Dependency Transformer Grammars: Integrating Dependency Structures into Transformer Language Models

Anonymous ACL submission

Abstract

Syntactic Transformer language models aim to achieve better generalization through simultaneously modeling syntax trees and sentences. While prior work has been focusing on adding constituency-based structures to Transformers, we introduce Dependency Transformer Grammars (DTGs), a new class of Transformer language model with explicit dependency-based inductive bias. DTGs simulate dependency transition systems with constrained attention patterns by modifying attention masks, incorporate the stack information through relative positional encoding, and augment dependency arc representation with a combination of token embeddings and operation embeddings. When trained on a dataset of sentences annotated with dependency trees, DTGs achieve better generalization while maintaining comparable perplexity with Transformer language model baselines. DTGs also outperform recent constituency-based models, showing that dependency can better guide Transformer language models. Our code will be publicly available upon acceptance.

1 Introduction

Transformer language models have shown strong performance on language modeling tasks and a broad spectrum of downstream tasks (Radford et al., 2019; Devlin et al., 2019; Brown et al., 2020). Despite the great power of the Transformer architecture (Vaswani et al., 2017), it lacks the inductive biases of syntactic structures, which has been hypothesized to improve generalization (Everaert et al., 2015). A straightforward way to incorporate such biases into Transformers is explicit modeling of syntactic structures.

Inspired by earlier work of generative parsing as language modeling that integrates syntactic structures into RNNs (Dyer et al., 2016; Choe and Charniak, 2016), recent studies have focused on adapting this method to Transformer architectures (Qian et al., 2021; Yoshida and Oseki, 2022; Sartran et al., 2022; Murty et al., 2023). The models proposed by these studies are categorized as syntactic language models because they jointly model the distribution of surface strings and their corresponding syntactic trees. Experiments show that these models achieve competitive perplexity in language modeling and gain better syntactic generalization, supporting the above hypothesis on the benefits of introducing inductive bias of syntactic structures. However, the structural supervision that has been used in all these models is based on constituency trees and it is unclear of the performance of dependency-based Transformer syntactic language models. Different from constituency structures, which model recursive syntactic compositions, dependency structures focus more on the relationship between tokens, which is similar to the self-attention mechanism in Transformer, hinting at potential synergy between the two.

In this paper, we propose Dependency Transformer Grammars (DTGs), dependency-based syntactic language models that learn joint distributions of sentences and dependency trees. DTGs introduce an inductive bias of dependency structures to Transformers by (i) modeling transition sequences of transition-based dependency parsers instead of sentences, (ii) simulating the stack operations in transition-based dependency parsers through modification of attention masks, (iii) incorporating the stack information of transition-based systems through relative positional encoding of stack depth, and (iv) representing head-dependent relations through a combination of head token embeddings and transition operation embeddings. Following a line of previous work in generative dependency parsing (Titov and Henderson, 2007; Cohen et al., 2011; Buys and Blunsom, 2015), the generative formulation of our model is based on the *arc-standard* system (Nivre, 2004), which builds a dependency tree in a bottom-up manner. We also explore models using other dependency transition systems for comparison.

Our experiments show that DTGs achieve comparable perplexity in language modeling and improved syntactic generalization on both the BLiMP benchmark (Warstadt et al., 2020) and the SG test suites (Hu et al., 2020) over Transformer language model baselines. Furthermore, DTGs outperform constituency-based syntactic language models in both language modeling and syntactic generalization.

In summary, our contributions are as follows.

- We propose dependency-based syntactic language models, DTGs, to incorporate dependency inductive bias into Transformers.
- We primarily build DTGs using the *arc-standard* transition system, while we also study the usage of other dependency transition systems.
- Experimental results on two syntactic generalization benchmarks show the benefits of introducing inductive bias of dependency structures.

2 Preliminaries: Transition-based Dependency Parsing

Given a sentence, transition-based dependency parsing predicts a sequence of predefined transitions between states that incrementally build a dependency parse tree. A state contains a *stack* σ with token i on the top, a *buffer* β with j at its leftmost side, and a set A of dependency arcs, denoted as $(\sigma|i, j|\beta, A)$.

In this work, we focus on unlabeled projective dependency parsing for the simplicity of its transition systems. There are several different transition systems for projective dependency parsing, as shown in Table 1. *Arc-standard* (Nivre, 2004) is a widely used transition system that defines three transitions: SHIFT, LEFTARC and RIGHTARC. *Arc-standard* builds dependency trees in a bottom-up manner, that is, every token is not connected to its head token until it gathers all of its dependents. *Arc-eager* (Nivre, 2003) is another transition system that adds one more transition: POP. The main difference between *arc-standard* and *arc-eager* lies in the scope of arcs. *Arc-standard* only allows inducing arcs in the stack while *arc-eager* eases the restriction by defining arc transitions between the stack and the buffer. As a result, dependency trees are no longer built from bottom to up in *arc-eager*. A later system *arc-hybrid* (Kuhlmann et al., 2011) combines LEFTARC in *arc-eager* and RIGHTARC in *arc-standard*. Another more recent system *arc-swift* (Qi and Manning, 2017) extends arc-inducing to non-local cases: transition LEFTARC/RIGHTARC[k] in *arc-swift* can be seen as $k-1$ POP operations followed by one arc-inducing in *arc-eager*.

The above dependency parsing transition systems can be changed into a generative form, such that they generate sentences along with their associated dependency trees. The main change to the transition systems is that tokens need to be generated instead of being shifted from the buffer. Specifically, in *arc-standard* we substitute SHIFT with a token generation transition GEN, while retaining the other transitions (Titov and Henderson, 2007; Cohen et al., 2011; Buys and Blunsom, 2015). Other systems require additional efforts to obtain a generative form because they contain the usage of the buffer head in LEFTARC and/or RIGHTARC before shifting it to the stack. Simply replacing SHIFT with GEN cannot ensure the existence of the two tokens involved in a newly generated arc. Therefore, we need to insert a GEN' transition,[1] which generates a new token but puts it in the buffer, before any LEFTARC/RIGHTARC transition that involves an ungenerated token. The SHIFT transitions are omitted because any generated token will be shifted to the stack once a new token is generated.

