A Systematic Study of Compositional Syntactic Transformer Language Models

Yida Zhao[1,2], **Hao Xve**[1,2], **Xiang Hu**[3], **Kewei Tu**[1,2] *
School of Information Science and Technology, ShanghaiTech University[1]
Shanghai Engineering Research Center of Intelligent Vision and Imaging[2]
Ant Group[3]
{zhaoyd2023,xvehao,tukw}@shanghaitech.edu.cn, aaron.hx@antgroup.com

Abstract

Syntactic language models (SLMs) enhance Transformers by incorporating syntactic biases through the modeling of linearized syntactic parse trees alongside surface sentences. This paper focuses on compositional SLMs that are based on constituency parse trees and contain explicit bottom-up composition of constituent representations. We identify key aspects of design choices in existing compositional SLMs and propose a unified framework encompassing both existing models and novel variants. We conduct a comprehensive empirical evaluation of all the variants in our framework across language modeling, syntactic generalization, summarization, dialogue, and inference efficiency. Based on the experimental results, we make multiple recommendations on the design of compositional SLMs. Our code is released at https://github.com/zhaoyd1/compositional_SLMs.

1 Introduction

Transformer language models (LMs) have achieved remarkable success on various NLP tasks (Devlin et al., 2019; Radford et al., 2019; Brown et al., 2020; Ouyang et al., 2022). While the Transformer architecture (Vaswani et al., 2017) is highly powerful, it lacks the inductive bias of syntactic structures, which is believed to be critical for effective generalization (Everaert et al., 2015). Syntactic language models (SLMs) (Qian et al., 2021; Yoshida and Oseki, 2022; Sartran et al., 2022; Murty et al., 2023; Zhao et al., 2024; Hu et al., 2024) incorporate such syntactic biases into Transformers with a straightforward method: modeling linearized syntactic parse trees along with the surface sentences.

A major class of SLMs, which we call compositional SLMs, are based on constituency parse trees and contain explicit composition of sub-constituent representations to form constituent representations (Sartran et al., 2022; Yoshida and Oseki, 2022; Hu et al., 2024). These compositional SLMs differ in some key aspects, including the form of parse trees, the tree linearization strategy, the composition function, and the attention masking scheme. However, the specific impact of these aspects on SLM performance in language modeling and downstream tasks remains under-explored.

In this paper, we propose a unified framework of compositional SLMs that encompasses all these aspects. Our framework subsumes not only existing models as special cases but also more than ten novel variants. We then conduct a systematic empirical comparison of all the variants in language modeling, syntactic generalization, summarization and dialogue (as two representative downstream tasks), and inference efficiency. The experimental results indicate that, compared with the Transformer LM baseline, compositional SLMs may underperform in language modeling, but the top-performing variants demonstrate significantly improved syntactic generalization, summarization, and dialogue performance, thus confirming the benefit of incorporating syntactic biases and explicit composition. We also observe significant performance and efficiency differences between the variants and make several recommendations on the design choices of compositional SLMs, such as discouraging sub-constituent masking and encouraging the combination of a specialized composition function and binary parse trees.

In summary, our contributions are two-fold:

- We identify key aspects of design choices seen in existing compositional SLMs and propose a unified framework encompassing both existing models and novel variants.
- We conduct a comprehensive empirical evaluation of all the variants within our framework across a range of metrics, which leads

* Corresponding author

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 7070–7083

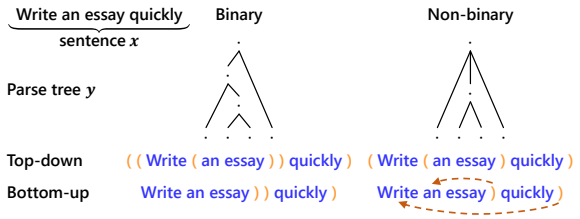


Figure 1: An example sentence, its binary and non-binary parse trees, and their linearizations produced by the two methods. For the bottom-up linearization of the non-binary tree, arcs are used to point from ")" actions to their corresponding start positions.

to multiple recommendations on the design of compositional SLMs.

2 Compositional SLM: A Framework

This section defines a framework that subsumes existing compositional SLMs as special cases. We start with an overview of the framework before delving into details of four key aspects of design choices.

A compositional syntactic language model (SLM) defines a joint distribution of sentences $\mathbf{x}$ and their constituency parse trees $\mathbf{y}$. For simplicity, we focus on unlabeled constituency trees in this paper. Following previous work on generative parsing and SLMs (Dyer et al., 2016; Choe and Charniak, 2016; Sartran et al., 2022), we define a sequence of actions $\boldsymbol{a} = (a_0, a_1, ..., a_{L-1})$ of length L that construct $(\mathbf{x}, \mathbf{y})$ in a left-to-right manner, where a_i is an action that either generates a token in $\mathbf{x}$ or indicates bracketing within a parse tree in $\mathbf{y}$. We say $\boldsymbol{a}$ is a *linearization* of $(\mathbf{x}, \mathbf{y})$. Figure 1 shows examples of two types of constituency parse trees and two linearization methods, which will be explained in section 2.1 and 2.2 respectively. The joint probability of $(\mathbf{x}, \mathbf{y})$ can then be computed in an autoregressive way:

$$p(\mathbf{x}, \mathbf{y}) = p(\boldsymbol{a}) = \prod_i p(a_i | \boldsymbol{a}_{<i})$$

A Transformer is utilized to model the autoregressive generation of action sequence $\boldsymbol{a}$.

A compositional SLM also leverages explicit composition that calculates a composed representation for each nonterminal constituent in a constituency parse tree from the representations of its sub-constituents. How the composed representation is calculated and integrated into the Transformer will be discussed in section 2.3. Since the information of the sub-constituents is contained in the composed representation, once the composition is done, the sub-constituents can be optionally masked in the Transformer for subsequent action generation (section 2.4).

2.1 Parse Tree Binarization

A linguistically defined constituency tree $\mathbf{y}$ is generally a non-binary tree. However, previous studies on SLMs (Murty et al., 2023; Hu et al., 2024) often model binarized parse trees, highlighting the potential practical benefits of binarization. We consider both options in our framework. (i) **Non-binary trees**, denoted as `Nb`. We eliminate all unary chains from any constituency trees, so structures like "*(quickly)*" are simplified to "*quickly*". (ii) **Binary trees**, denoted as `Bi`. We convert any non-binary tree into the Chomsky normal form using left binarization.

