Tetra-Tagging: Word-Synchronous Parsing with Linear-Time Inference

Nikita Kitaev and **Dan Klein**
Computer Science Division
University of California, Berkeley
{kitaev, klein}@cs.berkeley.edu

Abstract

We present a constituency parsing algorithm that, like a supertagger, works by assigning labels to each word in a sentence. In order to maximally leverage current neural architectures, the model scores each word's tags in parallel, with minimal task-specific structure. After scoring, a left-to-right reconciliation phase extracts a tree in (empirically) linear time. Our parser achieves 95.4 F1 on the WSJ test set while also achieving substantial speedups compared to current state-of-the-art parsers with comparable accuracies.

1 Introduction

Recent progress in NLP, and practical machine learning applications more generally, has been driven in large part by increasing availability of compute. These advances are made possible by an ecosystem of specialized hardware accelerators such as GPUs and TPUs, highly tuned kernels for executing particular operations, and the ability to amortize computational costs across tasks through approaches such as pre-training and multi-task learning. This places particular demands for a model to be efficient: it must parallelize, it must maximally use standard subcomponents that have been heavily optimized, but at the same time it must adequately incorporate task-specific insights and inductive biases.

Against this backdrop, constituency parsing stands as a task where custom architectures are prevalent and parallel execution is limited. State-of-the-art approaches use custom architecture components, such as the tree-structured networks of RNNG (Dyer et al., 2016) or the per-span MLPs in chart parsers (Stern et al., 2017; Kitaev et al., 2019). Approaches to inference range from autoregressive generation, to cubic-time CKY, to A* search – none of which are readily parallelizable. Our goal is to demonstrate a parsing algorithm that makes effective use of the latest hardware. The desiderata for our approach are *(a)* to maximize parallelism, *(b)* to minimize task-specific architecture design, and *(c)* to lose as little accuracy as possible compared to a state-of-the-art highly-specialized model. To do this, we propose an algorithm that reduces parsing to tagging, where all tags are predicted in parallel using a standard model architecture such as BERT (Devlin et al., 2019). Tagging is followed by a minimal inference procedure that is fast enough to schedule on the CPU because it runs in linear time with low constant factors (subject to mild assumptions).

2 Related Work

Label-based parsing A variety of approaches have been proposed to mostly or entirely reduce parsing to a sequence labeling task. One family of these approaches is *supertagging* (Bangalore and Joshi, 1999), which is particularly common for CCG parsing. CCG imposes constraints on which supertags may form a valid derivation, necessitating complex search procedures for finding a high-scoring sequence of supertags that is self-consistent. An example of how such a search procedure can be implemented is the system of Lee et al. (2016), which uses A* search. This search procedure is not easily parallelizable on GPU-like hardware, and has a worst-case serial running time that is exponential in the sentence length. Gómez-Rodríguez and Vilares (2018) propose a different approach that fully reduces parsing to sequence labeling, but the label set size is unbounded: it expands with tree depth and related properties of the input, rather than being fixed for any given language. There have been attempts to address this by adding redundant labels, where the model learns to switch between tagging schemes in an attempt to avoid the problem of unseen labels (Vilares et al., 2019), but that only increases the label inventory rather than restricting it to a finite set. Our approach, on the other hand, uses just 4 labels in its simplest formulation (hence the name *tetra-tagging*).

Shift-reduce transition systems A number of parsers proposed in the literature can be categorized as *shift-reduce* parsers (Henderson, 2003; Sagae and Lavie, 2005; Zhang and Clark, 2009; Zhu et al., 2013). These systems rely on generating sequences of actions, which need not be evenly distributed throughout the sentence. For example, the construction of a deep right-branching tree might involve a series of *shift* actions (one per word in the sentence), followed by equally many consecutive *reduce* actions that all cluster at the end of the sentence. Due to the uneven alignment between actions and locations in a sentence, neural network architectures in recent shift-reduce systems (Vinyals et al., 2015; Dyer et al., 2016; Liu and Zhang, 2017) generally follow an encoder-decoder approach with autoregressive generation rather than directly assigning labels to positions in the input. Our proposed parser is also transition-based, but there are guaranteed to be exactly two decisions to make between one word and the next. This fixed alignment allows us to predict all actions in parallel rather than autoregressively.

Chart parsing Chart parsers fundamentally operate over *span-aligned* rather than *word-aligned* representations. For instance, the size of the chart in the CKY algorithm (Cocke, 1970; Kasami, 1966; Younger, 1967) is quadratic in the length of the sentence, and the algorithm itself has cubic running time. This is true for both classical methods and more recent neural approaches (Durrett and Klein, 2015; Stern et al., 2017). The construction of a chart involves a non-trivial (quadratic) computation that is specialized to parsing, and implementing the CKY algorithm on a hardware accelerator is a nontrivial and hardware-specific task.