We can use an oracle to extract a transition sequence from a dependency parse tree: An arc-inducing transition is generated whenever possible, and a POP transition (in *arc-eager*) is generated when it is impossible to generate other transitions, i.e., the transition preference order is LEFTARC/RIGHTARC > GEN > POP.

3 Model

DTG follows the generative form of the *arc-standard* dependency transition system and generates a sequence of transitions that construct a sentence $\mathbf{x}$ and its dependency tree $\mathbf{y}$ incrementally. The sequence consists of three types of transitions:

- **GEN**(x): generating a token, which corresponds to the GEN operation in generative *arc-*

[1] To simplify, we will refer to GEN' as GEN, which can be distinguished according to transition systems.

arc-standard	*arc-hybrid*
Shift $(\sigma, i\vert\beta, A) \Rightarrow (\sigma\vert i, \beta, A)$	**Shift** $(\sigma, i\vert\beta, A) \Rightarrow (\sigma\vert i, \beta, A)$
LArc $(\sigma\vert i\vert j, \beta, A) \Rightarrow (\sigma\vert j, \beta, A \cup \{(j \rightarrow i)\})$	**LArc** $(\sigma\vert i, j\vert\beta, A) \Rightarrow (\sigma, j\vert\beta, A \cup \{(j \rightarrow i)\})$
RArc $(\sigma\vert i\vert j, \beta, A) \Rightarrow (\sigma\vert i, \beta, A \cup \{(i \rightarrow j)\})$	**RArc** $(\sigma\vert i\vert j, \beta, A) \Rightarrow (\sigma\vert i, \beta, A \cup \{(i \rightarrow j)\})$
arc-eager	*arc-swift*
Shift $(\sigma, i\vert\beta, A) \Rightarrow (\sigma\vert i, \beta, A)$	**Shift** $(\sigma, i\vert\beta, A) \Rightarrow (\sigma\vert i, \beta, A)$
LArc $(\sigma\vert i, j\vert\beta, A) \Rightarrow (\sigma, j\vert\beta, A \cup \{(j \rightarrow i)\})$	**LArc**$[k]$ $(\sigma\vert i_k\vert \ldots \vert i_1, j\vert\beta, A) \Rightarrow (\sigma, j\vert\beta, A \cup \{(j \rightarrow i_k)\})$
RArc $(\sigma\vert i, j\vert\beta, A) \Rightarrow (\sigma\vert i\vert j, \beta, A \cup \{(i \rightarrow j)\})$	**RArc**$[k]$ $(\sigma\vert i_k\vert \ldots \vert i_1, j\vert\beta, A) \Rightarrow (\sigma\vert i_k\vert j, \beta, A \cup \{(i_k \rightarrow j)\})$
Pop $(\sigma\vert i, \beta, A) \Rightarrow (\sigma, \beta, A)$	

Table 1: Transitions defined by different transition systems (adapted from Qi and Manning (2017))

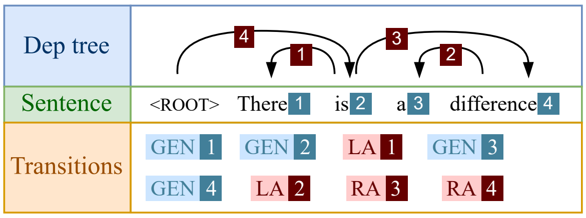


Figure 1: An example sentence with its dependency tree and transition sequence. Numbers in blue and red are indices of tokens and arcs respectively.

standard and is exactly what a standard Transformer decoder does at each step;

- **LEFTARC** or **LA**: inducing an arc from the most recent unconnected token (i.e., a token that has not been connected to its head) to the second most recent unconnected token, which corresponds to the LEFTARC operation in *arc-standard*;

- **RIGHTARC** or **RA**: inducing an arc from the second most recent unconnected token to the most recent unconnected token, which corresponds to the RIGHTARC operation in *arc-standard*.

An example is shown in Figure 1.

We write $\boldsymbol{\alpha}(\mathbf{x}, \mathbf{y}) = (\alpha_0, \alpha_1, ..., \alpha_{T-1})$ as the transition sequence of length T of sentence $\mathbf{x}$ and parse tree $\mathbf{y}$, where each α_t belongs to one of the three types mentioned above. DTG is a Transformer decoder that models the distribution of $\boldsymbol{\alpha}(\mathbf{x}, \mathbf{y})$ in the manner of causal language modeling, that is, $p(\boldsymbol{\alpha}(\mathbf{x}, \mathbf{y})) = \prod_i p(\alpha_i|\boldsymbol{\alpha}_{<i})$. It differs from a standard Transformer in several aspects in order to incorporate the dependency inductive bias, including attention masks, positional encoding, augmented representation of arcs, and constrained generation, which we discuss in the following subsections.

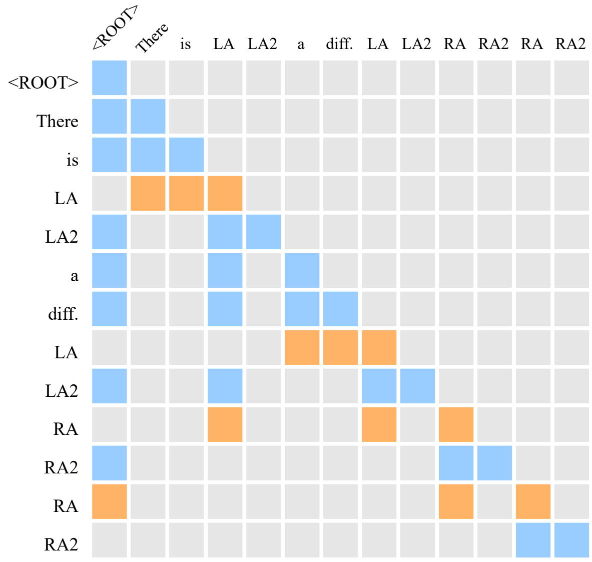
3.1 Arc-Standard via Attention Mask

DTGs generate the transition sequence autoregressively. A standard Transformer language model makes predictions based on the complete generation history. In contrast, to incorporate the dependency inductive bias into DTGs, we generate transitions based on the stack in *arc-standard*. The stack is encoded into the model with different attention forms and is updated by input transitions.

