Sneaking Syntax into Transformer Language Models with Tree Regularization 🌲

Ananjan Nandi, Christopher D. Manning & Shikhar Murty
Department of Computer Science
Stanford University
Stanford, CA 94305, USA
{ananjan,smurty}@stanford.edu

Abstract

While compositional accounts of human language understanding are based on a hierarchical tree-like process, neural models like transformers lack a direct inductive bias for such tree structures. Introducing syntactic inductive biases could unlock more robust and data-efficient learning in transformer language models (LMs), but existing methods for incorporating such structure greatly restrict models, either limiting their expressivity or increasing inference complexity. This work instead aims to *softly* inject syntactic inductive biases into given transformer *circuits*, through a structured regularizer. We introduce TREEREG, an auxiliary loss function that converts bracketing decisions from silver parses into a set of differentiable orthogonality constraints on vector hidden states. TREEREG integrates seamlessly with the standard LM objective, requiring no architectural changes. LMs pre-trained with TREEREG on natural language corpora such as WikiText-103 achieve up to 10% lower perplexities on out-of-distribution data and up to 9.5 point improvements in syntactic generalization, requiring less than half the training data to outperform standard LMs. TREEREG still provides gains for pre-trained LLMs: Continued pre-training of Sheared Llama with TREEREG results in improved syntactic generalization, and fine-tuning on MultiNLI with TREEREG mitigates degradation of performance on adversarial NLI benchmarks by 41.2 points. We release all code to guide future research[1].

1 Introduction

A substantial body of research (Crain and Nakayama, 1987; Pallier et al., 2011; van Schijndel et al., 2013; Hale et al., 2018) suggests that human language processing is inherently *hierarchical*: syntactic rules combine word-level meanings to give rise to semantics for larger constituents, which are then combined into sentences. In contrast, transformers (Vaswani et al., 2017), the architecture behind large language models (LLMs), allow for arbitrary, non-hierarchical routing of information while processing an input sentence.

LLMs are pre-trained on massive amounts of data to achieve reasonable generalization (Achiam et al., 2023; Dubey et al., 2024), but recent studies find that state-of-the-art LLMs still struggle with compositional generalization (Guo et al., 2020; Berglund et al., 2024; Hale and Stanojević, 2024), the ability to understand familiar words in novel contexts. Prior work (Socher et al., 2013; Eriguchi et al., 2016) proposed explicitly tree-structured models capable of data-efficient compositional generalization, but they are too constrained and not amenable to pre-training at scale.

Consequently, people have proposed inductive biases that encourage hierarchical computation in transformers, including Syntactic Language Models (SLMs, Sartran et al., 2022; Murty et al., 2023b) that model a joint distribution over words and syntax trees. While SLMs often improve data efficiency and syntactic generalization, they have several drawbacks: They often involve additional parameters to model syntax, constrain the attention mechanisms of the underlying model, or involve more complex and slower inference methodologies.

In this work, we instead devise a new differentiable loss function that *softly* injects syntactic inductive biases into a given *circuit* of the transformer: TREEREG. These biases are soft structural constraints that aim to ensure that the hidden state representation from the transformer circuit for each constituent in an input sentence is *maximally orthogonal* to that of its surrounding context (Figure 1). TREEREG is simply added as a regularizer to the LM loss during training. Crucially, an LM trained with TREEREG is *completely indistinguishable* from a standard LM in both architecture and inference mechanism.

[1] https://github.com/ananjan-nandi-9/tree_regularization

Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 8006–8024

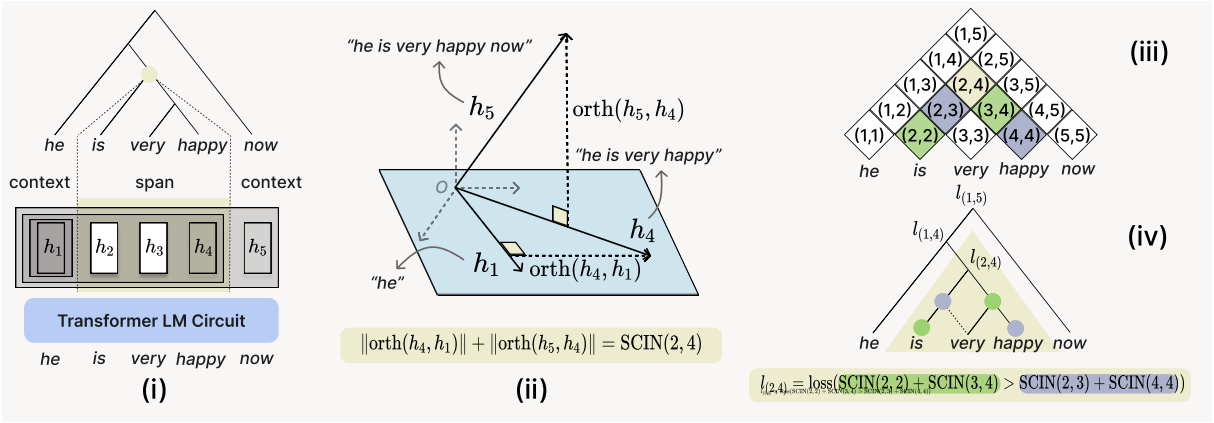

Figure 1: TREEREG loss ($\mathcal{L}_{\text{TR}}$) computation for $S =$ "he is very happy now". **(i)** Computation of vector hidden states $\boldsymbol{h}_i$ by passing S as input to some circuit of the LM. $\boldsymbol{h}_i$ is the representation for the prefix of S ending at i. **(ii)** Span Contextual Independence Score (SCIN, § 3.1) computation for "is very happy". Orthogonality constraints are enforced between span representation $\boldsymbol{h}_4$ and its context $\boldsymbol{h}_1$ and $\boldsymbol{h}_5$. **(iii)** Chart of SCIN for all spans in S. **(iv)** Possible bracketings of "is very happy" are ("is very", "happy") with score $\text{SCIN}(2,3) + \text{SCIN}(4,4)$ and ("is", "very happy") with score $\text{SCIN}(2,2) + \text{SCIN}(3,4)$. Loss for this span ($l_{(2,4)}$) encourages the second bracketing. $\mathcal{L}_{\text{TR}} = l_{(1,5)} + l_{(1,4)} + l_{(2,4)}$ includes analogous losses for spans "he is very happy" and "he is very happy now".

The use of TREEREG improves syntactic generalization across model and data scales. On diagnostic sentence transformation tasks such as tense inflection and question formation (McCoy et al., 2020), it significantly enhances grokking behavior (§ 4). It also improves syntactic generalization on the BLiMP (Warstadt et al., 2020) and SyntaxGym (Gauthier et al., 2020) test suites by up to 3.2 and 9.5 points, both when used during pre-training from scratch (§ 5), and during continued pre-training (§ 6.1). LMs with TREEREG surpass the syntactic generalization of baseline LMs with less than half the training data (§ 5.2). Most importantly, TREEREG improves out-of-distribution language understanding. When used during pre-training of LMs on fully parsed (§ 5.1) as well as a mixture of parsed and unparsed (§ 5.3) data, TREEREG reduces out-of-distribution perplexities by up to 9.2%. When TREEREG is used for fine-tuning an LLM on MultiNLI (Williams et al., 2018), it mitigates performance drops by up to 41.2 points on adversarial NLI benchmarks (§ 6.2). We release all code at `https://github.com/ananjan-nandi-9/tree_regularization`.

2 Background and Notation

Tree Structures in Language Processing. Consider a sentence $S = \{x_1, x_2, \ldots, x_n\}$ where x_j is the j^{th} token in the sentence. Let tree $T(S)$ over sentence S be a set of spans $S_{a;b} = \{x_a, \ldots, x_b\}$ such that if $S_{a;b}$ and $S_{b+1;c} \in S$, then $S_{a;c} \in S$. We say $S_{a;c}$ is *split* at index b to get spans $S_{a;b}$ and $S_{b+1;c}$ of $T(S)$. Prior work (Pallier et al., 2011; van Schijndel et al., 2013; Hale et al., 2018) suggests that human language processing is hierarchical—sentences contain "constituents" or spans of words that act as semantic units, with smaller constituents ("is", "very happy") recursively combining to form larger ones ("is very happy"), building up meaning incrementally. The constituency parse $\hat{T}(S)$ is the tree form by all constituents of S. Tree-structured models (Socher et al., 2013; Tai et al., 2015) build sentence representations through recursively bottom-up computation over constituency parses, whereas for transformers, any hierarchically structured computation is at most implicitly learned in the attention layers.

Hierarchical Structure in Transformers. Recent evidence suggests that pre-trained transformers behave similarly to tree-structured models on certain tasks (Murty et al., 2023c,a). The *tree projection* metric from Murty et al. (2023c) quantifies how well a transformer's computation can be approximated by a tree-structured model. It is based on the idea that bottom-up computations on trees create context-invariant representations at intermediate nodes. Therefore, the tree built from maximally context-invariant spans aligns most closely with the transformer's computation. Another line of work builds Syntactic Language Models (Sartran et al., 2022; Murty et al., 2023b; Hu et al., 2024) that learn joint distributions over inputs and their parses to introduce syntactic inductive biases into transformer language models. These ap-

proaches involve substantial changes to the transformer architecture—explicitly modifying attention patterns, as well as changing the output space or inference mechanisms. TREEREG provides an alternative, biasing transformer circuits towards hierarchical structure through regularization.

