency structures and dependency relations. **DAMT-based** System (Fan et al., 2022) is a DDP system that uses a multi-task learning method. This system combines the advantages of transition-based DDP systems and graph-based DDP systems, and uses XLNet (Yang et al., 2019) to extract EDU features.

LLM-based Systems: To verify the effectiveness of the code-based DDP system proposed in this paper, we implement a **Tuple-based** DDP system. The input of the tuple-based DDP system is the semantic unit of the current sample, and the output is the collection of all possible dependency relation tuples in the current sample. Each dependency relation tuple is represented in the form of a five-tuple: (head node semantic unit index, head node semantic unit text, dependent node semantic unit index, dependent node semantic unit text, semantic relation label).

3.4 Main Results

According to the experimental results in Table 2, we can draw several conclusions.

The code-based DDP system has achieved competitive results on DDP tasks. On the SU-CDTB test set with more training samples, the code-based DDP system has reached a level comparable to multiple PLM-based DDP systems. On the Sci-CDTB test set with fewer training samples, the effect of the code-based DDP system has surpassed that of the PLM-based DDP system. These experimental results indicate that the code-based DDP system may have a better ability to learn from small scale samples.

The code-based DDP system performs better than the tuple-based DDP system on DDP tasks. In several experiments based on LLM, whether using baichuan2-7b-base or deepseek-llm-7b-base, the code-based DDP system performs better than the tuple-based DDP system on both test sets. These experimental results suggest that LLM might more easily understand data in code form, and the structural information in the code-based DDP system may help improve the performance of DDP.

Introducing SDP tasks and performing CL-based instruction tuning can significantly improve the performance of code-based DDP systems. These experimental results indicate that SDP tasks have a positive effect on DDP tasks. Furthermore, we believe that the unified dependency parsing task prompt build a bridge between different forms of dependency parsing tasks, enabling LLM to learn the common features of dependency parsing tasks from different dependency parsing tasks, thereby realizing cross-task transfer.

3.5 Detailed Analysis

Comparison of Different Instruction Tuning Strategies: To verify the effectiveness of the CL-based instruction tuning strategy, we implement two additional instruction tuning strategies for comparative experiments.

The first is multi-task learning-based (MTL-based) instruction tuning. Under this strategy, we merge the training data of all tasks together for training. Throughout the training process, we do not adjust the weight of each sample's loss based

Systems		SU-CDTB		Sci-CDTB	
		UAS	LAS	UAS	LAS
SML-based	**Graph**	58.5	41.5	33.8	17.5
	Vanilla Transition-based	80.3	58	52.5	27.6
	Two-stage	80.3	58.7	52.5	27.6
PLM-based	**BERT-based Two-stage**	82.2	**64.8**	59.5	35.5
	BERT-based Biaffine	80.1	59.8	58.8	34.9
	DAMT-based	80.6	59.9	59.8	36.8
LLM-based	**Tuple-based (baichuan2-7b-base)**	80.5	59.1	71.2	39.9
	Tuple-based (deepseek-llm-7b-base)	80.1	57.5	59.9	29.5
	Code-based (baichuan2-7b-base)	84.3	62.2	76.2	49.4
	Code-based (deepseek-llm-7b-base)	82.6	59.4	62.5	29.9
	Code-based (baichuan2-7b-base) + CL	**85.1**	64.5	**78.3**	**51.2**
	Code-based (deepseek-llm-7b-base) + CL	84.5	61.3	70.5	39.9

Table 2: Results of all systems on DDP tasks.

Strategies	SU-CDTB		Sci-CDTB	
	UAS	LAS	UAS	LAS
Code-based	84.3	62.2	76.2	49.4
Code-based + CL	**85.1**	**64.5**	**78.3**	**51.2**
Code-based + MTL	84.6	62.1	76.3	49.6
Code-based + MSL	84.9	63.9	77.7	49.9

Table 3: Results of different instruction tuning strategies on DDP tasks. In these experiments, we use the baichuan2-7b-base model for training.

on the difficulty of the sample and the current progress of training steps. The second is multi-stage learning-based (MST-based) instruction tuning. Under this strategy, we train the model in two stages. Firstly, we carry out the first stage of training on SDP tasks. Then, based on the model obtained from the first stage of training, we perform the second stage of training on DDP tasks.

