Physics of Language Models: Part 1, Context-Free Grammar

Zeyuan Allen-Zhu
zeyuanallenzhu@meta.com
Meta FAIR Labs

Yuanzhi Li
Yuanzhi.Li@mbzuai.ac.ae
Mohamed bin Zayed University of AI

May 24, 2023*

Abstract

We design experiments to study *how* generative language models, like GPT, learn context-free grammars (CFGs)— diverse language systems with a tree-like structure capturing many aspects of natural languages, programs, and human logics. CFGs are as hard as pushdown automata, and can be ambiguous so that verifying if a string satisfies the rules requires dynamic programming. We construct synthetic data and demonstrate that even for very challenging CFGs, pre-trained transformers can learn to generate sentences with near-perfect accuracy and remarkable *diversity*.

More importantly, we delve into the *physical principles* behind how transformers learns CFGs. We discover that the hidden states within the transformer implicitly and *precisely* encode the CFG structure (such as putting tree node information exactly on the subtree boundary), and learn to form "boundary to boundary" attentions that resemble dynamic programming. We also cover some extension of CFGs as well as the robustness aspect of transformers against grammar mistakes. Overall, our research provides a comprehensive and empirical understanding of how transformers learn CFGs, and reveals the physical mechanisms utilized by transformers to capture the structure and rules of languages.

1 Introduction

Language systems are composed of many structures, such as grammar, coding, logic, and so on, that define how words and symbols can be arranged and manipulated to form meaningful and valid expressions. These structures reflect the logic and reasoning of human cognition and communication, and enable the generation and comprehension of diverse and complex expressions.

Language models [8, 9, 26, 29, 38] are neural network models that aim to learn the probability distribution of natural language and to generate natural language texts. Language models, such as GPT [28], can follow the structures of natural language or codes [32, 36] very well, even for relatively small models [6]. However, how do language models learn these structures? What mechanisms and representations do language models use to capture the rules and patterns of the language systems? Despite some recent theoretical progress in understanding the inner workings of language models [5, 15, 19, 20, 42], most of them are limited to rather simple settings and cannot account for the complex structure of languages.

In this paper, we study such structural learning by examining the **physical principles** behind how generative language models can learn probabilistic context-free grammars (CFGs) [4, 10, 18,

*We would like to thank Lin Xiao, Sida Wang and Hu Xu for many helpful conversations.

22, 31]. This is a class of grammars that can generate a large and diverse set of **highly structured** expressions. CFGs consist of a set of *terminal (T) symbols*, a set of *nonterminal (NT) symbols*, a root symbol, and a set of production rules. Each production rule has the form $a \to bcd\ldots$, where a is an NT symbol and $bcd\ldots$ is a string of symbols from T or NT. A string belongs to the language generated by a CFG, if there is a sequence of rules that transform the root symbol into the string of T symbols. For example, the following CFG generates the language of balanced parentheses:

$$s \to ss \mid (s) \mid \varnothing$$

where $\varnothing$ denotes the empty string. Examples in the language include $\varnothing$, `()`, `(())`, `()()`, `((()))`.

We use transformer [38] as the *generative language model*, which is a neural network model that has achieved remarkable results in various language processing tasks, such as machine translation, text summarization, and text generation. Transformer is based on self-attention, allowing the model to learn dependencies between tokens in a sequence. In GPT-based generative models, we have multiple layers of transformer *decoders*. Each decoder consists of a self-attention, which attends to leftward tokens, and feed-forward sublayers. The decoders process the input sequence and generate a sequence of hidden states, which are then passed to the next layer. The final layer, called the language model head (LMHead), converts the hidden states of the last decoder layer into predictions for the next token in the sequence.

It is known that transformers can encode some CFGs, especially those that correspond to the grammar of natural languages [3, 13, 21, 23, 33, 39, 41, 43]. However, these cited works only consider relatively simple grammar trees, and CFGs can go much beyond grammar to capture even some *human logics*. For instance, CFGs are as powerful as pushdown automata [34]; Blackjack and other card games can be described using CFGs [11, 24]. Most importantly, the *physical mechanism* behind how such CFGs can be efficiently learned by transformers remains unclear.

For a generative language model to learn a long CFG (e.g. *hundreds of tokens*), it needs to **efficiently learn many non-trivial, long-distance planning**. The model cannot just generate tokens that are "locally consistent." For example, to generate a string with balanced parentheses, the model must keep track of the number and type of open and close parentheses *globally*. Imagine, for complex CFGs, even verifying that a sequence satisfies a *given* CFG may require dynamic programming: to have a *memory* and a mechanism to *access* the memory in order to verify the hierarchical structure of the CFG. Learning CFGs is thus a significant challenge for the transformer model, and it tests the model's ability to learn and generate complex and diverse expressions.

In this paper, we pre-train GPT-2 [29] on the language modeling task over a large corpus of strings, sampled from a few (very non-trivial!) CFGs that we construct with different levels of difficulties— see Figure 1 for an example. We test the model by feeding it with *prefixes* of samples from the CFG, and observe if it can generate completions that perfectly,belong to the CFG. We measure accuracy and *diversity* of this generation process.

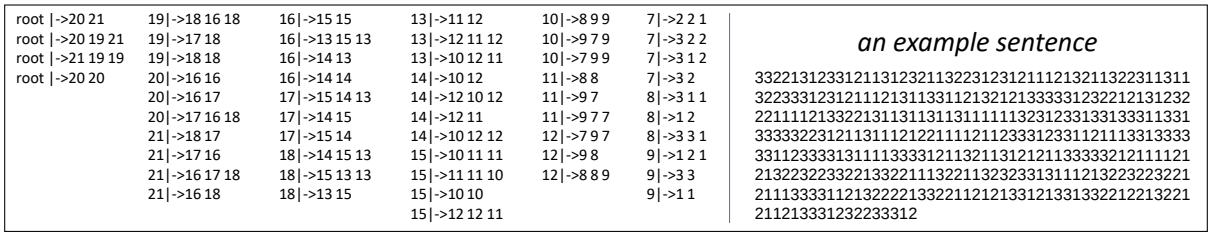


Figure 1: An example CFG that we experiment with in this paper. Although the CFG is of of depth 7, it is capable of generating sufficiently long and ambiguous instances; after all, even when the CFG rules are given, the canonical way to decide if a string x belongs to the CFG language $x \in L(\mathcal{G})$ is via dynamic programming.

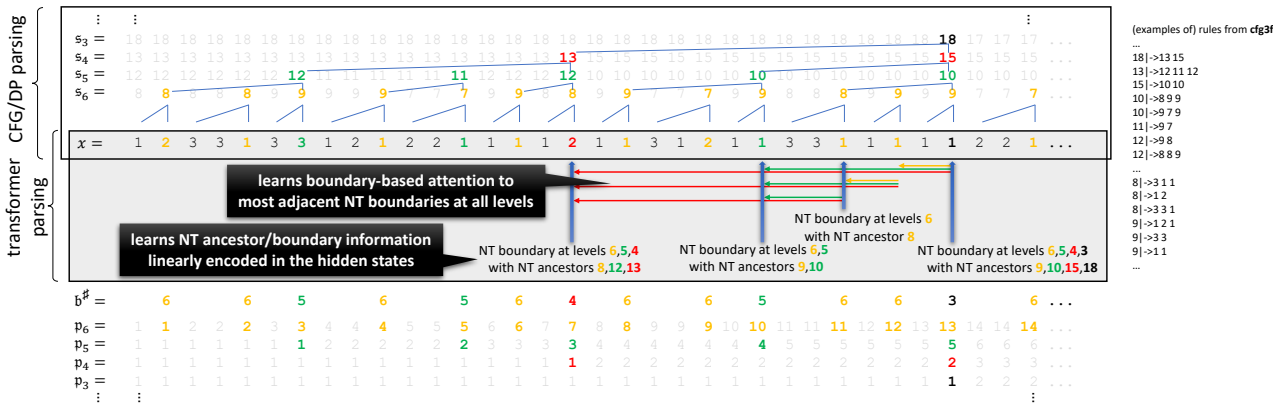


Figure 2: An example string x from $\mathcal{G} = \mathsf{cfg3f}$. We define NT ancestor symbols $\mathfrak{s}_\ell(i)$ and ancestor indices $\mathfrak{p}_\ell(i)$ for each position x_i at level ℓ. Bold symbols represent NT boundaries at different CFG levels ℓ.

- We confirm that GPT-2 is capable of achieving near-perfect CFG generation accuracies, and has a diversity that is much greater than the number of model parameters.

As the **key contribution** of our paper, we focus on analyzing *how transformer recovers the structures of the underlying CFG*, by looking at the attention patterns and the hidden states of the model. In particular,

- We devise a method to verify that the hidden states of the trained model linearly encode the hidden NT information (in fact, encoding the "NT symbol" exactly at "NT boundary") almost perfectly. This is a non-trivial observation, as pre-training does not reveal the CFG or the hidden NTs. Rather, after pre-training, GPT automatically learns such syntactic information.
- We propose several methods to *statistically* visualize and quantify the attention pattern. We show that GPT learns two types of attentions. Position-based attention enables the model to learn the regularity and periodicity of the CFG, and to generate tokens based on their positions. Boundary-based attention enables the model to learn the hierarchical and recursive structure of the CFG, and to generate tokens based on the NT symbols and rules.
- We argue that our findings suggest GPT-like models learn CFGs by *implementing standard dynamic programming*. We discover that boundary-based attention allows a token to attend to its "most adjacent" NT symbols in the CFG tree— via attending to the exact ending token of such NT symbols, even if the two tokens can be hundreds of tokens apart. This resembles dynamic programming, in which the CFG parsing on a sequence $1...i$ needs to be "concatenated" with another sequence $i+1...j$ in order to form a solution to a larger problem on $1...j$. See Figure 2 for an illustration of the parsing process of a pre-trained GPT-2 on CFGs. Having boundary-based attentions allows GPT to accurately retrieve the NT symbol information stored on the i-th position to position j.

To broaden the scope of our paper, we also consider implicit CFGs [27], where each T symbol is now a bag of tokens, and data is generated by randomly sampling tokens from the bag. Implicit CFGs allow one to capture some syntactic or semantic categories of the tokens (such as whether the word is a verb or noun).

- We show that the model learns implicit CFGs simply by using its token embedding layer to encode the (hidden) T symbol information.

Last but not least, we study model robustness [25, 37] using CFGs. Given a corrupted prefix,

we test whether the model can correct errors and continue generating according to the valid CFG. Robustness is an important property of language models, as it reflects the generalization and adaptation ability of the network to handle real-world data, such as those with grammar mistakes.

- We show that GPT models pre-trained using only grammatically correct data can only achieve moderate robust accuracy. Adding as little as 10% data perturbation, or even allowing *all* the training samples to have grammar mistakes, the robust accuracy significantly improves. This finding suggests that it can be *beneficial* to include low-quality data during pre-training.
- Using CFGs, we show when pre-trained with perturbed data, transformer learns a "mode switch" between intentionally writing and not writing grammar mistake. This explains why in practice, when feeding a large pre-trained language model (such as `LLaMa-30B`) with prompts of grammar mistakes, it tends to generates more grammar mistakes when using high temperature.

2 Context-Free Grammars

A probabilistic context-free grammar (CFG) is a formal system that defines a distribution of strings using production rules. It has four components: terminal symbols ($\mathbf{T}$), nonterminal symbols ($\mathbf{NT}$), a root symbol ($root \in \mathbf{NT}$), and production rules ($\mathcal{R}$). We write a context-free grammar as $\mathcal{G} = (\mathbf{T}, \mathbf{NT}, \mathcal{R})$ and use $L(\mathcal{G})$ to denote the distribution of strings generated by $\mathcal{G}$.

2.1 Definition and Notations

In this paper, we consider L-level CFGs with the following structure. Each level $\ell \in [L]$ corresponds to a set of symbols $\mathbf{NT}_\ell$, where $\mathbf{NT}_\ell \subseteq \mathbf{NT}$ for $\ell < L$. Additionally, $\mathbf{NT}_L = \mathbf{T}$ and $\mathbf{NT}_1 = \{root\}$. Without loss of generality, we assume the symbols at different levels are disjoint $\mathbf{NT}_i \cap \mathbf{NT}_j = \varnothing$ for $i \neq j$; and we consider rules of length 2 or 3, denoted as $\mathcal{R} = (\mathcal{R}_1, \dots, \mathcal{R}_{L-1})$, where each $\mathcal{R}_\ell$ consists of a collection of rules in the form:

$$r = (a \mapsto b \circ c \circ d) \quad \text{or} \quad r = (a \mapsto b \circ c) \quad \text{for} \quad a \in \mathbf{NT}_\ell \quad \text{and} \quad b, c, d \in \mathbf{NT}_{\ell+1}$$

For a non-terminal symbol $a \in \mathbf{NT}$, we say $a \in r$ if $r = (a \mapsto \star)$ for some rule. (The symbol $\circ$ is used for clarity to indicate concatenation of symbols.) For each non-terminal symbol $a \in \mathbf{NT}$, we define its associated set of rules as $\mathcal{R}(a) = \big\{r \mid r \in \mathcal{R}_\ell \wedge a \in r\big\}$. The *degree* of a is given by $|\mathcal{R}(a)|$. The *size* of the CFG is represented as $(|\mathbf{NT}_1|, |\mathbf{NT}_2|, \dots, |\mathbf{NT}_L|)$.

Generating from CFG. To generate samples x from $L(\mathcal{G})$, we follow the steps below:

1. Begin with the *root* symbol $\mathbf{NT}_1$.
2. For each layer $\ell < L$, maintain a sequence of symbols $s_\ell = \big(s_{\ell,1}, \cdots, s_{\ell,m_\ell}\big)$.
3. To generate the symbols of the next layer, for each $s_{\ell,i}$, randomly sample one rule $r \in \mathcal{R}(s_{\ell,i})$ with uniform probability.[1] Replace $s_{\ell,i}$ with $b \circ c \circ d$ if $r = (s_{\ell,i} \mapsto b \circ c \circ d)$, or with $b \circ c$ if $r = (s_{\ell,i} \mapsto b \circ c)$. Denote the resulting sequence as $s_\ell = \big(s_{\ell+1,1}, \cdots, s_{\ell+1,m_{\ell+1}}\big)$.
4. During the generation process, when a rule $s_{\ell,i} \mapsto s_{\ell+1,j} \circ s_{\ell+1,j+1}$ is applied, define the parent $\mathsf{par}_{\ell+1}(j) = \mathsf{par}_{\ell+1}(j+1) \stackrel{\text{def}}{=} i$ (and similarly if the rule of $s_{\ell,i}$ is of length 3).
5. Define NT ancestor indices $\mathfrak{p} = (\mathfrak{p}_1(i), \dots, \mathfrak{p}_L(i))_{i \in [m_L]}$ and NT ancestor symbols $\mathfrak{s} = (\mathfrak{s}_1(i), \dots, \mathfrak{s}_L(i))_{i \in [m_L]}$— see also Figure 2 for an illustration

$$\mathfrak{p}_L(j) \stackrel{\text{def}}{=} j \ , \quad \mathfrak{p}_\ell(j) \stackrel{\text{def}}{=} \mathsf{par}_{\ell+1}(\mathfrak{p}_{\ell+1}(j)) \quad \text{and} \quad \mathfrak{s}_\ell(j) \stackrel{\text{def}}{=} s_{\ell,\mathfrak{p}_\ell(j)}$$

[1] We consider uniform case for simplicity, so there are no rules that appear with extremely small probability. Our observations easily extend to non-uniform cases as long as the distributions are not very unbalanced.

The final generated string is $x = s_L = (s_{L,1}, \cdots, s_{L,m_L})$ and we write $x_i = s_{L,i}$. The length of x is denoted as $\mathbf{len}(x) = m_L$. We use the notation $(x, \mathfrak{p}, \mathfrak{s}) \sim L(\mathcal{G})$ to represent a string x along with its associated NT ancestor indices and symbols, sampled according to the described generation process. Alternatively, we use $x \sim L(\mathcal{G})$ when the $\mathfrak{p}$ and $\mathfrak{s}$ are evident from the context.