Evaluating Syntactic Generalization. Syntactic generalization refers to an LM's ability to apply syntactic rules implicitly learned during training to new, unseen data. We use the BLiMP (Warstadt et al., 2020) and SyntaxGym (SG, Gauthier et al., 2020) test suites to evaluate LMs for syntactic generalization. In BLiMP, models are presented with pairs of grammatical and ungrammatical sentences that differ minimally, and are evaluated on assigning lower perplexity to the grammatical sentences. The SG test suites evaluate six distinct syntactic phenomena, with each suite involving a particular inequality constraint related to the surprisal of different continuations of input sentences based on their prefixes. These constraints are rooted in theories of syntax; for example, assigning higher surprisal to the third word in the sentence "I know who you introduced to them." compared to "I know that you introduced to them."

3 Our Approach

TREEREG is a differentiable loss function that injects the hierarchical structure of input constituency parses into transformer circuits. Inspired by Murty et al. (2023c), TREEREG specifies a set of soft constraints to ensure that constituent representations are maximally independent of surrounding context. For any span, we quantify this independence with the Span Contextual Independence Score (SCIN, § 3.1). TREEREG maximizes SCIN for spans corresponding to input constituents while simultaneously minimizing it for other spans (§ 3.2). In an optimally trained model, the tree with the highest cumulative SCIN thus recovers the constituency parse. We then provide a greedy algorithm to extract the parse tree induced by TREEREG from a given model for any input sentence. A deeper discussion of the design choices underlying our method can be found in Appendix A.

3.1 Span Contextual Independence Score

Let $\boldsymbol{h}^l_{a,1}, \boldsymbol{h}^l_{a,2}, \ldots, \boldsymbol{h}^l_{a,n}$ be L_2-normalized vector hidden states for each token in S produced by attention head a at layer l of a causal auto-regressive transformer LM with multi-headed self-attention. Since most tasks benefit from a blend of hierarchical and unstructured computation, we only apply TREEREG on the circuit formed by a given subset of attention heads A at a given layer l.

Span vectors. For any $j \leq n$, $\boldsymbol{h}^l_{A,j}$ is the concatenation ($\#$) of vector representations from the attention heads in A at layer l for prefix $x_{\leq j}$:

$$\boldsymbol{h}^l_{A,j} = \#_{a \in A} \boldsymbol{h}^l_{a,j} \tag{1}$$

Then, for span $S_{i;j}$, $\boldsymbol{h}^l_{A,i-1}$ contains information before the start of $S_{i;j}$, or the *context* for the span.

Contextual Independence. We operationalize contextual independence in terms of orthogonality constraints between span representations. In particular, for $S_{i;j}$ to be contextually invariant, we expect $\boldsymbol{h}^l_{A,j}$ to be independent of, or *orthogonal to* $\boldsymbol{h}^l_{A,i-1}$. We also constrain $\boldsymbol{h}^l_{A,j+1}$ to be orthogonal to $\boldsymbol{h}^l_{A,j}$, as the information in $S_{i;j}$ is expected to be independent of what comes after it in the sentence. Therefore, we define SCIN^l_A for $S_{i;j}$ at layer l for attention heads in A as

$$\begin{aligned}\text{SCIN}^l_A(i,j) = &\|\text{orth}(\boldsymbol{h}^l_{A,j}, \boldsymbol{h}^l_{A,i-1})\| \\ &+ \|\text{orth}(\boldsymbol{h}^l_{A,j+1}, \boldsymbol{h}^l_{A,j})\|,\end{aligned} \tag{2}$$

where $\|\|$ is the L_2 norm, and $\text{orth}(x, y) = x - (x^\top y)y$ for any two normalized vectors x and y. For the sake of completeness, $\|\text{orth}(h_j, h_{-1})\|\ \forall j$ and $\|\text{orth}(h_{k+1}, h_k)\|$ are assumed to be 0. Since l and A are fixed during training, we simplify SCIN^l_A to SCIN from here on.

3.2 TREEREG Loss

Given sentence S, the TREEREG loss ($\mathcal{L}_{\text{TR}}$) biases the model towards driving up SCIN for all spans in the constituency parse $\hat{T}(S)$ while driving down SCIN for other spans. Specifically, for constituent $S_{i;j} \in \hat{T}(S)$ split at index p, we first compute split scores $s(i, q, j)$ for $i \leq q \leq j-1$ as

$$\begin{aligned}s(i,q,j) = &\text{SCIN}(i,q) \\ &+ \text{SCIN}(q+1,j).\end{aligned} \tag{3}$$

We then use these scores to compute a span-level log loss $l_{(i,j)}$. These losses are computed as

$$l_{(i,j)} = \log\left[\sum_{q=i}^{j-1} \exp s(i,q,j)\right] - s(i,p,j). \tag{4}$$

$\mathcal{L}_{\text{TR}}$ is then the sum of span-level losses $l_{(i,j)}$ for all constituents in $\hat{T}(S)$,

$$\mathcal{L}_{\text{TR}} = \sum_{S_{i;j} \in \hat{T}(S)} l_{(i,j)}. \quad (5)$$

In practice, $\mathcal{L}_{\text{TR}}$ is computed recursively top-down on $\hat{T}(S)$ following Alg. 1. $\mathcal{L}_{\text{TR}}$ is added as an auxiliary loss to the LM loss during training, resulting in the training objective $\mathcal{L}_{\text{LM}} + \alpha\mathcal{L}_{\text{TR}}$ where $\mathcal{L}_{\text{LM}}$ is the LM loss. Since both $\mathcal{L}_{\text{TR}}$ and $\mathcal{L}_{\text{LM}}$ are cross-entropy losses, α can generally be set to 1. $\mathcal{L}_{\text{TR}}$ and $\mathcal{L}_{\text{LM}}$ can also consume data from different datasets. For example, we can perform LM on a large pre-training dataset while passing batches from a smaller, parsed dataset to $\mathcal{L}_{\text{TR}}$.

Recovering parses during inference. During inference, we can use a top-down greedy decoding algorithm to recover the unique parse tree encoded in the TREEREG constraints for any given S from the hidden states of a circuit. Given SCIN scores for all spans in S, we recover the tree that maximizes the sum of SCIN scores across its spans. Specifically, the split for any span $S_{i;j}$ happens at the index $\hat{p}$ that maximizes $s(i, q, j)$,

$$\hat{p} = \arg\max_{q \in [i, j-1]} s(i, q, j). \quad (6)$$

$S_{i;\hat{p}}$ and $S_{\hat{p};j}$ can then be recursed on to obtain more constituents of this *induced* parse tree. Details of this method can be found in Alg. 2.

3.3 Implementation Details

SCIN computation does not require additional model calls and can be performed during the forward pass that computes $\mathcal{L}_{\text{LM}}$. SCIN scores for all spans in a sentence can be calculated simultaneously and efficiently using vectorization, as detailed in Appendix C. Additionally, updates from $\mathcal{L}_{\text{LM}}$ and $\mathcal{L}_{\text{TR}}$ can occur at different frequencies during training, for example, $\mathcal{L}_{\text{TR}}$ may be applied once every k steps of $\mathcal{L}_{\text{LM}}$. Despite the quadratic time complexity of computing $\mathcal{L}_{\text{TR}}$ (which scales with the square of input token count), these factors collectively ensure that TREEREG remains efficient in practice. As per our implementation, there is an increase of around 25% in training time if $\mathcal{L}_{\text{TR}}$ is applied once every 10 $\mathcal{L}_{\text{LM}}$ steps, on 25% of attention heads. This excludes the time required for parsing the data used in TREEREG. The parses are either included with the dataset (BLLIP-LG, MultiNLI) or can be computed once and reused for every training run.

4 Warm-up: Improving Grokking on Sentence Transformation Tasks

We start by training transformer LMs on two diagnostic tasks derived from PCFGs (See Appendix D for examples and data statistics). In Tense Inflection (TI), the model is provided with an input in the past tense, and required to generate the same input in the present tense. In Question Formation (QF), the model is required to transform a declarative sentence into a question. For QF, we report first word accuracy of the decoded question, and for TI, we report the fraction of test inputs for which the target verb is correctly inflected.

Setup. We train 4-layer transformer LMs for 500k steps (Base LM), performing extended training to enable grokking as reported in prior work (Murty et al., 2023a). For TREEREG LM, we autoparse both datasets with the Berkeley Neural Parser (Benepar, Kitaev et al., 2022), and use $\mathcal{L}_{\text{TR}}$ on 2 out of 8 attention heads using hidden states from layer 2 of the LM, once every 20 steps of $\mathcal{L}_{\text{LM}}$. To account for variance across training runs, we report averages as well as best performance at the end of training across 4 seeds (Table 1). We also report the training iteration (averaged over all runs) after which the performance on the test set converges.

Results. LMs trained with TREEREG grok faster and achieve higher performance than Base LMs on average, across datasets. Notably, on QF, TREEREG LM obtains a 57.5 pt gain over a standard LM (generalizing perfectly) while also grokking over 10 times faster on average.