According to the experimental results in Table 3, we find that **the CL-based instruction tuning strategy outperforms the other two instruction tuning strategies on both test sets**. These experimental results indicate that the CL-based instruction tuning strategy can better enhance the LLM's learning of DDP tasks, thereby effectively avoiding multi-task conflicts, catastrophic forgetting, and other problems present in the other two training methods.

Comparison of Different Types of Models: In order to compare the effects of different types of LLMs on DDP tasks, we chose several different types of LLMs released by DeepSeek to train Code-based DDP systems and conduct a comparative analysis. The deepseek-llm-7b-base model (**Base Model**) is trained from scratch on a vast dataset of 2T tokens in both English and Chinese. The deepseek-llm-7b-chat model (**Chat Model**) [3] is initialized from deepseek-llm-7b-base and fine-tuned on extra instruction data. The deepseek-coder-7b-instruct model (**Code Model**) [4] is initialized from deepseek-llm-7b-base and fine-tuned on extra code data and instruction data.

According to the experimental results in Figure 4, firstly, we find that the performance of Code model and Base model on DDP tasks is significantly better than that of Chat model. This indicates that the Chat model has lost a certain degree of code ability after enhancing its dialogue ability. Secondly, although the Code model performs slightly worse than the Base model on the SU-CDTB test set, the Code model's performance on the Sci-CDTB test set is significantly better than the Base model. This suggests that **further enhancing the code ability of LLM can help improve the performance of Code-based DDP on DDP tasks**, especially on tasks with relatively less training data.

Comparison of Different Linearization Orders: To compare the effects of data obtained from different linearization orders on DDP tasks, we use data obtained from DFS linearization, BFS linearization, and a mixture of DFS and BFS linearization to train Code-based DDP systems.

According to the experimental results in Figure 5, we find that whether the data obtained by using the DFS linearization or the BFS linearization, the performance of the Code-based DDP system on

[3] https://huggingface.co/deepseek-ai/deepseek-llm-67b-chat

[4] https://huggingface.co/deepseek-ai/deepseek-coder-7b-instruct-v1.5

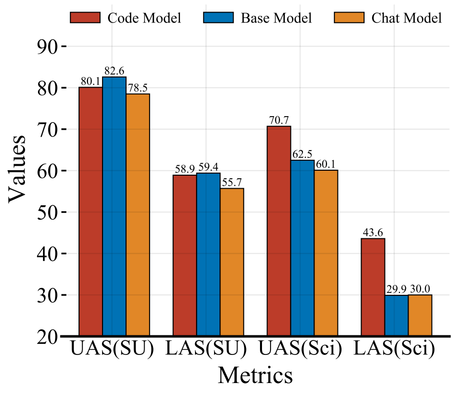


Figure 4: Results of different types of models on DDP tasks. SU stands for the SU-CDTB dataset. Sci stands for the Sci-CDTB dataset.

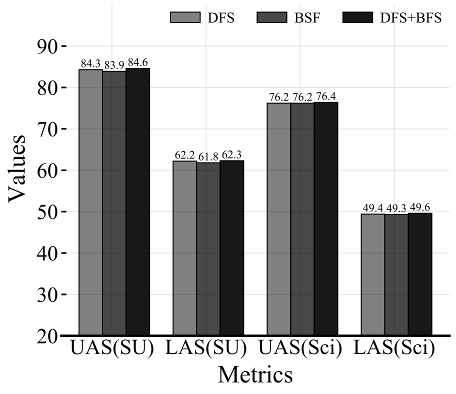


Figure 5: Results of different linearization orders on DDP tasks. SU stands for the SU-CDTB dataset. Sci stands for the Sci-CDTB dataset.

DDP tasks is basically the same. This indicates that the Code-based DDP system may not be sensitive to data obtained from different linearization orders. Furthermore, training the Code-based DDP system with data obtained from a mixed with DFS and BFS linearization did not achieve a significant improvement on DDP tasks. This indicates that there is no significant information gain for the Code-based DDP system from data obtained from different linearization orders.

