CONSTITUENCY PARSING VIA FEW SHOT PROMPTING

By

ANINDITA CHAVAN

A thesis submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Master of Science

Graduate Program in Computer Science

Written under the direction of

Dr. Karl Stratos

And approved by

________________________________________

________________________________________

________________________________________

________________________________________

New Brunswick, New Jersey

May 2023
Constituency parsing is a fundamental task in natural language processing (NLP) that involves analyzing the grammatical structure of a sentence and identifying its constituent parts, such as noun phrases, verb phrases, and clauses. Prompting pretrained large language models (LLMs) has dramatically improved the state of the art in NLP across many tasks. This leads us to the substantial interest in analyzing the syntactic knowledge that LLMs learn. In this work, we use prompting in zero shot and chain-of-thought setting on various LLMs with multiple prompt formats. The experiments carried out on Codex asking for direct parse trees given a sentence gives an F1 score of 62.8 which beats the current state-of-the-art by 5 points.
ACKNOWLEDGMENTS

I would like to express my heartfelt gratitude to my advisor, Dr. Karl Stratos, for his invaluable guidance and support throughout my thesis and research work. His knowledge, expertise, unwavering encouragement and insightful feedback helped me bring my project to fruition. I would also like to extend my sincere appreciation to all of the professors in the Computer Science Department who have contributed to my knowledge that helped me in this project. Last but not least, I want to express my deep gratitude to my family members for their constant love, support, and encouragement throughout my graduate studies.
<table>
<thead>
<tr>
<th>TABLE OF CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
</tr>
<tr>
<td>Acknowledgments</td>
</tr>
<tr>
<td>List of Tables</td>
</tr>
<tr>
<td>List of Figures</td>
</tr>
<tr>
<td>Chapter 1: Introduction and Background</td>
</tr>
<tr>
<td>1.1 What is constituency parsing?</td>
</tr>
<tr>
<td>1.2 Context-free grammars</td>
</tr>
<tr>
<td>1.2.1 Formal definition</td>
</tr>
<tr>
<td>1.2.2 Applications of CFGs</td>
</tr>
<tr>
<td>1.3 Ambiguity</td>
</tr>
<tr>
<td>1.3.1 Causes of Ambiguity:</td>
</tr>
<tr>
<td>1.3.2 Addressing Ambiguity:</td>
</tr>
<tr>
<td>1.4 Dataset: English Penn Treebank (PTB)</td>
</tr>
<tr>
<td>1.4.1 Overview</td>
</tr>
<tr>
<td>1.4.2 Structure</td>
</tr>
<tr>
<td>Chapter 2: Related Work</td>
</tr>
</tbody>
</table>
## 2.1 Deep Inside-Outside Recursive Autoencoder (DIORA) . . . . . . . . . . . . 7