When a GEN transition comes, the transition system pushes a new token onto the stack and then gathers the stack information to generate the next transition, which we realize by the first attention form, **STACK** attention. When a transition changing the dependency structure comes, i.e., a LEFTARC/RIGHTARC transition, the stack is updated in two steps: (i) pop two tokens from the stack and designate one as the head of the other and (ii) push the head token back onto the stack. The two steps are realized by the second form of attention, **COMPOSE** attention, which updates the representation of the head by consuming its dependent but ignoring everything else to reflect the newly induced dependency arc. Then all the stack information is gathered for generating the next transition, which is again realized by **STACK** attention. Therefore, two forms of attention are required for one transition. As each transition can only use one form of attention in Transformer, we duplicate the arc transitions, namely LEFTARC/RIGHTARC and LEFTARC2/RIGHTARC2. The former encodes dependency information with **COMPOSE** attention and makes no generation, while the latter triggers the generation of the next transition with **STACK** attention. After the duplication, the sequence length increases from T to T'. We denote the new sequence as $\boldsymbol{\alpha}'$, which is the exact input sequence

i	Input	Attn. Mask	Prediction
0	<ROOT>	STACK	GEN(There)
1	There	STACK	GEN(is)
2	is	STACK	LEFTARC
3	LEFTARC + is	COMPOSE	-
4	LEFTARC2 + is	STACK	GEN(a)
5	a	STACK	GEN(difference)
6	difference	STACK	LEFTARC
7	LEFTARC + difference	COMPOSE	-
8	LEFTARC2 + difference	STACK	RIGHTARC
9	RIGHTARC + is	COMPOSE	-
10	RIGHTARC2 + is	STACK	RIGHTARC
11	RIGHTARC + <ROOT>	COMPOSE	-
12	RIGHTARC2 + <ROOT>	STACK	<END>

(a) Transition sequence after duplicating LEFTARC/RIGHTARC transitions. We do not have to make predictions for positions 3, 7, 9, 11.



(b) Attention mask. Tokens are simplified for a tight view. We use orange to represent **COMPOSE** and blue to represent **STACK**.

Figure 2: Transition sequence and attention masks of an example sentence

Algorithm 1 COMPOSE/STACK attention

Require: $\boldsymbol{\alpha}'$ sequence of transitions
Ensure: $\mathbf{A} \in \mathbb{R}^{T' \times T'}$ attention mask

$S \leftarrow []$ ▷ Empty stack
$\mathbf{A} \leftarrow 0$
for $i \leftarrow 0$ to T' **do**
  if type($\boldsymbol{\alpha}'[i]$) = LEFTARC or
   type($\boldsymbol{\alpha}'[i]$) = RIGHTARC **then** ▷ COMPOSE
   $A_{ii} \leftarrow 1$
   l$\leftarrow S.pop()$
   r$\leftarrow S.pop()$
   $A_{il} \leftarrow 1$
   $A_{ir} \leftarrow 1$
   $S.push(i)$ ▷ View transition i as the head token
  else ▷ STACK
   if type($\boldsymbol{\alpha}'[i]$) ≠ LEFTARC2 and
    type($\boldsymbol{\alpha}'[i]$) ≠ RIGHTARC2 **then**
    $S.push(i)$
   end if
   for $j \in S$ **do**
    $A_{ij} \leftarrow 1$
   end for
  end if
end for
return A ▷ Attention mask

of our model. Note that this does not change the distribution of $\boldsymbol{\alpha}(\mathbf{x}, \mathbf{y})$, as the generation sequence remains unchanged. An example of the expanded transition sequence and the corresponding attention forms is shown in Figure 2a.

The two forms of attention can be realized by leveraging different attention masks. We represent the attention masks as $\mathbf{A} \in \mathbb{R}^{T' \times T'}$, where $A_{ij} = 1$ means position j can be attended from i and $A_{ij} = 0$ means position j is masked from i. Our models generate transitions in an autoregressive manner, so the attention mask is causal, i.e., $A_{ij} = 0$ for $j > i$.

STACK attention is performed at each position i which needs to predict a new transition, i.e., $\alpha_i' \in \{\mathrm{GEN}(x), \mathrm{LEFTARC2}, \mathrm{RIGHTARC2}\}$. From position i, we attend to all the unmasked positions before i (including i) to collect all the information on the stack for generation.

COMPOSE attention is performed at each position i where $\alpha_i' \in \{\mathrm{LEFTARC}, \mathrm{RIGHTARC}\}$. From position i, we attend to the positions of the most recent two unmasked tokens, i.e., the top two tokens on the stack in *arc-standard*, which forms a head-dependent pair. Then we mask the two attended positions from subsequent positions, effectively popping the two tokens from the stack. The newly computed representation serves as a substitute for the head token that has absorbed the information of its dependent and is pushed back onto the stack.

Algorithm 1 shows how to compute attention masks for a transition sequence as described above. We also show attention masks of an example transition sequence in Figure 2b.

3.2 Relative Positional Encoding

We design the positional encoding for DTGs based on the relative positional encoding in Transformer-XL (Dai et al., 2019). In Transformer-XL, the positional encoding is based on the distance between the attending position i and the attended position j, i.e., $\mathbf{R}_{ij} = i - j$. In DTGs, we modify the formulation to reflect the stack information. Crucially, $\mathbf{R}_{ij}$ is only computed when $A_{ij} = 1$. For **STACK** attention, we define $d(i)$ as the depth in the stack, which increases from the top to the bottom. We then define $\mathbf{R}_{ij} = d(i) - d(j)$. For **COMPOSE** attention, we define two positions, 0 and -1, to distinguish between the head and the dependent to be composed, i.e., $\mathbf{R}_{ij} = 0$ if token j is the head token and $\mathbf{R}_{ij} = -1$ if token j is the dependent token. The new representation computed with **COMPOSE** inherits the depth of the head token, i.e., $d(i) = d(j)$ if token j is the composed head token.