Model	Avg. Acc. (↑)	Best Acc. (↑)	itr. (↓)
Tense Inflection (TI)			
Base LM	47.2 ± 16.7	71.1	427k ± 41k
TREEREG LM	**90.4 ± 6.3**	**98.3**	**391k ± 35k**
Question Formation (QF)			
Base LM	42.1 ± 15.4	66.9	460k ± 7k
TREEREG LM	**99.6 ± 0.7**	**100.0**	**43k ± 26k**

Table 1: Evaluations on TI and QF tasks. We report averaged test accuracy across seeds (**Avg. Acc.**), the best test accuracy (**Best Acc.**), and the average iteration after which test performance converges (**itr.**). TREEREG LMs grok faster and achieve better performance.

Model	Syntactic Generalization (↑)		PPL (↓)		Modified Architecture	Inference Overhead
	BLiMP	SG	BLLIP	PTB		
Base LM	72.2	71.9	21.6	49.1	-	-
PLM	75.1	80.2	29.8	-	✗	✓
TG	-	**82.5**	30.3	-	✓	✓
PUSHDOWN LM	**75.6**	82.3	**19.9**	-	✓	✓
TREEREG LM	**74.8**	**80.0**	22.3	**44.6**	✗	✗

Table 2: Syntactic Generalization on BLiMP and SG test suites, and perplexities (**PPL**) on BLLIP-LG test set (**BLLIP**) and PTB test set (**PTB**) for models trained on BLLIP-LG. Results for PLM, TG and PUSHDOWN LM are taken from Murty et al., 2023b. TREEREG LMs show better syntactic generalization and better generalization to PTB compared to the Base LM.

5 Sentence-Level Language Modeling

Can TREEREG provide gains when introduced in pre-training? We investigate this under settings where data passed to $\mathcal{L}_{\text{LM}}$ and $\mathcal{L}_{\text{TR}}$ is the same (§ 5.1) and different (§ 5.3).

5.1 Language Modeling on BLLIP-LG

Setup. We train 16-layer LMs on the BLLIP-LG dataset (Hu et al., 2020) (Base LM, detailed hyperparameters in Appendix F). For TREEREG LM, we use $\mathcal{L}_{\text{TR}}$ at layer 12 of the LM, on 2 out of 8 attention heads, once every 10 steps of $\mathcal{L}_{\text{LM}}$, on the already-parsed BLLIP-LG dataset. A discussion of the layer and attention heads used for $\mathcal{L}_{\text{TR}}$ can be found in Appendix G. We benchmark syntactic generalization using the BLiMP and SG test suites (See § 2). To evaluate out-of-distribution generalization, we report perplexity on the Penn TreeBank (PTB; Marcus et al., 1993) test set.

To the best of our knowledge, all prior approaches to induce hierarchical computations in autoregressive causally-trained transformer language models do not leave the underlying transformer architecture untouched, and thus are not directly comparable with TREEREG. Nevertheless, we also report results for Parsing as Language Model (PLM; Qian et al., 2021), Transformer Grammars (TG; Sartran et al., 2022), and Pushdown Layers (PUSHDOWN LM; Murty et al., 2023b) in Table 2.

Results. TREEREG achieves a 2.6 point (pt) gain on BLiMP and 8.1 pt overall gain on SG test suites. Figure 2 shows TREEREG improves over the baseline on 4 out of 6 SG test suites, with a notable 17 pt gain on Licensing. TREEREG LM also generalizes better to PTB with 9.2% lower perplexity, with a marginal 0.7 pt perplexity increase on BLLIP-LG.

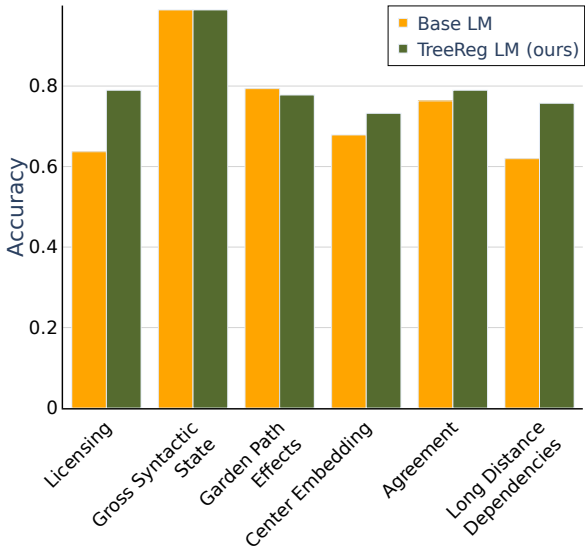

Figure 2: Comparing TREEREG LM with Base LM from Table 2 on SG test suites. TREEREG LM outperforms the Base LM on 4 out of 6 test suites, with 1 tie.

While PLM, TG and PUSHDOWN LM outperform TREEREG at syntactic generalization on BLiMP and SG test suites, they require additional parameters or incur inference overheads. Except for Pushdown LM, these models also have substantially higher perplexities on the BLLIP-LG test set compared to TREEREG, showing the benefits of the unchanged transformer architecture.

5.2 Sample Efficiency

Setup. To measure sample efficiency gains from TREEREG, we train 16-layer LMs with the same configuration as § 5.1 on [10, 50, 100]% of BLLIP-LG. Only parses of the data used for LM is used for TREEREG in each case. We present syntactic generalization results on SG in Figure 3.

Results. While syntactic generalization improves with more training data for both the Base and

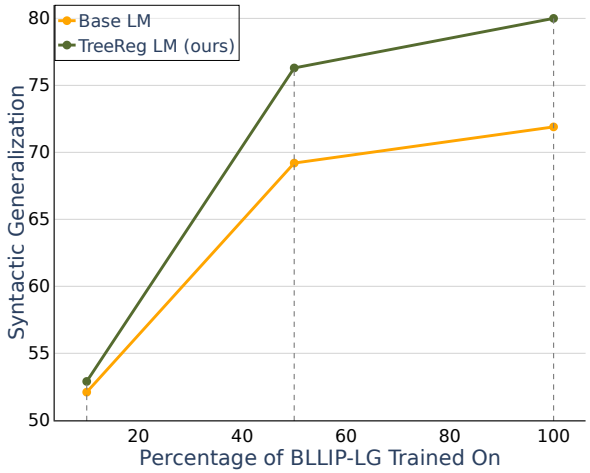

Figure 3: Plot of Syntactic Generalization on SG test suites vs Percentage of BLLIP-LG data used to train LMs from scratch. TREEREG LM exceeds the maximum syntactic generalization performance of Base LM with less than 50% of the data.

TREEREG LMs, TREEREG LMs outperform the baseline across all settings. Moreover, the performance gap widens as the amount of training data increases. Notably, TREEREG surpasses the baseline's maximum syntactic generalization performance on SG by 4.4 points when trained on only 50% of the data, demonstrating substantially more sample-efficient syntactic generalization. Therefore, even though TREEREG increases training time by 25% compared to a baseline trained for the same number of iterations (see § 3.3), it reaches better syntactic generalization in only 62.5% of the baseline's training time.

5.3 Language Modeling on WikiText

Setup. Auto-parsing text corpora may not always be practical for large-scale language modeling, particularly when the corpus is very large or when the text lacks clear syntactic structure, as is the case for web data. Therefore, we experiment with a setting where the pre-training and parsed data come from different data sources. We train a 12-layer GPT-2-small model (Base LM) on chunks of 1024 tokens from WikiText-103 (**?**) with the exact hyperparameters and tokenization of GPT-2 small (with the addition of dropout ($= 0.1$) to prevent overfitting). For TREEREG LM, we reuse the BLLIP-LG parses and use $\mathcal{L}_{\text{TR}}$ at layer 4, on 3 out of 12 attention heads, once every 10 steps of $\mathcal{L}_{\text{LM}}$. We also test a setting where LM training is additionally performed on one batch from BLLIP-LG for every 20 batches from WikiText-103 (Table 3) for both Base LM and TREEREG LM. PTB is again used to evaluate out-of-distribution generalization.

Model	Syntactic Generalization (↑)		PPL (↓)	
	BLiMP	SG	WikiText	PTB
No LM on BLLIP-LG				
Base LM	71.0	69.4	**17.5**	**331.5**
TREEREG LM	**72.5**	**73.7**	17.8	411.1
LM on BLLIP-LG				
Base LM	71.5	67.2	**18.3**	53.2
TREEREG LM	**74.1**	**76.7**	19.7	**50.8**

Table 3: Syntactic Generalization on BLiMP and SG test suites, and perplexities on the WikiText and PTB test sets for GPT-2-small models trained on WikiText, with and without LM on BLLIP-LG. TREEREG LMs show better syntactic generalization in both settings.

Results. We observe improvements in syntactic generalization with TREEREG in both settings, obtaining up to 2.6 pt gain on BLiMP and 9.5 pt overall gain on SG test suites. The gains are relatively smaller when LM training is not performed on BLLIP-LG, and perplexities on PTB increase by 24% over the baseline. In this setting, we find that perplexities are also high on BLLIP-LG, averaging around 300. As a result, we conjecture that the hidden states used in TREEREG are not adequately trained in this setting, leading to worse out-of-distribution perplexities.

In the other setting, Base LM struggles to utilize the LM training on BLLIP-LG, resulting in a 2.2 pt overall drop on SG. In contrast, TREEREG shows an overall increase of 3 pt on SG test suites. Perplexities on PTB are again 2.4 pts lower with TREEREG in this setting compared to Base LM, showing better generalization from the BLLIP-LG LM training.