2.2 Linearization Methods

We consider two linearization methods for converting a constituency parse tree into an action sequence: top-down and bottom-up. **Top-down** linearization (Dyer et al., 2016; Sartran et al., 2022; Yoshida and Oseki, 2022), denoted as `Dn`, constructs a tree using pre-order traversal from the root to the terminals, with each nonterminal visited right before its children. In contrast, **bottom-up** linearization (Hu et al., 2024), denoted as `Up`, constructs a tree using post-order traversal from the leaf terminals to the root, with each nonterminal visited right after all its children are visited.

The choice between top-down and bottom-up linearization results in different action spaces. In top-down linearization, there are three types of actions: (i) opening a nonterminal (indicating the start of a new constituent), represented by "(", (ii) generating a terminal (a new token), represented directly by the token, and (iii) closing a nonterminal (indicating the end of the current constituent), represented by ")". On the other hand, bottom-up linearization does not require action (i), leaving only the other two actions. Consequently, bottom-up linearization is shorter than top-down linearization. Figure 1 presents examples of the two linearization methods on binary and non-binary trees.

As illustrated in the figure, a special case arises for **bottom-up linearization of a non-binary tree**, which has not been studied in previous work: for each closing-nonterminal action ")", the start of the current constituent is unknown and hence needs to be predicted. Concretely, if action a_k is pre-

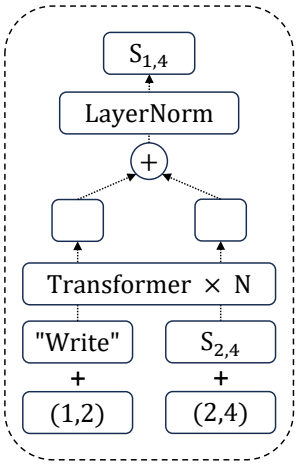


Figure 2: The Transformer-based external composition function f. The example input consists of the representations and position embeddings of two sub-constituents: "Write" and "an essay".

dicted to be ")", we additionally predict the start position $s_k \in C_k$ given prefix $\boldsymbol{a}_{<k}$, where $C_k \subseteq \{1, \cdots, k-1\}$ is the set of feasible start positions, at each of which is either a token or ")" of a closed nonterminal that is not subsumed by any closed nonterminals yet at step $< k$. In the example of "*Write an essay) quickly)*", the feasible start position set of the first ")" is $\{1, 2, 3\}$. For position i, we concatenate the outputs of the first two attention heads in the final layer of the Transformer as its representation $\mathbf{h}_i$. We then compute $p(s_k \mid \boldsymbol{a}_{<k})$ as follows:

$$p(s_k = i \mid \boldsymbol{a}_{<k}) \propto \begin{cases} \exp\left(\mathbf{h}_{k-1}^{\top} \Theta \mathbf{h}_i\right) & \text{if } i \text{ in } C_k \\ 0 & \text{otherwise} \end{cases}$$

where Θ is an extra learnable matrix. We then define $p(a_k \mid \boldsymbol{a}_{<k})$ as the product of probabilities of predicting ")" and the start position:

$$p(a_k = (")", s_k) \mid \boldsymbol{a}_{<k}) = p(")" \mid \boldsymbol{a}_{<k}) \cdot p(s_k \mid \boldsymbol{a}_{<k})$$

Another special case worthy of additional discussion is **top-down linearization of a binary tree**. For a binary tree, each constituent has a fixed length of two and therefore having both the opening-nonterminal and closing-nonterminal actions seems redundant. However, we decide to preserve both actions as explained in Appendix A.

2.3 Composition Function

When a constituent is completed, i.e., a ")" is generated, we use a composition function to compose all its sub-constituents into a single representation, which is then integrated into the Transformer and influences subsequent action generation. Previous studies use two different methods for this purpose: internal composition functions (Sartran et al., 2022) and external composition functions (Yoshida and Oseki, 2022; Hu et al., 2024).

An **internal** composition function, denoted as `In`, regards each composition as an additional action within the action sequence $\boldsymbol{a}$ and relies on the Transformer for composition computation. Specifically, for each predicted a_k = ")", we directly input a ")" to the Transformer at step k and set the attention mask such that only the sub-constituents of the current constituent can be attended to, thus forcing the computed hidden states to represent the composition of these sub-constituents. Note that step k has an attention range limited to a single constituent and hence uninformative for predicting the next action. Therefore, we input a duplicate ")" to the Transformer at step $k+1$, allow attention to the full context (more details in section 2.4), and output the next action prediction. The duplicate ")" is then permanently masked in subsequent steps.

The internal composition function, as described above, integrates the composition process into the Transformer, thus simplifying implementation and enabling parallelized training as in a standard Transformer LM. Its downside is that recursive composition through multiple Transformer layers has a receptive-field limitation as explained by section 2.3 of Sartran et al. (2022)) and the action sequence length is increased by the number of duplicate ")".

An **external** composition function, denoted as `Ex`, employs an additional module f with separate parameters from the Transformer. Specifically, for each predicted α_k = ")", module f takes as input the representations of the sub-constituents, which are either token embeddings or representations previously computed by the module, and outputs a single representation of the current constituent:

$$S_{p_0,p_m} = f(S_{p_0,p_1}, \cdots, S_{p_{m-1},p_m})$$

where $p_0, \ldots, p_m$ are left-inclusive and right-exclusive indexes of the sub-constituent spans. The newly composed representation S_{p_0,p_m} is then used as the input embedding at step k in the Transformer. We adopt the Transformer-based composition function from GPST (Hu et al., 2024) shown in Figure 2.

The external composition function leverages the composed representation as the input to the Trans-

former, thereby avoiding the limitation of recursive composition in the internal composition function. However, it requires implementing and running an external module in addition to the Transformer. During training, all the compositions in the parse trees of the training set are typically pre-computed before the parallelized training of the Transformer, resulting in a slightly increased training time.

2.4 Sub-Constituent Masking

After each composition, we may follow Sartran et al. (2022) and prevent subsequent steps from directly accessing information about already composed sub-constituents, creating a syntactic bottleneck that encourages learning informative compositions. On the other hand, from a language modeling perspective, allowing access to sub-constituent information, as done in Hu et al. (2024), could enhance performance by providing additional context. Therefore, we consider two contrasting settings: (i) **Mask** the already composed sub-constituents, denoted as `M`, and (ii) **No mask** for the already composed sub-constituents, denoted as `Nm`. An example illustrating different mask patterns combined with different composition functions is presented in Figure 3. Note that the choice of `M` or `Nm` does not affect the mask used for internal composition described in section 2.3.