Left-corner parsing To achieve all of our desiderata, we combine aspects of the previously-mentioned approaches with ideas drawn from a long line of work on left-corner parsing (Rosenkrantz and Lewis, 1970; Nijholt, 1979; van Schijndel et al., 2013; Noji et al., 2016; Shain et al., 2016, *inter alia*). Much of past work highlights the benefits of a left-corner formulation for *memory efficiency*, with implications for psycholinguistic plausibility of the approach. We, on the other hand, demonstrate how to leverage these same considerations to achieve parallel tagging and linear *time complexity* of the subsequent inference procedure. Further, past work has used grammars (Rosenkrantz and Lewis, 1970), or transformed labeled trees (Johnson, 1998; Schuler et al., 2010). On the other hand, it is precisely the lack of an explicit grammar that allows us to formulate our linear-time inference algorithm.

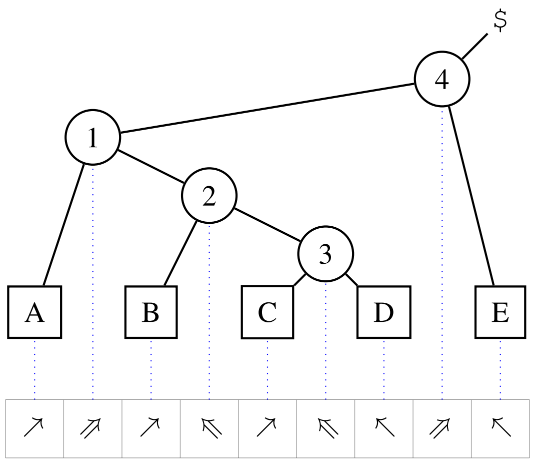


Figure 1: An example tree with the corresponding labels. The nonterminal nodes have been numbered based on an in-order traversal.

3 Method

To introduce our method, we first restrict ourselves to only consider unlabeled full binary trees (where every node has either 0 or 2 children). We defer the discussion of labeling and non-binary structure to Section 3.5.

3.1 Trees to tags

Consider the example tree shown in Figure 1. The tree is fully binarized and consists of 5 terminal symbols (A,B,C,D,E) and 4 nonterminal nodes (1,2,3,4). For any full binary parse tree, the number of nonterminals will always be one less than the number of words, so we can construct a one-to-one mapping between nonterminals and fenceposts (i.e. positions between words): each fencepost is matched with the shortest span that crosses it.

For each node, we calculate the *direction of its parent*, i.e. whether the node is a left-child or a right-child. Although the root node in the tree does not have a parent, by convention we treat it as though it were a left-child (in Figure 1, this is denoted by the dummy parent labeled $).

Our scheme associates each word and fencepost in the sentence with one of four labels:

- "↗": This terminal node is a left-child.
- "↖": This terminal node is a right-child.
- "⇗": The shortest span crossing this fencepost is a left-child.
- "⇖": The shortest span crossing this fencepost is a right-child.

We refer to our method as **tetra-tagging** because it uses only these four labels to represent binary bracketing structure.

3.2 Model

Given a sentence with n words, there are altogether $2n - 1$ decisions (each with two options). By the construction above, it is evident that every tree has one (and only one) corresponding label representation. To reduce parsing to tagging, we simply use a neural network to predict which tag to select for each of the $2n - 1$ decisions required.

Our implementation predicts these tag sequences from pre-trained BERT word representations. Two independent projection matrices are applied to the feature vector for the last sub-word unit within each word: one projection produces scores for actions corresponding to that word, and the other for actions at the following fencepost. A softmax loss is applied, and the model is trained to maximize the likelihood of the correct action sequence.

3.3 Tags to trees: transition system

To map from label sequences back to trees, we reinterpret the four labels ("↗", "↖", "⇗", "⇖") as actions in a left-corner transition system. The transition system maintains a *stack* of partially-constructed trees, where each element of the stack is one of the following: (a) a terminal symbol, i.e. a word; (b) a complete tree; or (c) a tree with a single empty slot, denoted by the special element ∅. An empty slot must be the rightmost leaf node in its tree, but may occur at any depth.

The tree operations used are: *(a)* MAKE-NODE(*left-child*, *right-child*), which creates a new tree node; and *(b)* COMBINE(*parent-tree*, *child-tree*), which replaces the empty slot ∅ in the parent tree with the child tree.