6 Continued Pre-training and Fine-tuning

Next, we apply TREEREG to LLMs. In particular, we use the Sheared Llama-1.3B model of Xia et al. (2024) and explore TREEREG under continued pre-training and fine-tuning settings.

6.1 Continued Pre-training on BLLIP-LG

Setup. We perform continued pre-training of Sheared Llama-1.3B (Base LM) on BLLIP-LG, with a context window of 512 tokens, to get CPT LM. For TREEREG LM, BLLIP-LG parses are used, and $\mathcal{L}_{\text{TR}}$ is applied once every 5 steps of $\mathcal{L}_{\text{LM}}$, on 6 out of 24 attention heads at layer 16 of the LLM. We report scores on the BLiMP and SG test suites and perplexity on PTB (Table 4).

Model	Syntactic Generalization (↑)		PPL (↓)	
	BLiMP	SG	BLLIP	PTB
Base LM	73.6	80.7	23.8	42.5
CPT LM	80.7	83.9	**9.24**	15.2
TreeReg CPT LM	**81.9**	**85.5**	9.41	**14.3**

Table 4: Syntactic Generalization on BLiMP and SG test suites, and perplexities on BLLIP-LG and PTB test sets for Sheared Llama-1.3B (**LM**) continued pre-trained (**CPT**) on BLLIP-LG. With TreeReg, CPT LM shows better syntactic generalization and PTB perplexities.

Model	MultiNLI (↑)		Adversarial (↑)	
	Matched	Mismatched	MoNLI	MED
Base LM	35.5	35.1	50.3	50
FT LM	65.9	66.3	1.9	43.1
TreeReg FT LM	**68.1**	**68.0**	**43.5**	**45.8**

Table 5: NLI accuracies on MultiNLI test splits and two **Adversarial** NLI evaluation datasets for Sheared Llama-1.3B (**LM**) finetuned (**FT**) on MultiNLI. TreeReg results in more gains on MultiNLI and lower decreases on the adversarial benchmarks.

Results. TreeReg results in a 1.2 pt gain on BLiMP and 1.6 pt overall gain on SG test suites, along with a 0.9 pt decrease in PTB perplexity. These improvements are less significant than those reported in § 5.1, which we conjecture is due to the greater challenge in rewiring the unstructured inductive biases learned by the LLM during large-scale pre-training.

6.2 Fine-tuning on MultiNLI

Can TreeReg prevent LLMs fine-tuned on classification tasks from learning spurious shortcuts? To answer this, we consider Natural Language Inference (NLI), a task that involves classifying the logical relationship between a *premise* and a *hypothesis* sentence as *entailment*, *contradiction*, or *neutral*. Prior work (McCoy et al., 2019; Geiger et al., 2020) suggests that models often learn spurious shortcuts when trained on broad coverage NLI datasets, causing them to fail on diagnostic out-of-distribution datasets.

Setup. We finetune Sheared Llama-1.3B (Base LM) on MultiNLI (Williams et al., 2018) (training details and evaluation methodology in Appendix E) to get FT LM. For TreeReg LM, we binarize the parses of the premise and hypothesis sentences provided in the dataset and apply $\mathcal{L}_{\text{TR}}$ at layer 16 on 6 out of 24 attention heads, every 10 steps of $\mathcal{L}_{\text{LM}}$. We report accuracy on the MultiNLI test set, which includes a *matched* split from the same sources as the training data, and a *mismatched* split from different sources. We also present results (Table 5) on two diagnostic datasets: MoNLI (Geiger et al., 2020) and MED (Yanaka et al., 2019).

Results. We note improvements of up to 2.2 pt on MultiNLI from TreeReg. Next, we find a decrease of 48.4 pt on MoNLI and 6.9 pt on MED when Base LM is finetuned on MultiNLI. With TreeReg, the finetuned model is substantially more robust, with a moderate decrease of 6.8 pt on MoNLI and 4.2 pt on MED compared to Base LM. We conclude that TreeReg discourages the LM from learning spurious shortcuts for this task.

7 Analysis

Parsing. TreeReg induces a tree-structured inductive bias in transformer LMs. We explore how well these induced trees align with given constituency parses. In particular, we use the procedure detailed in § 3.2 to recover induced parses from TreeReg LM (§ 5.1) and TreeReg CPT LM (§ 6.1) trained on BLLIP-LG. We evaluate parsing on the auto-parsed BLLIP-LG and the manually annotated PTB test sets. Additionally, to estimate parsing on domains other than Newswire, we evaluate parsing on the 4000 Questions dataset (Judge et al., 2006). Since TreeReg induces unlabeled binarized parses, we report unlabeled F1 scores against binarized parses in Table 6.

Model	BLLIP (↑)	PTB (↑)	4kQ (↑)
TreeReg LM	**95.2**	88.4	91.6
TreeReg CPT LM	94.6	89.1	**94.5**
Kitaev et al., 2022	-	**94.7**	87.7

Table 6: Unlabeled F1 scores against parses from the BLLIP-LG, PTB and 4000 Questions (4kQ) test sets for TreeReg LMs trained from scratch (**TreeReg LM**) and continued pre-trained from Sheared Llama-1.3B (**TreeReg CPT LM**) on BLLIP-LG. F1 scores are near or above 90 for all datasets. We also present unlabeled F1 scores from Benepar (Kitaev et al., 2022) for comparison.

Although TreeReg only implicitly encodes constituency parses in transformer hidden states, we observe that it demonstrates remarkable parsing capabilities. Parses induced by TreeReg align closely with given parses on all datasets, with F1 scores near or above 90. Parsing per-

formance on PTB and 4000 Questions is slightly better with TREEREG CPT LM, likely due to improved representations from Sheared Llama-1.3B (which benefits from large-scale pre-training) on these datasets, that are relatively out-of-distribution from the BLLIP-LG training data. Notably, induced parses from TREEREG align better with silver parses than Benepar (Kitaev et al., 2022) on the 4000 Questions datasets, by 6.8 F1 pts. We provide some example parses on each dataset in Appendix H.

Emergent structure across layers. We investigate the evolution of tree structure across layers in models trained with TREEREG. To do this, we first infer trees for the BLLIP-LG test set from the circuit formed by the attention heads used to train TREEREG, at each layer of the 16-layer TREEREG LM from § 5.1 (trained with $\mathcal{L}_{\text{TR}}$ at layer 12). We report unlabeled F1 scores of parses recovered at every layer against given parses in Figure 4.

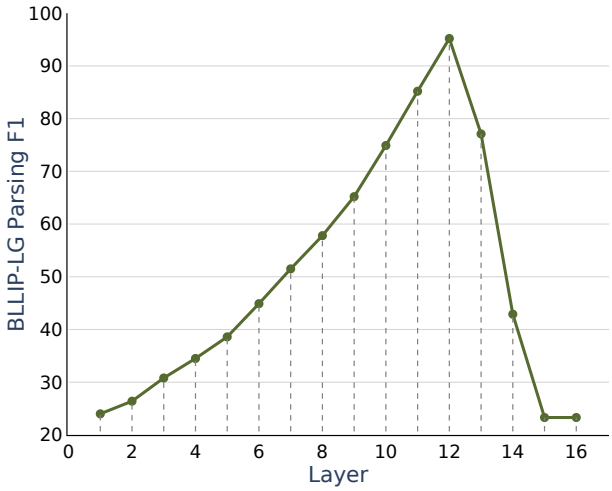

Figure 4: Unlabeled F1 scores on the BLLIP-LG test set for parse trees induced from every layer of a 16-layer TREEREG LM trained on BLLIP-LG with $\mathcal{L}_{\text{TR}}$ at layer 12. Circuits become increasingly tree-structured till layer 12, then rapidly become unstructured.

The F1 scores reveal that at layer 1, the parses are nearly random. As we progress to layer 12, they increasingly align with given parses. Beyond layer 12, there is a sharp drop in F1 scores back to near-random by layer 16. This suggests that the circuit involved in $\mathcal{L}_{\text{TR}}$ becomes increasingly aligned with given parses up to layer 12, and then rapidly loses this structure. We conclude that to perform next-word prediction, TREEREG LMs implement a non-syntactic function over the syntactic representations built at layer 12. These syntactic representations are, in turn, built gradually over the preceding layers of the transformer, even in the absence of explicit supervision from TREEREG at these layers.

Training on Randomized Parses. To assess if the gains from TREEREG are specifically due to hierarchical computation that align closely with constituency parses, we conduct a controlled experiment. We train a 16-layer LM using the exact setup described in § 5.1, but instead of using gold parses, we provide TREEREG with random parses of BLLIP-LG sentences (Randomized TREEREG LM). These randomized parses are generated top-down recursively, by choosing random split points for each sentence span. We then compare performance with the TREEREG LM and Base LM on the BLiMP and SG test suites (Table 7).

Model	BLiMP (↑)	SG (↑)
Base LM	72.2	71.9
TREEREG LM	**74.8**	**80**
Randomized TREEREG LM	73.4	71.8

Table 7: Results on BLiMP and SG test suites for LMs trained from scratch on BLLIP-LG, with silver and randomized parses (**Randomized TREEREG LM**) passed to TREEREG. Performance decreases with randomized parses compared to silver parses.

As expected, overall performance on the SG test suites drops marginally compared to Base LM when randomized parses are fed to TREEREG. On the other hand, BLiMP accuracies increase by 1.2 pts even in this setting. We hypothesize that this discrepancy arises from BLiMP's less granular sentence-level perplexity-based evaluation, in contrast to the surprisal-based approach used in SG.