2.5 Variants Within the Framework

Our framework specifies two options for each of the four key aspects and hence contains sixteen distinct SLMs, each named based on its configuration across the four aspects. For example, `Bi-Dn-In-M` represents an SLM that models linearized binary trees in a top-down manner with an internal composition function and sub-constituent masking. We use the symbol `#` to denote any option within a particular aspect. For instance, `Bi-#-#-#` signifies an SLM that models binary trees, regardless of the choices made in the other aspects.

Compositional SLMs from previous studies can be accommodated within our framework with minor modifications: (i) Transformer Grammars (Sartran et al., 2022) are classified as `Nb-Dn-In-M` if modeling unlabeled trees. (ii) Composition Attention Grammars (Yoshida and Oseki, 2022) are classified as `Nb-Dn-Ex-M` if we change the composition function from a bidirectional LSTM to a Transformer. (iii) Generative Pretrained Structured Transformers (Hu et al., 2024) are classified as `Bi-Up-Ex-Nm` if we set the depth of token layers to zero, i.e., we task type layers to predict both actions and tokens. Apart from these three models, the other thirteen SLMs within our framework are novel SLM variants not studied before.

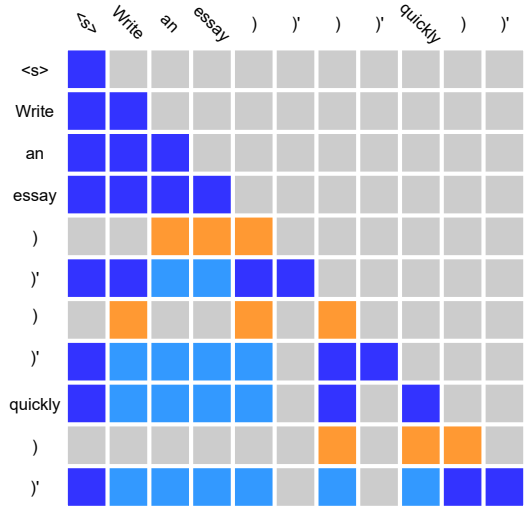
2.6 Inference

The space of token generation (type ii actions) is much larger than that of structure generation actions (type i & iii actions) in SLMs, leading to an imbalance between their probabilities. Word-synchronous beam search, first introduced by Stern et al. (2017), groups beams by the length of generated tokens instead of the whole action sequence, forcing SLMs to generate high-entropy tokens. We implement word-synchronous beam search for each SLM variant in our framework. There are two cases in which our implementation deviates from the standard implementation: (i) For `Nb-#-#-#`, the number of nonterminals is not fixed given a sentence. We apply an additional hyperparameter n_c as the maximum number of nonterminals, preventing models from composing too many times. (ii) For `#-Dn-#-#`, the model tends to generate a lot of successive "(" because structure generation is of low entropy. We also apply a hyperparameter p_c as the maximum number of consecutive generation of "(".

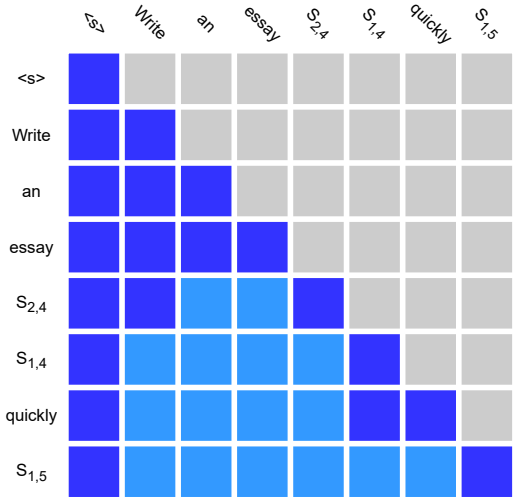
3 Experiments

We compare the sixteen compositional SLMs from our framework with two Transformer baselines: (i) **GPT2-token**, a traditional language model of token sequences, and (ii) **GPT2-tree**, a syntactic language model of linearized trees without explicit composition. Following the setting in Sartran et al. (2022), GPT2-tree models non-binary trees in a top-down manner. We train all the models from scratch on the same corpus with comparable parameter sizes. We first evaluate all the models on language modeling and syntactic generalization. Then, we select eight best-performing compositional SLMs on both tasks, along with GPT2-token and GPT2-tree baselines, for further evaluation on summarization and dialogue—two generation tasks that we consider representative of downstream applications. Finally, we compare the inference efficiency of SLMs within the word-synchronous beam search setup.

We also report experiments on additional baselines in Appendix D, including variants of GPT2-tree based on trivial trees and different binarization/linearization options.



(a) Attention masks for modeling a binary tree with the internal composition function. ")'" represents a duplication of its preceding ")".



(b) Attention masks for modeling a binary tree with the external composition function. $S_{2,4} = f(\text{an}, \text{essay})$, $S_{1,4} = f(\text{Write}, S_{2,4})$, $S_{1,5} = f(S_{1,4}, \text{quickly})$.

Figure 3: Examples of different mask patterns combined with different composition functions. We use gray for masked positions, orange for the attention ranges of internal compositions, dark blue for ordinary attended positions, light blue for already composed positions that are only accessible in **Nm**.

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). We use an off-the-shelf CRF constituency parser (Zhang et al., 2020) , implemented in *Supar*[1], to reparse the dataset and obtain silver constituency trees for training. All the silver trees parsed or sampled in the rest of the experiments are also produced with the same parser. Left-binarization is done with *nltk*[2]. Note that we model each sentence as a whole during both training and evaluation, and cutoff can only take place between two sentences to maintain the integrity of parse trees.

Hyper-parameters. Following GPT-2_{small} (Radford et al., 2019), we use 768-dimensional embeddings, a vocabulary size of 50257, 3072-dimensional hidden layer representations, 12 Transformer layers, and 12 attention heads for all SLMs and baselines. To maintain comparable parameter numbers between internal and external composition functions, we use a relatively small Transformer as the external composition function for **#-#-Ex-#**, setting the input dimension to 256 and the number of layers to 4, following (Hu et al., 2024). Token embeddings are down-scaled before composition and the constituent representations are up-scaled before fed into the main SLM Transformer. The module increases the total parameter number by only 5%, which we believe does not significantly affect the comparability between models. We discuss other training details and training variances in Appendix B.