Decoding uses Algorithm 1; an example derivation is shown in Figure 2.

Algorithm 1 Decoding algorithm

```
Input: A list of words (words) and a corresponding list of
       tetra-tags (actions)
Output: A parse tree
stack ← []
buffer ← words
for action in actions do
    switch action do
        case "↗"
            leaf ← POP-FIRST(buffer)
            stack ← PUSH-LAST(stack, leaf)
        end case
        case "↖"
            leaf ← POP-FIRST(buffer)
            stack[−1] ← COMBINE(stack[−1], leaf)
        end case
        case "⇗"
            stack[−1] ← MAKE-NODE(stack[−1], ∅)
        end case
        case "⇖"
            tree ← POP-LAST(stack)
            tree ← MAKE-NODE(tree, ∅)
            stack[−1] ← COMBINE(stack[−1], tree)
        end case
    end switch
end for        ▷ The stack should only have one element
return stack[0]
```

Each action in the transition system is responsible for adding a single tree node onto the stack: the actions "↗" and "↖" do this by shifting in a leaf node, while the actions "⇗" and "⇖" construct a new non-terminal node. The transition system maintains the invariant that the topmost stack element is a complete tree if and only if a leaf node was just shifted (i.e. the last action was either "↗" or "↖"), and all other stack elements have a single empty slot.

The actions "↖" and "⇖" both make use of the COMBINE operation to fill an empty slot on the stack with a newly-introduced node, which makes the new node a right-child. New nodes from the actions "↗" and "⇗", on the other hand, are introduced directly onto the stack and can become left-children via a later MAKE-NODE operation. As a result, the behavior of the four actions ("↗", "↖", "⇗", "⇖") matches the label definitions from the previous section.

3.4 Inference

The goal of inference is to select the sequence of labels that is assigned the highest probability by the tagging model. It should be noted that not all sequences of labels are valid under our transition system. In particular:

- The first action must be "↗", because the stack is initially empty and the only valid action is to shift the first word in the sentence from the buffer onto the stack.

- The action "⇖" relies on there being more than one element on the stack (lines 17-19 of Algorithm 1).
- After executing all actions, the stack should contain a single element. Due to the invariant that the top stack element after a "↗" or "↖" action is always a tree with no empty slots, this single stack element is guaranteed to be a complete tree that spans the full sentence.

We observe that the validity constraints for our transition system can be expressed entirely in terms of the *number* of stack elements at each point in the derivation, and do not depend on the precise structure of those elements. This property enables an optimal and efficient dynamic program for finding the valid sequence of labels that has the highest probability under the model.

The dynamic program maintains a table of the highest-scoring parser state for each combination of *number of actions taken* and *stack depth*. Prior to taking any actions, the stack must be empty. The algorithm then proceeds left-to-right through the sentence to fill in highest-scoring stack configurations after action 1, 2, etc. The dynamic program can be visualized as finding the shortest path through a graph like Figure 3, where each action-count/stack-depth combination is represented by a node, and a transition is represented by an edge with weight equal to the model-predicted score of the associated tag.

The time complexity of this dynamic program depends on the number of actions (which is $2n - 1$, where n is the length of the sentence), as well as the maximum possible depth of the stack (d). A left-corner transition system has the property that stack depth tends to be small for parse trees of natural language (Abney and Johnson, 1991; Schuler et al., 2010). In practice, the largest stack depth observed at any point in the derivation for any tree in the Penn Treebank is 8. By comparison, the median sentence length in the data is 23, and the longest sentence contains over 100 words.

As a result, we can cap the maximum stack depth allowed in our inference procedure to $d = 8$, which means that the $O(nd^2)$ time complexity of inference is effectively $O(n)$. In other words, our inference procedure will, in practice, take linear time in the length of the sentence.

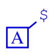
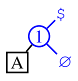
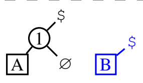
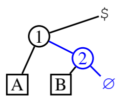
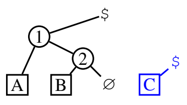
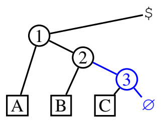
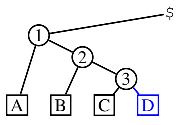
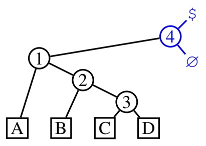
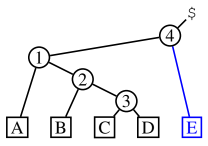
Action	Stack	Buffer
(0)	*empty*	A B C D E
(1) ↗		B C D E
(2) ⇗		B C D E
(3) ↗		C D E
(4) ⇖		C D E
(5) ↗		D E
(6) ⇖		D E
(7) ↖		E
(8) ⇗		E
(9) ↖		*empty*

Figure 2: An example derivation under our transition system.