## 2.2 Ordered Neurons (ON): . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

## 2.3 Probabilistic Context-Free Grammar (PCFG): . . . . . . . . . . . . . . . 8

## 2.4 Unsupervised Recurrent Neural Network Grammars (URNNG): . . . . . . 8

## 2.5 Generative Adversarial Network (GAN)-based Parsing . . . . . . . . . . . 8

## 2.6 Inside-Outside Recursive Autoencoder (IORA): . . . . . . . . . . . . . . . 9

### Chapter 3: SOTA: Unsupervised Parsing via Constituency Tests . . . . . . . 10

## 3.1 What are constituency tests? . . . . . . . . . . . . . . . . . . . . . . . . 10

## 3.2 Unsupervised Parsing via Constituency Tests . . . . . . . . . . . . . . . 11

### Chapter 4: Scoring spans using GPT-2 . . . . . . . . . . . . . . . . . . . . . . 14

### Chapter 5: Scoring spans by prompting T0++ . . . . . . . . . . . . . . . . . 16

### Chapter 6: Prompting InstructGPT . . . . . . . . . . . . . . . . . . . . . . . 19

### Chapter 7: Prompting InstructGPT: Chain-of-Thought Prompting . . . . . . 21

### Chapter 8: Prompting Codex . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

### Chapter 9: Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

### References . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26
LIST OF TABLES

3.1 Examples of Constituency Tests ............................................. 11

4.1 Results for constituency parsing using GPT-2 .......................... 15

5.1 Results for prompting using T0pp ......................................... 18

6.1 Results for prompting using InstructGPT ................................. 20

7.1 Results using Chain-of-Thought prompting .............................. 22

8.1 Results for prompting using Codex ....................................... 23
LIST OF FIGURES

1.1 Tree Structure: Constituency Parsing ........................................ 1
1.2 Ambiguity in constituency parsing: Multiple derivations ............... 4
3.1 Step-by-step approach to parsing via constituency tests .................. 12
4.1 Constituency parsing using GPT-2 ........................................... 14
4.2 Using sentence probability to score spans ................................ 15
5.1 Scoring spans by prompting T0++ .......................................... 16
5.2 T0pp prompt format ............................................................. 17
6.1 Prompting InstructGPT: Asking for direct parse ............................ 19
7.1 Prompting InstructGPT: Chain-of-Thought Prompting ................. 21
1.1 What is constituency parsing?

Constituency parsing is a fundamental task in natural language processing that involves identifying the hierarchical structure of a sentence. There are two primary approaches to constituency parsing: supervised and unsupervised. In supervised constituency parsing, a model is trained on a large corpus of labeled sentences, where each sentence is annotated with its corresponding parse tree. The model then uses this labeled data to learn to predict the correct parse tree for new, unseen sentences. On the other hand, unsupervised constituency parsing does not rely on labeled data, instead it tries to find the underlying structure of the sentences by clustering words or phrases based on various similarity metrics.

Recent work in constituency parsing has focused on improving the accuracy and effi-
ciency of both supervised and unsupervised approaches. In supervised parsing, researchers have explored various neural network architectures, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformer-based models. These models have shown promising results in capturing complex syntactic structures and achieving state-of-the-art performance on several benchmark datasets. However, they require a large amount of labeled data to train, which can be costly and time-consuming to obtain.

In unsupervised parsing, researchers have focused on developing algorithms that can learn the underlying structure of sentences without relying on labeled data. Clustering algorithms such as hierarchical agglomerative clustering and k-means have been used to group words or phrases based on their similarity in context. Recent unsupervised models have incorporated neural network architectures to better capture semantic and syntactic information in the clustering process.

1.2 Context-free grammars

Context-free grammars (CFGs) are a type of formal grammar commonly used in computational linguistics and computer science to model the syntax of natural language and programming languages.

1.2.1 Formal definition

A context-free grammar is a four-tuple \( G = (V, \Sigma, R, S) \) where:

- \( V \) is a finite set of non-terminal symbols or variables, such as \( S, \) NP, VP, etc. These symbols represent syntactic categories that can be replaced by other symbols in the grammar.

- \( \Sigma \) is a finite set of terminal symbols or terminals, such as words, punctuation, etc. These symbols represent the lexical elements of the language being modeled.

- \( R \) is a set of production rules, each of the form \( A \rightarrow \alpha \), where \( A \) is a non-terminal
symbol and $\alpha$ is a string of terminals and/or non-terminals. These rules define how non-terminal symbols can be replaced by other symbols in the grammar.

- $S$ is a start symbol, which is a non-terminal symbol that represents the root of the derivation tree.

The production rules in a CFG specify how non-terminal symbols can be replaced by other symbols. For example, a production rule $NP \rightarrow \text{Det } N$ might specify that an NP (noun phrase) can be formed by replacing a Det (determiner) with a N (noun). Using a set of such rules, the CFG can generate a set of strings of terminals that form valid sentences in the language being modeled.

1.2.2 Applications of CFGs

- **Parsing**: CFGs can be used to parse natural language sentences and determine their syntactic structure. This involves constructing a derivation tree for the sentence, which shows how it can be generated from the CFG. Parsing can be done using algorithms such as the CYK algorithm, Earley parser, or chart parsing.