Dependence on amount of parsed data. We train 16-layer LMs on BLLIP-LG using TREEREG, with the same setup as § 5.1, but vary the percentage of parsed BLLIP-LG used for TREEREG across [1, 5, 10, 20, 40, 60, 80, 100]%. We report the overall SG test suite performance for these settings in Figure 5.

As the amount of parsed data provided to TREEREG increases, we see a general trend of improvement in syntactic generalization on the SG test suites. Remarkably, we see improvements over Base LM from TREEREG even when it is provided with parses for only 1% of the data used for LM.

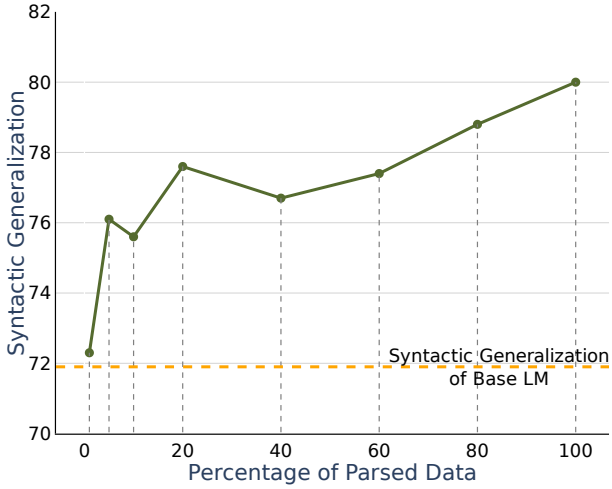

Figure 5: Syntactic Generalization on SG test suites vs Percentage of parsed BLLIP-LG data provided to TREEREG, for LMs trained from scratch on BLLIP-LG. Even with 1% of the data, TREEREG LMs have better syntactic generalization than baseline LMs.

8 Other Related Work

Generalization Failures of Transformer LMs. A substantial body of research has pointed out the limitations of transformer-based LMs in achieving robust compositional generalization. They have been shown to struggle with simple tasks, such as determining the parity of binary strings (Bhattamishra et al., 2020; Chiang and Cholak, 2022), balancing bracket sequences (Hahn, 2020), as well as more complex diagnostic natural language tasks (McCoy et al., 2020; Geiger et al., 2020; Dziri et al., 2023). Large-scale pre-training does not fully resolve these issues. For instance, Berglund et al. (2024) shows that GPT-4 (Achiam et al., 2023) fails to generalize on bijective relationships (for example, answering "Who is Tom Cruise's mother?" correctly but not "Who is Mary Lee South's son?"). Hale and Stanojević, 2024 also find that Gemini Pro (Team et al., 2023) does not learn syntactic universals such as the Final-over-Final condition (Sheehan et al., 2017) for low-resource languages such as Basque.

Tree-structure in Neural Networks. Building on theories of tree-structured human language processing, prior work explores various tree-structured model architectures (Socher et al., 2013; Tai et al., 2015; Le and Zuidema, 2015; Dyer et al., 2016; Shen et al., 2019; Hu et al., 2021). These models often excel at data-efficient compositional generalization due to their inherently hierarchical computation. However, they have been largely overshadowed by non-hierarchical pre-trained transformer-based LLMs (Brown et al., 2020; Achiam et al., 2023).

Linguistic Structure in Transformers. One promising alternative to tree-structured models is to inject syntactic inductive biases into transformers either by jointly modeling sentences and parse structure (Qian et al., 2021; Sartran et al., 2022; Murty et al., 2023b, among others), or via constraints on self-attention (Strubell et al., 2018; Wang et al., 2019; Deshpande and Narasimhan, 2020; Sartran et al., 2022). Although these models bring data efficiency and generalization improvements, they often require additional parameters, impose rigid constraints on attention mechanisms, or complicate inference. To the best of our knowledge, TREEREG is the first approach to introduce explicit, albeit soft, syntactic biases into the transformer without modifying the architecture. This is achievable because standard transformers can encode hierarchical languages of bounded depth (Yao et al., 2021) when trained appropriately.

9 Conclusion

We propose TREEREG, a structured regularizer that injects soft syntactic inductive biases into given circuits of a transformer LM, converting constituency parses of input sentences into differentiable orthogonality constraints on vector hidden states. Without requiring any architectural changes to the transformer, models pre-trained with TREEREG achieve up to 10% lower perplexities on out-of-distribution data and enhance syntactic generalization by up to 9.5 points on standard test suites. When applied to pre-trained LLMs, TREEREG improves syntactic generalization during continued pre-training and mitigates degradation on adversarial NLI benchmarks by up to 41.2 points when used during fine-tuning. TREEREG more than doubles the sample efficiency of syntactic generalization and does not require the entire training dataset to be parsed, making it practical and efficient. To the best of our knowledge, this work is among the first to translate insights from a mechanistic interpretability-related work (Murty et al., 2023c) into an actionable modeling innovation. We leave unsupervised alternatives to TREEREG, pre-training at scale with TREEREG, and application of TREEREG to languages other than English as future work.

Limitations

TREEREG requires constituency-parsed datasets, which might be difficult to obtain for languages other English due to the lack of easily available constituency parsers. This also makes TREEREG inapplicable to languages that do not have constituency structure, and we limit the experiments presented here to English. TREEREG also introduces some additional hyperparameters that require tuning, such as the layer and subset of attention heads on which TREEREG is be applied. As a heuristic, applying TREEREG at the middle layer of the model, on 25% of the attention heads, tends to work well across settings (see our ablations in Appendix G for more details). Finally, while we optimize TREEREG computation through vectorization, it still adds some computational overhead during training.

Acknowledgments

We thank Ananth Agarwal, Jasper Jian, Derek Chong, Harshit Joshi, Martijn Bartelds and other members of the Stanford NLP Group, and the reviewers for discussions and feedback.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Zeyuan Allen-Zhu and Yuanzhi Li. 2023. Physics of language models: Part 1, learning hierarchical language structures. *arXiv preprint arXiv:2305.13673*.

Lukas Berglund, Meg Tong, Maximilian Kaufmann, Mikita Balesni, Asa Cooper Stickland, Tomasz Korbak, and Owain Evans. 2024. The reversal curse: LLMs trained on "A is B" fail to learn "B is A". In *Proceedings of the International Conference on Learning Representations*.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020. On the Ability and Limitations of Transformers to Recognize Formal Languages. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

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

David Chiang and Peter Cholak. 2022. Overcoming a theoretical limitation of self-attention. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

John Cocke. 1969. *Programming languages and their compilers: Preliminary notes*. New York University.

Stephen Crain and Mineharu Nakayama. 1987. Structure dependence in grammar formation. *Language*, pages 522–543.

Ameet Deshpande and Karthik Narasimhan. 2020. Guiding attention for self-supervised learning with transformers. In *Findings of the Association for Computational Linguistics: EMNLP*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang (Lorraine) Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. 2023. Faith and fate: Limits of transformers on compositionality. In *Proceedings of the International Conference in Neural Information Processing Systems*.

Akiko Eriguchi, Kazuma Hashimoto, and Yoshimasa Tsuruoka. 2016. Tree-to-sequence attentional neural machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Jon Gauthier, Jennifer Hu, Ethan Wilcox, Peng Qian, and Roger Levy. 2020. Syntaxgym: An online platform for targeted evaluation of language models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics: System Demonstrations*.

Atticus Geiger, Kyle Richardson, and Christopher Potts. 2020. Neural natural language inference models partially embed theories of lexical entailment and negation. In *Proceedings of the BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*.

Yinuo Guo, Zeqi Lin, Jian-Guang Lou, and Dongmei Zhang. 2020. Hierarchical poset decoding for compositional generalization in language. In *Proceedings of the International Conference on Neural Information Processing Systems*.

Michael Hahn. 2020. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171.

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

John T. Hale and Miloš Stanojević. 2024. Do LLMs learn a true syntactic universal? In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

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 Annual Meeting of the 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. *arXiv preprint arXiv:2403.08293*.

Xiang Hu, Haitao Mi, Zujie Wen, Yafang Wang, Yi Su, Jing Zheng, and Gerard de Melo. 2021. R2D2: Recursive transformer based on differentiable tree for interpretable hierarchical language modeling. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

John Judge, Aoife Cahill, and Josef van Genabith. 2006. QuestionBank: Creating a corpus of parse-annotated questions. In *Proceedings of the International Conference on Computational Linguistics and Annual Meeting of the Association for Computational Linguistics*.

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

Nikita Kitaev, Thomas Lu, and Dan Klein. 2022. Learned incremental representations for parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Phong Le and Willem Zuidema. 2015. Compositional distributional semantics with long short term memory. In *Proceedings of the Joint Conference on Lexical and Computational Semantics*.

Mitch Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330.

R Thomas McCoy, Robert Frank, and Tal Linzen. 2020. Does syntax need to grow on trees? sources of hierarchical inductive bias in sequence-to-sequence networks. *Transactions of the Association for Computational Linguistics*, 8:125–140.

Tom McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher Manning. 2023a. Grokking of hierarchical structure in vanilla transformers. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*.

Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher Manning. 2023b. Pushdown layers: Encoding recursive structure in transformer language models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher D Manning. 2023c. Characterizing intrinsic compositionality in transformers with tree projections. In *Proceedings of the International Conference on Learning Representations*.