Definition 2.1. *Given a sample $(x, \mathfrak{p}, \mathfrak{s}) \sim L(\mathcal{G})$, for each symbol x_i, we say that x_i is the NT-end boundary (or* **NT boundary / NT end** *for short) at level $\ell \in [L-1]$, if and only if: $\mathfrak{p}_\ell(i) \neq \mathfrak{p}_\ell(i+1)$ or $i = \mathbf{len}(x)$. We denote $\mathfrak{b}_\ell(i) \stackrel{\text{def}}{=} \mathbb{1}_{x_i \text{ is the NT boundary at level } \ell}$ as the indicator function of the NT-end boundary, and the deepest NT-end of i as— see also Figure 2 —*

$$\mathfrak{b}^\sharp(i) = \min_{\ell \in \{2,3,\ldots,L-1\}} \{\mathfrak{b}_\ell(i) = 1\} \quad \text{or } \perp \text{ if the set is empty } .$$

3 Transformer and CFGs

In this section, we focus on the *generative aspect* of the transformer. Given prefixes of strings in $L(\mathcal{G})$, can the transformer accurately complete the sequence by generating strings that are fully correct within the support of $L(\mathcal{G})$? For example, if the grammar represents balanced parentheses and the input sequence is "`((`", both the outputs "`))`" and "`)())`" would be considered grammatically correct while "`())`" would not. Moreover, it is important to assess whether the generated outputs exhibit sufficient diversity, ideally matching the probabilistic distribution of strings within $L(\mathcal{G})$.

Transformer can perfectly learn CFG. We construct multiple CFGs $\mathcal{G}$ of varying difficulty levels and generate a large corpus $\{x^{(i)}\}_{i \in [N]}$ by sampling from $L(\mathcal{G})$. We train 12-layered GPT-2 networks [29] F on this corpus, where each terminal symbol is treated as a separate token. We pre-train the model using a standard auto-regressive task (details in Section A.3). For evaluation, we test F by having it generate completions for prefixes $x_{:c} = (x_1, x_2, \cdots, x_c)$ from strings x that are (freshly) generated from $L(\mathcal{G})$. Denoting by $F(x_{:c})$ the tokens generated by F on input prefix $x_{:c}$, we measure the *generation accuracy* as:

$$\mathbf{Pr}_{x \sim L(G) + \text{randomness of } F}[(x_{:c} \circ F(x_{:c})) \in L(\mathcal{G})]$$

Throughout the paper, we utilize multinomial sampling without beam search for generation. Unless explicitly mentioned, we consistently employ a temperature $\tau = 1$ during generation, which ensures that the output reflects the unaltered distribution learned by the transformer.

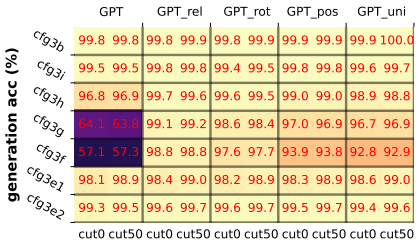
DATASETS. We construct seven CFGs of depth $L = 7$ and their details are in Section A.1:

- hard datasets cfg3b, cfg3i, cfg3h, cfg3g, cfg3f are of sizes $(1, 3, 3, 3, 3, 3, 3)$, with increasing difficulties cfg3b < cfg3i < cfg3h < cfg3g < cfg3f.
- easy datasets cfg3e1 and cfg3e2 are of sizes resp. $(1, 3, 9, 27, 81, 27, 9)$ and $(1, 3, 9, 27, 27, 9, 4)$.

The sequences generated by these CFGs are of length up to $3^6 = 729$. Intuitively, having more (terminal or nonterminal) symbols makes the task of learning CFG easy due to less ambiguity. Thus, we primarily focus on the harder datasets generated by cfg3b, cfg3i, cfg3h, cfg3g, cfg3f.

MODELS. We use `GPT` to denote the vanilla GPT2 small architecture (12-layered, 12-head, 768-dims). Since GPT2 is known to have weak performance due to its absolute positional embedding, we also implemented two *modern variants*. We use $\texttt{GPT}_{\mathsf{rel}}$ to denote `GPT` equipped with relative positional attention [12], and $\texttt{GPT}_{\mathsf{rot}}$ to denote `GPT` equiped with rotary positional embedding [7, 35].

For specific purposes in subsequent sections, we also introduce two weaker variants of `GPT`. $\texttt{GPT}_{\mathsf{pos}}$ replaces the attention matrix with a matrix based solely on tokens' relative positions, while



Observation: CFGs with less ambiguity (cfg3e1, cfg3e2) are easy to learn; to perfectly learn those hard CFGs, modern transformer variants ($\texttt{GPT}_{\texttt{rel}}$ or $\texttt{GPT}_{\texttt{pos}}$) that make use "relative positional embedding" have to be adopted. We also consider two weaker variants: $\texttt{GPT}_{\texttt{pos}}$ and $\texttt{GPT}_{\texttt{uni}}$; they have attention matrices solely based on token positions and serve specific purposes in subsequent sections.

Figure 3: Generation accuracy across multiple CFG datasets that we construct at different levels of difficulty. Cut $= 0$ means to generate from start and $= 50$ means to generate from a valid prefix of length 50.

$\texttt{GPT}_{\texttt{uni}}$ employs a constant, uniform average of past tokens from various window lengths as the attention matrix. Detailed explanations of these variants can be found in Section A.2.

RESULTS. Figure 3 displays the generation accuracies for cuts $c = 0$ and $c = 50$. These correspond to generating a fresh string from the start and completing a prefix of length 50, respectively. The results demonstrate that the pre-trained transformers can generate near-perfect strings that adhere to the CFG rules, even at a high temperature $\tau = 1$.

Generation diversity. A potential concern is whether the trained transformer simply memorizes a subset of strings from the CFG, resulting in fixed patterns in its outputs. To further evaluate its learning capability, we measure the diversity of the generated strings. Higher diversity indicates a better understanding of the grammar rules. For example, in the matching bracket example, the transformer should be able to generate various outputs (e.g., `))`, `()))`, `)())`, `))()`) rather than being limited to a single pattern (e.g., `)))`). Since "diversity" is influenced by the length of the input prefix, the length of the output, and the CFG rules, we want to carefully define what we measure.

Given a sample pool $x^{(1)}, ..., x^{(M)} \in L(\mathcal{G})$, for every symbol $a \in \mathbf{NT}_{\ell_1}$ and some later level $\ell_2 \geq \ell_1$ that is closer to the leaves, we wish to define a *multi-set* $\mathcal{S}_{a\to\ell_2}$ that describes *all possible generations from* $a \in \mathbf{NT}_{\ell_1}$ *to* $\mathbf{NT}_{\ell_2}$ in this sample pool. Formally,

Definition 3.1. *For $x \in L(\mathcal{G})$ and $\ell \in [L]$, we use $\mathfrak{s}_\ell(i..j)$ to denote the sequence of NT ancestor symbols at level $\ell \in [L]$ from position i to j with distinct ancestor indices:*[2]

$$\mathfrak{s}_\ell(i..j) = (\mathfrak{s}_\ell(k))_{k\in\{i,i+1,...,j\}\ s.t.\ \mathfrak{p}_\ell(k)\neq\mathfrak{p}_\ell(k+1)}$$

Definition 3.2. *For symbol $a \in \mathbf{NT}_{\ell_1}$ and some layer $\ell_2 \in \{\ell_1, \ell_1+2, \ldots, L\}$, define multi-set*[3]

$$\mathcal{S}_{a\to\ell_2}(x) = \left[\!\!\left[\mathfrak{s}_{\ell_2}(i..j) \,\middle|\, \forall i,j, i \leq j \text{ such that } \mathfrak{p}_{\ell_1}(i-1) \neq \mathfrak{p}_{\ell_1}(i) = \mathfrak{p}_{\ell_1}(j) \neq \mathfrak{p}_{\ell_1}(j+1) \wedge a = \mathfrak{s}_{\ell_1}(i) \right]\!\!\right]$$

and we define the multi-set union $\mathcal{S}_{a\to\ell_2} = \bigcup_{i\in[M]} \mathcal{S}_{a\to\ell_2}(x^{(i)})$.

(Above, when $x \sim L(\mathcal{G})$ is generated from the ground-truth CFG, then the ancestor indices and symbols $\mathfrak{p}, \mathfrak{s}$ are defined in Section 2.1. If $x \in L(\mathcal{G})$ is an output from the transformer F, then we let $\mathfrak{p}, \mathfrak{s}$ be computed using dynamic programming, breaking ties lexicographically.)

We use $\mathcal{S}^{\mathsf{truth}}_{a\to\ell_2}$ to denote the ground truth $\mathcal{S}_{a\to\ell_2}$ when $x^{(1)}, \ldots, x^{(M)}$ are i.i.d. sampled from the real distribution $L(\mathcal{G})$, and denote by

$$\mathcal{S}^{F}_{a\to\ell_2} = \bigcup_{i\in[M'] \text{ and } x^{(i)}_{:c}\circ F(x^{(i)}_{:c})\in L(\mathcal{G})} \mathcal{S}_{a\to\ell_2}\left(x^{(i)}_{:c} \circ F(x^{(i)}_{:c})\right)$$

[2]With the understanding that $\mathfrak{p}_\ell(0) = \mathfrak{p}_\ell(\mathbf{len}(x)+1) = \infty$.

[3]Throughout this paper, we use $[\![\cdot]\!]$ to denote multi-sets that allow multiplicity, such as $[\![1,2,2,3]\!]$. This allows us to conveniently talk about its collision count, number of distinct elements, and set average.

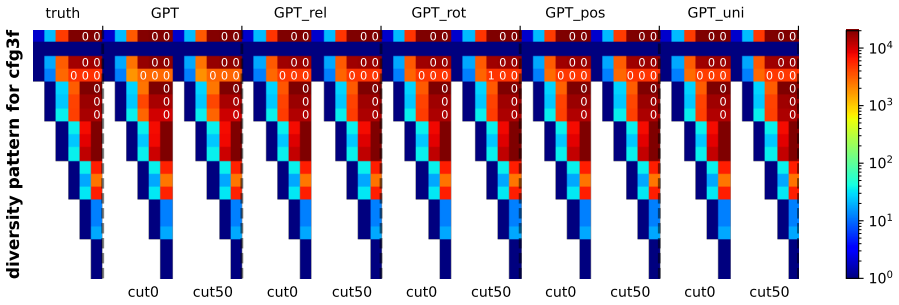


Figure 4: Comparing the generation diversity $\mathcal{S}^{\text{truth}}_{a\to\ell_2}$ and $\mathcal{S}^{F}_{a\to\ell_2}$ across different learned GPT models ($c = 0$ or $c = 50$). Rows correspond to NT symbols a and columns correspond to $\ell_2 = 2, 3, \ldots, 7$. Colors represent the number of distinct elements in $\mathcal{S}^{\text{truth}}_{a\to\ell_2}$, and the white numbers represent the collision counts (if not present, meaning there are more than 5 collisions). More experiments in Appendix B.1.

Observation. We use $M = 20000$ samples. The diversity pattern from the pre-trained transformer matches that of the ground-truth. For instance, there is a symbol $a \in \mathbf{NT}_2$ capable of generating $\Omega(M^2)$ distinct sequences to level $\ell_2 = 5$ satisfying the CFG (not to say to the T-level $\ell_2 = 7$); this is already more than the number of parameters in the model. Therefore, we conclude that the pre-trained model **does not rely on simply memorizing** a small set of patterns to learn the CFGs.

that from the transformer F. For a fair comparison, for each F and p, we pick an $M' \geq M$ such that $M = \left|\left\{i \in [M'] \mid x^{(i)}_{:p} \circ F(x^{(i)}_{:p}) \in L(\mathcal{G})\right\}\right|$ so that F is capable of generating exactly M sentences that perfectly satisfy the CFG rules.

Intuitively, for x's generated by the transformer model, the larger the number of distinct sequences in $\mathcal{S}^{F}_{a\to\ell_2}$ is, the more diverse the set of NTs at level ℓ_2 (or Ts if $\ell_2 = L$) the model can generate starting from NT a. Moreover, in the event that $\mathcal{S}^{F}_{a\to\ell_2}$ has only distinct sequences (so collision count $= 0$), then we know that the generation from $a \to \ell_2$, with good probability, should include at least $\Omega(M^2)$ possibilities using a birthday paradox argument. [4]

For such reason, it can be beneficial if we compare the *number of distinct sequences* and the *collision counts* between $\mathcal{S}^{F}_{a\to\ell_2}$ and $\mathcal{S}^{\text{truth}}_{a\to\ell_2}$. Note we consider all $\ell_2 \geq \ell_1$ instead of only $\ell_2 = L$, because we want to better capture model's diversity at all CFG levels.[5] We present our findings in Figure 4 with $M = 20000$ samples.

Distribution Comparison. To fully learn a CFG, it is also crucial to match the distribution of generating probabilities. This can be challenging to measure, but we have performed a simple test on the marginal distributions $p(a, i)$, which represent the probability of symbol $a \in \mathbf{NT}_\ell$ appearing at position i (i.e., the probability that $\mathfrak{s}_\ell(i) = a$). We observe a strong alignment between the generation probabilities and the ground-truth distribution. See Appendix B.2 for details.

4 How Do Transformers Learn CFGs?

In this section, we delve into the learned representation of the transformer to understand *how* it encodes CFGs. We employ various measurements to probe the representation and gain insights.

Standard way to solve CFGs. Given CFG $\mathcal{G}$, recall the canonical way to verify if a sequence x satisfied $L(\mathcal{G})$ is to use dynamic programming (DP) [30, 34]. Given starting and ending indices

[4] A CFG of depth L, even with constant degree and constant size, can generate $2^{2^{\Omega(L)}}$ distinct sequences.

[5] A model might generate a same NT symbol sequence s_{L-1}, and then generate different Ts randomly from each NT. In this way, the model still generates strings x's with large diversity, but $\mathcal{S}^{F}_{a\to L-1}(x)$ is small. If $\mathcal{S}^{F}_{a\to\ell_2}$ is large for every ℓ_2 and a, then the generation from the model is *truely diverse at any level of the CFG.*

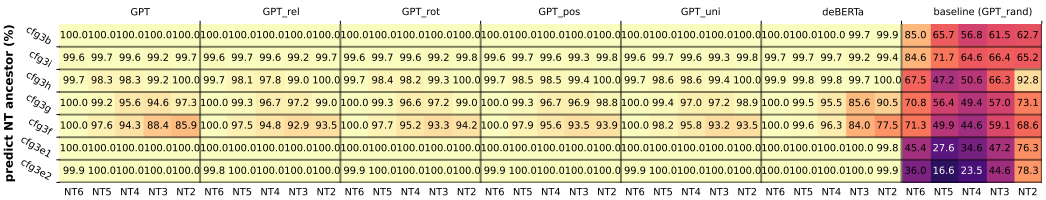


Figure 5: After pre-training, hidden states of generative models implicitly encode the NT ancestors information.

It also encodes NT boundaries, see Appendix C.1; and such information is discovered gradually and *hierarchically*, across layers and training epochs, see Appendix C.2 and C.3. We compare against a baseline which is the encoding from a random GPT. We also compare against DeBERTa, illustrating that BERT-like models are less effective in learning NT information at levels close to the CFG root.

$i \le j$, and any non-terminal symbol $a \in \mathbf{NT}_\ell$, we can use $\mathsf{DP}(i,j,a)$ to define whether or not $x_i \circ x_{i+1} \cdots \circ x_j$ can be generated from symbol a following the CFG rules. Mathematically, for any sequence $x \sim L(\mathcal{G})$ satisfying the CFG, it must satisfy (recall the NT-boundary $\mathfrak{b}_\ell$ and the NT-ancestor $\mathfrak{s}_\ell$ notions are in Section 2.1)

$$\mathfrak{b}_\ell(i-1) = 1, \mathfrak{b}_\ell(j) = 1, \forall k \in [i,j), \mathfrak{b}_\ell(k) = 0 \text{ and } \mathfrak{s}_\ell(i) = s \implies \mathsf{DP}(i,j,a) = 1$$

The dynamic programming recursively finds all $\mathsf{DP}(i,j,a)$ from $a \in \mathbf{NT}_{L-1}$ to $a \in \mathbf{NT}_1$. This DP process is illustrated in the upper half of Figure 2 and it relies on two important quantities: the NT-boundary $\mathfrak{b}_\ell$ and the NT-ancestor $\mathfrak{s}_\ell$. Our main goal of this section is to show that **the transformer learns to parse the CFG according to the lower half of Figure 2**.