3.1 Document-Level Language Modeling

Dataset. We evaluate all the models on the testing split of BLLIP-LG from Hu et al. (2020).

Setup. Since SLMs model $p(\mathbf{x}, \mathbf{y})$, the joint probability of sentences and parse trees, we compute the probability of a sentence as $p(\mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{x}, \mathbf{y})$. It is impossible to compute the summation exactly due to the large space of possible constituency trees, so we follow Sartran et al. (2022) to approximate it using a relatively small set of trees sampled from a proposal model. We use a CRF parser as the proposal model and sample 300 unlabeled constituency 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 a lower bound of the true value (hence leading to an upper bound of perplexity).

For document-level language modeling, we compute the probability of a document consisting of M

[1] https://github.com/yzhangcs/parser
[2] https://www.nltk.org/

Model	PPL† (↓)	SG (↑)
GPT2-token	**17.31**	64.1
GPT2-tree	19.97	73.1
Bi-Up-Ex-Nm	20.51	80.1
Bi-Up-Ex-M	24.15	**82.4**
Bi-Up-In-Nm	19.99	77.5
Bi-Up-In-M	21.32	79.7
Bi-Dn-Ex-Nm	23.62	80.2
Bi-Dn-Ex-M	27.21	80.9
Bi-Dn-In-Nm	22.02	79.4
Bi-Dn-In-M	26.50	80.9
Nb-Up-Ex-Nm	23.85	40.8
Nb-Up-Ex-M	24.07	51.8
Nb-Up-In-Nm	19.36	79.6
Nb-Up-In-M	22.01	73.4
Nb-Dn-Ex-Nm	20.88	51.1
Nb-Dn-Ex-M	25.15	51.9
Nb-Dn-In-Nm	18.11	78.1
Nb-Dn-In-M	22.30	75.6

Table 1: Perplexity (PPL) and syntactic generalization (SG) results of our models and baselines. †: All the reported PPLs except that of GPT2-token are upper bounds of the true values.

sentences. When computing $p(\mathbf{x}^i|\mathbf{x}^0, \cdots, \mathbf{x}^{i-1})$, the probability of the i-th sentence in the document conditioned on its $i-1$ preceding sentences, in theory we have to marginalize over all the i parse trees, each having 300 samples, which demands unacceptable computational costs. Following Sartran et al. (2022), we approximate this by greedily choosing a single tree $\mathbf{y}$ that maximizes $p(\mathbf{x}, \mathbf{y})$ for each of the preceding $i-1$ sentences, serving as a single-path prefix for the i-th sentence.

Results. We report the perplexity of all the models in Table 1. All the SLMs, including GPT2-tree, show higher perplexity than GPT2-token baselines, seemingly implying that document-level language modeling may not benefit from the inductive bias of syntax. However, this observation is inconclusive because the reported SLM PPLs are upper bounds of the true values. Another observation is that only **Nb-Dn-In-Nm** and **Nb-Up-In-Nm** outperform GPT2-tree, and **Bi-Up-In-Nm** shows comparable performance with GPT2-tree. Since the main difference between GPT2-tree and our models is that it does not involve explicit composition, this observation indicates that explicit composition may not be critical for language modeling and only helps in certain configurations.

Comparing compositional SLMs in our framework, we have two major findings: (i) Fixing the first three aspects, **#-#-#-Nm** consistently shows significantly better language modeling performance than **#-#-#-M**, which is to be expected because less information is directly available at each generation step in the setting of **M**, making it harder for next token prediction. (ii) **#-#-In-#** achieves lower perplexity than **#-#-Ex-#**, showing that directly reusing parameters of the main Transformer for composition is a better choice for language modeling than using a small external composition function.

3.2 Syntactic Generalization

Dataset. We evaluate all the models on the syntactic generalization (SG) task (Hu et al., 2020), consisting of test suites for six fine-grained syntactic phenomena.

Setup. Each test suite is evaluated by a specific inequality formula, which requires computing the surprisal values, i.e., $-\log p(\mathbf{x}_t|\mathbf{x}_{<t})$. We compute the surprisal values for SLMs using the word-synchronous beam search described in section 2.6. As the target token $\mathbf{x}_t$ is given, we modify the algorithm by directly predicting the given token. The beam size is set to 300. The maximum number of nonterminals n_c is dynamically set to the length of each sentence and the maximum number of consecutive opening-nonterminal actions p_c is set to 3. Further details on the selection of these hyperparameters are provided in Appendix C.

Results. The overall results are reported in Table 1. We also plot the detailed performance over each of the six syntactic phenomena in Appendix E. Most of the SLMs outperform the GPT2-token baseline with a significant gain in the SG score, which is to be expected because of their explicit modeling of syntax. Furthermore, most of the compositional SLMs outperform GPT2-tree, proving that explicit composition is helpful to SLMs in syntactic modeling.

It is notable that four compositional SLMs have extremely low SG scores and they all belong to the configuration of **Nb-#-Ex-#**, i.e., modeling non-binary trees with an external composition function. This is likely because the relatively small external composition model fails to capture the complicated interactions among varying numbers of sub-constituents. In sharp contrast, external composition functions applied to binary trees (**Bi-#-Ex-#**) achieve impressive SG scores, occupying four of the top five spots, suggesting that they are expressive enough to handle binary composition and even outperform internal composition functions of much larger sizes. A similar trend can be observed on

Model	Xsum				DailyDialog			
	R-1	R-2	R-L	R-AVG	R-1	R-2	R-L	R-AVG
GPT2-token	27.14	7.67	21.65	18.82	14.02	3.82	13.31	10.38
GPT2-tree	**29.59**	**9.47**	**23.58**	**20.88**	**14.99**	3.83	**14.31**	**11.04**
Bi-Up-Ex-Nm	29.04	8.95	23.01	20.33	14.14	**4.01**	13.61	10.59
Bi-Up-Ex-M	23.48	5.75	18.84	16.02	12.44	3.00	11.69	9.04
Bi-Up-In-Nm	28.93	8.97	22.97	20.29	13.01	3.33	12.19	9.51
Bi-Up-In-M	24.84	6.64	19.88	17.12	12.05	2.78	11.40	8.74
Nb-Up-In-Nm	29.05	9.06	23.21	20.44	13.81	3.68	13.09	10.19
Nb-Up-In-M	24.30	6.28	19.45	16.68	11.92	2.93	11.33	8.72
Nb-Dn-In-Nm	29.48	9.40	23.54	20.81	13.99	3.85	13.36	10.40
Nb-Dn-In-M	26.10	7.31	20.98	18.07	12.50	3.31	11.90	9.23

Table 2: Results on the summarization and dialogue tasks.

internal composition functions regarding the relative difficulty of modeling non-binary composition in comparison with binary composition (i.e., **Nb-#-In-#** vs. **Bi-#-In-#**), although to a much lesser extent.