4 Related Works

Dependency parsing: Common dependency parsing methods can be divided into three categories: graph-based methods (Dozat and Manning, 2017, 2018; Zhang et al., 2020), transition-based methods (Wang et al., 2022b; Zhou and Feng, 2022), and autoregressive generative methods (Li et al., 2018; He and Choi, 2023). The graph-based method treats the dependency parsing task as a graph search problem. This method requires a scoring model to calculate the scores of each edge in the dependency structure. The transition-based method treats the dependency parsing task as a sequence-to-action conversion problem (Sun et al., 2022). This method usually requires different transition systems to be designed for different types of dependency structures. The autoregressive generative method treats the dependency parsing task as a sequence-to-sequence generation problem. This method usually requires linearizing the dependency structure to fit the modeling method of autoregressive generative models. Thanks to the powerful generation capabilities of LLMs, we also try to use the autoregressive generative method to build our dependency parsing system. Unlike previous work, we first attempt to use code to represent different types of dependency parsing tasks, and use a curriculum-learning based instruction tuning strategy for joint training of multiple dependency parsing tasks.

Code-based Methods for NLP: Given that LLMs are typically pre-trained using code data and have developed strong capabilities in code comprehension and generation, consequently, some researchers have embarked on exploring the use of code for complex NLP tasks, aiming to harness the inherent code processing strengths of LLMs. Wang et al. (2023) use code language to represent event extraction tasks, improving the performance of LLM in event extraction tasks. Bi et al. (2023) use code language to represent triple extraction tasks for automatic construction of knowledge graphs. Li et al. (2024) propose a code-based unified information extraction method and verify the effectiveness of this method in few-shot learning scenarios and instruction tuning scenarios. Inspired by above works, we first try to use code language to represent more complex dependency paring tasks.

5 Conclusion

In order to better utilize LLMs to alleviate the low-resource dilemma of DDP tasks, we propose a code-based unifed dependency parsing method. Specifically, we regard the process of dependency parsing as a search process of dependency paths and design a code-based unifed dependency parsing task prompt based on this search process. Furthermore, we use a curriculum learning-based instruction tuning strategy to jointly train SDP tasks and DDP tasks under different annotation shema, achieving the goal of using SDP tasks to enhance

DDP tasks. Experimental results show that our code-based DDP system can take advantage of LLM's strengths in code understanding, enabling it to achieve good performance on DDP tasks.

Limitations

The main limitations of our research include the following aspects: 1) Computational resource limitations. When selecting models, we only consider models with a parameter size of 7B for testing, without exploring models with larger parameter sizes; in terms of training strategies, we only adopt the parameter-efficient fine-tuning method, without attempting the full parameter fine-tuning method. 2) Test set limitations. Our experiments only involved two representative Chinese DDP test sets, without extending to more languages or domains. 3) Experimental scenario limitations. We only verify the effectiveness of the proposed method in the supervised fine-tuning scenario, without conducting experiments in the in-context learning scenario.

We plan to conduct experiments on more models and test sets in future work and verify the effectiveness of our proposed method in the in-context learning scenario.

Acknowledgements

This research project is supported by the National Natural Science Foundation of China (No.62306045), the Humanities and Social Science Project of the Ministry of Education (24YJAZH119), Science Foundation of Beijing Language and Culture University (supported by "the Fundamental Research Funds for the Central Universities") (18ZDJ03).

References

Zhen Bi, Jing Chen, Yinuo Jiang, Feiyu Xiong, Wei Guo, Huajun Chen, and Ningyu Zhang. 2023. Codekgc: Code language model for generative knowledge graph construction. *CoRR*, abs/2304.09048.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Wanxiang Che, Yanqiu Shao, Ting Liu, and Yu Ding. 2016. SemEval-2016 task 9: Chinese semantic dependency parsing. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1074–1080, San Diego, California. Association for Computational Linguistics.

Yi Cheng and Sujian Li. 2019. Zero-shot Chinese discourse dependency parsing via cross-lingual mapping. In *Proceedings of the 1st Workshop on Discourse Structure in Neural NLG*, pages 24–29, Tokyo, Japan. Association for Computational Linguistics.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Timothy Dozat and Christopher D. Manning. 2018. Simpler but more accurate semantic dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 484–490, Melbourne, Australia. Association for Computational Linguistics.

Mohamed Elgaar and Hadi Amiri. 2023. Ling-CL: Understanding NLP models through linguistic curricula. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13526–13542, Singapore. Association for Computational Linguistics.

Yaxin Fan, Peifeng Li, Fang Kong, and Qiaoming Zhu. 2022. A distance-aware multi-task framework for conversational discourse parsing. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 912–921, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Han He and Jinho D. Choi. 2023. Unleashing the true potential of sequence-to-sequence models for sequence tagging and structure parsing. *Transactions of the Association for Computational Linguistics*, 11:582–599.