4.1 Finding 1: Transformers' Hidden States Encode NT Ancestors and NT Boundaries

Inspired by DP, we test whether pre-trained transformer F, besides being able to generate grammatically correct sequences, *also* implicitly encodes the NT ancestor and boundary information. And if so, *where* does F store such information.

Given input x, let $E_{l,i}(x) \in \mathbb{R}^d$ be the hidden state of the decoder at layer l and position i. Then, we first ask whether one can use a *linear* function to predict $\big(\mathfrak{b}_1(i), \ldots, \mathfrak{b}_L(i)\big)_{i \in [\mathbf{len}(x)]}$ and $\big(\mathfrak{s}_1(i), \ldots, \mathfrak{s}_L(i)\big)_{i \in [\mathbf{len}(x)]}$ using only $\big(E_{l,i}(x)\big)_{i \in [\mathbf{len}(x)]}$.

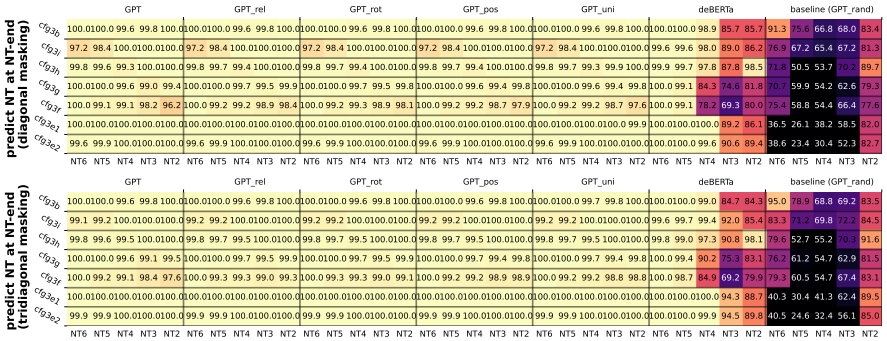
Meaningfulness. Recall the transformer F (and hence the hidden state $E_{l,i}(x)$) is trained solely from the auto-regressive task. One should not expect it to learn the CFG tree exactly, let alone $\mathfrak{b}$ or $\mathfrak{s}$— after all, there are equivalent CFG representations that may capture the same $L(\mathcal{G})$.

However, is it possible that the hidden states have already implicitly encoded $\mathfrak{b}$ and $\mathfrak{s}$, up to some equivalence class? In this section, we use linear probing to test whether $E_{l,i}(x)$ encodes the structural information of the CFG **up to linear transformation**.

Our linear function. Specifically, let us fix a layer l in the transformer. We train a collection of linear functions $f_r \colon \mathbb{R}^d \to \mathbb{R}^{|\mathbf{NT}|}$, where $r \in [H]$ for a parameter H denoting the number of "heads". To predict the NT ancestor symbol at position i, we apply:

$$G_i(x) = \textstyle\sum_{r \in [H], k \in [\mathbf{len}(x)]} w_{r,i \to k} \cdot f_r(E_{l,k}(x)) \in \mathbb{R}^{|\mathbf{NT}|} \tag{4.1}$$

where $w_{r,i\to k} \stackrel{\text{def}}{=} \frac{\exp(\langle P_{i,r}, P_{k,r}\rangle)}{\sum_{k' \in [\mathbf{len}(x)]} \exp(\langle P_{i,r}, P_{k',r}\rangle)}$ for trainable parameters $P_{i,r} \in \mathbb{R}^{d'}$. Here, G_i can be viewed as the output of the i-th token after a multi-head attention layer, but the attention only depends



Observation. BERT-like (encoder-only) transformers, such as DeBERTa, trained on a masked language modeling (MLM) task, do not store deep NT ancestor information at the NT boundaries.

Figure 6: Generative pre-trained transformer encodes NT ancestors almost exactly <u>at</u> NT boundaries.

on the *position of the tokens* (so is fixed and independent of input sequence x).

We train $G_i(x) \in \mathbb{R}^{|\mathbf{NT}|}$ using the (multi-class) cross-entropy loss to predict $\big(\mathfrak{s}_\ell(i)\big)_{\ell\in[L]}$. We emphasize here that, despite having position-based attention, $G_{l,i}(x)$ is still a linear function over $\big(E_{l,k}(x)\big)_{k\in[\mathbf{len}(x)]}$ (i.e., the linear weights do not depend on x and only depend positions i and k).

Completely analogously, we also train a some $G'_i(x) \in \mathbb{R}^L$ using the (multi-class) logistic loss to predict the values $\big(\mathfrak{b}_\ell(i)\big)_{\ell\in[L]}$. We give details in Section A.4.

Results. Our experiments in Figure 5 suggest that, through pre-training, the learned generative models *almost perfectly encode* the NT ancestor and NT boundary information in its hidden states at the last transformer layer, up to a *linear* transformation.

4.2 Finding 2: Transformer's Hidden States Encode NT Ancestors <u>At</u> NT Boundaries

Using attention weights $w_{r,i\to k}$ in (4.1) to probe a *generative model* is necessary because if a token x_i is close to the start of the string, or a starting token of a subtree in the CFG, it can be impossible to extract i's ancestor information by only looking at $E_{l,i}(x)$. However, if we only focus on the tokens x_i on the NT-end boundaries, (1) does the pre-trained transformer know it is an NT-end boundary token, and (2) does the pre-trained transformer know the corresponding NT ancestor's symbol, by only looking at $E_{l,i}(x)$ or at most $E_{l,i\pm1}(x)$? To do so, we replace $w_{r,i\to k}$ in (4.1) with a mask $w_{r,i\to k} \cdot \mathbb{1}_{|i-k|\le\delta}$ for $\delta \in \{0,1\}$. From Figure 6, we see that with very small $\delta = 1$, the hidden states can already achieve almost perfect accuracy. Thus, the information of position i's NT ancestor symbols and NT-end boundaries are encoded very locally around position i.

Related works. Our probing method is similar to the pioneering work by Hewitt and Manning [13], where they use linear probing to test whether the hidden states of BERT can largely correlate with the distance metric associated with the parse tree (similar to the NT-distance in our language). Subsequent works [3, 21, 23, 33, 39, 41, 43] also study various probing methods to provide more hints that CFGs from natural languages can be approximately learned by BERT-like transformers.

Instead, we use synthetic data to show that linear probing *near-perfectly* recovers NT ancestors and boundaries, for even very hard CFGs capable of generating strings of more than hundreds of tokens. The main difference is that we study pre-training *generative* language models. For a BERT-like, non-generative model pre-trained using via language-modeling (MLM), such as the modern variant DeBERTa [12], it is less effective in learning *deep* NT information (i.e., close to the CFG root), see Figure 5. (This may not be surprising, as the MLM task may only need the transformer to learn those NT rules with respect to, say 20, neighboring tokens.) More importantly, BERT-like models simply *do not* store deep NT information *exactly* at the NT boundaries (see Figure 6).

Our findings here, together with Section 5, shall be strong evidence that generative language models such as GPT-2 learn to perform dynamic-programming-like approach to generate CFGs; while encoder-based models, usually trained via MLM, are less capable of learning harder/deeper CFGs.

5 How Do Transformers Learn NTs?

After confirming that the transformer learns the NT ancestors and boundaries, in this section, we further investigate how the transformer uses such information by looking at the attention patterns. We show that the attention patterns of the transformer reflect the syntactic structure and rules of the CFG, and that the transformer uses different attention heads to learn NTs at different CFG levels. We focus on two types of attentions: position-based attention and boundary-based attention.

5.1 Position-Based Attention

We observe that transformer exhibits a prominent position-based attention pattern, where the attention weights are primarily determined by the *relative distance* of the tokens rather than their content or meaning. This suggests that transformer learns the regularity and periodicity of the CFG through positional information and uses it to generate accordingly.

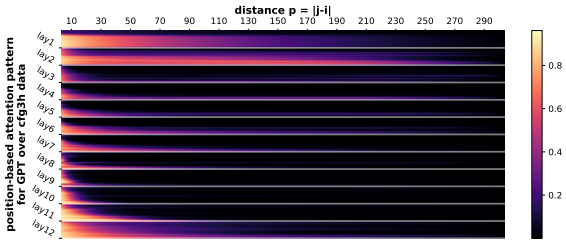


Figure 7: Position-based attention pattern of `GPT` trained on `cfg3h`. The 12 rows in each block represent attention heads. See Appendix D.1 for more experiments.

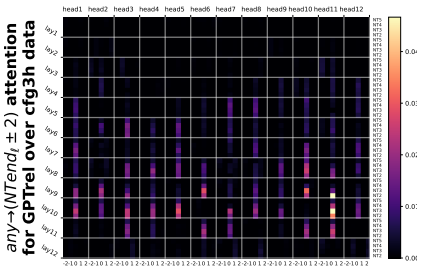
Formally, let $A_{l,h,j\to i}(x)$ for $j \geq i$ denote the attention weight for positions $j \to i$ at layer l and head h-th of the transformer, on input sequence x. In Figure 7 we visualize the attention pattern by plotting the cumulative attention weights. That is, for each layer l, head h, and distance $p \geq 0$, we compute average of partial sum $\sum_{1\leq i'\leq i} A_{l,h,j\to i'}(x)$ over all data x and pairs i, j with $j - i = p$.

We observe that the attention pattern is strongly correlated with the relative distance $p = j - i$ (this is true even if one uses the vanilla `GPT` which uses absolute positional embedding). Moreover, the attention pattern is *multi-scale*, meaning some attention heads focus on shorter distances while some focus on longer ones.

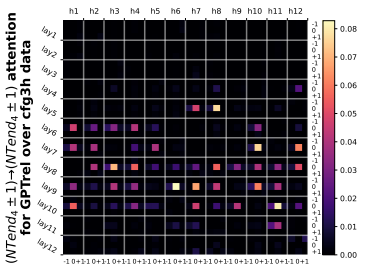
Motivated by this finding, we additionally investigate whether position-based attention *alone* is sufficient for learning CFGs. In Figure 3, we see that $\texttt{GPT}_{\texttt{pos}}$ (or even $\texttt{GPT}_{\texttt{uni}}$) can indeed perform quite well, much better than the vanilla `GPT`, though still not comparable with the full power of $\texttt{GPT}_{\texttt{rel}}$. This also gives evidence for why relative-position based transformer variants (such as $\texttt{GPT}_{\texttt{rel}}$, $\texttt{GPT}_{\texttt{rot}}$, DeBERTa) have much better practical performances comparing to their based models (`GPT` or BERT).

5.2 Boundary-Based Attention

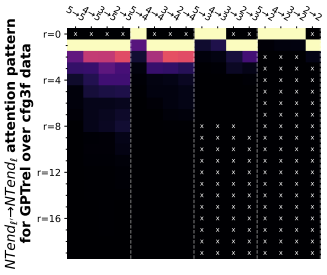
Next, we propose to *remove* the position-bias from the attention matrix to study what is left over. We observe that the transformer also learns a strong boundary-based attention pattern, which means **tokens on the NT-end boundaries typically attend to the "most adjacent" NT-end boundaries, just like standard dynamic programming for parsing CFGs (see Figure 2)**. This type of attention pattern allows the transformer to better learn the hierarchical



(a) $B_{l,h,j\to i}$ for $i+\delta$ at NT-end in CFG level ℓ. Rows represent $\ell = 2, 3, 4, 5$ and columns represent $\delta = -2, -1, 0, 1, 2$.



(b) $B_{l,h,j\to i}$ for $i+\delta_1, j+\delta_2$ at NT-ends in CFG level $\ell = 4$. Rows / columns represent $\delta_1, \delta_2 = -1, 0, +1$.



(c) $B^{\text{end}\to\text{end}}_{l,h,\ell'\to\ell,r}$ for NT-ends between CFG levels $\ell' \to \ell$. Rows represent r and columns $\ell' \to \ell$. "×" means empty entries.

Figure 8: Attention has a strong bias towards " NT-end at level ℓ' to the most adjacent NT-end at ℓ", for even different ℓ, ℓ'. For definitions see Section 5.2, and more experiments see Appendix D.2, D.3 and D.4.

and recursive structure of the CFG, and to generate the output tokens based on the NT symbols and rules.

Formally, given a sample pool $\{x^{(n)}\}_{n\in[N]} \in L(\mathcal{G})$ we compute for every layer l, head h,

$$\overline{A}_{l,h,p} = Average[\![A_{l,h,j\to i}(x^{(n)}) \mid \forall n \in N, \forall 1 \le i \le j \le \mathbf{len}(x^{(n)}) \text{ s.t. } j - i = p]\!] \ ,$$

which is the attention between any token pairs of distance p averaged over the sample pool. To remove position-bias, in this subsection, we only look at $B_{l,h,j\to i}(x) \stackrel{\text{def}}{=} A_{l,h,j\to i}(x) - \overline{A}_{l,h,j-i}$. Our observation can be understood in three steps.

- First, $B_{l,h,j\to i}(x)$ has a very strong bias towards *tokens i that are at NT ends*. Indeed, in Figure 8(a) we present the value of $B_{l,h,j\to i}(x)$, averaged over data x, over pairs i, j where $i+\delta$ is the deepest NT-end on level ℓ (in symbols, $\mathfrak{b}^\sharp(i+\delta) = \ell$). We see that the attention weights are the largest when $\delta = 0$ and drop rapidly to the surrounding tokens.
- Second, $B_{l,h,j\to i}(x)$ also has a strong bias towards pairs i, j *when both i, j are at NT ends* on some level ℓ. In Figure 8(b), we present the value of $B_{l,h,j\to i}(x)$ averaged over data x, over pairs i, j where $\mathfrak{b}_\ell(i+\delta_1) = \mathfrak{b}_\ell(j+\delta_2) = 1$ for $\delta_1, \delta_2 \in \{-1, 0, 1\}$.
- Third, $B_{l,h,j\to i}(x)$ has a strong bias towards *token pairs i, j that are "adjacent" NT-ends*. Since i and j may be NT-ends at different levels, we have to define "adjacency" carefully as follows.

  We introduce a notion $B^{\text{end}\to\text{end}}_{l,h,\ell'\to\ell,r}$, to capture $B_{l,h,j\to i}(x)$ averaged over samples x and all token pairs i, j such that, they are at deepest NT-ends on levels ℓ, ℓ' respectively (in symbols, $\mathfrak{b}^\sharp(i) = \ell \wedge \mathfrak{b}^\sharp(j) = \ell'$), and of distance r based on the ancestor indices at level ℓ (in symbols, $\mathfrak{p}_\ell(j) - \mathfrak{p}_\ell(i) = r$).

  In Figure 8(c), we observe that $B^{\text{end}\to\text{end}}_{l,h,\ell'\to\ell,r}$ is a *decreasing function of* r, and attains its largest when $r = 0$ (or $r = 1$ for those pairs $\ell' \to \ell$ that do not have the $r = 0$ entry).[6]

Therefore, we conclude that tokens corresponding to NT-ends at level ℓ' statistically have (much) higher attention weights to its *most adjacent* NT-ends at every level ℓ, even after removing the position-bias.[7] This should be reminiscent of dynamic programming.

[6] For any token pair $j \to i$ with $\ell = \mathfrak{b}^\sharp(i) \ge \mathfrak{b}^\sharp(j) = \ell'$— so i is at an NT-end closer to the root comparing to j— it satisfies $\mathfrak{p}_\ell(j) - \mathfrak{p}_\ell(i) \ge 1$ so their distance r is strictly positive.

[7] If one does not remove position-bias, such statement might be empty as the position-bias may favor "adjacent" anything, including NT-end pairs.