For sub-constituent masking, we observe that **Bi-#-#-M** always performs better than **Bi-#-#-Nm**, confirming that the information bottleneck created by sub-constituent masking can benefit syntactic modeling. On the other hand, when it comes to non-binary trees (excluding worst-performing **Nb-#-Ex-#**), we observe that **Nb-#-In-M** performs worse than **Nb-#-In-Nm**. Considering that non-binary composition is more difficult as discussed above, we may conclude that sub-constituent masking is useful to syntactic modeling only when composition is effective.

3.3 Downstream Tasks

3.3.1 Summarization

Dataset. We conduct experiments on the BBC extreme dataset (Xsum) (Narayan et al., 2018) to assess the performance of SLMs in terms of summarization abilities.

Setup. We truncate the documents and their summaries to 600 and 70, respectively, and concatenate them with a short prompt "*Summary:*". Following Hu et al. (2024), we finetune each model for 15 epochs with a batch size of 16 on the training split of Xsum. ROUGE (Lin and Hovy, 2003) is employed as the evaluation metric. To evaluate SLMs, we apply the word-synchronous beam search to top-k random sampling with k set to 2. The maximum number of nonterminals n_c is dynamically set to the length of each sentence and the maximum number of consecutive opening-nonterminal actions p_c is set to 5. For all the SLMs, the input contains the linearization of the sentences in the document and their corresponding silver parse trees.

We only conduct experiments on eight compositional SLMs and discard the other eight as explained below: (i) The four models of **Nb-#-Ex-#** show poor performance on language modeling and syntactic generalization, indicating a failure in composition learning. (ii) The four models of **Bi-Dn-#-#** model both the opening-nonterminal and the closing-nonterminal actions. For binary trees, these two actions are redundant, and predicting one of them for each constituent is enough (as done in **Bi-Up-#-#**). The four models also show poor language modeling performance.

Results. The results are presented in Table 2. First of all, **#-#-#-Nm** significantly outperforms **#-#-#-M**, which is consistent with the language modeling results and highlights the importance of direct access to composed sub-constituents in generation tasks.

Second, all the four SLMs of **#-#-#-Nm** outperform GPT2-token, which can be attributed to two possible reasons: (i) SLMs may have better generation abilities than GPT2-token. (ii) GPT2-token only receives the input text as the prompt, while SLMs receive additional information—linearized parse trees of the input text. Regardless of the reasons, The results suggest that SLMs aided by an off-the-shelf parser have great potential in downstream generation tasks.

Finally, GPT2-tree achieves the best scores while compositional SLMs of **#-#-#-Nm** show comparable or slightly lower scores, suggesting again that explicit composition is not critical in generation tasks.

Model	bsz-10	bsz-30	bsz-100	bsz-300
GPT2-tree	2.06	2.94	3.98	5.81
Bi-Up-Ex-Nm	1.28	1.46	1.77	2.62
Bi-Up-Ex-M	1.33	1.49	1.97	3.23
Bi-Up-In-Nm	4.28	4.81	5.95	8.69
Bi-Up-In-M	4.28	4.80	5.91	8.70
Bi-Dn-Ex-Nm	1.20	1.33	2.13	3.76
Bi-Dn-Ex-M	1.14	1.61	2.44	4.53
Bi-Dn-In-Nm	3.79	4.46	6.03	9.73
Bi-Dn-In-M	3.91	4.44	5.93	10.43
Nb-Up-Ex-Nm	1.35	1.83	2.53	4.53
Nb-Up-Ex-M	3.04	3.51	4.71	7.74
Nb-Up-In-Nm	7.98	8.90	11.23	16.97
Nb-Up-In-M	10.52	11.12	13.97	20.84
Nb-Dn-Ex-Nm	0.93	1.25	1.70	3.35
Nb-Dn-Ex-M	1.11	1.71	2.66	5.17
Nb-Dn-In-Nm	3.99	5.03	7.22	11.37
Nb-Dn-In-M	4.43	5.34	7.72	12.17

Table 3: Inference time (in seconds, lower is better). "**bsz**" refers to beam size.

3.3.2 Dialogue

Dataset. We conduct experiments on the Dailydialog dataset (Li et al., 2017) to assess the performance of SLMs in terms of dialogue abilities.

Setup. In each dialogue, all the utterances except for the last one are fed as the prompt, and the last utterance is the generation target. We add a special <sep> to indicate the start of each utterance. We truncate the prompt utterances and the target to 600 and 150, respectively. We finetune each model for 5 epochs with a batch size of 16 on the training split of Dailydialog. The other setups are exactly the same as in the summarization experiments.

Results. The results are presented in Table 2. Similar to the findings in the summarization task, **#-#-#-Nm** significantly outperforms **#-#-#-M** and GPT2-tree achieves the best scores. Different from the summarization task, however, only **Bi-Up-Ex-Nm** clearly outperforms GPT2-token among the four **#-#-#-Nm** SLMs. Nonetheless, the improved performance of GPT2-tree and **Bi-Up-Ex-Nm** still shows the potential of SLMs in downstream generation tasks.

3.4 Inference Efficiency

Setup. In practice, SLMs reply on word-synchronous beam search to generate meaningful sentences and proper tree structures. Therefore, we compare the inference efficiency of all the SLMs with word-synchronous beam search. GPT2-token is not considered here because it only generates tokens without the need for synchronization of structures (see Appendix H for additional experiments comparing the efficiency gap between GPT2-token with SLMs). We follow the same setup in the syntactic generalization experiment. The sentence is fixed in advance, ensuring a fair comparison between the inference time of different SLMs. We evaluate the SLMs on five sentences of 20 tokens each. For beam sizes of {10, 30, 100, 300}, we repeat the inference for 5 times and compute the average time for each. We also choose one sentence and count the number of model forward calls for each beam size as a supplement in Appendix F. All the efficiency evaluation experiments are run on a single H800 GPU unless specifically mentioned.