- **Machine Translation**: CFGs can be used to model the syntax of different languages and facilitate machine translation between them. This involves creating a CFG for each language, and then mapping the rules between the two grammars to enable translation.

- **Text-to-Speech Synthesis**: CFGs can be used to model the syntax of spoken languages and facilitate text-to-speech synthesis. This involves creating a CFG for the language’s phonology, which specifies how the sounds of the language are formed from phonemes, and a CFG for the language’s prosody, which specifies the rhythm, stress, and intonation of the language. In addition to these applications, CFGs are also used in programming languages to model their syntax and enable compilers to translate code into machine-readable form.
1.3 Ambiguity

Ambiguity is a common problem in constituency parsing. It arises when a sentence can be parsed into multiple valid tree structures, each representing a different interpretation of the sentence’s meaning.

![Ambiguity in constituency parsing: Multiple derivations](image)

**Figure 1.2: Ambiguity in constituency parsing: Multiple derivations**

1.3.1 Causes of Ambiguity:

- **Structural Ambiguity**: This occurs when a sentence can be parsed into two or more tree structures that have the same words in different orders. For example, the sentence "I saw the man with the telescope" can be parsed as "I saw the man with the telescope" or "I saw the man who had the telescope."

- **Lexical Ambiguity**: This occurs when a word has multiple meanings that can lead to different interpretations of a sentence’s syntax. For example, the word "bank" can refer to a financial institution or a riverbank, leading to different syntactic structures for sentences like "I went to the bank."
• **Attachment Ambiguity**: This occurs when a word or phrase can be attached to different parts of the sentence’s structure, leading to different interpretations. For example, the sentence ”John saw the woman with the binoculars” can be parsed as ”John saw the woman using binoculars” or ”John saw the woman who had binoculars.”

1.3.2 Addressing Ambiguity:

• **Probabilistic Parsing**: This involves assigning probabilities to the possible tree structures of a sentence, based on their likelihood of representing the sentence’s intended meaning. Probabilistic parsing can be done using statistical models such as Hidden Markov Models (HMMs) or Conditional Random Fields (CRFs).

• **Rule-Based Disambiguation**: This involves using rules to disambiguate between the possible tree structures of a sentence. For example, a rule might specify that an adjective should be attached to the noun it modifies, rather than a nearby noun or verb.

• **Semantic Role Labeling**: This involves identifying the roles of words in a sentence (such as subject, object, or modifier), and using this information to disambiguate between the possible tree structures. Semantic role labeling can be done using supervised learning algorithms, such as Support Vector Machines (SVMs) or neural networks.

• **Ensemble Techniques**: This involves combining the outputs of multiple parsing models, each using a different algorithm or feature set, to improve the accuracy and robustness of the parsing process.

1.4 **Dataset: English Penn Treebank (PTB)**

The English Penn Treebank (PTB) [1] dataset is a widely-used benchmark dataset in natural language processing (NLP) that contains parsed sentences from the Wall Street Journal
section of the Penn Treebank corpus. In this work, we are using the English PTB test set for evaluation and hand-picked examples from the PTB train set for generating prompts.

1.4.1 Overview

The PTB dataset was first released in the early 1990s, and has since become one of the most widely-used datasets for research in NLP. It contains approximately 2.4 million words of text, comprising over 49,000 sentences, that have been hand-annotated with syntactic parse trees. The dataset covers a range of text genres, including news articles, editorials, and reviews, and has been extensively used for research in areas such as parsing, part-of-speech tagging, named entity recognition, and sentiment analysis.