Christophe Pallier, Anne-Dominique Devauchelle, and Stanislas Dehaene. 2011. Cortical representation of the constituent structure of sentences. *Proceedings of the National Academy of Sciences*, 108(6):2522–2527.

Peng Qian, Tahira Naseem, Roger Levy, and Ramón Fernandez Astudillo. 2021. Structural guidance for transformer language models. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

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.

Michelle Sheehan, Theresa Biberauer, Ian Roberts, and Anders Holmberg. 2017. *The Final-Over-Final Condition: A Syntactic Universal*, volume 76. MIT Press.

Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. 2019. Ordered neurons: Integrating tree structures into recurrent neural networks. In *Proceedings of the International Conference on Learning Representations*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

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

Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the International Conference on Neural Information Processing Systems*.

Yaushian Wang, Hung-Yi Lee, and Yun-Nung Chen. 2019. Tree transformer: Integrating tree structures into self-attention. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*.

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.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. 2024. Sheared llama: Accelerating language model pre-training via structured pruning. In *Proceedings of the International Conference on Learning Representations*.

Hitomi Yanaka, Koji Mineshima, Daisuke Bekki, Kentaro Inui, Satoshi Sekine, Lasha Abzianidze, and Johan Bos. 2019. Can neural networks understand monotonicity reasoning? In *Proceedings of the ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*.

Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. 2021. Self-attention networks can process bounded hierarchical languages. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

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

A Discussion of Design Decisions

A.1 Last Hidden State as Representation for Prefix

We use the hidden state for the last token in a span as an easily computable representation of the span. There is some evidence suggesting that in the middle layers of an autoregressive transformer language model, the burden of computation of representations tends towards the right, indicating that intermediate hidden states summarize corresponding prefixes to some extent. For instance, Allen-Zhu and Li, 2023 find that when a transformer is trained on generations from a Probabilistic Context-Free Grammar (PCFG), ancestors of non-terminals are encoded at the hidden state of the last token in the span produced from the non-terminal. Therefore, we leverage the prefix information summarized in the hidden state of the last token of a span and rely on the LM's substantial expressive capabilities to embed TREEREG's orthogonality constraints into its hidden states.

A.2 Orthogonality for Operationalizing Contextual Independence

We choose to operationalize contextual independence between spans as orthogonality of correspoding hidden states, to design an efficiently computable loss function. There exist other ways of operationalizing contextual independence but these are inefficient and often require multiple forward passes on the transformer. For instance, Murty et al., 2023c defines contextual independence as the similarity between the sum of hidden states for a span with unconstrained attention, compared to those from a transformer with masked attention for

the prefix of the span. Another efficient alternative is using the negative absolute value of cosine similarity between span vector representations to represent contextual independence. This does not work as well, resulting in a syntactic generalization of 76.4 on SG test suites (a drop of 3.6 points) and 70.6 on BLiMP (a drop of 4.2 points) when used during the pretraining of a 16-layer LM from scratch on BLLIP-LG (§ 5.1).

TreeReg softly encourages higher SCIN for constituents of the parse tree, and lower SCIN for other constituents. However, a low SCIN score indicates greater similarity between span vectors, due to a smaller orthogonal component between them, which can result in degenerate hidden states. In practice, this is mitigated by opposing pressure from $\mathcal{L}_{\text{LM}}$, that encourages divergent hidden states for different tokens in the sentence. To confirm this, we compute the average SCIN for constituent and non-constituent spans in the PennTreeBank test dataset, for the Base LM and TREEREG LM trained in § 5.1. For the TREEREG LM, the average SCIN for constituents is 1.29, and that for non-constituents is 0.98. In contrast, for the Base LM, the average SCIN for constituents is 0.66, and that for non-constituents is 0.65. We therefore conclude that the average SCIN for non-constituents is substantially positive for a model trained with TREEREG, and in fact is even higher than that for a model trained without TREEREG.

A.3 Computation of SCIN

Our method for computation of SCIN involves the addition of two terms, the first one being $\|\text{orth}(\boldsymbol{h}_j, \boldsymbol{h}_{i-1})\|$, which captures the contextual independence between the information in a span and its prefix. The second term is $\|\text{orth}(\boldsymbol{h}_{j+1}, \boldsymbol{h}_j)\|$, which is a measure of how much the suffix of a span is dependent on the information contained in the span. The information in constituent spans should be mostly self-contained in a tree-structured computation, and therefore the hidden state after the span is expected to be independent of it. While this could result in enforced orthogonalities between consecutive words that may belong to the same constituent, TREEREG only enforces soft constraints, and there is opposing pressure from $\mathcal{L}_{\text{LM}}$ mitigating any undesirable behavior. We also found that TREEREG improves syntactic generalization even without this future-masking constraint, reaching 77.4 on SG test suites (a drop of 2.6 points) and 72.8 on BLiMP (a drop of 2.0 points) when used during pre-training of a 16-layer LM from scratch on BLLIP-LG (§ 5.1).

A.4 Greedy Decoding for Recovering Parses

An alternative to the proposed top-down greedy decoding algorithm for recovering induced parse trees (§3.2) is a dynamic programming approach like the CKY algorithm (Cocke, 1969; Kasami, 1966; Younger, 1967). However, while computing $\mathcal{L}_{\text{TR}}$ during training, we only apply supervision on spans that are possible splits of constituents in the silver parse tree. For example, when applying TREEREG to "the quick brown fox jumped over the river", there is no supervision on the SCIN for the span "fox jumped over", as "fox" is a part of the constituent "the quick brown fox", and "jumped over" is a part of the constituent "jumped over the river". A globally optimal parsing algorithm such as CKY will also consider SCIN for spans that had no supervision on similar spans during training, and these can be arbitrarily large, resulting in worse parsing performance.

B Algorithms for Computing TREEREG Loss and Induced Parse Trees

Alg. 1 details the procedure used to compute TREEREG loss for a given sentence. Alg. 2 details the procedure used to recover induced parsed trees from TREEREG-trained models.

Algorithm 1: TREEREG Loss

function loss($SCIN, P, i, j$);
// Start: loss($SCIN, P, 0, n$)
Input :
$SCIN$: Dictionary of SCIN for all spans
P: Dictionary of split points of constituents in $\hat{T}(S)$
i: Starting index of span $S_{i;j}$
j: Ending index of span $S_{i;j}$
Output : $l_{i,j}$, contribution to $\mathcal{L}_{\text{TR}}$ from $S_{i;j}$
if $j - i < 1$ **then**
  /* If span has length 1 or 2, only one possible split */
  return 0;
scores = $[SCIN(i, q) + SCIN(q + 1, j)$ for $q \in [i, j - 1]]$
p = $P(S_{i;j})$
span_loss = $cross\_entropy(scores, p)$
left_loss = loss($SCIN, P, i, p$)
right_loss = loss($SCIN, P, p + 1, j$)
return left_loss + span_loss + right_loss

Algorithm 2: TREEREG Probe

```
function parse(SCIN, i, j);
// Start: parse(SCIN, 0, n)
Input :
SCIN : Dictionary of SCIN for all spans,
i: Starting index of span S_{i;j}
j: Ending index of span S_{i;j}
Output : Constituents in parse tree induced
         by SCIN for S_{i;j}
if i == j then
    /* If span has length 1,
       terminate recursion */
    return S_{st;st};
split_point = argmax_{q∈[i,j−1]}
  SCIN(i, q) + SCIN(q + 1, j)
left_constituents =
  parse(SCIN, i, split_point)
right_constituents =
  parse(SCIN, split_point + 1, j)
return S_{i;j} + left_constituents +
  right_constituents
```

C Efficient Computation of SCIN

In this section, we detail our approach to efficiently compute SCIN for all spans in a sentence simultaneously through vectorization. For sentence S, let $\mathbf{H}$ be the tensor containing all hidden states $\boldsymbol{h}^l_{A,j}$:

$$\mathbf{H} = [\boldsymbol{h}^l_{A,1}, \ldots, \boldsymbol{h}^l_{A,n}] \tag{7}$$

Note that $\mathbf{H}$ has dimension (n, kd), where d is the dimensionality of the hidden states from a single attention head of the model. We create a tensor $\mathbf{O}$ whose elements correspond to orthogonals dropped from vectors in $\mathbf{H}$ to every other vector in $\mathbf{H}$, with dimensionality (n, n, kd). In PyTorch, this can be implemented as follows:

$$\mathbf{H'} = \mathbf{H}.\text{unsqueeze}(\text{dim} = 1) \tag{8}$$

$$\mathbf{D} = \text{torch.sum}(\mathbf{H} * \mathbf{H'}, \text{dim} = -1) \tag{9}$$

$$\mathbf{O} = \mathbf{H} - \mathbf{D}.\text{unsqueeze}(\text{dim} = -1) * \mathbf{H'} \tag{10}$$

We then have:

$$\text{orth}(\boldsymbol{h}^l_{A,j}, \boldsymbol{h}^l_{A,i}) = \mathbf{O}[i, j, :] \tag{11}$$

and:

$$\begin{aligned}\text{SCIN}(i, j) = &\|\mathbf{O}[i-1, j, :]\| \\ &+\|\mathbf{O}[j, j+1, :]\|\end{aligned} \tag{12}$$

where $\|\|$ is the L_2 norm as before.