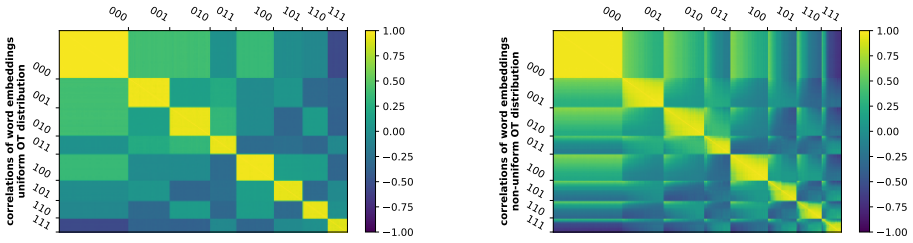


Figure 9: Language models learn implicit CFGs by using word embeddings to encode the (hidden) terminal symbol.

We present word embedding correlations for GPT pre-trained on an implicit CFG with $|\mathbf{T}| = 3$ and vocabulary size $|\mathbf{OT}| = 300$. There are 300 rows/columns each representing an observable token $a \in \mathbf{OT}$. Label $ijk \in \{0,1\}^3$ in the figure indicates whether a is in $\mathbf{OT}_t$ for the three choices $t \in \mathbf{T}$.

6 Extensions of CFGs

6.1 Implicit CFG

In an *implicit CFG*, terminal symbols represent bags of tokens with shared properties. For example, a terminal symbol like *noun* corresponds to a distribution over a bag of nouns, while *verb* corresponds to a distribution over a bag of verbs. These distributions can be non-uniform and overlapping, allowing tokens to be shared between different terminal symbols. During pre-training, the model learns to associate tokens with their respective syntactic or semantic categories, without prior knowledge of their specific roles in the CFG.

Formally, we consider a set of *observable tokens* $\mathbf{OT}$, and each terminal symbol $t \in \mathbf{T}$ in $\mathcal{G}$ is associated with a subset $\mathbf{OT}_t \subseteq \mathbf{OT}$ and a probability distribution $\mathcal{D}_t$ over $\mathbf{OT}_t$. The sets $(\mathbf{OT}_t)_t$ can be overlapping. To generate a string from this implicit CFG, after generating $x = (x_1, x_2, \ldots, x_m) \sim L(\mathcal{G})$, for each terminal symbol x_i, we independently sample one element $y_i \sim \mathcal{D}_{x_i}$. After that, we observe the new string $y = (y_1, y_2, \cdots, y_m)$, and let this new distribution be called $y \sim L_O(\mathcal{G})$

We pre-train language models using samples from the distribution $y \sim L_O(\mathcal{G})$. During testing, we evaluate the success probability of the model generating a string that belongs to $L_O(\mathcal{G})$, given an input prefix $y_{:c}$. Or, in symbols,

$$\mathbf{Pr}_{y \sim L_O(\mathcal{G}) + \text{randomness of } F}\left[(y_{:c} \circ F(y_{:c})) \in L_O(\mathcal{G})\right] \ ,$$

where $F(y_{:c})$ represents the model's generated completion given prefix $y_{:c}$. (We again use dynamic programming to determine whether the output string is in $L_O(\mathcal{G})$.) Our experiments show that language models can learn implicit CFGs also very well. By visualizing the weights of the word embedding layer, we observe that the embeddings of tokens from the same subset $\mathbf{OT}_t$ are grouped together (see Figure 9), indicating that transformer learns implicit CFGs by using its token embedding layer to encode the hidden terminal symbol information. Details are in Appendix E.

6.2 Robustness on Corrupted CFG

One may also wish to pre-train a transformer to be *robust* against errors and inconsistencies in the input. For example, if the input data is a prefix with some tokens being corrupted or missing, then one may hope the transformer to correct the errors and still complete the sentence following the correct CFG rules. Robustness is an important property, as it reflects the generalization and adaptation ability of the transformer to deal with real-world training data, which may not always follow the CFG perfectly (such as having grammar errors).

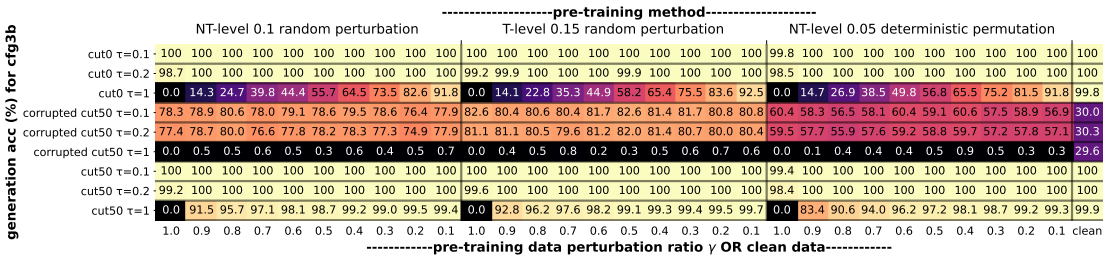


Figure 10: Generation accuracies for models pre-trained cleanly VS pre-trained over perturbed data, on clean or corrupted prefixes with cuts $c = 0$ or $c = 50$, using generation temperatures $\tau = 0.1, 0.2, 1.0$.

Observation. In Rows 4/5, by comparing against the last column, we see it is *beneficial* to include low-quality data (e.g. grammar mistakes) during pre-training. The amount of low-quality data could be little ($\gamma = 0.1$ fraction) or large (*every training sentence may have grammar mistake*). The transformer also learns a "mode switch" between the "correct mode" or not; details in Section 6.2.

To test robustness, for each input prefix $x_{:c}$ of length c that belongs to the CFG, we randomly select a set of positions $i \in [c]$ in this prefix— each with probability ρ— and flip them i.i.d. with a random symbol in $\mathbf{T}$. Call the resulting prefix $\widetilde{x}_{:c}$. Next, we feed the *corrupted prefix* $\widetilde{x}_{:c}$ to the transformer F and compute its generation accuracy in the uncorrupted CFG: $\mathbf{Pr}_{x\sim L(\mathcal{G}),\,F}[(x_{:c} \circ F(\widetilde{x}_{:c})) \in L(\mathcal{G})]$.

We not only consider clean pre-training, but also some versions of *robust pre-training*. That is, we randomly select $\gamma \in [0, 1]$ fraction of the training data and perturb them before feeding into the pre-training process. We compare three types of data perturbations.[8]

- (T-level random perturbation). Each x_i w.p. 0.15 we replace it with a random symbol in $\mathbf{T}$.
- (NT-level random perturbation). Let $\ell = L - 1$ and recall $s_\ell = \left(s_{\ell,1}, s_{\ell,2}, \ldots, s_{\ell,m_{L-1}}\right)$ is the sequence of symbols at NT-level ℓ. For each $s_{\ell,i}$, w.p. 0.10 we perturb it to a random symbol in $\mathbf{NT}_\ell$; and then generate $x = s_L$ according to this perturbed sequence.
- (NT-level deterministic perturbation). Let $\ell = L - 1$ and fix a permutation π over symbols in $\mathbf{NT}_\ell$. For each $s_{\ell,i}$, w.p. 0.05 we perturb it to its next symbol in $\mathbf{NT}_{L-1}$ according to π; and then generate $x = s_L$ according to this perturbed sequence.

We focus on $\rho = 0.15$ with a wide range of perturbation rate $\tau = 0.0, 0.1, \ldots, 0.9, 1.0$. We present our findings in Figure 10. Noticeable observations include:

- Rows 4/5 of Figure 10 suggest that GPT models are not so robust (e.g., $\sim 30\%$ accuracy) when training over clean data $x \sim L(\mathcal{G})$. If we train from perturbed data— *both* when $\gamma = 1.0$ so all data are perturbed, *and* when $\gamma = 0.1$ so we have a tiny fraction of perturbed data— GPT can achieve $\sim 79\%, 82\%$ and 60% robust accuracies respectively using the three types of data perturbations (Rows 4/5 of Figure 10). This suggest that it is actually *beneficial* in practice to include corrupted or low-quality data during pre-training.
- Comparing Rows 3/6/9 of Figure 10 for temperature $\tau = 1$, we see that pre-training teaches the language model to actually include a *mode switch*. When given a correct prefix it is in the *correct mode* and completes the sentence with a correct string in the CFG (Row 9); when given corrupted prefixes, it *always* completes sentences with grammar mistakes (Row 6); when given no prefix it generates corrupted strings with probability close to γ (Row 3).

[8] One can easily extend our experiments by considering other types of data corruption (for evaluation), and other types of data perturbations (for training). We refrain from doing so because it is beyond the scope of this paper.

- Comparing Rows 4/5 to Row 6 in Figure 10 we see that high robust accuracy is achieved when generating using low temperatures τ.[9] This should not be surprising given that the language model learned a "mode switch." Using low temperature encourages the model to, for each next token, pick a more probable solution. This allows it to achieve good robust accuracy *even when* the model is trained totally on corrupted data ($\gamma = 1.0$).

  Please note this is consistent with practice: when feeding a pre-trained large language model (such as LLaMA-30B) with prompts of grammar mistakes, it tends to produce texts also with (even new!) grammar mistakes when using a large temperature.

Our experiments seem to suggest that, additional instruct fine-tuning may be necessary, if one wants the model to *always* be in the "correct mode." This is beyond the scope of this paper.

7 Conclusion

In this paper, we designed experiments to investigate how generative language models based on transformers, specifically GPT-2, learn and generate context-free grammars (CFGs). By pre-training such model on CFGs with varying difficulty levels, we examined its ability to generate accurate and *diverse* strings according to the grammar rules. We believe our findings offer a comprehensive and empirical understanding of how transformers learn CFGs, uncovering the underlying *physical mechanisms* that enable the models to capture the hierarchical structure of CFGs and resemble dynamic programming. This research contributes to the fields by providing insights into how language models can effectively learn and generate complex and diverse expressions. It also offers valuable tools for interpreting and understanding the inner workings of these models. Our study opens up several exciting avenues for future research, including:

- Exploring other classes of grammars, such as context-sensitive grammars [16, 40], and investigating how the network can learn context-dependent information.
- Studying the transfer and adaptation of the learned network representation to different domains and tasks, especially using low-rank update [14], and evaluating how the network can leverage the grammatical knowledge learned from the CFGs.
- Extending the analysis and visualization of the network to other aspects of the languages, such as the semantics, the pragmatics, and the style.

We hope that our paper will inspire more research on understanding and improving the generative language models based on transformers, and on applying them to natural language, coding problems, and beyond.

[9] Recall, when temperature $\tau = 0$ the generation is greedy and deterministic; when $\tau = 1$ it reflects the unaltered distribution learned by the transformer; when $\tau > 0$ s small it encourages the transformer to output "more probable" tokens.

Appendix

A Experiment Setups

A.1 Dataset Details

We construct seven synthetic CFGs of depth $L = 7$ of different levels of learning difficulties. One can easily imagine the greater the number of T/NT symbols are, the harder it is to learn the CFG. For such reason, to test the power of language models to the extreme, we mainly focus on cfg3b, cfg3i, cfg3h, cfg3g, cfg3f of sizes $(1, 3, 3, 3, 3, 3, 3)$, with increasing difficulties. Their details are given in Figure 11:

- In cfg3b we construct the CFG with degree $|\mathcal{R}(a)| = 2$ for every NT a; we also made sure in any generation rule, consecutive pairs of T/NT symbols are distinct.
- In cfg3i we let $|\mathcal{R}(a)| = 2$ for every NT a; we drop the distinctness requirement to make the data harder than cfg3b.
- In cfg3h we let $|\mathcal{R}(a)| \in \{2, 3\}$ for every NT a to make the data harder than cfg3i.
- In cfg3g we let $|\mathcal{R}(a)| = 3$ for every NT a to make the data harder than cfg3h.
- In cfg3f we let $|\mathcal{R}(a)| \in \{3, 4\}$ for every NT a to make the data harder than cfg3g.

The sequences generated by such CFGs are of typical length around $250 \sim 500$.

Remark A.1. From the examples in Figure 11 it is perhaps clear that for such grammars $\mathcal{G}$ of depth 7, it is already very non-trivial even for a human being to prove that a string x belongs to $L(\mathcal{G})$, even when the CFG is known. The canonical way of showing $x \in L(\mathcal{G})$ is via dynamic programming.

Remark A.2. cfg3f is a dataset that is right at the difficulty boundary in which GPT2-small is capable of learning (see subsequent subsections for training parameters). One can certainly consider deeper and harder CFGs, and this requires training over a larger network for a longer time. We refrain from doing so because our findings are sufficiently convincing at the level of cfg3f.

At the same time, to demonstrate that transformers can learn CFGs with larger $|\mathbf{NT}|$ or $|\mathbf{T}|$, we construct datasets cfg3e1 and cfg3e2 respectively of size $(1, 3, 9, 27, 81, 27, 9)$ and $(1, 3, 9, 27, 27, 9, 4)$. We ignore their details since they are long to describe.

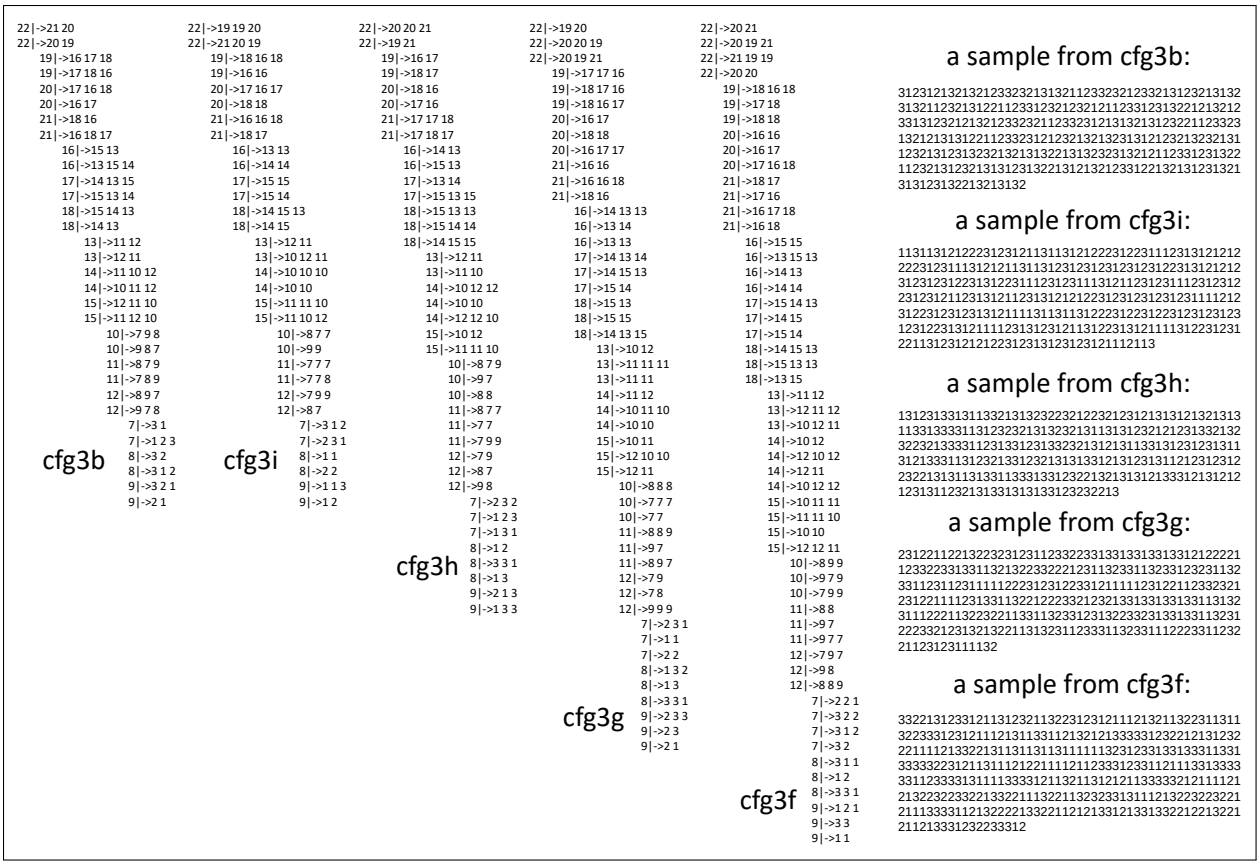


Figure 11: The context-free grammars cfg3b, cfg3i, cfg3h, cfg3g, cfg3f that we primarily use in this paper, together with a sample string from each of them.