3.3 Arc Representation

In standard language models, generated tokens are fed back into models as history. For arc-inducing transitions in DTGs, the generated transitions have surface forms of `LEFTARC` or `RIGHTARC` while the tokens ought to be pushed back are the head tokens. We propose to feed a combination of `LEFTARC`/`RIGHTARC` and the head token via summing the embedding of these two parts. This formulation stems from the following two considerations: (i) the attention in DTGs cannot distinguish between `LEFTARC` and `RIGHTARC`, so the embedding of `LEFTARC`/`RIGHTARC` acts as an indicator of the arc direction; (ii) the representation computed with **COMPOSE** is viewed as a substitute of the composed head token by subsequent positions, so we add the embedding of the head token to bias the representation.

3.4 Other Transition Systems via Attention Mask

We also design the attention mechanism for generative *arc-eager* and *arc-swift* and name the resulting models DTG-eager and DTG-swift. We do not work on generative *arc-hybrid* because its transition sequences are exactly the same as that of generative *arc-standard*.

For DTG-eager, we make two modifications based on DTG: (i) Change the **COMPOSE** attention of `RIGHTARC` by not masking the position of the dependent token because in *arc-eager*, the dependent token can still induce arcs to subsequent tokens. (ii) For transition `POP`, we define **POP-STACK** attention, which pops the stack top. The stack top is the second most recent unmasked token in most cases, and the most recent one is the head of the buffer. However, if all tokens have been generated and thus the buffer is empty, the stack top is the most recent unmasked token.

For DTG-swift, `LEFTARC` and `RIGHTARC` are decorated by an additional positive number k. This affects ranges of attending and masking in **COMPOSE** attention. That is, we attend to not only the head-dependent pair but also the $k - 1$ tokens between them, and we mask all these $k + 1$ tokens for subsequent positions.

More details and examples of these two models are provided in Appendix A.

3.5 Constraints on Inference

We define several constraints on transition generation during DTGs inference to make it consistent with the corresponding transition-based dependency parsing systems:

- For all the systems, the `LEFTARC` and `RIGHTARC` transition can only be generated if at least two tokens exist in the stack.
- For *arc-eager*, `POP` can only be generated if the top of the stack has been recognized as a right dependent of some head token.
- For *arc-swift*, the value of k in `LEFTARC`/`RIGHTARC`[k] must not exceed the size of the stack.

4 Experiments

We compare DTGs with DTG-eager, DTG-swift, two Transformer-XL baselines, and constituency-based syntactic Transformer language models. The two Transformer-XL baselines follow those of Sartran et al. (2022): (i) **TXL (tokens)** is a standard Transformer-XL that generates sentences only, and (ii) **TXL (trans)** is Transformer-XL that generates transition sequences just like DTG, but uses standard attention masks and positional encoding. Constituency-based syntactic Transformer language models include: (i) the "generative parsing as language modeling" of Qian et al. (2021) (**PLM**), (ii) Transformer Grammars of Sartran et al. (2022) (**TG**) and (iii) Pushdown Layers of Murty et al. (2023) (**Pushdown**).

Dataset and Preprocessing All the models are trained on the BLLIP-LG dataset of Charniak et al. (2000), with training splits from Hu et al. (2020). For our models, we obtain unlabeled projective

	Model	PPL (↓)	BLiMP (↑)	SG (↑)
Models without syntactic inductive bias				
	TXL (tokens)	14.8	75.3	76.6
Constituency-based models				
	PLM	29.8$^{\diamond}$	75.1	80.2
	TG	18.4$^{\clubsuit}$	73.5$^{\clubsuit}$	82.5
	Pushdown	19.9$^{\diamond}$	75.6	82.3
Dependency-based models				
	TXL (trans)	**14.4**	**77.3**	81.1
Ours	DTG-eager	15.5	75.2	-
Ours	DTG-swift	15.0	76.2	-
Ours	DTG	14.9	76.1	**83.9**

Table 2: Results of our models and baselines. ◇: Results are taken from prior work and are only for reference due to differences in tokenization. ♣: We rerun the code from the original work (Sartran et al., 2022) and obtain better perplexity than the reported result in it. All results for PLM and Pushdown are taken from Murty et al. (2023). The SG result for TG is taken from Sartran et al. (2022).

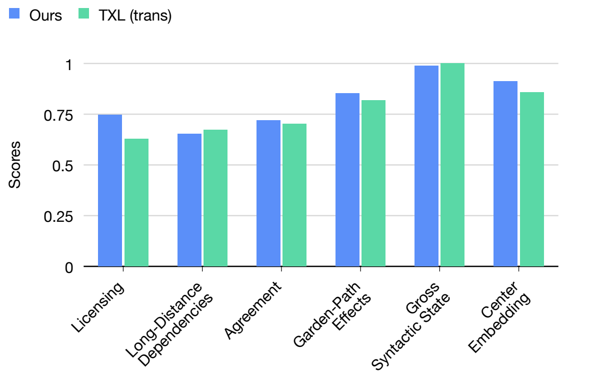


Figure 3: Scores on the six circuits of the SG test suites.

dependency trees by parsing the dataset with a Biaffine-roberta parser (Dozat and Manning, 2017) implemented in *Supar*[2]. Tokenization is performed with the same scheme as in Sartran et al. (2022) with SentencePiece (Kudo and Richardson, 2018). Note that we model each sentence independently in all the experiments.

Training Details We use the same hyperparameters as in Sartran et al. (2022) for training our models, using 16-layer models with 252M parameters. To accelerate the training of token embeddings, we add a multiplier of 2.0 to the learning rate of embedding weights. More details can be found in Appendix B.

4.1 Sentence-Level Language Modeling

Setup For syntactic language models that jointly model the distributions of sentences and syntactic trees, i.e., $p(\mathbf{x}, \mathbf{y})$, we compute the string probability $p(\mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{x}, \mathbf{y})$. It is impossible to compute $p(\mathbf{x})$ precisely due to the large space of all possible trees, so we follow Sartran et al. (2022) to approximate it using a relatively small set of trees sampled from a proposal model $q(\mathbf{y}|\mathbf{x})$. For our depdendency-based models, we use the Biaffine-roberta (Dozat and Manning, 2017) parser as the proposal model to sample 300 unlabeled projective dependency trees without replacement as a proposal tree set $\mathbf{Y}'$. $p(\mathbf{x})$ is then approximated by $\sum_{\mathbf{y} \in \mathbf{Y}'} p(\mathbf{x}, \mathbf{y})$, which is an exact lower bound of the true value of $p(\mathbf{x})$ (hence leading to an upper bound of perplexity). We evaluate the models by sentence-level perplexity.