D Dataset Statistics

In this section, we provide statistics for all datasets used in our experiments in Table 8. We also provide some examples from the Tense Inflection, Question Formation, MultiNLI, MoNLI and MED datasets.

Dataset	Train	Val	Test
Tense Inflection	100,000†	1,000†	10,000†
Question Formation	100,000†	1,000†	10,000†
BLLIP-LG	42,000,000*	36,000*	72,000*
WikiText-103	1,801,350†	3,760†	4,358†
MultiNLI	392,702†	20,000†	20,000†

Table 8: Statistics for each dataset used in our paper. †: Number of data points. *: Number of tokens when tokenized using the GPT-2 tokenizer.

D.1 Tense Inflection Examples

- **Input.** my quail upon my peacocks **remembered** her unicorns.
- **Output.** my quail upon my peacocks **remembers** her unicorns.
- **Input.** the vultures upon my unicorn **waited**.
- **Output.** the vultures upon my unicorn **wait**.

D.2 Question Formation Examples

- **Input.** my raven that doesn't sleep **does** change.
- **Output. does** my raven that doesn't sleep change?
- **Input.** our orangutans who do swim **don't** eat.
- **Output. don't** our orangutans who do swim eat?

D.3 MultiNLI Examples

- **Premise.** Conceptually cream skimming has two basic dimensions - product and geography.
- **Hypothesis.** Product and geography are what make cream skimming work.
- **Label. Neutral**
- **Premise.** How do you know? All this is their information again.
- **Hypothesis.** This information belongs to them.

- **Label. Entailment**
- **Premise.** Fun for adults and children.
- **Hypothesis.** Fun for only children.
- **Label. Contradiction**

D.4 MoNLI Examples

- **Premise.** The man does not own a dog.
- **Hypothesis.** The man does not own a mammal.
- **Label. Neutral**
- **Premise.** The man does not own a mammal.
- **Hypothesis.** The man does not own a dog.
- **Label. Entailment**

D.5 MED Examples

- **Premise.** No delegate finished the report on time.
- **Hypothesis.** No delegate finished the report.
- **Label. Neutral**
- **Premise.** Some delegates finished the survey on time.
- **Hypothesis.** Some delegates finished the survey.
- **Label. Entailment**

E NLI Setup

We now discuss the training and evaluation setup used for our NLI experiments. We format the training examples using the following prompt:

> Determine if the hypothesis is an entailment, contradiction or neutral in relation to the premise. If the hypothesis directly follows from the premise, it is an entailment. If the hypothesis directly contradicts the premise, it is a contradiction. Any relationship that does not fit in with the above definitions is considered neutral. Return A if the second sentence is an entailment, B if it is a contradiction, and C if it is neutral.
>
> ### Premise
>
> {premise}
>
> ### Hypothesis
>
> {hypothesis}
>
> ### Answer

We frame training as a classification task on the final logit from the LLM when the prompt above is passed as input. Instead of using a classification head, we directly apply cross-entropy loss over the log-probabilities of 'A', 'B', and 'C' as possible continuations of the prompt, with the groundtruth class label as the target. During evaluation, we select the continuation with the highest log-probability at the final logit, with 'A', 'B' and 'C' as choices. Note that we train on MultiNLI, which includes examples of the entailment, contradiction and neutral labels. On the other hand, MoNLI and MED datasets treat NLI as a two-class problem (entailment or neutral). However, we retain all three classes as options during inference on these datasets, for consistency with training.

F Experiment Hyperparameters

In this section, we detail the hyperparameters used for all of our main experiments. All experiments with LMs trained from scratch use tied input and output weight matrices. For all experiments, linear warmup is used for learning rates for the first 10% of training steps, after which cosine decay is used to reach a learning rate of 0 at the end of training. We use the AdamW optimizer with epsilon of 1e-7 and clip gradients to a maximum L_2 norm of 1.0.

F.1 § 4

- Number of layers: 4
- Number of attention heads: 8
- Hidden dimension: 512
- Training steps: 500,000
- Batch size: 8
- Learning rate: 1e-4
- Weight decay: 0.01
- Dropout: 0.1

F.2 § 5.1

- Number of layers: 16
- Number of attention heads: 8
- Hidden dimension: 512
- Training steps: 100,000
- Batch size: 160
- Learning rate: 1e-4
- Weight decay: 0.01
- Dropout: 0.1

F.3 § 5.3

- Number of layers: 12
- Number of attention heads: 12
- Hidden dimension: 768
- Training steps: 40,000
- Batch size: 480
- Learning rate: 6e-4
- Weight decay: 0.1
- Dropout: 0.1

F.4 § 6.1

- Training steps: 10,000
- Batch size: 32
- Learning rate: 2e-5
- Weight decay: 0.01
- Dropout: 0.1

F.5 § 6.2

- Training steps: 20,000
- Batch size: 32
- Learning rate: 1e-5
- Weight decay: 0.01
- Dropout: 0.1

G Selection of Layer and Number of Attention Heads for TreeReg

What is the ideal layer and number of attention heads for the $\mathcal{L}_{\text{TR}}$ computation? To explore, we vary the layer used in TreeReg as [2,4,6,8,10,12,14] and the number of attention heads used as [2,4,8] out of 8 in the experiment setup described in Section 5.1 for the training of a 16-layer LM from scratch on the BLLIP-LG dataset. For the resulting models, we report the overall SyntaxGym performance in Figure 6 and PTB perplexities in Figure 7.

We find that applying TreeReg at layer 12 on 2 attention heads results in the highest overall SyntaxGym performance as well as lowest PTB perplexity. With a couple of exceptions (4 attention heads at layer 4 and 6), TreeReg consistently outperforms the Base LM in both SyntaxGym performance and PTB perplexities across hyperparameter settings. Both syntactic generalization and out-of-distribution perplexities show generally improving trends till layer 12, after which they start degrading.

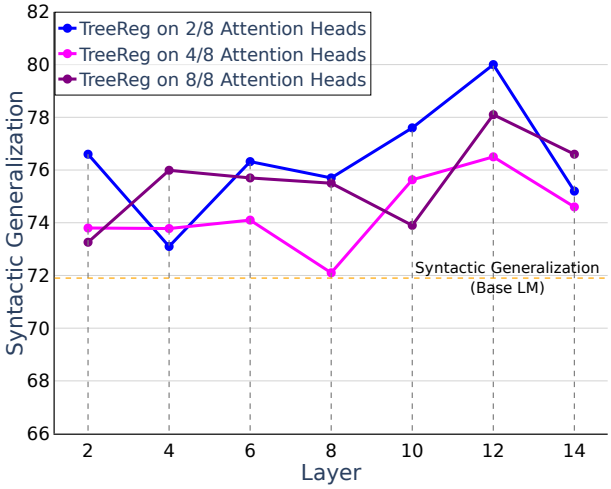

Figure 6: Syntactic Generalization on SG test suites for different layers and number of attention heads used in TreeReg, for LMs trained from scratch on BLLIP-LG.

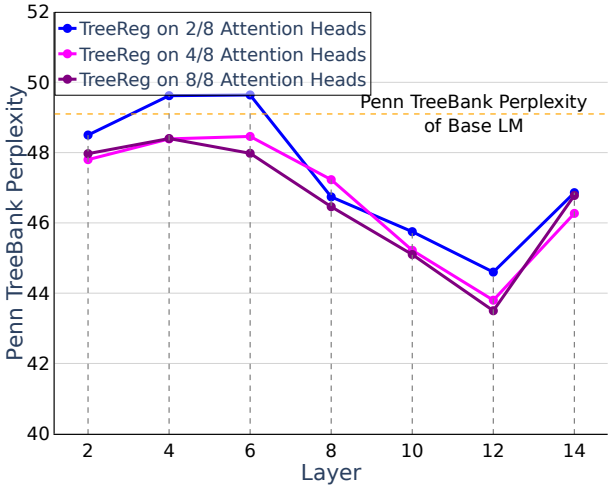

Figure 7: PTB Perplexity on SG test suites for different layers and number of attention heads used in TreeReg, for LMs trained from scratch on BLLIP-LG.

This finding suggests that TreeReg LMs represent syntax most naturally at intermediate layers, on a small subset of attention heads.

H Examples of Parses Induced by TreeReg

In this section, we provide some examples of parses induced from the TreeReg LM trained in §5.1 on the BLLIP-LG, PTB and 4000 Questions test sets.

H.1 BLLIP-LG

Figure 8 is an example where the induced parse from TreeReg matches the silver parse. Figure 9 and Figure 10 represents a case where the induced parse differs from the silver parse.

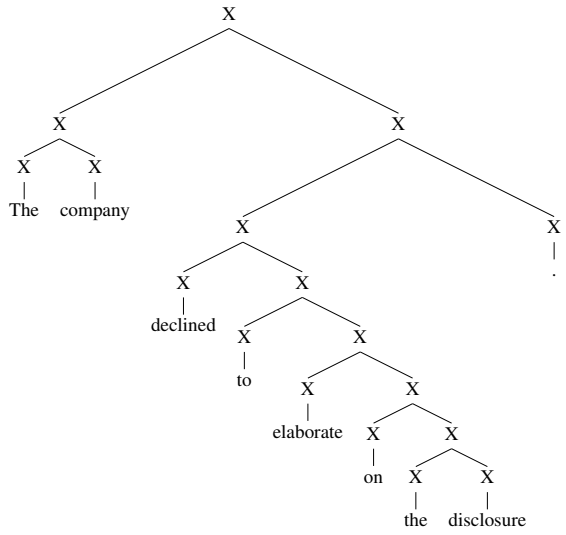

Figure 8: Silver and induced parse for "The company declined to elaborate on the disclosure."