Observation. Although those CFGs are only of depth 7, they are capable of generating sufficiently long and hard instances; after all, even when the CFG rules are given, the canonical way to decide if a string x belongs to the CFG language $x \in L(\mathcal{G})$ may require dynamic programming.

A.2 Model Architecture Details

We let `GPT` be the vanilla GPT2-small architecture [29], with 12 layers, 12 attention heads per layer, and 768(=12 × 64) hidden dimensions. We pre-train `GPT` over the aforementioned datasets from random initialization. As a baseline comparison, we also implemented DeBERTa [12] and resize it to match that of GPT2— so also with 12 layers, 12 attention heads, and 768 dimensions.

Architecture size. We have also played with models of smaller or larger sizes, and noticed that their learning capabilities scale with the difficulty levels of the CFGs. For a fair comparison, and for better reproducibility, we decide to focus primarily on 12-layer, 12-head, 768 dimensions and the so-constructed transformers have 85.87M parameters.

Modern GPTs with relative attention. More recent research [7, 12, 35] demonstrated that transformers can have a significant performance boost using attentions that are based on tokens' *relative* position differences, as opposed to absolute positions like in the original GPT2 [29] or BERT [17]. There are two lines of approaches to achieve this. One is to exactly use a "relative positional embedding layer" on $|j-i|$ when calculating the attention $j \to i$ (or a bucket embedding to save space). This approach is the most effective but trains slower. The other is to apply a rotary

positional embedding (RoPE) transformation [35] on the hidden states; this is known to be slightly worse than the relative approach, but can be trained much faster.

We have implemented both. We simply adopt the RoPE implementation from the GPT-NeoX-20B project (and the default parameters), but downsize it to GPT2 small. We call this architecture $\texttt{GPT}_{\texttt{rot}}$. We did not find a standard implementation of GPT using relative attention, so we have re-implemented GPT2 using the relative attention framework from DeBERTa [12]. (Recall, DeBERTa is one of BERT variants that made great use of relative positional embeddings.) We call this architecture $\texttt{GPT}_{\texttt{rel}}$.

Weaker GPTs using only position-based attention. For analysis purpose, we also include two much weaker variants of GPT, where the attention matrix *only depends* on the token positions, but not on the input sequences or hidden embeddings. In other words, the attention pattern remains *fixed* for all input sequences.

We implement $\texttt{GPT}_{\texttt{pos}}$, which is a variant of $\texttt{GPT}_{\texttt{rel}}$ but restricting the attention matrix to be computed only using the (trainable) relative positional embedding. This can be viewed as a GPT variant that *makes the maximum use of the position-based attention*. We also implement $\texttt{GPT}_{\texttt{uni}}$, which is a 12-layer, 8-head, 1024-dim transformer, where the attention matrix is *fixed*; for each $h \in [8]$, the h-th head *always* use a fixed, uniform attention over the previous $2^h - 1$ tokens. This can be viewed as a GPT variant that *makes the simplest use of the position-based attention*.

Remark A.3. It should not be surprising that $\texttt{GPT}_{\texttt{pos}}$ or $\texttt{GPT}_{\texttt{uni}}$ perform much worse than other GPT models on real-life wikibook pre-training. However, once again, we use them only for *analysis purpose* in this paper, as we wish to demonstrate what is the maximum power of GPT when only using position-based attention to learn CFGs, and what is the marginal effect when one goes *beyond* position-based attention.

Features from random transformer. Finally we also consider a randomly-initialized $\texttt{GPT}_{\texttt{rel}}$, and use those random features for the purpose of predicting NT ancestors and NT ends. This serves as a baseline, and can be viewed as the power of the so-called (finite-width) neural tangent kernel [2]. We call this $\texttt{GPT}_{\texttt{rand}}$.

A.3 Pre-Training Details

For each sample $x \sim L(\mathcal{G})$ we append it to the left with a BOS token and to the right with an EOS token. Then, following the tradition of language modeling (LM) pre-training, we concatenate consecutive samples and randomly cut the data to form sequences of a fixed window length 512.

As a baseline comparison, we also applied DeBERTa on a masked language modeling (MLM) task for our datasets. We use standard MLM parameters: 15% masked probability, in which 80% chance of using a masked token, 10% chance using the original token, and 10% chance using a random token.

We use standard initializations from the huggingface library. For GPT pre-training, we use AdamW with $\beta = (0.9, 0.98)$, weight decay 0.1, learning rate 0.0003, and batch size 96. We pre-train the model for 100k iterations, with a linear learning rate decay.[10] For DeBERTa, we use learning rate 0.0001 which is better and 2000 steps of learning rate linear warmup.

Throughout the experiments, we perform pre-training only using fresh samples from the CFG datasets (thus using 4.9 billion tokens $= 96 \times 512 \times 100k$). We have also tested pre-training with a finite training set of $100m$ tokens; and the conclusions of this paper stay similar. To make this

[10] We have slightly tuned the parameters to make pre-training go best. We noticed for training GPTs over our CFG data, a warmup learning rate schedule is not needed.

paper clean, we choose to stick to the infinite-data regime in this version of the paper, because it enables us to make negative statements (for instance about the vanilla GPT or DeBERTa, or about the learnability of NT ancestors / NT boundaries) without worrying about the sample size.

As for the reproducibility of our result, we did not run each pre-train experiment more than once (or plot any confidence interval). This is because, rather than repeating our experiments identically, it is obviously more interesting to use the resources to run it against different datasets and against different parameters. We pick the best model using the perplexity score from each pre-training task. When evaluating the generation accuracy in Figure 3, we have generated more than 20000 samples for each case, and present the diversity pattern accordingly in Figure 4.

A.4 Predict NT ancestor and NT boundary

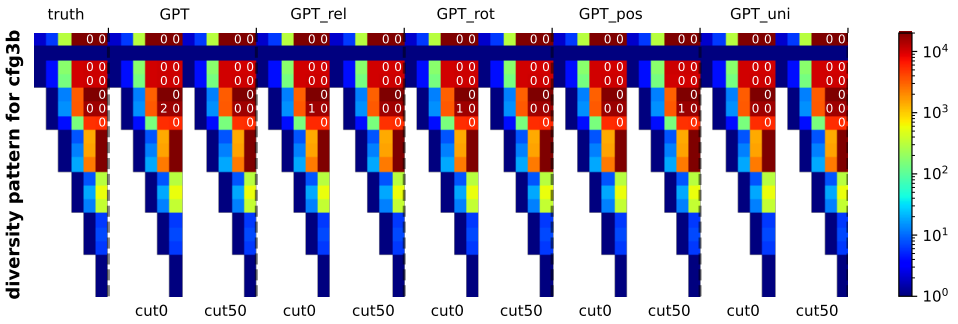
Recall from Section 4.1 that we have proposed to use a linear function to probe whether or not the hidden states of a transformer, implicitly encodes the NT ancestor and NT boundary information for each token position. Since this linear function is of dimension 512×768— where 512 comes from the sequence length and 768 comes from the hidden dimension size— recall in (4.1), we have proposed to use position-based attention to construct such linear function. This significantly reduces sample complexity and makes it much easier to find the linear function.

In our implementation, we choose 16 heads and hidden dimension $d' = 1024$ when constructing this position-based attention in (4.1). We have also tried other parameters but the NT ancestor/boundary prediction accuracies are not very sensitive to such architecture change. We again use AdamW with $\beta = (0.9, 0.98)$ but this time with learning rate 0.003, weight decay 0.001, batch size 60 and train for 30k iterations.

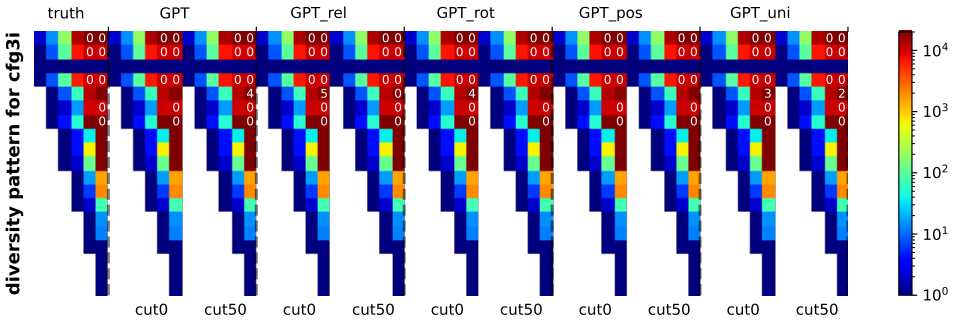
B More Experiments on Generation

B.1 Generation Diversity

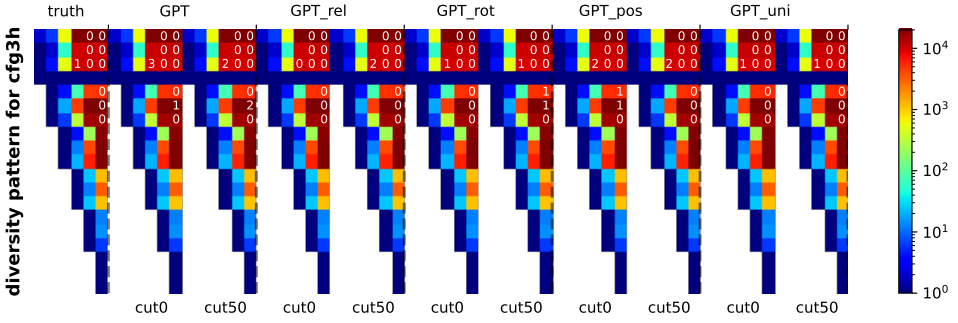
Recall we have included the diversity pattern for models pre-trained over cfg3f in Figure 4. Below in Figure 12 we present that for cfg3b, cfg3i, cfg3h, cfg3g, in Figure 13 for cfg3e1, and in Figure 14 for cfg3e2. We note that not only for hard, ambiguous datasets, also for those less ambiguous (cfg3e1, cfg3e2) datasets, language models are capable of generating very diverse outputs.



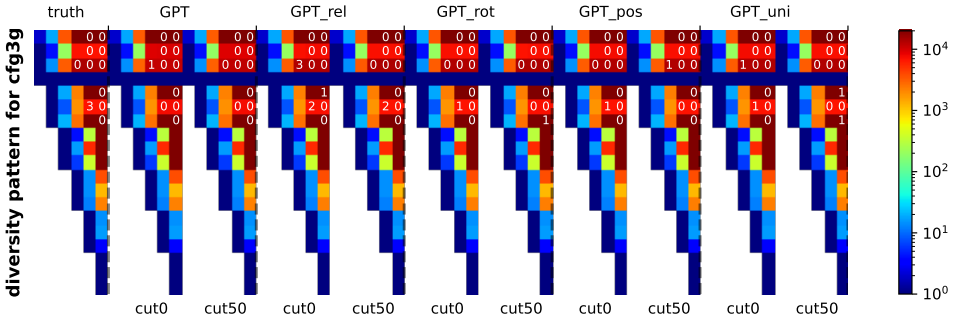
(a) cfg3b dataset



(b) cfg3i dataset



(c) cfg3h dataset



(d) cfg3g dataset

Figure 12: Comparing the generation diversity $\mathcal{S}^{\mathsf{truth}}_{a\to\ell_2}$ and $\mathcal{S}^{F}_{a\to\ell_2}$ across different learned GPT models (and for $c = 0$ or $c = 50$). Rows correspond to NT symbols a and columns correspond to $\ell_2 = 2, 3, \ldots, 7$. Colors represent the number of distinct elements in $\mathcal{S}^{\mathsf{truth}}_{a\to\ell_2}$, and the white numbers represent the collision counts (if not present, meaning there are more than 5 collisions).

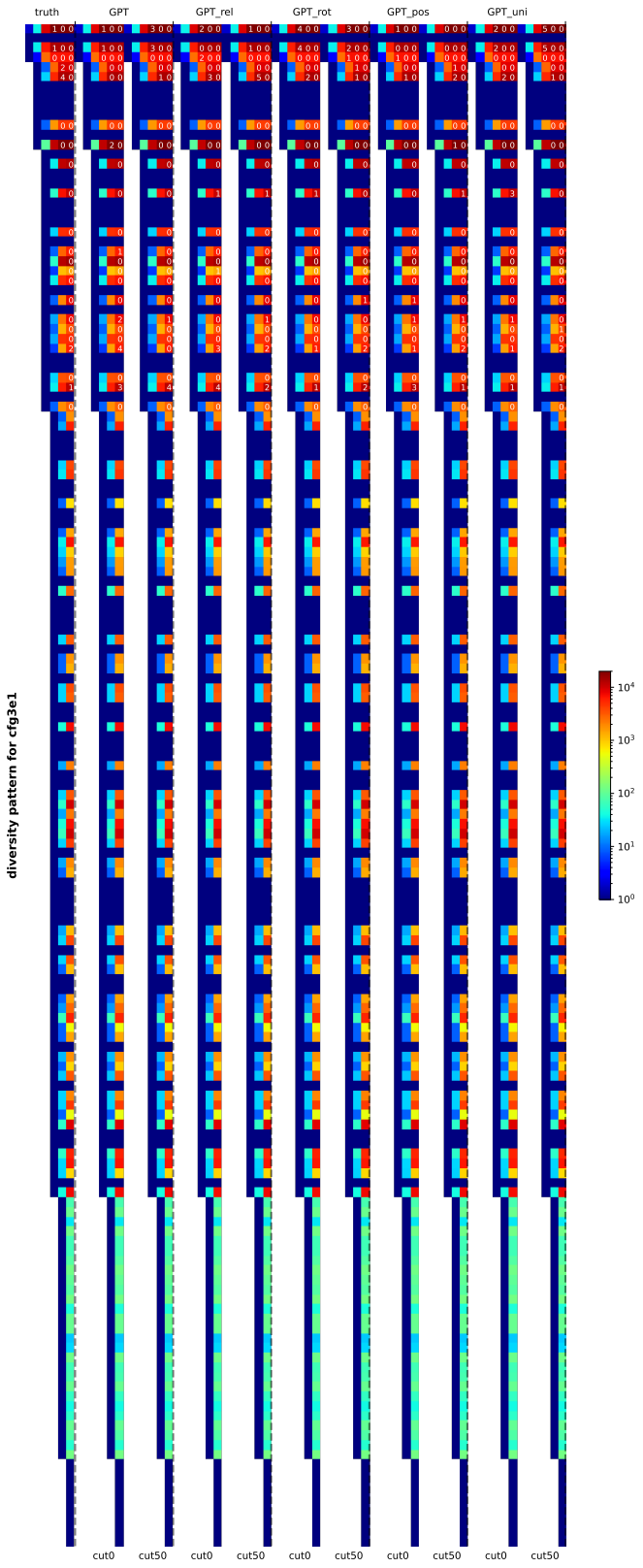


Figure 13: Comparing the generation diversity $\mathcal{S}^{\mathsf{truth}}_{a\to\ell_2}$ and $\mathcal{S}^{F}_{a\to\ell_2}$ across different learned GPT models (and for $c=0$ or $c=50$). Rows correspond to NT symbols a and columns correspond to $\ell_2 = 2, 3, \ldots, 7$. Colors represent the number of distinct elements in $\mathcal{S}^{\mathsf{truth}}_{a\to\ell_2}$, and the white numbers represent the collision counts (if not present, meaning there are more than 5 collisions). This is for the `cfg3e1` dataset.