1.4.2 Structure

The PTB dataset is organized into sections, where each section consists of a set of files that contain parsed sentences in a specific format. The most commonly used format is the "bracketed parse tree" format, where each sentence is represented as a tree structure, with each node in the tree representing a constituent of the sentence (such as a noun phrase or verb phrase). Each node is labeled with a part-of-speech tag (such as "NN" for noun or "VB" for verb), and is connected to its parent node and child nodes by labeled arcs. The PTB dataset also includes other formats, such as the "dependency parse" format, where each word in the sentence is represented as a node, and edges between nodes represent syntactic dependencies between words.
CHAPTER 2

RELATED WORK

Recent work in unsupervised parsing has focused on developing novel algorithms that can leverage large amounts of unannotated data to identify the underlying syntactic structure of sentences. These approaches hold promise for advancing our understanding of language and improving the performance of downstream natural language processing tasks.

2.1 Deep Inside-Outside Recursive Autoencoder (DIORA)

DIORA [2] is an unsupervised parsing algorithm that learns to generate parse trees by optimizing a reconstruction loss on a pre-trained autoencoder. The model generates parse trees by recursively applying a set of production rules until all terminal symbols have been generated. The key innovation of DIORA is its use of a deep inside-outside recursive neural network architecture that allows the model to capture long-range dependencies and better model the hierarchical structure of natural language sentences. DIORA has achieved state-of-the-art performance on a number of unsupervised parsing benchmarks, but it can be computationally expensive and requires a pre-trained autoencoder.

2.2 Ordered Neurons (ON):

ON [3] is an unsupervised parsing algorithm that uses a novel permutation scheme to capture long-range dependencies and better model the hierarchical structure of natural language sentences. The model generates parse trees by recursively applying a set of production rules to the sentence, but instead of processing the words in a fixed order, the model permutes the order of the words based on their syntactic relationships. This allows the model to capture long-range dependencies and better model the hierarchical structure of
natural language sentences. ON has achieved state-of-the-art performance on a number of unsupervised parsing benchmarks, and it has the added advantage of being computationally efficient and easy to implement. However, ON requires a pre-processing step to identify the syntactic relationships between words in the sentence.

2.3 Probabilistic Context-Free Grammar (PCFG):

PCFG [4] is a generative probabilistic model that is commonly used for unsupervised parsing. It assigns a probability to each parse tree based on the probability of its production rules and the probability of the terminal symbols in the sentence. PCFG generates parse trees by recursively applying a set of production rules until all terminal symbols have been generated. While PCFG is relatively simple and efficient, it does not capture the complex linguistic phenomena that are present in natural language.

2.4 Unsupervised Recurrent Neural Network Grammars (URNNG):

URNNG [5] is an unsupervised parsing algorithm that uses recurrent neural networks (RNNs) to generate parse trees. The model is trained on unannotated data using a likelihood-based loss function that encourages the model to generate parse trees that are consistent with the input sentence. URNNG generates parse trees by recursively applying a set of RNN-based production rules until all terminal symbols have been generated. By leveraging the expressive power of neural networks, URNNG can generate parse trees that are more accurate and linguistically plausible than traditional unsupervised parsing algorithms.

2.5 Generative Adversarial Network (GAN)-based Parsing

GAN-based parsing is an unsupervised parsing algorithm that uses GANs to generate parse trees. The model consists of a generator network that generates parse trees and a discriminator network that distinguishes between real and generated parse trees. The generator is
trained to generate parse trees that fool the discriminator, while the discriminator is trained to accurately distinguish between real and generated parse trees. By leveraging the power of GANs, this approach can generate parse trees that are more accurate and linguistically plausible than traditional unsupervised parsing algorithms, but it can be challenging to train and requires a large amount of data.

2.6 Inside-Outside Recursive Autoencoder (IORA):

IORA [6] is an unsupervised parsing algorithm that learns to generate parse trees by optimizing a reconstruction loss on a pre-trained autoencoder. The model generates parse trees by recursively applying a set of production rules until all terminal symbols have been generated. The model is trained to minimize the reconstruction error between the input sentence and the reconstructed sentence generated by the parse tree. IORA can generate parse trees that are more accurate and linguistically plausible than traditional unsupervised parsing algorithms, but it requires a pre-trained autoencoder and can be computationally expensive.
3.1 What are constituency tests?