Peng Li, Tianxiang Sun, Qiong Tang, Hang Yan, Yuanbin Wu, Xuanjing Huang, and Xipeng Qiu. 2023. CodeIE: Large code generation models are better few-shot information extractors. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15339–15353, Toronto, Canada. Association for Computational Linguistics.

Sujian Li, Liang Wang, Ziqiang Cao, and Wenjie Li. 2014. Text-level discourse dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 25–35, Baltimore, Maryland. Association for Computational Linguistics.

Zixuan Li, Yutao Zeng, Yuxin Zuo, Weicheng Ren, Wenxuan Liu, Miao Su, Yucan Guo, Yantao Liu, Xiang Li, Zhilei Hu, et al. 2024. Knowcoder: Coding structured knowledge into llms for universal information extraction. *arXiv preprint arXiv:2403.07969*.

Zuchao Li, Jiaxun Cai, Shexia He, and Hai Zhao. 2018. Seq2seq dependency parsing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3203–3214, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Tianyi Liu, Yansong Feng, and Dongyan Zhao. 2023. Learning dynamic representations for discourse dependency parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 14253–14263. Association for Computational Linguistics.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. 2022. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3470–3487, Dublin, Ireland. Association for Computational Linguistics.

Noriki Nishida and Yuji Matsumoto. 2022. Out-of-domain discourse dependency parsing via bootstrapping: An empirical analysis on its effectiveness and limitation. *Transactions of the Association for Computational Linguistics*, 10:127–144.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the Eighth International Conference on Parsing Technologies*, pages 149–160, Nancy, France.

Donovan Ong, Jian Su, Bin Chen, Anh Tuan Luu, Ashok Narendranath, Yue Li, Shuqi Sun, Yingzhan Lin, and Haifeng Wang. 2022. Is discourse role important for emotion recognition in conversation? *Proceedings of the AAAI Conference on Artificial Intelligence*, page 11121–11129.

Oscar Sainz, Iker García-Ferrero, Rodrigo Agerri, Oier Lopez de Lacalle, German Rigau, and Eneko Agirre. 2023. Gollie: Annotation guidelines improve zero-shot information-extraction. *CoRR*, abs/2310.03668.

Yanqiu Shao, Wanxiang Che, Ting Liu, and Yu Ding. 2023. The construction of a chinese semantic dependency graph bank. In *Chinese Language Resources: Data Collection, Linguistic Analysis, Annotation and Language Processing*, pages 211–226. Springer.

Tianxiang Sun, Xiangyang Liu, Xipeng Qiu, and Xuanjing Huang. 2022. Paradigm shift in natural language processing. *Int. J. Autom. Comput.*, 19(3):169–183.

Jialong Tang, Hongyu Lin, Meng Liao, Yaojie Lu, Xianpei Han, Le Sun, Weijian Xie, and Jin Xu. 2021. From discourse to narrative: Knowledge projection for event relation extraction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

Peiyi Wang, Liang Chen, Tianyu Liu, Damai Dai, Yunbo Cao, Baobao Chang, and Zhifang Sui. 2022a. Hierarchical curriculum learning for AMR parsing. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 333–339, Dublin, Ireland. Association for Computational Linguistics.

Xingyao Wang, Sha Li, and Heng Ji. 2023. Code4Struct: Code generation for few-shot event structure prediction. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3640–3663, Toronto, Canada. 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.

Yuxuan Wang, Wanxiang Che, Jiang Guo, and Ting Liu. 2022b. A neural transition-based approach for semantic dependency graph parsing. *Proceedings of the AAAI Conference on Artificial Intelligence*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Jingxuan Yang, Kerui Xu, Jun Xu, Si Li, Sheng Gao, Jun Guo, Nianwen Xue, and Ji-Rong Wen. 2021. A joint model for dropped pronoun recovery and conversational discourse parsing in chinese conversational speech. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 5754–5764.

Yu Zhang, Zhenghua Li, and Min Zhang. 2020. Efficient second-order TreeCRF for neural dependency parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3295–3305, Online. Association for Computational Linguistics.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Yifei Zhou and Yansong Feng. 2022. Improve discourse dependency parsing with contextualized representations. In *Findings of the Association for Computational Linguistics: NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 2250–2261. Association for Computational Linguistics.