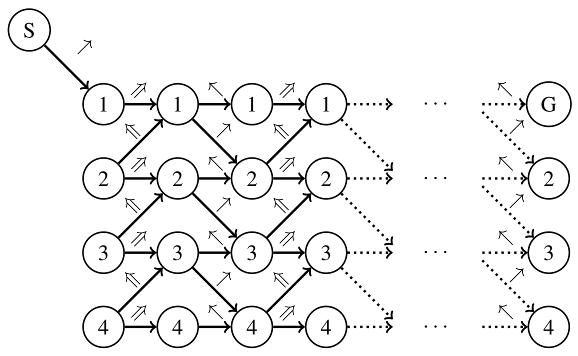


Figure 3: Paths in this grid correspond to sequences of tags, where paths starting at S and arriving at G are valid trees. Numbers represent the number of elements on the stack.

	Sents/s	Hardware	F1
Vilares et al. (2019)	942	1x GPU	91.13
Kitaev et al. (2019)*	39	1x GPU	95.59
Zhou and Zhao (2019)*	–	–	95.84
This work*	1200	1x TPU v3-8	95.44

Table 1: Comparison of F1 scores and inference speeds on the WSJ test set. *Models using BERT$_{\text{LARGE}}$ (Devlin et al., 2019) word representations fine-tuned from the same initial parameters.

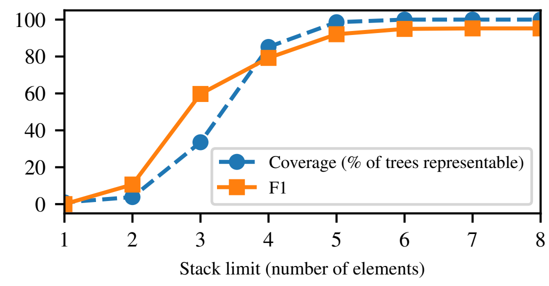


Figure 4: With a modest maximum stack size, the tetra-tagging transition system has near-complete coverage of the development data. Our parser's F1 score closely tracks the fraction of gold trees that can be represented.

3.5 Handling of labels and non-binary trees

Each of our four actions creates a single node in the binary tree. Labeling a node can therefore be incorporated into the corresponding action; for example, the action "$\nearrow$ S" will construct an S node that is a left-child in the tree. We do not impose any constraints on valid label configurations, so our inference procedure remains virtually unchanged.

To handle non-binary trees, we first collapse all unary chains by introducing additional labels. For example, a clause that consists only of a verb phrase would be assigned the label S-VP. We then ensure that each non-terminal node has exactly two children by applying fully right-branching binarization, where a dummy label is introduced and assigned to nodes generated as a result of binarization. During inference, a post-processing step undoes these transformations.

4 Results

Our proposed parser is designed to rank syntactic decisions entirely in parallel, with inference reduced to a minimal linear-time algorithm. Its neural architecture consists almost entirely of BERT layers, with the only additions being two trainable projection matrices. To verify our approach, we train our parser on the Penn Treebank (Marcus et al., 1993) and evaluate its efficiency and accuracy when running on Cloud TPU v3 hardware.

In Table 1, we compare with two classes of recent work. The parser by Vilares et al. (2019) is one of the fastest reported in the recent literature, but it trails the state-of-the-art model by more than 4 F1 points. In contrast, models by Zhou and Zhao (2019) and Kitaev et al. (2019) achieve the highest-reported numbers when fine-tuning from the same initial BERT$_{\text{LARGE}}$ checkpoint that we use to train our tetra-tagger. However, these latter models are slower than our tetra-tagging approach and feature inference algorithms with high polynomial complexity that are difficult to adapt to accelerators such as the TPU. Our approach is able to achieve both high throughput and high F1, with only small losses in accuracy compared to the best BERT-based approaches.

In Figure 4, we plot the parser's accuracy across different settings for the maximum stack depth. The F1 score rapidly asymptotes as the stack size limit is increased, which validates our claim that inference can run in linear time.