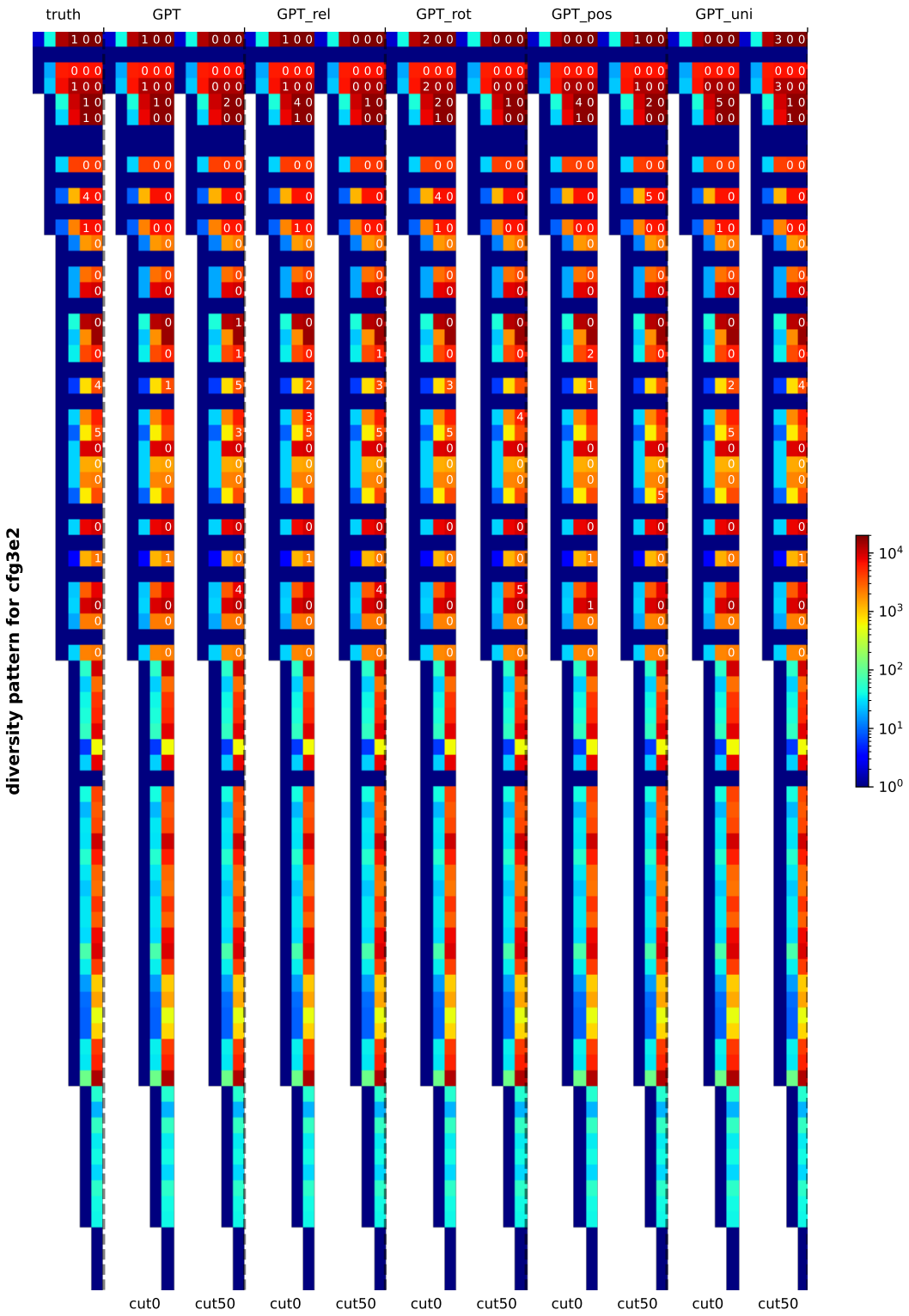
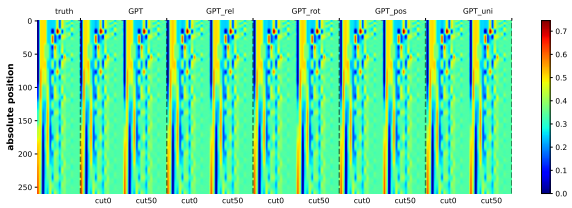


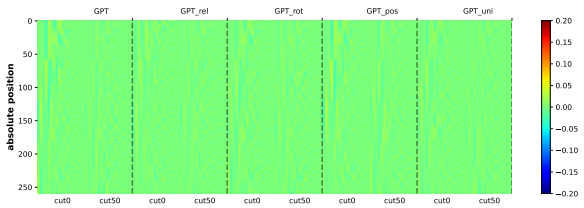
Figure 14: Comparing the generation diversity $\mathcal{S}^{\text{truth}}_{a\to\ell_2}$ and $\mathcal{S}^{F}_{a\to\ell_2}$ across different learned GPT models (and for $c = 0$ or $c = 50$). Rows correspond to NT symbols a and columns correspond to $\ell_2 = 2, 3, \ldots, 7$. Colors represent the number of distinct elements in $\mathcal{S}^{\text{truth}}_{a\to\ell_2}$, and the white numbers represent the collision counts (if not present, meaning there are more than 5 collisions). This is for the cfg3e2 dataset.

B.2 Marginal Distribution Comparison

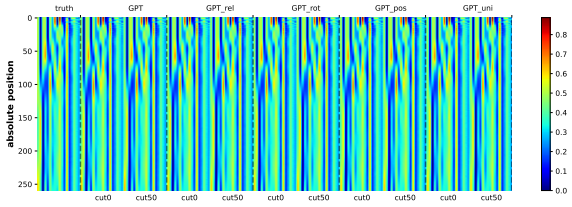
In order to effectively learn a CFG, it is also important to match the distribution of generating probabilities. While measuring this can be challenging, we have conducted at least a simple test on the marginal distributions $p(a, i)$, which represent the probability of symbol $a \in \mathbf{NT}_\ell$ appearing at position i (i.e., the probability that $\mathfrak{s}_\ell(i) = a$). We observe a strong alignment between the generated probabilities and the ground-truth distribution. See Figure 15.



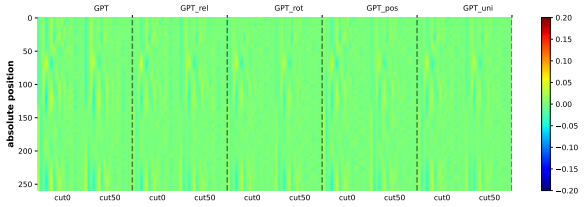
(a) cfg3b dataset; marginal distribution



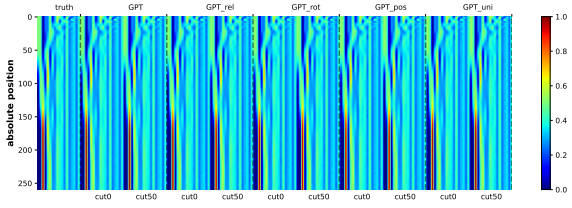
(b) cfg3b dataset; marginal distribution - ground truth



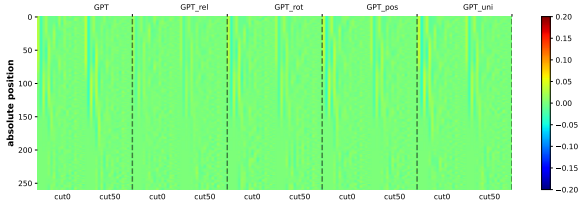
(c) cfg3i dataset; marginal distribution



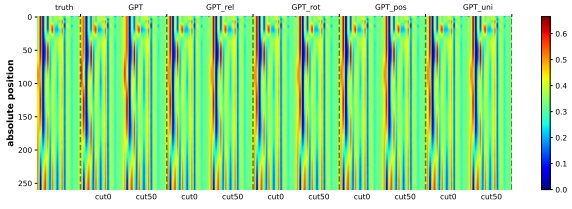
(d) cfg3i dataset; marginal distribution - ground truth



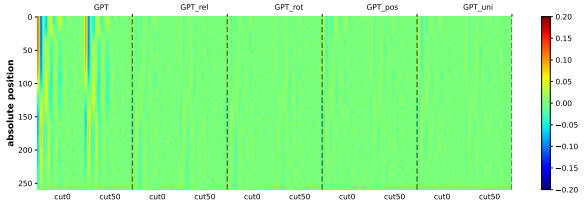
(e) cfg3h dataset; marginal distribution



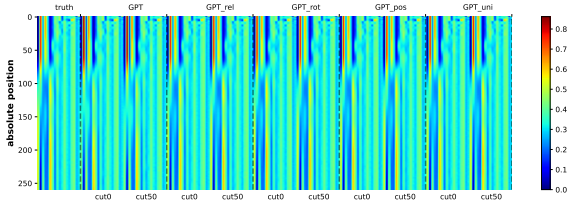
(f) cfg3h dataset; marginal distribution - ground truth



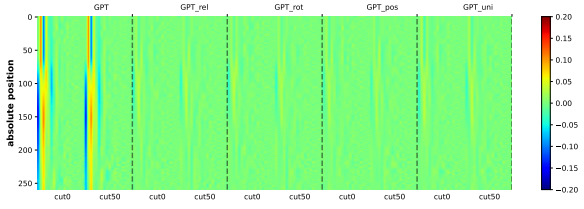
(g) cfg3g dataset; marginal distribution



(h) cfg3g dataset; marginal distribution - ground truth



(i) cfg3f dataset; marginal distribution



(j) cfg3f dataset; marginal distribution - ground truth

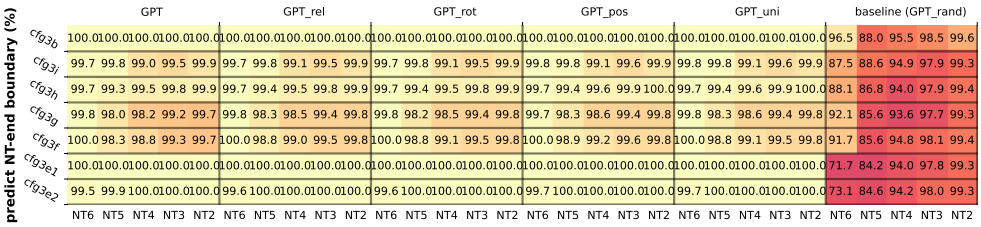
Figure 15: Marginal distribution $p(a, i)$ difference between a trained model and the ground-truth, for an NT/T symbol a (column) at position i (row). Figures on the left compare the marginal distribution of the ground-truth against those generated from 5 models $\times$ 2 cut positions ($c = 0/c = 50$). Figures on the right showcase the marginal distribution *difference* between them and the ground-truth. It is noticeable from the figures that GPT did not learn cfg3g and cfg3f well. This is consistent with the generation accuracies in Figure 3.

C More Experiments on NT Ancestor and NT Boundary Predictions

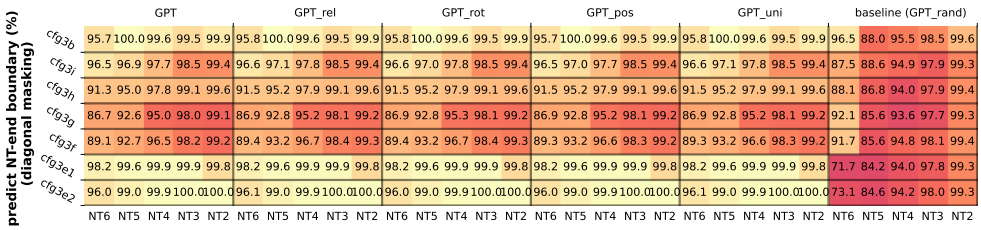
C.1 NT Ancestor and NT Boundary Predictions

Earlier in Figure 5 we have confirmed that the hidden states (of the last transformer layer) have implicitly encoded very accurate knowledge of the NT ancestor symbols $\mathfrak{s}_\ell(i)$ for each CFG level ℓ and token position i. In Figure 16(a) we also demonstrate the same conclusion holds for the NT-end boundary information $\mathfrak{b}_\ell(i)$.

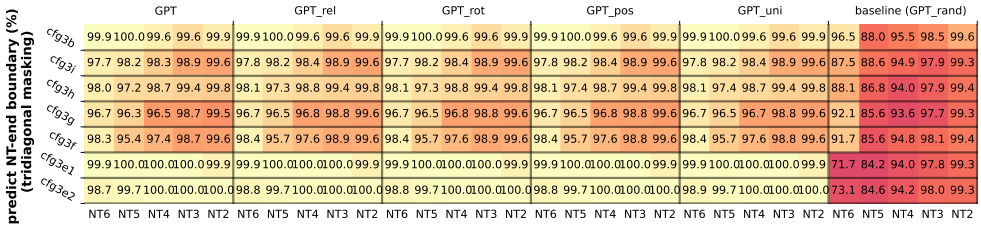
Furthermore, recall in Figure 6 we also confirmed that at any NT boundary, transformer has also locally (either exactly or $\delta = \pm 1$) encoded clear information about the NT ancestor symbol. Note that is a conditional statement— conditioning on NT boundaries, NT ancestors can be predicted— so in principle, one must also verify that the prediction task for NT boundary is already successful to begin with. We include such missing experiments in Figure 16(b) and Figure 16(c).



(a) Predicting NT boundaries



(b) Predicting NT boundaries with diagonal masking



(c) Predicting NT boundaries with tridiagonal masking

Figure 16: After pre-training, hidden states of generative models implicitly encode the NT boundary information, up to a linear transformation (*c.f.* that of the NT ancestor information in Figure 5).

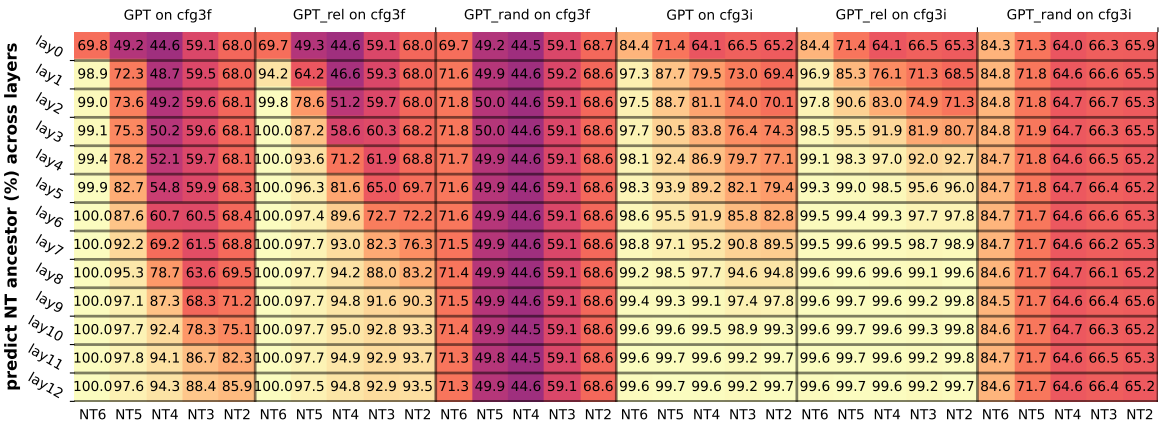
Observation. Such NT-end boundary information— i.e., $\mathfrak{b}_\ell(i) = 1$ for position i and NT level ℓ— is largely stored *locally* near the hidden state at position $i \pm 1$. This implies, the transformer actually *knows*, with a very good accuracy, that "position i is already the end of NT on level ℓ", by just reading all the texts until this position (possibly peeking one more to its right).

Remark 1. It may be mathematically necessary to peek more than 1 tokens to decide if a position i is at an NT boundary, due to CFG's ambiguity. But, in most cases, that can be decided quite early.

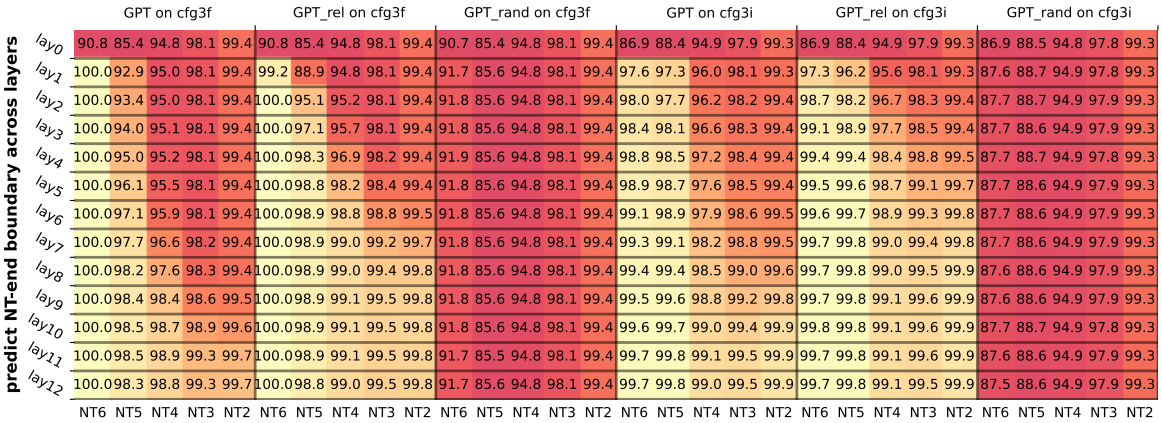
Remark 2. Predicting NT boundary is a very *biased* binary classification task. For levels ℓ that are close to the CFG root, most symbols are not at NT boundary for that level ℓ (see Figure 2). For such reason, in the *heatmap color* of the figures above, we have *normalized* the columns with respect to NT2..NT6 differently, to reflect this bias.