H.2 PTB

Figure 11 is an example where the induced parse from TREEREG matches the silver parse. Figure 12 and Figure 13 represents a case where the induced parse differs from the silver parse.

H.3 4000 Questions

Figure 14 is an example where the induced parse from TREEREG matches the silver parse. Figure 15 and Figure 16 represents a case where the induced parse differs from the silver parse.

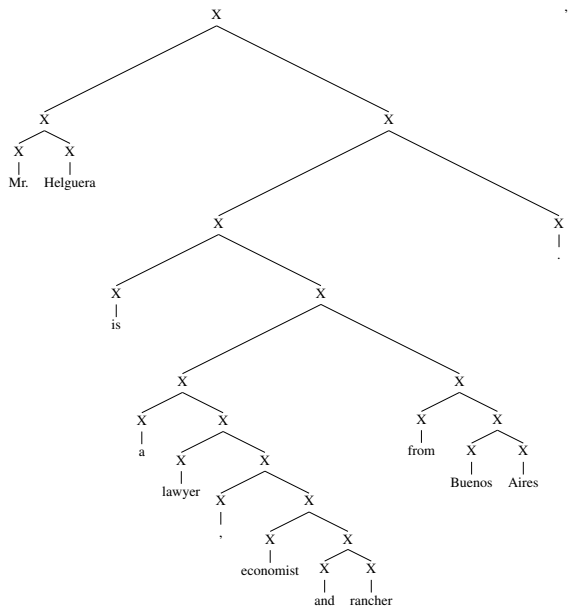

Figure 9: Silver parse for "Mr. Helguera is a lawyer, economist and rancher from Buenos Aires."

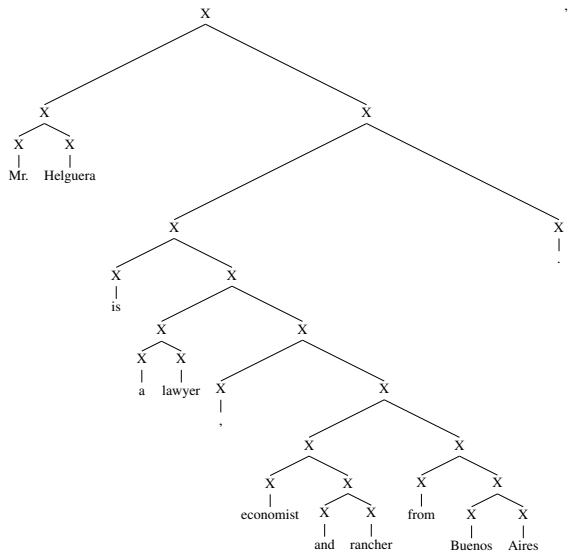

Figure 10: Induced parse for "Mr. Helguera is a lawyer, economist and rancher from Buenos Aires."

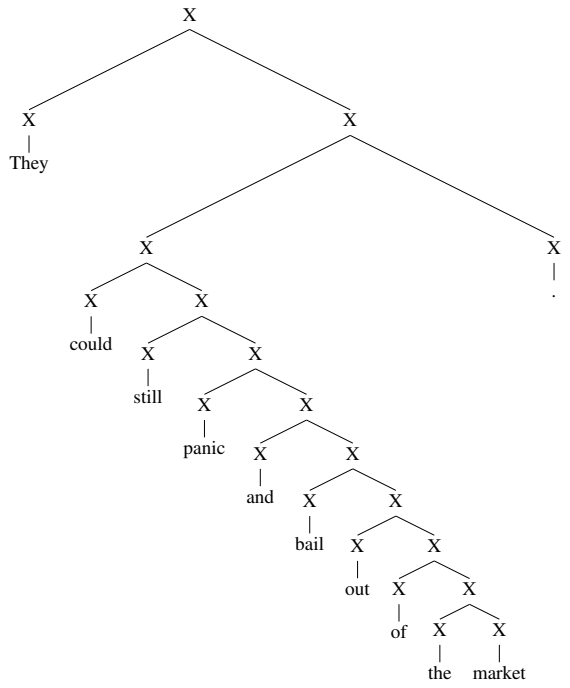

Figure 11: Silver and induced parse for "They could still panic and bail out of the market."

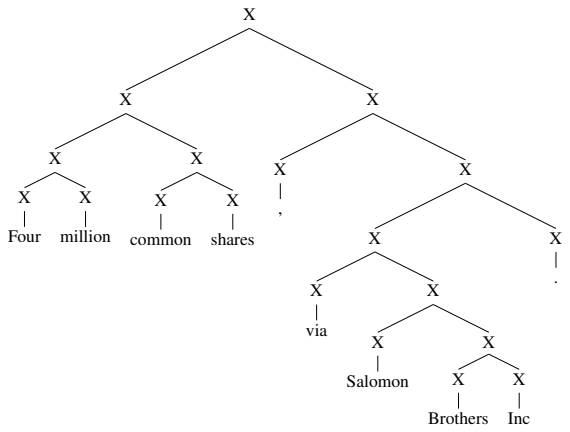

Figure 12: Silver parse for 'Four million common shares, via Salomon Brothers Inc."

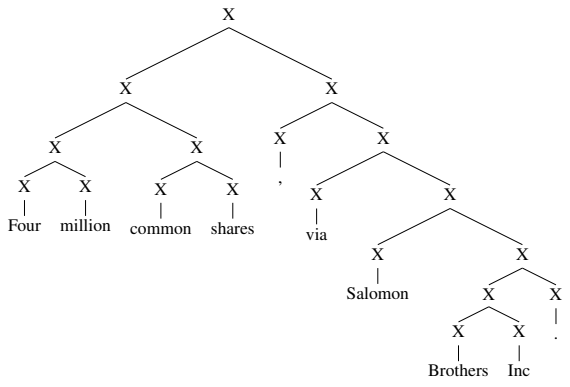

Figure 13: Induced parse for "Four million common shares, via Salomon Brothers Inc."

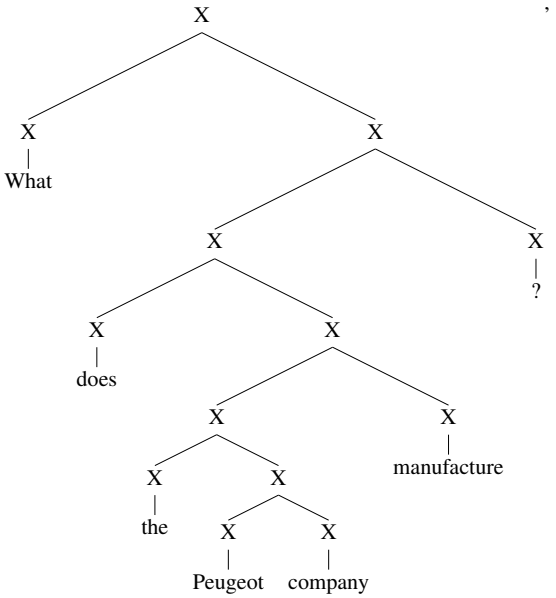

Figure 14: Silver and induced parse for "What does the Peugeot company manufacture?"

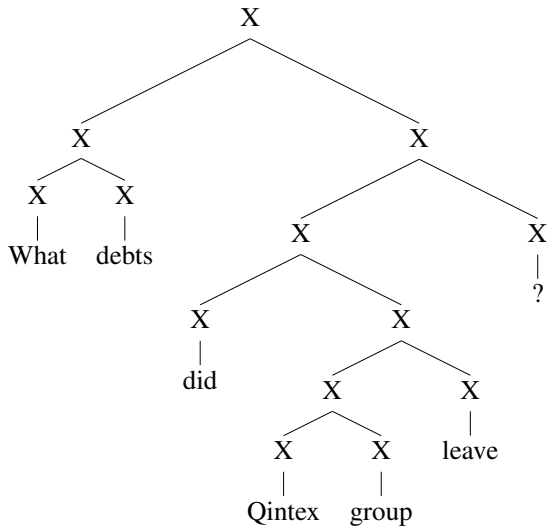

Figure 15: Silver parse for "What debts did Qintex group leave?"

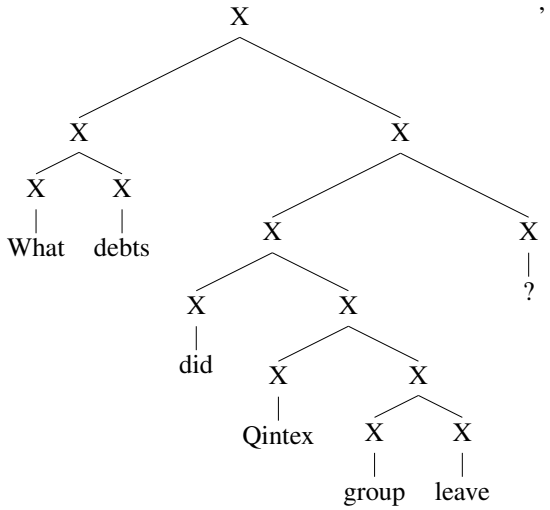

Figure 16: Induced parse for “What debts did Qintex group leave?”