Results. The results are shown in Table 3. All the models with external composition functions (**Ex**) show much less inference time than those with internal ones (**In**), because the external composition module is smaller and faster than reusing the main Transformer for composition. Moreover, a large gap in inference time exists between **#-#-Ex-M** and **#-#-Ex-Nm**. This is because when **Nm** is combined with **Ex**, the attention mask becomes a simple casual mask which can be accelerated by various optimization methods in `scaled_dot_product_attention` of PyTorch.

The **Nb-Up-#-#** models require more time than the **Nb-Dn-#-#** models, despite having shorter action sequences. We also find that **Bi-#-#-#** is generally faster than **Nb-#-#-#**, which is quite surprising because a non-binary tree usually has fewer nonterminals than a binary tree and thus a shorter linearization. We speculate that these two observations result from an intrinsic problem of applying word-synchronous beam search to modeling linearized non-binary trees, which we discuss further in Appendix G.

3.5 Overall Observations

Based on the overall experimental results, we recommend several design choices for compositional SLMs:

(i) SLMs without sub-constituent masks generally outperform their masked counterparts in both effectiveness and efficiency except for a potentially small disadvantage in syntax-focused tasks.

(ii) External composition excels for efficiency. If efficiency is the main concern, modeling binary trees with an external composition function is a good choice with decent performance on all the tasks.

(iii) Binary trees align better with bottom-up linearization in terms of both efficiency and performance, while non-binary trees seem to be more compatible with top-down linearization.

(iv) Modeling non-binary trees using an external composition function can result in suboptimal performance on certain tasks.

4 Related Work

Augmenting language models with syntactic bias has been a longstanding area of research. One line of work focuses on SLMs that jointly model the distribution of sentences and their structures (Chelba, 1997; Roark, 2001; Henderson, 2004; Choe and Charniak, 2016; Kim et al., 2019; Dyer et al., 2016). More recent SLMs are mostly based on Transformers. Among them, TGs (Sartran et al., 2022), CAGs (Yoshida and Oseki, 2022), and GPSTs (Hu et al., 2024) are closely related to our work as they are constituency-based SLMs with explicit composition. There are also recent studies not covered by our framework: Zhao et al. (2024) propose dependency-based SLMs, and Qian et al. (2021) and Murty et al. (2023) study constituency-based SLMs without explicit composition. Another line of work augments language models with learnable structures, such as stack-structured memory where syntax patterns are learned from data rather than predefined (Joulin and Mikolov, 2015; Yogatama et al., 2018; DuSell and Chiang, 2021, 2023), and learning structural attention patterns (Kim et al., 2017; Wang et al., 2019; Shen et al., 2021, 2022).

5 Conclusion

We propose a unified framework for compositional syntactic language models (SLMs) that encompasses four key aspects of design choices. Instances of this framework include not only existing models but also over ten novel variants. Our experiments demonstrate that compositional SLMs outperform the Transformer language model baseline in syntactic generalization and summarization, underscoring the potential of syntactic biases with explicit composition. Furthermore, we comprehensively compare the performance and efficiency of all the SLM variants, resulting in recommendations on the design of compositional SLMs.

Limitations

Our framework is currently limited to unlabeled constituency trees and is tested on a relatively small corpus using a GPT-2 backbone due to limited computational resources. Future research will explore other syntactic structures, larger corpora, and more advanced Transformer backbones. The composition functions employed in our framework show suboptimal performance when modeling non-binary trees. This limitation arises from the simplicity of their architecture and the absence of an explicit learning target to guide the composition process beyond the language modeling loss. We identify these as two key areas for enhancing the composition function.

For training and inference, most compositional SLMs are unable to readily leverage recent advancements in Transformer efficiency, such as Flash-Attention (Dao et al., 2022), due to their specific attention patterns. Additionally, we approximate the probability of a sentence by greedily selecting a single-path prefix and marginalizing over 300 sampled trees. Although this approach is commonly used in SLM studies, it is time-consuming and only provides an upper bound for the perplexity metric. We plan to explore more efficient approximation methods in future work.

Acknowledgement

This work was supported by the robotic AI-Scientist platform of Chinese Academy of Sciences and by the HPC platform of ShanghaiTech University.

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.

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 Euro-*

pean 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.

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.

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.

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.

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.

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.

Xiang Hu, Pengyu Ji, Qingyang Zhu, Wei Wu, and Kewei Tu. 2024. Generative pretrained structured transformers: Unsupervised syntactic language models at scale. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2640–2657, Bangkok, Thailand. 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.

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.

Yanran Li, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, and Shuzi Niu. 2017. DailyDialog: A manually labelled multi-turn dialogue dataset. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 986–995, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Chin-Yew Lin and Eduard Hovy. 2003. Automatic evaluation of summaries using n-gram co-occurrence statistics. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 150–157.

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.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium. Association for Computational Linguistics.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*.

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.

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.

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.

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.

Yu Zhang, Houquan Zhou, and Zhenghua Li. 2020. Fast and accurate neural crf constituency parsing. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4046–4053. International Joint Conferences on Artificial Intelligence Organization. Main track.

Yida Zhao, Chao Lou, and Kewei Tu. 2024. Dependency transformer grammars: Integrating dependency structures into transformer language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1543–1556, Bangkok, Thailand. Association for Computational Linguistics.

A Binary Trees and Top-down Linearization

When combining top-down linearization and binary trees (i.e., **`Bi-Dn-#-#`** as defined in section 2.5), for each constituent that is predicted with an opening-nonterminal action, there is no need to predict its closing-nonterminal action, because whenever two of its sub-constituents are generated, the constituent automatically ends. Thus, we may omit closing-nonterminal actions in **`Bi-Dn-#-#`**. However, this omission would cause (different) problems in both **`Bi-Dn-Ex-#`** and **`Bi-Dn-In-#`**. We use input sequence "<bos> (A (B C))" (whose output sequence is "(A (B C)) <eos>") as our running example to show the problems of such omission:

- For **`Bi-Dn-Ex-#`**, if we omit any closing-nonterminal action, the target sequence becomes "(A (B C <eos>". What is the corresponding input sequence? A natural choice is "<bos> (A (B C", but then the two representations composed from (B C) and (A (B C)) never get a chance to be input into the transformer as required in compositional SLMs. If we change the last input from C to one of the composed representations, then C is never input into the transformer, which is intuitively as bad if not worse.