Constituency tests are a set of techniques used to determine the internal structure of a sentence by identifying its constituent parts. The goal of constituency tests is to determine which words in a sentence group together to form larger units or constituents, and to identify the hierarchical relationships between these constituents.

There are several common types of constituency tests, including:

1. **Substitution Test**: In this test, a word or phrase in the sentence is replaced with a pronoun or other substitute, and the resulting sentence is checked to see if it is grammatical.

2. **Movement Test**: In this test, a word or phrase in the sentence is moved to a different position in the sentence, and the resulting sentence is checked to see if it is grammatical.

3. **Coordination Test**: In this test, two or more words or phrases in the sentence are joined by a coordinating conjunction such as ”and” or ”or”, and the resulting sentence is checked to see if it is grammatical.

By applying these and other constituency tests, linguists can gain insights into the internal structure of a sentence and develop more accurate models of language syntax.
<table>
<thead>
<tr>
<th>Constituency Test</th>
<th>Example Sentence</th>
<th>Example Test</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Substitution Test</td>
<td>The dog barked loudly.</td>
<td>Replace ”the dog” with “it”.</td>
<td>It barked loudly (Grammatical)</td>
</tr>
<tr>
<td>Movement Test</td>
<td>The cat chased the mouse.</td>
<td>Move ”the mouse” to the beginning of the sentence.</td>
<td>The mouse was chased by the cat (Grammatical)</td>
</tr>
<tr>
<td>Coordination Test</td>
<td>John likes to read books and play sports.</td>
<td>Join ”to read books” and ”play sports” with ”and”.</td>
<td>John likes to read books and play sports (Grammatical)</td>
</tr>
<tr>
<td>Clefting Test</td>
<td>It was the cat that chased the mouse.</td>
<td>Divide the sentence into two parts: ”It was” and ”the cat that chased the mouse”.</td>
<td>Both parts are grammatical.</td>
</tr>
</tbody>
</table>

*Table 3.1: Examples of Constituency Tests*

### 3.2 Unsupervised Parsing via Constituency Tests

The paper proposes an unsupervised approach to constituency parsing, which aims to automatically learn a parse tree structure that represents the underlying grammatical structure of a given sentence. Constituency parsing is a fundamental task in natural language processing, as it enables a range of downstream applications, such as machine translation, text generation, and sentiment analysis.

The proposed approach is based on the use of constituency tests, which are a set of linguistic phenomena that can be used to test whether a proposed parse tree is a valid constituent structure for a sentence. The paper provides examples of several types of constituency tests, including substitution, movement, coordination, and clefting. For example, the substitution test involves replacing a proposed constituent with a pronoun to see if the sentence remains grammatical. If the sentence remains grammatical, it suggests that the proposed constituent is a valid constituent structure.

The unsupervised parsing method consists of two stages: (1) generating candidate parse trees from raw text using a probabilistic model, and (2) ranking the candidate parse trees
Figure 3.1: Step-by-step approach to parsing via constituency tests

based on how well they pass a set of constituency tests. The authors use a generative model called the probabilistic context-free grammar (PCFG) to generate parse trees from raw text. The PCFG is a probabilistic model that assigns probabilities to each rule in the grammar, based on the frequency of that rule in a corpus of training data.

Once candidate parse trees are generated, they are evaluated using a set of hand-designed constituency tests. The authors propose a measure called the constituency score, which is based on the number of constituency tests that a parse tree passes. The authors also propose a variant of the constituency score, called the length-weighted constituency score, which gives higher weight to longer constituents. The length-weighted constituency score takes into account the intuition that longer constituents are more likely to be valid constituent structures.