Results We report the perplexity of all the models in Table 2. DTG achieves comparable perplexity with TXL (tokens) and DTG-swift, outperforming DTG-eager. TXL (trans) achieves lower perplexity than TXL (tokens) even though the reported result of TXL (trans) is an upper bound of its true perplexity. It shows that jointly modeling dependency trees and sentences is helpful for sentence-level language modeling.

The perplexity upper bound of DTG can be seen to be lower than that of TG. There are two possible interpretations of this result: (i) Dependency trees give better guidance than constituency trees in syntactic language modeling. (ii) 300 trees may be too few to get an accurate approximation of perplexity when sampling from a large set of possible trees. Evaluating DTG and TG requires samples of unlabeled projective dependency trees and labeled constituency trees, respectively. The number of the former is much smaller than the number of the latter. Therefore, sampling 300 trees may give a much tighter perplexity upper bound for DTG than for TG, resulting in a gap in the reported results. Unfortunately, it requires nontrivial work to distinguish between the two possibilities and we leave it for future work.

4.2 Syntactic Generalization

To measure the syntactic generalization, we evaluate our models on BLiMP (Warstadt et al., 2020) and SG test suites (Hu et al., 2020).

[2] https://github.com/yzhangcs/parser

Setup on BLiMP BLiMP contains 67 generalization tests, each with 1000 sentence pairs. Each sentence pair consists of a grammatical sentence and an ungrammatical sentence. Models are evaluated by whether they assign a higher probability to the grammatical one. We use the same setup as in section 4.1, sampling 300 trees for each sentence and calculating a lower bound of marginal probability $p(\mathbf{x})$ for comparison.

Setup on SG SG consists of test suites for six fine-grained syntactic phenomena. Each test suite has a specific inequality formula for evaluation. These inequalities are based on incremental natural processing, requiring computing the surprisal values, i.e., $-\log p(x_t|x_{<t})$. We implement the word-synchronous beam search (Stern et al., 2017; Hale et al., 2018) to get the marginal probability at each token t and calculate the surprisal value. We fix the beam size at 300.

Results The results are reported in Table 2. For BLiMP, we found that most of the constituency-based syntactic language models perform comparably with our baseline TXL (tokens), while DTG, DTG-swift, and TXL (trans) outperform them. For SG, all syntactic language models perform better than TXL (tokens), and DTG achieves the highest score. These results show that explicit modeling of syntactic structures is helpful for better generalization in Transformer language models, and dependency relations may lead to greater improvements in generalization than constituency compositions.

We further compare TXL (trans) with DTG. The SG scores of 6 circuits are shown in Figure 3. In SG, DTG achieves a much higher average score than TXL (trans) and outperforms TXL (trans) in 4 circuits while maintaining comparable scores in the other 2 circuits. On the other hand, TXL (trans) performs better than DTG on BLiMP. We believe it is because BLiMP evaluates semantic knowledge in addition to syntactic knowledge as detailed in Warstadt et al. (2020), even though BLiMP is used as a syntactic testset in previous work of syntactic language models (Qian et al., 2021; Murty et al., 2023). Syntax-motivated attention masking in DTG, while helpful in syntactic modeling, hinders acquisition of semantic information. Please refer to Appendix C for more discussion. It is thus an interesting future direction to integrate syntactic language models with standard language models so as to get the best of both worlds.

Model	UAS (↑)
Biaffine-roberta	96.9
TXL (trans)	97.0
DTG	97.0

Table 3: UAS on the PTB test set.

Model	PPL (↓)	BLiMP (↑)
w	15.1	75.9
arc	15.2	75.8
w+arc	**14.9**	**76.1**

Table 4: Results of different arc representations.

4.3 Parse Reranking

Setup We study to what extent DTG and TXL (trans) have learned to produce correct dependency structures. We still sample 300 trees with the Biaffine-roberta parser and rerank them using the two models. We convert human-annotated constituency trees in the Penn Treebank (PTB) (Marcus et al., 1993) test split into dependency trees with CoreNLP 3.3.0 (Manning et al., 2014) and then evaluate the UAS of the reranked trees on them.

Result We present the results in Table 3. TXL (trans) and DTG both achieve a slightly higher score than the proposal model Biaffine-roberta. Note that both models are trained on the dependency parse trees produced by Biaffine-roberta. The results show that both models successfully learn about dependency structures from Biaffine-roberta.

5 Analysis

5.1 Arc Representation

We compare three different representations of `LEFTARC/RIGHTARC` in DTG : (i) the default formulation of summing the `LEFTARC/RIGHTARC` embedding and the embedding of the head token x, (denoted as *w* + *arc*); (ii) the embedding of the `LEFTARC/RIGHTARC` alone (denoted as *arc*); (iii) the embedding of the head token alone (denoted as *w*). DTG models with these representations are trained and evaluated with the same setting as in section 4.

The result is reported in Table 4. The default formulation outperforms the other two representations, showing that both the head token embedding and the `LEFTARC/RIGHTARC` embedding play a positive role in arc representation.

Parser	PPL (↓)	BLiMP (↑)
Biaffine	15.1	76.0
Biaffine-roberta	**14.9**	**76.1**

Table 5: Results of using different external parsers.

5.2 Dependency Parses for Training

We use an external parser to provide dependency trees in the training data and sample 300 trees in sentence probability evaluation. Here, we study how the quality of the external parser affects our model's performance. We compare two parsers, vanilla Biaffine without pre-trained token embeddings and Biaffine-roberta,[3] as the external parser used in training and evaluation. Note that Biaffine-roberta is more accurate than vanilla Biaffine.

The result is reported in Table 5. We see an improvement in both perplexity and generalization when using a better parser.