- For **`Bi-Dn-In-#`**, the input sequence is "<bos> (A (B C))'))'" and the target sequence is

"(A (B C) _) _ <eos>". If we omit closing-nonterminal actions, then the input and target sequences become "<bos> (A (B C)')'" and "(A (B C _ _ <eos>". We need full context for <eos> prediction, so the two internal compositions (with attention limited to sub-constituents) of (B C) and (A (B C)) must taken place on the input C and)'. In other words, we have to compose B and C when the input is C. This is problematic because C is not even encoded by the transformer before its composition.

Solving all these problems requires non-trivial redesign of SLMs, which may be potentially inconsistent with the framework proposed in the paper. We leave this for future work.

B Training Details and Variances

SLMs model linearized trees, which consist of more action tokens than traditional token sequences. To ensure a fair comparison, we train all SLMs with a cutoff length of 2048 and GPT-2 with a cutoff length of 1024. All models are trained with a fixed learning rate of 5e-5 , and we modify the batch size for each model to fit within the available GPU memory. We spent 4 NVIDIA A6000 GPUs for each training, which lasted approximately 35 hours on average. To address training variance, we provide the evaluation results for **`Bi-Up-Ex-Nm`** as an example, as shown in Table 4, which was trained three times with different random seeds. The variance was found to be small and does not affect the experimental results presented in the paper.

C Hyperparameters Selection

The beam size of 300 is commonly used in previous studies (Qian et al., 2021; Murty et al., 2023; Hu et al., 2024), so we also fix the beam size at 300.

It is reasonable to set the maximum number of nonterminals n_c to be the length of a sentence because, for a binary tree, there are exactly $n_{token} - 1$ nonterminals with a sentence of length n_{token}. For a non-binary tree, there are even fewer nonterminals. Therefore, the length of a sentence is a good upper bound for the number of nonterminals n_c.

We tune the maximum number of consecutive opening-nonterminal actions p_c on the training set of BLLIP-LG. As the sentences in SG test suites are short in length (usually no more than 20 tokens), so we randomly choose 10 sentences from BLLIP-LG of no more than 15 tokens. We run word-synchronous beam search to get top-300 $p(\mathbf{x}, \mathbf{y})$ and approximate $p(\mathbf{x})$ by $\sum_{\mathbf{y}} p(\mathbf{x}, \mathbf{y})$. We tune p_c from 2 to 10 and find that when p_c is set to 3, $p(\mathbf{x})$ remains large for the 10 sentences. So we fix p_c to be 3 for SG evaluation. We also hypothesize that if the sentence is longer, there is likely to be more consecutive opening-nonterminal actions. Therefore, we set p_c to 5 for summarization as the summaries are longer.

Model	PPL (std)	SG (std)	R-AVG (std)
`Bi-Up-Ex-Nm`	20.51 (0.09)	80.1 (0.3)	20.33 (0.06)

Table 4: Mean and Variance.

Model	PPL_single† (↓)	PPL† (↓)	SG (↑)
GPT2-tree	**20.99**	**19.97**	73.1
Left-branching	22.20	21.93	54.2
Right-branching	21.34	21.22	42.5
GPT2-tree-Bi-Dn	-	22.68	78.1
GPT2-tree-Bi-Up	-	21.13	**78.6**
GPT2-tree-Nb-Up	-	22.23	36.0

Table 5: Perplexity (PPL) and syntactic generalization (SG) results of GPT2-tree, two trivial tree baselines, and three GPT2-tree variants. PPL_single means to estimate perplexity with a single linearized tree. †: All the reported PPLs are upper bounds of the true values.

D More Baselines

In this section, we experiment with a few variants of the GPT2-tree baseline.

D.1 Trivial Trees

We first introduce two trivial tree baselines to verify if the better performance of SLMs is achieved by incorporating real syntax or simply by leveraging more computation brought by longer sequences: (i) **Left-branching**: a SLM that models a linearized binary left-branching tree (e.g., "(((Write an) essay) quickly)"), and (ii) **Right-branching**: a SLM that models a linearized binary right-branching tree (e.g., "(Write (an (essay quickly)))").

The performance of the original GPT2-tree and these two new variants on language modeling and syntactic generalization is reported in Table 5. We additionally report **PPL_single** that is evaluated with a single proposal tree, which is the gold parse tree for the original GPT2-tree and the left/right-branching tree for the two variants. As shown in the table, these trivial tree baselines show worse per-

plexity and significantly worse SG scores. This suggests the importance of incorporating real syntax in the success of SLMs, which cannot be achieved with trivial tree structures.

D.2 Other Binarization and Linearization Options

In our main experiments, GPT2-tree is based on top-down linearized non-binary trees. There are three obvious variants of GPT2-tree: (i) GPT2-tree-**Bi-Dn**: modeling top-down linearized binary trees, (ii) GPT2-tree-**Bi-Up**: modeling bottom-up linearized binary trees, and (iii) GPT2-tree-**Nb-Up**: modeling bottom-up linearized non-binary trees. As shown in Table 5, GPT2-tree achieves the lowest perplexity and a medium SG score among the four. GPT2-tree-**Bi-Dn** and GPT2-tree-**Bi-Up** show higher perplexity, but they gain significant improvement in syntactic generalization, which is consistent with the performance of the corresponding compositional SLMs in section 3.2 (i.e., **Bi-#-#-#** all achieve impressive SG performance and **Bi-Up-Ex-Nm** achieves the highest SG score). Furthermore, though outperforming GPT2-tree on SG, these two variants still underperform most of their compositional SLM counterparts (**Bi-Dn-#-#** and **Bi-Up-#-#** respectively), which is consistent with our conclusion that explicit composition is helpful in syntactic generalization. The extremely bad SG performance of GPT2-tree-**Nb-Up** also coincides with the bad performance of **Nb-Up-Ex-Nm**, both of which apply additional modules to predict the start of constituents while failing in capturing the complicated interactions among varying numbers of sub-constituents (as also mentioned in section 3.2).

E Syntactic Generalization Details

We present the six detailed syntactic phenomenon scores of two baselines and the four best-performing SLMs in Figure 4. The results show that all four compositional SLMs gain consistent and significant improvement over two baselines on all six circuits. The results again strengthen the conclusion that properly modeling syntax and explicit composition are of help in syntactic generalization.