The authors evaluate their approach on a number of benchmark datasets, including the Penn Treebank and show that it achieves competitive performance compared to other unsupervised parsing methods including the Deep Inside-Outside Recursive Autoencoder
(DIORA), the Ordered Neurons (ON), and the Unsupervised Recurrent Neural Network Grammar (URNNG). The results show that the proposed approach achieves state-of-the-art performance of 62.8 F1. The approach is notable for its simplicity and flexibility, as it can be easily adapted to different languages and test sets.
CHAPTER 4
SCORING SPANS USING GPT-2

GPT-2[7] uses a transformer architecture that consists of multiple layers of self-attention mechanisms, which allows the model to capture long-range dependencies between words in a sentence. The self-attention mechanism allows the model to attend to different parts of the input sentence and give more weight to the words that are more relevant to the prediction of the next word. The model is trained on the objective of generating the next token given the previous sequence of words. This ability can be leveraged to generate text that is syntactically coherent.

In this approach, the span-scoring mechanism in the constituency-test based approach is replaced by GPT-2. Leveraging GPT-2’s ability of generating syntactically coherent text, we treat each span as a complete sentence and the score assigned to the span will be the probability score of the sentence assigned to it by the model. The hypothesis behind this scoring system is that the spans that are constituents will appear as “complete sentences” to the model and will naturally be scored higher that spans that are distituents.

Figure 4.1: Constituency parsing using GPT-2

```
The man ate his food
The man ate his food
The man ate his food
The man ate his food
The man ate his food

\* \* \* \* \*

GPT-2

P("the man")
P("man ate")
P("ate his")
P("his food")
P("the man ate")
P("man ate his food")

CKY

($) (the man) (ate (his food))
```
Figure 4.2: Using sentence probability to score spans

As shown in Figure 3.2, each span is assigned a score and the CKY algorithm is applied to build a tree with maximum score. However, this approach yields a low F1 score of 10.67 which means that longer sequences might also be assigned a higher score and simply verifying "coherency" does not guarantee that constituents will be score higher than distituents.

Table 4.1: Results for constituency parsing using GPT-2

<table>
<thead>
<tr>
<th>Model</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-2</td>
<td>10.67</td>
</tr>
</tbody>
</table>
CHAPTER 5
SCORING SPANS BY PROMPTING T0++

In the next approaches, we try to prompt LLMs for the task of constituency parsing. Prompting involves providing the model with a starting sentence or phrase, which it then continues to generate additional text. This can be used to generate text in a specific style or on a specific topic, or to complete a sentence or paragraph. By providing a specific prompt, the model is directed to focus on a specific task or objective, which can improve its accuracy and effectiveness. This is especially important in complex NLP tasks, where the model may otherwise struggle to identify the relevant information. Prompting can also provide additional context for the model to work with, which can improve its understanding of the input text. For example, providing a starting sentence or phrase can help the model generate text that is more coherent and relevant to the topic. It can be used to provide targeted feedback to the model, which can help improve its performance over time. For example, in interactive NLP applications, users can provide feedback on the model’s responses, which can be used to train the model to provide more accurate or relevant outputs.

The T0pp model is designed to be a general-purpose language model that can be
fine-tuned for a wide range of NLP tasks, including text classification, machine translation, question answering, and summarization, among others. The model can take in a variety of inputs, such as text, structured data, and even images, and generate outputs in a variety of formats, such as text, structured data, or programs. It is capable of generalizing well to new NLP tasks because it has been pre-trained on a large corpus of text data which allows it to capture the statistical patterns of the language. Our approach at prompting this model is to check whether it has enough context to decide whether a given span is a constituent or not. We try prompting in a zero shot and few shot setting with 4 different types of prompt format.

Using the prompt, we try to score each span by assigning it the conditional probability $P(\text{"yes" | span})$. After this, similar to the previous approach, we apply CKY again to generate the parse tree of the sentence. As seen in figure 4.1, we first prompt the model in a zero shot setting. In the first prompt, we directly ask the language model whether a given span is a constituent or not. In the second prompt, we decide to go one level deeper to ask the model whether the given span is a noun, verb or prepositional phrase. As seen in table 4.1, both these prompt perform poorly and give F1 score of 21.99 and 23.72 each.