6 Related Work

Augmenting language models with syntactic bias has been studied for a long time. One line of work adds constituency-based syntactic structures to language models through jointly modeling the distribution of sentences and structures (Chelba, 1997; Roark, 2001; Henderson, 2004; Choe and Charniak, 2016; Kim et al., 2019). The RNNG model (Dyer et al., 2016) is a representative work of syntactic language models, using recursive networks to build representations of phrases. More recent work of syntactic language models is based on Transformers (Qian et al., 2021; Yoshida and Oseki, 2022; Sartran et al., 2022; Murty et al., 2023). Qian et al. (2021) and Sartran et al. (2022) constrain the attention with syntactic bias, while Pushdown Layers (Murty et al., 2023) enforce structural constraints via gradient based learning. The above work is all based on constituency structures, and there has been some work considering dependency trees with simple neural networks (Titov and Henderson, 2007; Cohen et al., 2011; Buys and Blunsom, 2015; Mirowski and Vlachos, 2015). Most of them, however, focus more on generative dependency parsing while scratching the surface of a language modeling setting. A more general work is Prange et al. (2022), which both introduces constituency and dependency graphs to augment Transformer language modeling, but it requires given gold trees for generation. Following the work of generative dependency parsing and the constrained attention patterns used in Sartran et al. (2022) and other work (Strubell et al., 2018; Peng et al., 2019; Zhang et al., 2020; Nguyen et al., 2020; Fernandez Astudillo et al., 2020; Lou and Tu, 2023), we propose DTG, a novel class of dependency-based syntactic language models. It is the first syntactic language model that designs a dependency-based constrained attention mechanism for Transformers.

Another line of work augments models by learnable structures. Some studies integrate stack-structured memory into models, where updating patterns are learned from data rather than being dictated by predefined syntactic inductive bias (Joulin and Mikolov, 2015; Yogatama et al., 2018; DuSell and Chiang, 2021, 2023). Besides, some studies propose to learn structural attention patterns (Kim et al., 2017; Wang et al., 2019; Shen et al., 2021, 2022). For example, Kim et al. (2017) assumes that the attention scores are subject to linear-chain or tree conditional random fields (CRFs; Lafferty et al., 2001). These kinds of augmentation lead to better generalization but usually cost longer running time than naive counterparts.

Some other studies focus on examining the syntactic knowledge acquired by standard attention after pretraining (Htut et al., 2019; Kovaleva et al., 2019; Kim et al., 2020; Ravishankar et al., 2021). These studies have identified that certain attention heads align their attention patterns with syntactic structures, thereby providing pivot beliefs on the benefits of introducing syntactic inductive bias. In addition, some work re-invents attention using dependency structures and CRFs (Wu and Tu, 2023), motivating more linguistically principled studies.

7 Conclusion

We propose DTGs, a new type of syntactic language models that add explicit dependency bias into Transformers. DTGs simulate dependency transition systems with constrained attention patterns and incorporate stack information through relative positional encoding. Experiments show that DTGs surpass Transformer language model baselines and other constituency-based syntactic language models on syntactic generalization while maintaining competitive perplexity. This implies that the presence of dependency information does improve the performance of Transformer language models.

[3] Also from `https://github.com/yzhangcs/parser`

Limitations

DTGs rely on dependency trees for training, which are predicted by an external parser in this study. However, for languages lacking accurate dependency parsers, our methods might not offer benefits. Additionally, we restrict trees in our study to be in the Standard Dependency representation (de Marneffe and Manning, 2008) and only consider non-labeled projective dependency trees at the sentence level. The investigation of other dependency representations, such as Universal Dependencies (Nivre et al., 2020), more complex trees and document-level settings is lefted for future research.

For training and inference, DTGs cannot utilize some recent advancements for Transformers easily, including rotary position embeddings (Su et al., 2021) and Flash attention (Dao et al., 2022), due to our attention mask patterns and relative position encodings. Moreover, evaluating a sentence's probability with DTGs requires marginalizing over all possible trees, which is intractable. In this study, we approximate this by sampling 300 trees. However, this is still time-consuming and only provides an upper bound for the perplexity metric.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D 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 Ziegler, Jeffrey Wu, Clemens Winter, Chris 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*, volume 33, pages 1877–1901. Curran Associates, Inc.

Jan Buys and Phil Blunsom. 2015. Generative incremental dependency parsing with neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 863–869, Beijing, China. Association for Computational Linguistics.

Eugene Charniak, Don Blaheta, Niyu Ge, Keith Hall, John Hale, and Mark Johnson. 2000. Bllip 1987-89 wsj corpus release 1. *Linguistic Data Consortium*, 36.

Ciprian Chelba. 1997. A structured language model. In *35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 498–500, Madrid, Spain. Association for Computational Linguistics.

Do Kook Choe and Eugene Charniak. 2016. Parsing as language modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336, Austin, Texas. Association for Computational Linguistics.

Shay B. Cohen, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Exact inference for generative probabilistic non-projective dependency parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1234–1245, Edinburgh, Scotland, UK. Association for Computational Linguistics.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher R'e. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *ArXiv*, abs/2205.14135.

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The Stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8, Manchester, UK. Coling 2008 Organizing Committee.

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

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *International Conference on Learning Representations*.

Brian DuSell and David Chiang. 2021. Learning hierarchical structures with differentiable nondeterministic stacks. *arXiv preprint arXiv:2109.01982*.

Brian DuSell and David Chiang. 2023. Stack attention: Improving the ability of transformers to model hierarchical patterns. *arXiv preprint arXiv:2310.01749*.

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.

Martin BH Everaert, Marinus AC Huybregts, Noam Chomsky, Robert C Berwick, and Johan J Bolhuis. 2015. Structures, not strings: Linguistics as part of the cognitive sciences. *Trends in cognitive sciences*, 19(12):729–743.

Ramón Fernandez Astudillo, Miguel Ballesteros, Tahira Naseem, Austin Blodgett, and Radu Florian. 2020. Transition-based parsing with stack-transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1001–1007, Online. Association for Computational Linguistics.