F Number of Model Forward Calls

We record the number of the main Transformer forward calls of running word-synchronous beam search on a sentence of twenty tokens and present it in Table 6 as a supplement for inference time

Model	bsz-10	bsz-30	bsz-100	bsz-300
GPT2-tree	147	202	236	250
Bi-Up-Ex-Nm	165	175	179	182
Bi-Up-Ex-M	165	170	180	183
Bi-Up-In-Nm	309	329	341	345
Bi-Up-In-M	309	327	339	351
Bi-Dn-Ex-Nm	167	178	221	239
Bi-Dn-Ex-M	158	201	212	237
Bi-Dn-In-Nm	267	300	339	407
Bi-Dn-In-M	275	295	324	403
Nb-Up-Ex-Nm	215	268	328	356
Nb-Up-Ex-M	385	390	399	405
Nb-Up-In-Nm	547	587	617	659
Nb-Up-In-M	713	725	747	763
Nb-Dn-Ex-Nm	129	145	168	190
Nb-Dn-Ex-M	159	182	219	248
Nb-Dn-In-Nm	280	339	402	425
Nb-Dn-In-M	310	352	416	435

Table 6: The number of model forward calls.

results. We exclude the forward calls of the external composition function, as it has a much smaller parameter size and each forward pass takes significantly less time.

G Discussion on Inference Time of Modeling Linearized Non-binary Trees

As shown in Table 6, compared to **Bi-#-#-#**, **Nb-#-#-#** generally incurs more forward calls (except in the case of **Bi-Dn-Ex-Nm** vs. **Nb-Dn-Ex-Nm**). This discrepancy arises from word-synchronous beam search, where at each synchronous step, the top-k beams of non-binary trees tend to exhibit wide variation in the number of compositions, which is significantly higher than in binary trees. Consequently, during each pair of synchronous steps, some beams have few composed constituents, while others have many more. Those with fewer composed constituents perform multiple compositions, while others immediately generate a new token and wait for synchronization, leading to a consistently high number of forward calls during each pair of the synchronous steps, a phenomenon absent in binary settings. This phenomenon is more pronounced in **Nb-Up-#-#** than in **Nb-Dn-#-#**, resulting in more forward calls for **Nb-Up-#-#**. We consider this an intrinsic challenge of applying word-synchronous beam search to SLMs that model non-binary trees.

H Generation Efficiency

Experiments presented in this section are conducted with a single A800 GPU.

For a fair comparison between GPT2-token and SLMs, we use exactly the same setup as in the sum-

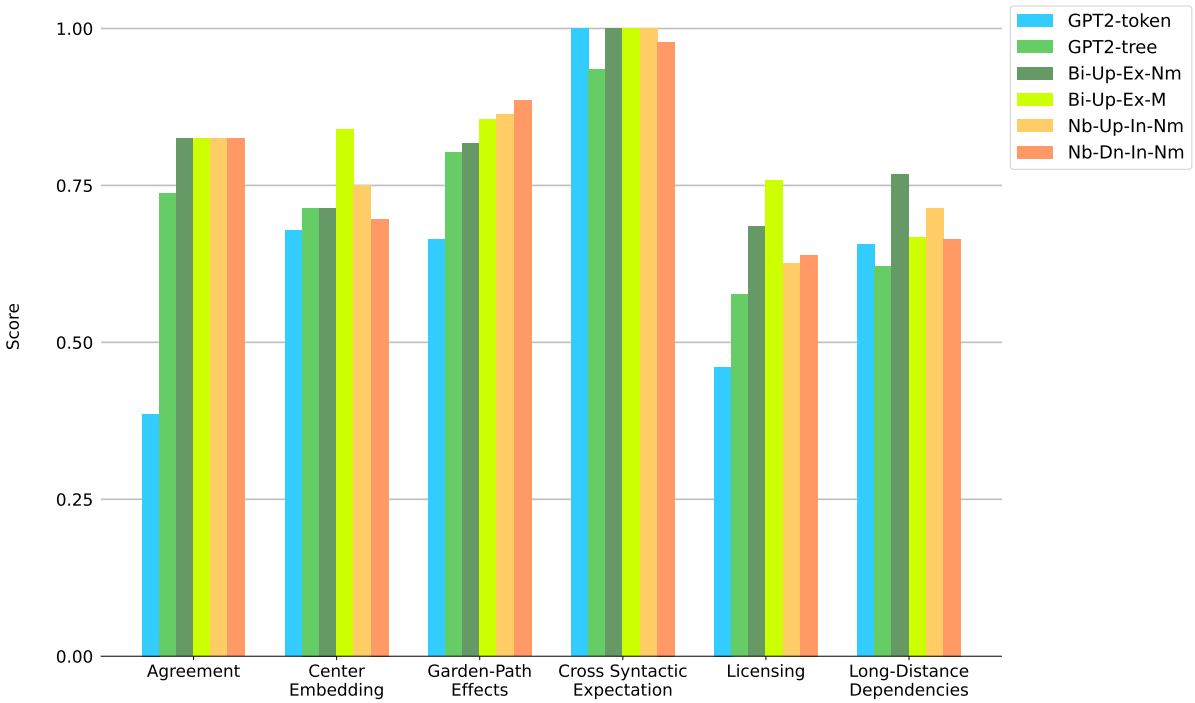


Figure 4: SG scores on each syntactic phenomenon.

Model	Time_Per_Token (ms)
GPT2-token	5.6
GPT2-tree	50.9
Bi-Up-Ex-Nm	26.1
Bi-Up-Ex-M	27.8
Bi-Up-In-Nm	57.1
Bi-Up-In-M	64.6
Nb-Up-In-Nm	65.5
Nb-Up-In-M	71.1
Nb-Dn-In-Nm	70.2
Nb-Dn-In-M	84.1

Table 7: The generation time (lower is better).

marization experiment (section 3.3.1), applying word-synchronous beam search to top-k random sampling (using a beam size of 2 and $k = 2$) for SLMs and use top-k random sampling ($k = 2$) for GPT2-token. We randomly pick 100 prompts with an average length of 106.16 (tokens) and run all the models to generate 100 tokens with each prompt. We report the average generation time per token in Table 7. The results show that all the SLMs show significantly lower efficiency than GPT2-token due to additionally encoding and generating syntactic structures, which is an intrinsic limitation of SLMs. On the other hand, `Ex` shows higher efficiency compared with `In`, which is consistent with the results in section 3.4.