**Prompt 3**
Q: Is the following span a grammatical constituent: [children ate the cake with a]?  
A: No  
Q: Is the following span a grammatical constituent: [ate the cake with a spoon]?  
A: Yes  
Q: Is the following span a grammatical constituent: [{ }]?  
A: Yes or No

**Prompt 4**
Sentence: the dog saw the cat  
Constituents: the, dog, saw, cat, the dog, the cat, saw the cat, the dog saw the cat  
Not constituents: dog saw, saw the, dog saw the, the dog saw  
Is the following span a constituent? Respond Yes or No  
Span: {}  

*Figure 5.2: T0pp prompt format*

Naturally, the next step in the process was to check whether providing some in-context examples to the model improves its performance. In prompt 3 and prompt 4, we try to prompt T0pp in a one-shot and two-shot setting. In the first prompt, we give the model an
example of a constituent and a distituent and then ask whether the given span is a grammatical constituent or not. In the second prompt, we provide an example sentence to the model and list all the constituents and distituents and then ask the model whether the given span is a constituent or not. However, it can be seen clearly in table 4.1 that both these prompts perform worse than the previous zero shot setting. This tells us that the in-context examples provided to the model are not enough for it to directly judge whether a given span of words is constituent or not. Table 4.1 shows us that prompt 2 that asks the model whether a given span is a noun, verb, prepositional phrase or not, gives the best performance.

<table>
<thead>
<tr>
<th>Model-T0pp</th>
<th>F1@k=0</th>
<th>F1@k=1</th>
<th>F1@k=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstructGPT</td>
<td>42.86</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

*Table 5.1: Results for prompting using T0pp*
CHAPTER 6
PROMPTING INSTRUCTGPT

Figure 6.1: Prompting InstructGPT: Asking for direct parse

OpenAI InstructGPT [10] is a large-scale generative language model developed by OpenAI, based on the GPT-3 architecture. Unlike traditional language models, which are trained on large amounts of text data to predict the next word in a sequence, InstructGPT is trained to generate instructions for a wide range of tasks, such as cooking, assembling furniture, or using software applications. The training process for InstructGPT involves feeding the model pairs of inputs and corresponding instructions, with the goal of having the model generate accurate and helpful instructions given a particular input. For example, the model might be trained on pairs of images of furniture and corresponding assembly instructions, or on pairs of software commands and corresponding instructions for how to use those commands. In this approach, we prompt InstructGPT by asking it for the direct parse. As it can be seen in figure 6.1, the in-context examples provided in the prompt is the sentence and its corresponding parse tree. This experiment was carried out in a 1-shot, 2-shot, 5-shot and 10-shot setting. The examples included in the prompt are hand-picked and contains sentences of varying length. The experiment was carried out with a different

Prompt format

Sentence: The finger-pointing has already begun.
Parse tree: (TOP (S (NP (DT The) (JJ finger-pointing)) (VP (VBZ has) (ADVP (RB already)) (VP (VBN begun))) (. .)))
.
.
Sentence: These stocks eventually reopened.
Parse tree: (TOP (S (NP (DT These) (NNS stocks)) (ADVP (RB eventually)) (VP (VBD reopened)) (. .)))
Sentence: The dog saw the cat.
Parse tree:
set of examples in each prompt and the best scores are reported in the below table. The results show that the model gives an F1-score of 45.002 for a 10-shot prompt.

<table>
<thead>
<tr>
<th>Model</th>
<th>F1@k=1</th>
<th>F1@k=2</th>
<th>F1@k=5</th>
<th>F1@k=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>InstructGPT</td>
<td>42.86</td>
<td>44.02</td>
<td>44.6</td>
<td><strong>45.002</strong></td>
</tr>
</tbody>
</table>

*Table 6.1: Results for prompting using InstructGPT*
CHAPTER 7
PROMPTING INSTRUCTGPT: CHAIN-OF-THOUGHT PROMPTING

Figure 7.1: Prompting InstructGPT: Chain-of-Thought Prompting

Chain-of-thought prompting is a technique used in natural language generation to generate coherent and contextually appropriate text based on a given prompt or topic. The approach involves breaking down the prompt into smaller, more specific subtopics or ideas, and generating text for each of these subtopics in a way that builds upon the previous ideas and leads to a coherent and well-structured narrative. It has a range of potential applications in areas such as natural language generation, text summarization, and question answering, where the ability to generate coherent and contextually appropriate responses is critical. One of the main benefits of this approach is its ability to generate text that is not only accurate and informative, but also engaging and easy to read, making it well-suited for a wide range of use cases and applications.