John Hale, Chris Dyer, Adhiguna Kuncoro, and Jonathan Brennan. 2018. Finding syntax in human encephalography with beam search. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2727–2736, Melbourne, Australia. Association for Computational Linguistics.

James Henderson. 2004. Discriminative training of a neural network statistical parser. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 95–102, Barcelona, Spain.

Phu Mon Htut, Jason Phang, Shikha Bordia, and Samuel R. Bowman. 2019. Do attention heads in bert track syntactic dependencies? *ArXiv*, abs/1911.12246.

Jennifer Hu, Jon Gauthier, Peng Qian, Ethan Wilcox, and Roger Levy. 2020. A systematic assessment of syntactic generalization in neural language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1725–1744, Online. Association for Computational Linguistics.

Armand Joulin and Tomas Mikolov. 2015. Inferring algorithmic patterns with stack-augmented recurrent nets. *Advances in neural information processing systems*, 28.

Taeuk Kim, Jihun Choi, Daniel Edmiston, and Sang goo Lee. 2020. Are pre-trained language models aware of phrases? simple but strong baselines for grammar induction. In *International Conference on Learning Representations*.

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. 2017. Structured attention networks. In *International Conference on Learning Representations*.

Yoon Kim, Alexander Rush, Lei Yu, Adhiguna Kuncoro, Chris Dyer, and Gábor Melis. 2019. Unsupervised recurrent neural network grammars. 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 1105–1117, Minneapolis, Minnesota. Association for Computational Linguistics.

Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. 2019. Revealing the dark secrets of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4365–4374, Hong Kong, China. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 673–682, Portland, Oregon, USA. Association for Computational Linguistics.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, page 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Chao Lou and Kewei Tu. 2023. AMR parsing with causal hierarchical attention and pointers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8942–8955, Singapore. Association for Computational Linguistics.

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

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.

Piotr Mirowski and Andreas Vlachos. 2015. Dependency recurrent neural language models for sentence completion. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 511–517, Beijing, China. Association for Computational Linguistics.

Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher Manning. 2023. Pushdown layers: Encoding recursive structure in transformer language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3233–3247, Singapore. Association for Computational Linguistics.

Xuan-Phi Nguyen, Shafiq Joty, Steven Hoi, and Richard Socher. 2020. Tree-structured attention with hierarchical accumulation. In *International Conference on Learning Representations*.

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.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain. Association for Computational Linguistics.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Jan Hajivc, Christopher D. Manning, Sampo Pyysalo, Sebastian Schuster, Francis M. Tyers, and Daniel Zeman. 2020. Universal dependencies v2: An evergrowing multilingual treebank collection. In *International Conference on Language Resources and Evaluation*.

Hao Peng, Roy Schwartz, and Noah A. Smith. 2019. PaLM: A hybrid parser and language model. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3644–3651, Hong Kong, China. Association for Computational Linguistics.

Jakob Prange, Nathan Schneider, and Lingpeng Kong. 2022. Linguistic frameworks go toe-to-toe at neuro-symbolic language modeling. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4375–4391, Seattle, United States. Association for Computational Linguistics.

Peng Qi and Christopher D. Manning. 2017. Arc-swift: A novel transition system for dependency parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 110–117, Vancouver, Canada. Association for Computational Linguistics.

Peng Qian, Tahira Naseem, Roger Levy, and Ramón Fernandez Astudillo. 2021. Structural guidance for transformer language models. 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)*, pages 3735–3745, Online. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Vinit Ravishankar, Artur Kulmizev, Mostafa Abdou, Anders Søgaard, and Joakim Nivre. 2021. Attention can reflect syntactic structure (if you let it). In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3031–3045, Online. Association for Computational Linguistics.

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

Laurent Sartran, Samuel Barrett, Adhiguna Kuncoro, Miloš Stanojević, Phil Blunsom, and Chris Dyer. 2022. Transformer grammars: Augmenting transformer language models with syntactic inductive biases at scale. *Transactions of the Association for Computational Linguistics*, 10:1423–1439.

Yikang Shen, Shawn Tan, Alessandro Sordoni, Peng Li, Jie Zhou, and Aaron Courville. 2022. Unsupervised dependency graph network. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4767–4784, Dublin, Ireland. Association for Computational Linguistics.

Yikang Shen, Yi Tay, Che Zheng, Dara Bahri, Donald Metzler, and Aaron Courville. 2021. StructFormer: Joint unsupervised induction of dependency and constituency structure from masked language modeling. 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)*, pages 7196–7209, Online. Association for Computational Linguistics.

Mitchell Stern, Daniel Fried, and Dan Klein. 2017. Effective inference for generative neural parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1695–1700, Copenhagen, Denmark. Association for Computational Linguistics.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5027–5038, Brussels, Belgium. Association for Computational Linguistics.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. 2021. Roformer: Enhanced transformer with rotary position embedding. *ArXiv*, abs/2104.09864.

Ivan Titov and James Henderson. 2007. A latent variable model for generative dependency parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 144–155, Prague, Czech Republic. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Yaushian Wang, Hung-Yi Lee, and Yun-Nung Chen. 2019. Tree transformer: Integrating tree structures into self-attention. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1061–1070, Hong Kong, China. Association for Computational Linguistics.

Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. 2020. BLiMP: The benchmark of linguistic minimal pairs for English. *Transactions of the Association for Computational Linguistics*, 8:377–392.

Haoyi Wu and Kewei Tu. 2023. Probabilistic transformer: A probabilistic dependency model for contextual word representation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 7613–7636, Toronto, Canada. Association for Computational Linguistics.

Dani Yogatama, Yishu Miao, Gabor Melis, Wang Ling, Adhiguna Kuncoro, Chris Dyer, and Phil Blunsom. 2018. Memory architectures in recurrent neural network language models. In *International Conference on Learning Representations*.

Ryo Yoshida and Yohei Oseki. 2022. Composition, attention, or both? In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 5822–5834, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Zhuosheng Zhang, Yuwei Wu, Junru Zhou, Sufeng Duan, Hai Zhao, and Rui Wang. 2020. Sg-net: Syntax-guided machine reading comprehension. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9636–9643.