5 Conclusion

We present a reduction from constituency parsing to a tagging task with two binary structural decisions and two labeling decisions per word. Remarkably, probabilities for these tags can be estimated fully in parallel by a simple classification layer on top of a neural network architecture such as BERT. We hope that this formulation can be useful as a simple and low-overhead way of integrating syntax into any neural NLP model, including for multi-task training and to predict syntactic annotations during inference. By reducing the task-specific architecture components to a minimum, our method can be rapidly adapted as new modeling techniques, efficiency optimizations, and hardware accelerators become available. Code for our approach is available at `github.com/nikitakit/tetra-tagging`.

Acknowledgments

This research was supported by DARPA through the XAI program and by the National Science Foundation under Grant No. 1618460. We would like to thank the Google Cloud TPU team for their hardware support. We are also grateful to the members of the Berkeley NLP group and the anonymous reviewers for their helpful feedback.

References

Steven P. Abney and Mark Johnson. 1991. Memory requirements and local ambiguities of parsing strategies. *Journal of Psycholinguistic Research*, 20(3):233–250.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

John Cocke. 1970. Programming languages and their compilers: Preliminary notes.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Greg Durrett and Dan Klein. 2015. Neural CRF parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 302–312, Beijing, China. Association for Computational Linguistics.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209, San Diego, California. Association for Computational Linguistics.

Carlos Gómez-Rodríguez and David Vilares. 2018. Constituent parsing as sequence labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1314–1324, Brussels, Belgium. Association for Computational Linguistics.

James Henderson. 2003. Inducing history representations for broad coverage statistical parsing. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 103–110.

Mark Johnson. 1998. Finite-state approximation of constraint-based grammars using left-corner grammar transforms. In *COLING 1998 Volume 1: The 17th International Conference on Computational Linguistics*.

Tadao Kasami. 1966. An efficient recognition and syntax-analysis algorithm for context-free languages. *Coordinated Science Laboratory Report no. R-257*.

Nikita Kitaev, Steven Cao, and Dan Klein. 2019. Multilingual constituency parsing with self-attention and pre-training. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3499–3505, Florence, Italy. Association for Computational Linguistics.

Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2016. Global neural CCG parsing with optimality guarantees. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2366–2376, Austin, Texas. Association for Computational Linguistics.

Jiangming Liu and Yue Zhang. 2017. In-order transition-based constituent parsing. *Transactions of the Association for Computational Linguistics*, 5:413–424.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Anton Nijholt. 1979. Structure preserving transformations on non-left-recursive grammars. In *International Colloquium on Automata, Languages, and Programming*, pages 446–459. Springer.

Hiroshi Noji, Yusuke Miyao, and Mark Johnson. 2016. Using left-corner parsing to encode universal structural constraints in grammar induction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 33–43, Austin, Texas. Association for Computational Linguistics.

Daniel J. Rosenkrantz and Philip M. Lewis. 1970. Deterministic left corner parsing. In *Switching and Automata Theory, 1970., IEEE Conference Record of 11th Annual Symposium On*, pages 139–152. IEEE.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 125–132, Vancouver, British Columbia. Association for Computational Linguistics.

William Schuler, Samir AbdelRahman, Tim Miller, and Lane Schwartz. 2010. Broad-coverage parsing using human-like memory constraints. *Computational Linguistics*, 36(1):1–30.

Cory Shain, William Bryce, Lifeng Jin, Victoria Krakovna, Finale Doshi-Velez, Timothy Miller, William Schuler, and Lane Schwartz. 2016. Memory-bounded left-corner unsupervised grammar induction on child-directed input. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 964–975, Osaka, Japan. The COLING 2016 Organizing Committee.

Mitchell Stern, Jacob Andreas, and Dan Klein. 2017. A minimal span-based neural constituency parser. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 818–827, Vancouver, Canada. Association for Computational Linguistics.

Marten van Schijndel, Andy Exley, and William Schuler. 2013. A model of language processing as hierarchic sequential prediction. *Topics in Cognitive Science*, 5(3):522–540.

David Vilares, Mostafa Abdou, and Anders Søgaard. 2019. Better, faster, stronger sequence tagging constituent parsers. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3372–3383, Minneapolis, Minnesota. Association for Computational Linguistics.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a Foreign Language. In *Advances in Neural Information Processing Systems 28*, pages 2755–2763. Curran Associates, Inc.

Daniel H. Younger. 1967. Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208.

Yue Zhang and Stephen Clark. 2009. Transition-based parsing of the Chinese treebank using a global discriminative model. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 162–171, Paris, France. Association for Computational Linguistics.

Junru Zhou and Hai Zhao. 2019. Head-driven phrase structure grammar parsing on Penn treebank. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2396–2408, Florence, Italy. Association for Computational Linguistics.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. 2013. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 434–443, Sofia, Bulgaria. Association for Computational Linguistics.