From the results obtained in the previous approach, it can be seen that InstructGPT performs well when asked for the direct parse of a sentence. Continuing on the line of Chain-of-Thought prompting, the idea is to give the model some context about how words are grouped together at each level in a bottom-up manner. This should give the model step-
by-step reasoning behind how constituency parsing is done. Both the prompts used in this experiments can be seen in figure 7.1 In the first prompt, the parse tree contains level by level grouping of words to form constituents. Each level is separated by a pipe symbol. The parse trees generated using this prompt could not be evaluated as they did not contain structure. The conclusion draw was that the examples provided to the model are too long and difficult to follow. To rectify this mistake, a variable subtree version of this prompt is created (prompt 2). Each span is encoded as a variable and the variable is substituted in the next level. The experiment conducted with this prompt gave an F1 score of 22.96 in a 2-shot setting and 26.26 in a 10-shot setting. This does not beat the previous approach of asking for direct parse tree.

<table>
<thead>
<tr>
<th>Model - InstructGPT</th>
<th>F1@k=2</th>
<th>F1@k=10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prompt 1</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Prompt 2</td>
<td>22.96</td>
<td>26.36</td>
</tr>
</tbody>
</table>

*Table 7.1: Results using Chain-of-Thought prompting*
CHAPTER 8
PROMPTING CODEX

OpenAI Codex is a powerful new language model created by OpenAI that is designed to enable developers to write code in natural language. Built on top of the GPT-3 language model, Codex is trained on a diverse range of codebases and programming languages, allowing it to understand and interpret a wide range of programming concepts and constructs. Some of the key features of Codex include the ability to auto-complete code, generate functions and classes, and provide context-aware suggestions for variable names and code snippets. The model is also able to understand and interpret programming concepts such as loops, conditionals, and data structures, allowing it to generate complex and sophisticated code with ease.

In this approach, we used the same prompt as the one used for InstructGPT (in the asking for direct parse setting) and obtained an F1-score of 67.97 which beats the current state-of-the-art. The experiment was also carried out with the same examples in the prompt but the trees provided were unlabelled and the model was asked to predict the unlabelled tree. However, this approach performs worse (63.002) than asking for direct parse. This is because the labels help the model apply reasoning while inferring the parse tree of the sentence.

<table>
<thead>
<tr>
<th>Model</th>
<th>same prompt</th>
<th>small sent</th>
<th>med sent</th>
<th>large sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codex (direct parse - labelled)</td>
<td>67.97</td>
<td>64.32</td>
<td>65.19</td>
<td>67.99</td>
</tr>
<tr>
<td>Codex (direct parse - unlabelled)</td>
<td>63.002</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Codex (CoT)</td>
<td>31.8</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

*Table 8.1: Results for prompting using Codex*

We also carried out experiments by varying the examples in the prompt. Table 8.1 shows that the model gives almost the same performance when large sentences are used.
We also tried evaluating Codex in the Chain-of-Thought (variable sub-tree) setting. Even if it performs better than InstructGPT, it is nowhere near beating the ”asking for direct parse tree” approach.
CHAPTER 9
CONCLUSION

In this research we try to use Large Language Models (LLMs) for the task of constituency parsing. We start with using GPT-2 to assign probability scores to each span and use these scores along with the CKY algorithm to generate parse trees. It can be seen that this approach fails and hence we move to prompting language models. We experiment with different types of prompts and language models and find out that the Codex model which is optimized for coding beats the state-of-the-art by 5 points. This suggests that large language models are aware about the underlying syntax of the language and since these parse trees appear like code, the Codex model gives the best performance.
REFERENCES