C.2 NT Predictions Across Transformer's Layers

As one may image, the NT ancestor and boundary information for smaller CFG levels ℓ (i.e., closer to CFG root) are only learned at those deeper transformer layers l. In Figure 17, we present this finding by calculating the *linear* encoding accuracies with respect to all the 12 transformer layers in GPT and $\texttt{GPT}_{\texttt{rel}}$. We confirm that generative models discover such information *hierarchically*.



(a) Predict NT ancestors, comparing against the $\texttt{GPT}_{\texttt{rand}}$ baseline



(b) Predict NT boundaries, comparing against the $\texttt{GPT}_{\texttt{rand}}$ baseline

Figure 17: Generative models discover NT ancestors and NT boundaries hierarchically.

C.3 NT Predictions Across Training Epochs

Moreover, one may conjecture that the NT ancestor and NT boundary information is learned *gradually* as the number of training steps increase. We have confirmed this in Figure 18. We emphasize that this does not imply layer-wise training is applicable in learning deep CFGs. It is crucial to train all the layers together, as the training process of deeper transformer layers may help backward correct the features learned in the lower layers, through a process called "backward feature correction" [1].

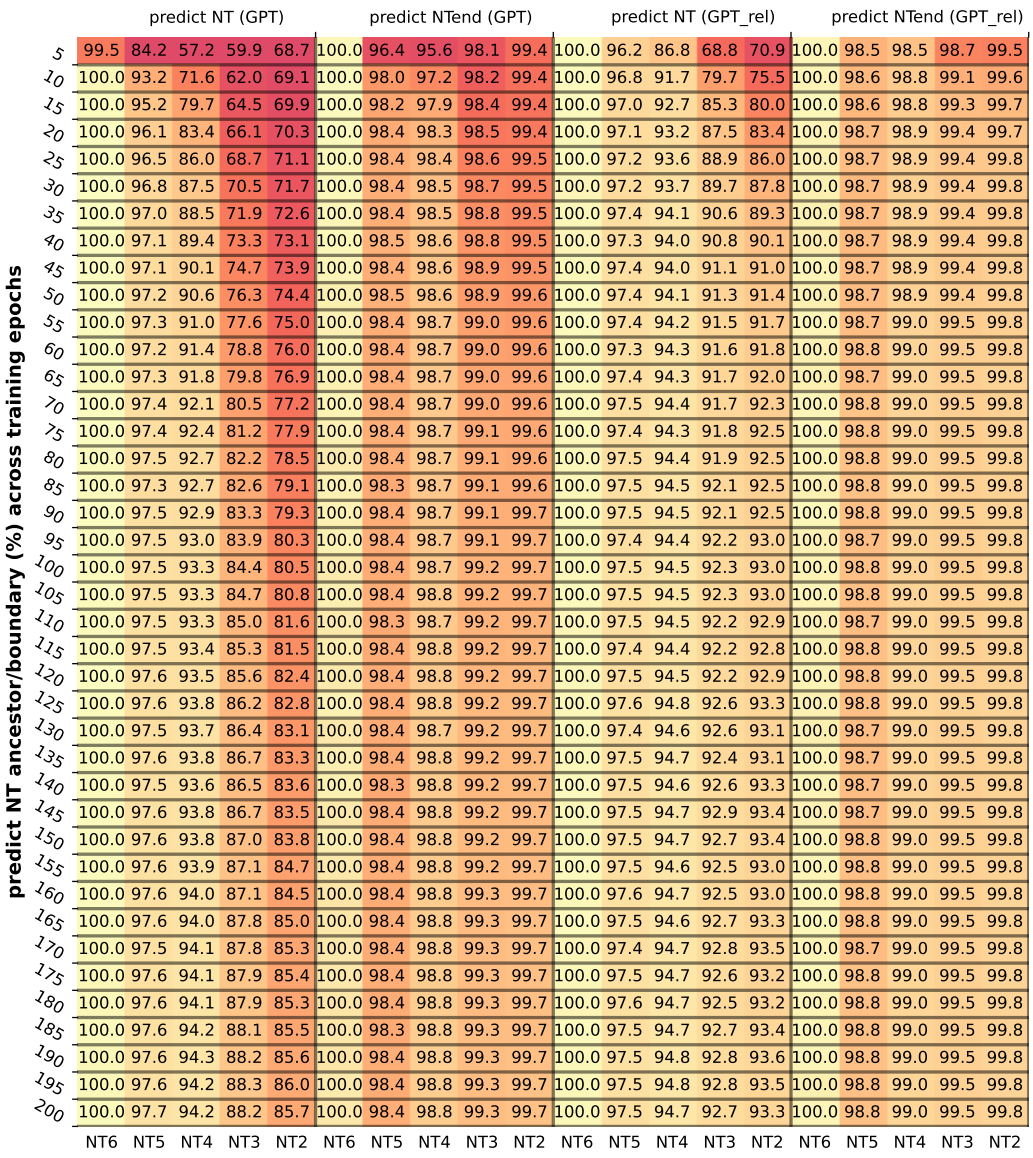
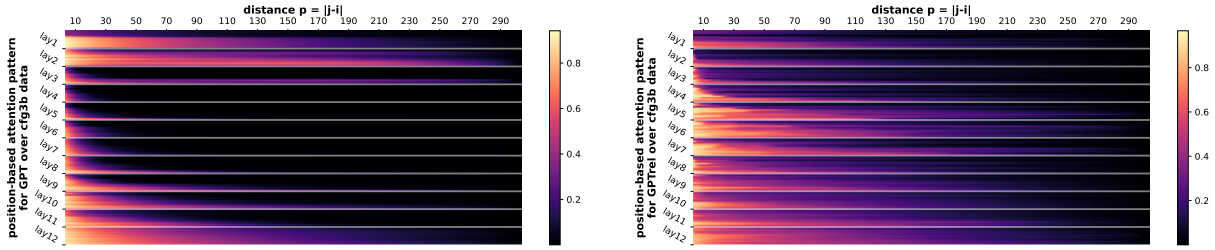


Figure 18: Generative models discover NT ancestors and NT boundaries gradually across training epochs (here 1 epoch equals 500 training steps). CFG levels closer to the leaves are learned faster, and their accuracies continue to increase as deeper levels are being learned, following a principle called "backward feature correction" in deep hierarchical learning [1].

D More Experiments on Attention Patterns

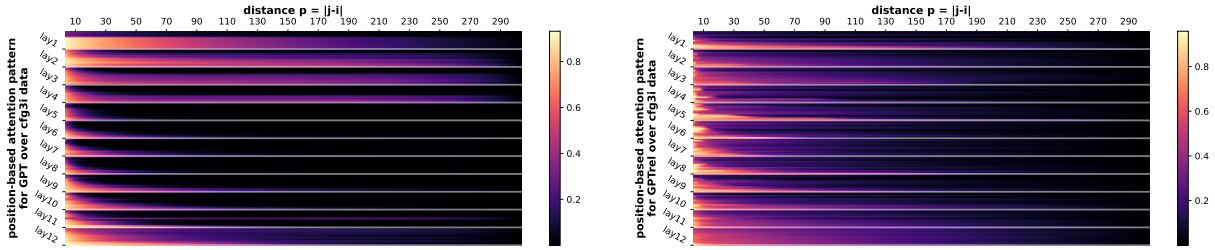
D.1 Position-Based Attention Pattern

Recall from Figure 7 we have shown that the attention weights between any two positions $j \to i$ have a strong bias in the relative difference $p = |j - i|$. Different heads or layers have different dependencies on p. Below in Figure 19, we give experiments for this phenomenon in more datasets and in both $\texttt{GPT}/\texttt{GPT}_{\mathsf{rel}}$.



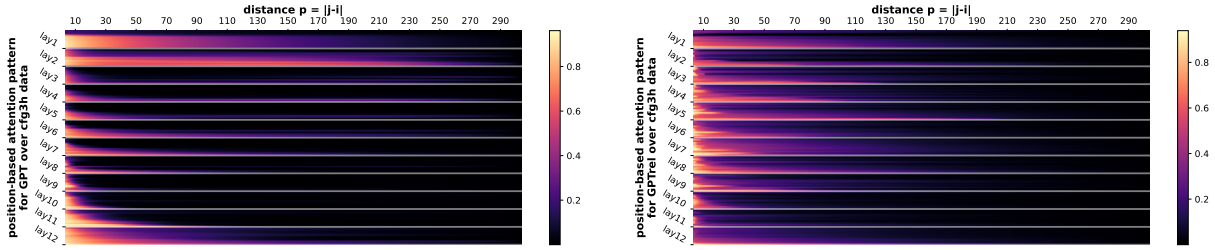
(a) GPT on cfg3b

(b) GPT_{rel} on cfg3b



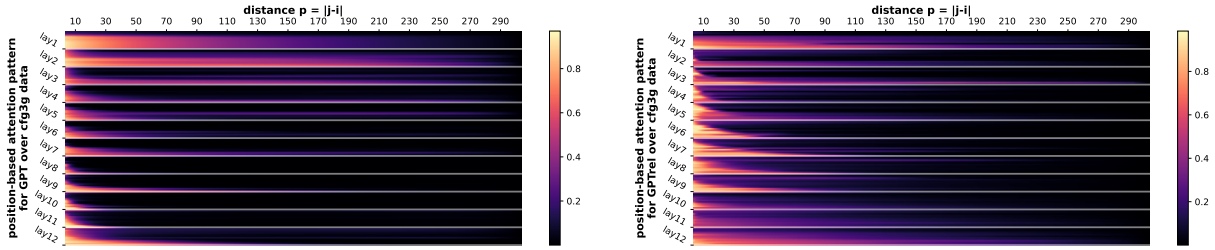
(c) GPT on cfg3i

(d) GPT_{rel} on cfg3i



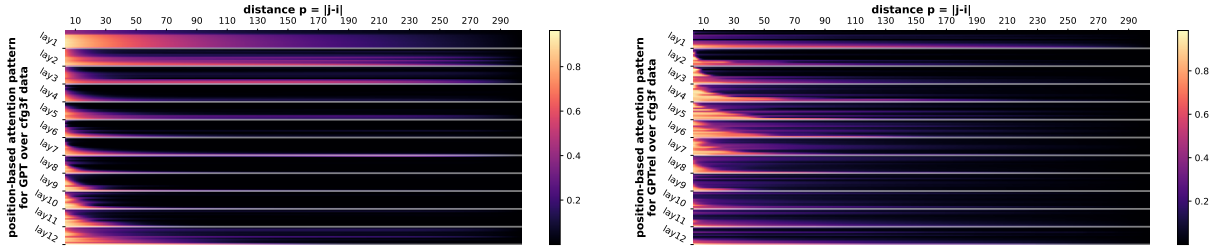
(e) GPT on cfg3h

(f) GPT_{rel} on cfg3h



(g) GPT on cfg3g

(h) GPT_{rel} on cfg3g



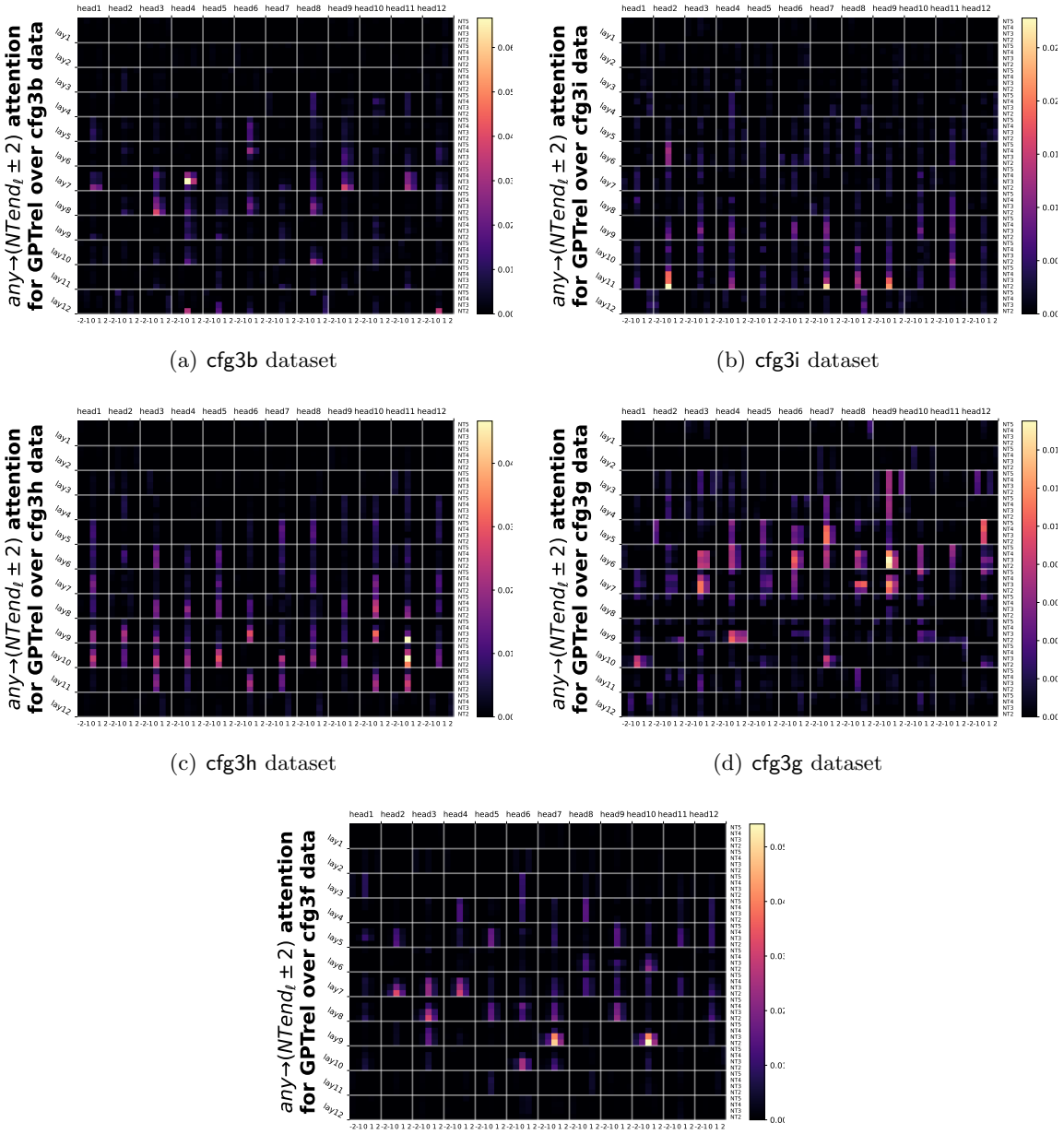
(i) GPT on cfg3f

(j) GPT_{rel} on cfg3f

Figure 19: Position-based attention pattern. The 12 rows in each layer represent 12 heads. Different heads or layers have different dependencies on p.

D.2 From Anywhere to NT-ends

Recall from Figure 8(a), we showed that after removing the position-bias $B_{l,h,j\to i}(x) \stackrel{\text{def}}{=} A_{l,h,j\to i}(x) - \overline{A}_{l,h,j-i}$, the attention weights have a very strong bias towards *tokens i that are at NT ends*. In Figure 20 we complement this experiment with more datasets.