A Examples of DTG-eager and DTG-swift

The example of DTG-eager is shown in Figure 4. The main difference with DTGs is the new transition `POP`. It directly masks the top of the stack and attends to other positions, denoted as **POPSTACK** attention.

The example of DTG-swift is shown in Figure 5. The newly introduced `ARC` number is represented within [].

B Other Experimental Details

Using subword tokenizers Previously, we always assume that each word corresponds to a single token. However, subword tokenizers (e.g., SentencePiece) may divide a word into several subtokens. In our work, we do not consider dependencies among subtokens within a word. All dependencies between words are converted to arcs between the last subtokens of these words. For masking, once a word should be masked, all of its subtokens are masked. For arc representation, we use the embedding of the last subtoken of the head word.

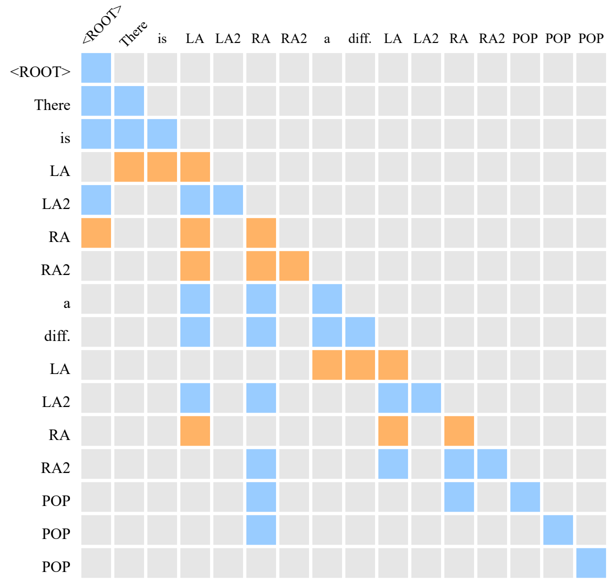
Computational costs We spent one NVIDIA A6000 GPU for each training, which lasted approximately 35 hours.

C Discussion on the Results of BLiMP

An example testcase in the QUANTIFIERS category of BLiMP is to judge whether "An actor arrived at at most six lakes" or "No actor arrived at at most six lakes" is acceptable. The correct answer is that the former is acceptable while the latter is not, because superlative quantifiers cannot embed under negation. A stardard Transformer language model could assign a lower probability to the second sentence because it could lower the probability of generating "at most" by attending to "No". In DTG , however, "No" as a determiner is absorbed into "actor" and hence masked from the attention when generating "at most". While doing this can be beneficial to syntactic generalization, it hinders semantic judgment in this case.

i	Input	Attn. Mask	Label
0	<ROOT>	STACK	GEN(There)
1	There	STACK	GEN(is)
2	is	STACK	LEFTARC
3	LEFTARC + is	COMPOSE	-
4	LEFTARC2 + is	STACK	RIGHTARC
5	RIGHTARC + <ROOT>	COMPOSE	-
6	RIGHTARC2 + <ROOT>	STACK	GEN(a)
7	a	STACK	GEN(difference)
8	difference	STACK	LEFTARC
9	LEFTARC + difference	COMPOSE	-
10	LEFTARC2 + difference	STACK	RIGHTARC
11	RIGHTARC + is	COMPOSE	-
12	RIGHTARC2 + is	STACK	POP
13	POP	STACK	POP
14	POP	STACK	POP
15	POP	STACK	<END>

(a) Transition sequence after duplicating LEFTARC/RIGHTARCs. We do not have to make predictions for positions 3, 5, 9, 11.

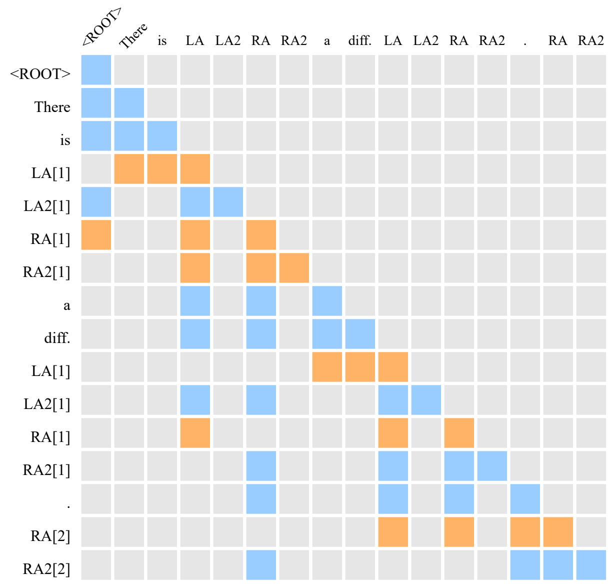


(b) Attention mask. Tokens are simplified for a tight view. We use orange to represent **COMPOSE** and blue to represent **STACK** and **POPSTACK**.

Figure 4: Arc-eager processing of an example sentence

i	Input	Attn. Mask	Label
0	<ROOT>	STACK	GEN(There)
1	There	STACK	GEN(is)
2	is	STACK	LEFTARC
3	LEFTARC[1] + is	COMPOSE	-
4	LEFTARC2[1] + is	STACK	RIGHTARC
5	RIGHTARC[1] + <ROOT>	COMPOSE	-
6	RIGHTARC2[1] + <ROOT>	STACK	GEN(a)
7	a	STACK	GEN(difference)
8	difference	STACK	LEFTARC
9	LEFTARC[1] + difference	COMPOSE	-
10	LEFTARC2[1] + difference	STACK	RIGHTARC
11	RIGHTARC[1] + is	COMPOSE	-
12	RIGHTARC2[1] + is	STACK	.
13	.	STACK	RIGHTARC
14	RIGHTARC[2] + is	COMPOSE	-
15	RIGHTARC2[2] + is	STACK	<END>

(a) Transition sequence after duplicating LEFTARC/RIGHTARCs. We do not have to make predictions for positions 3, 5, 9, 11, 14.



(b) Attention mask. Tokens are simplified for a tight view. We use orange to represent **COMPOSE** and blue to represent **STACK**. The number in [] is the ARC number of LEFTARC/RIGHTARC.

Figure 5: Arc-swift processing of an example sentence