(a) cfg3b dataset

(b) cfg3i dataset

(c) cfg3h dataset

(d) cfg3g dataset

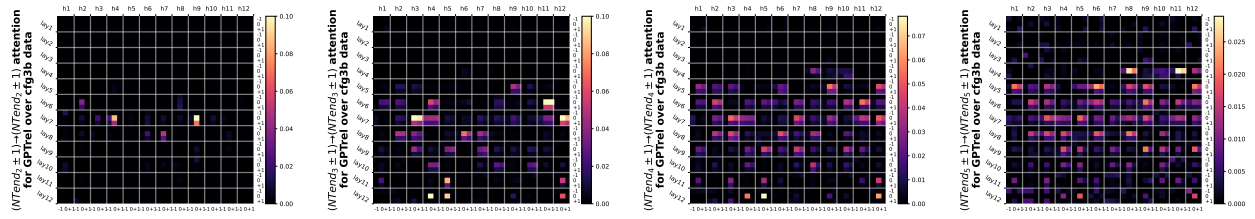
(e) cfg3f dataset

Figure 20: Attention weights $B_{l,h,j\to i}(x)$ averaged over data x and pairs i, j such that $i+\delta$ is at the NT-end in level ℓ of the CFG. In each cell, the four rows correspond to levels $\ell = 2, 3, 4, 5$, and the five columns represent $\delta = -2, -1, 0, +1, +2$.

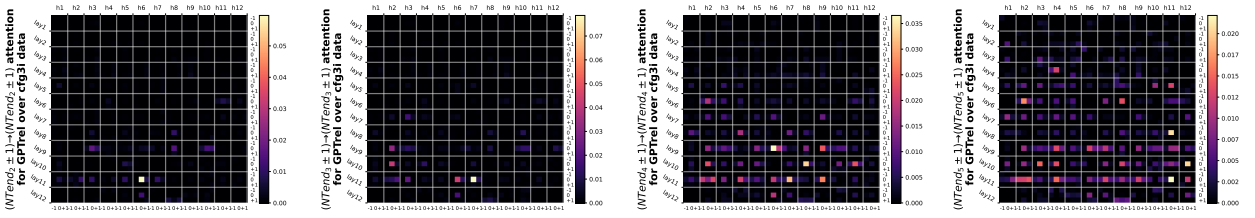
Observation. Attention is largest when $\delta = 0$ and drops rapidly to the surrounding tokens of i.

D.3 From NT-ends to NT-ends

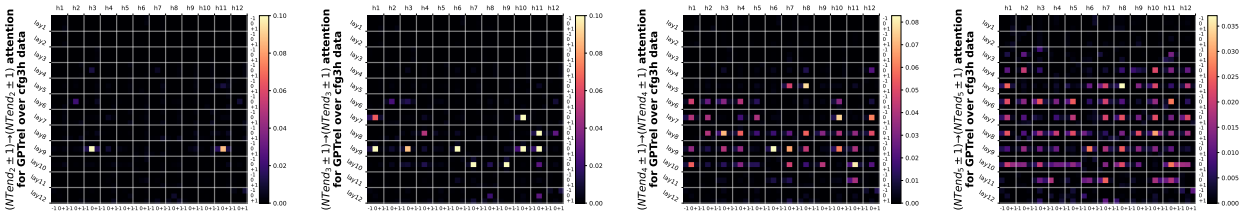
We mentioned in Section 5.2, not only tokens generally attend more to NT-ends, in fact, among those attentions, *NT-ends* also attend *more likely* to NT-ends. We include this experiment in Figure 21 for every different level $\ell = 2, 3, 4, 5$, between any two pairs $j \to i$ that are both at NT-ends for level ℓ.



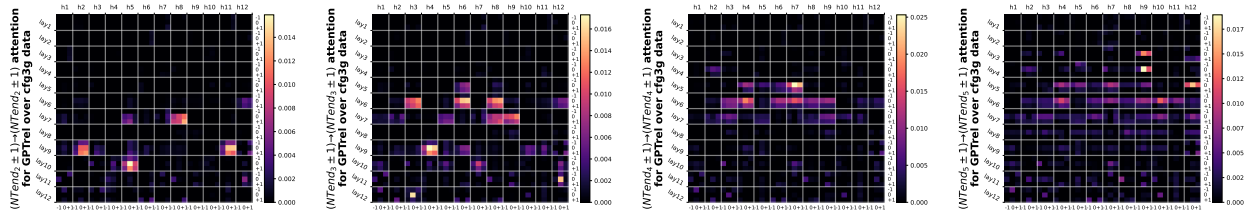
(a) cfg3b at level $\ell = 2$ (b) cfg3b at level $\ell = 3$ (c) cfg3b at level $\ell = 4$ (d) cfg3b at level $\ell = 5$



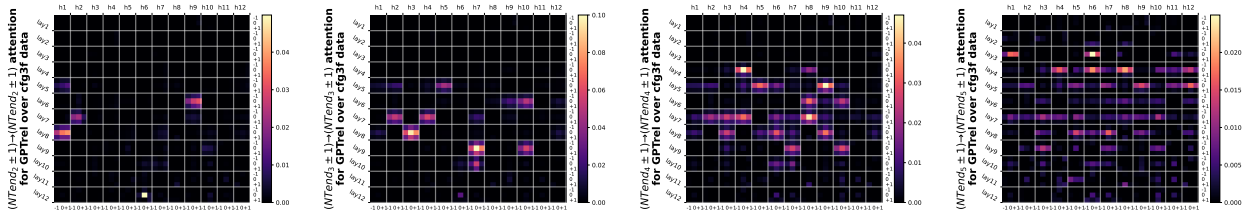
(e) cfg3i at level $\ell = 2$ (f) cfg3i at level $\ell = 3$ (g) cfg3i at level $\ell = 4$ (h) cfg3i at level $\ell = 5$



(i) cfg3h at level $\ell = 2$ (j) cfg3h at level $\ell = 3$ (k) cfg3h at level $\ell = 4$ (l) cfg3h at level $\ell = 5$



(m) cfg3g at level $\ell = 2$ (n) cfg3g at level $\ell = 3$ (o) cfg3g at level $\ell = 4$ (p) cfg3g at level $\ell = 5$



(q) cfg3f at level $\ell = 2$ (r) cfg3f at level $\ell = 3$ (s) cfg3f at level $\ell = 4$ (t) cfg3f at level $\ell = 5$

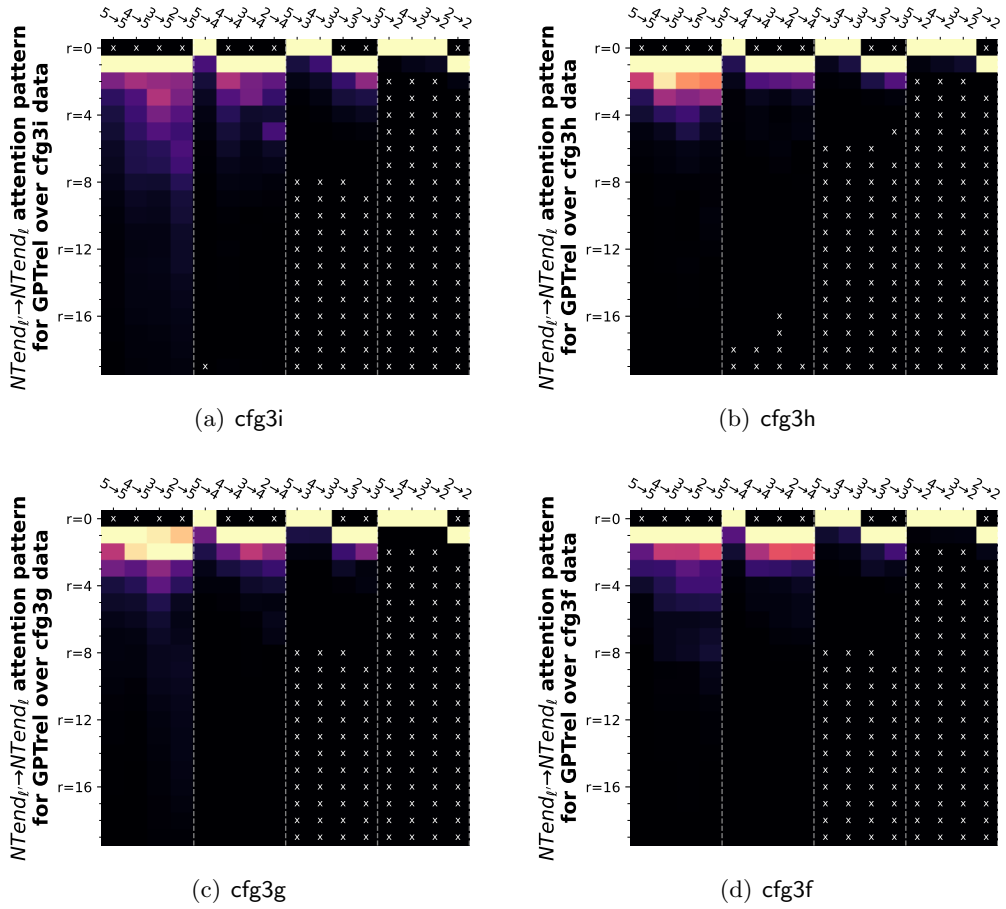
Figure 21: Attention pattern $B_{l,h,j\to i}(x)$ averaged over data x and pairs i, j such that $i + \delta_1$ and $j + \delta_2$ are at the NT-end boundaries in level ℓ of the CFG. In each block, the three rows correspond to $\delta_1 = -1, 0, +1$ and the three columns correspond to $\delta_2 = -1, 0, +1$.

Observation. Different transformer layer/head may be in charge of attending NT-ends at different levels ℓ. Also, it is noticeable that the attention value drops rapidly from $\delta_1 = \pm 1$ to $\delta_1 = 0$, but <u>*less so*</u> from $\delta_2 = \pm 1$ to $\delta_2 = 0$. This should not be surprising, as it may still be ambiguous to decide if position j is at NT-end *until* one reads few more tokens (see discussions under Figure 16).

D.4 From NT-ends to Adjacent NT-ends

In Figure 8(c) we have showcased that $B_{l,h,j\to i}(x)$ has a strong bias towards *token pairs i, j that are "adjacent" NT-ends*. We have defined what "adjacency" means in Section 5.2 and introduced a notion $B^{\text{end}\to\text{end}}_{l,h,\ell'\to\ell,r}$, to capture $B_{l,h,j\to i}(x)$ averaged over samples x and all token pairs i, j such that, they are at deepest NT-ends on levels ℓ, ℓ' respectively (in symbols, $\mathfrak{b}^\sharp(i) = \ell \wedge \mathfrak{b}^\sharp(j) = \ell'$), and of distance r based on the ancestor indices at level ℓ (in symbols, $\mathfrak{p}_\ell(j) - \mathfrak{p}_\ell(i) = r$).

Previously, we have only presented by Figure 8(c) for a single dataset, and averaged over all the transformer layers. In the full experiment Figure 22 we show that for more datasets, and Figure 23 we show that for individual layers.

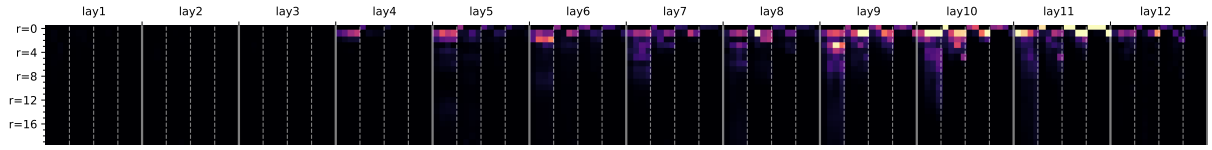


(a) cfg3i (b) cfg3h

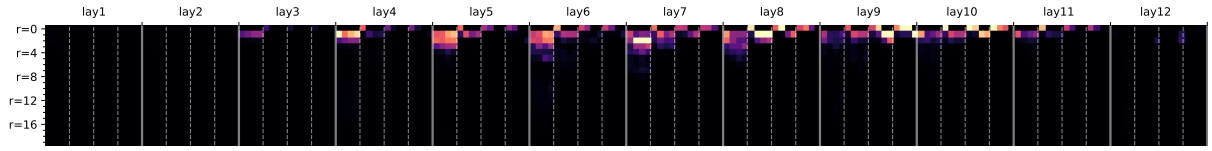
(c) cfg3g (d) cfg3f

Figure 22: Attention pattern $B^{\text{end}\to\text{end}}_{l,h,\ell'\to\ell,r}(x)$ averaged over layers l, heads h and data x. The columns represent $\ell' \to \ell$ and the rows represent r. "×" means empty entries.

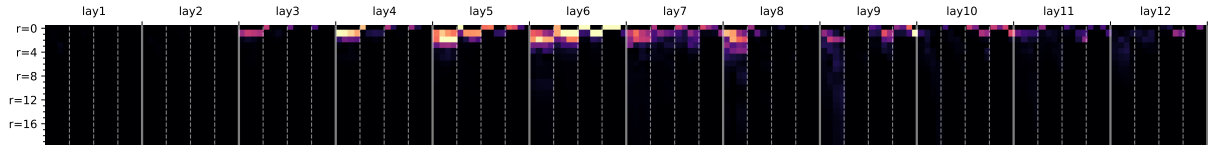
Remark. We present this boundary bias by looking at how close NT boundaries at level ℓ' attend to any other NT boundary at level ℓ. For some distances r, this "distance" that we have defined may be non-existing. For instance, when $\ell \geq \ell'$ one must have $r > 0$. Nevertheless, we see that the attention value, *even after removing the position bias*, still have a large correlation with respect to the smallest possible distance r, between every pairs of NT levels ℓ, ℓ'. This is a strong evidence that CFGs are implementing some variant of dynamic programming.



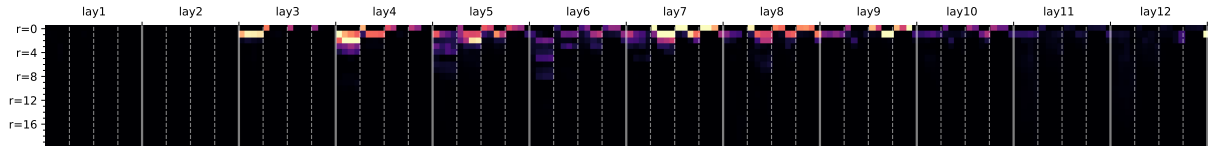
(a) cfg3i



(b) cfg3h



(c) cfg3g



(d) cfg3f

Figure 23: Attention pattern $B^{\text{end}\to\text{end}}_{l,h,\ell'\to\ell,r}(x)$ for each individual transformer layer $l \in [12]$, averaged over heads h and data x. The rows and columns are in the same format as Figure 22.

Observation. Different transformer layers are responsible for learning "NT-end to most adjacent NT-end" at different CFG levels.

E More Experiments on Implict CFGs

We study implicit CFGs where each terminal symbol $t \in \mathbf{T}$ is is associated a bag of observable tokens $\mathbf{OT}_t$. For this task, we study eight different variants of implicit CFGs, all converted from the exact same cfg3i dataset (see Section A.1). Recall cfg3i has three terminal symbols $|\mathbf{T}| = 3$:

- we consider a vocabulary size $|\mathbf{OT}| = 90$ or $|\mathbf{OT}| = 300$;
- we let $\{\mathbf{OT}_t\}_{t\in\mathbf{T}}$ be either disjoint or overlapping; and
- we let the distribution over $\mathbf{OT}_t$ be either uniform or non-uniform.

We present the generation accuracies of learning such implicit CFGs with respect to different model architectures in Figure 24, where in each cell we evaluate accuracy using 2000 generation samples. We also present the correlation matrix of the word embedding layer in Figure 9 for the $\texttt{GPT}_{\texttt{rel}}$ model (the correlation will be similar if we use other models).

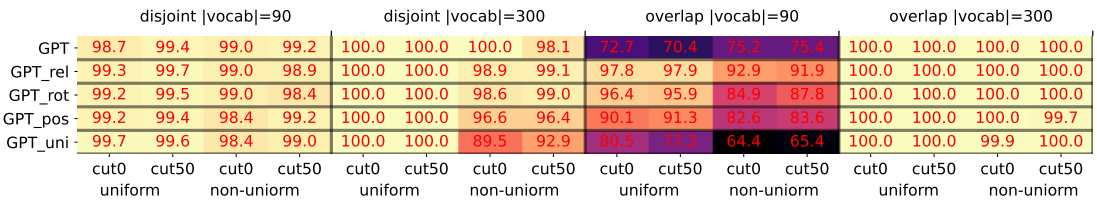
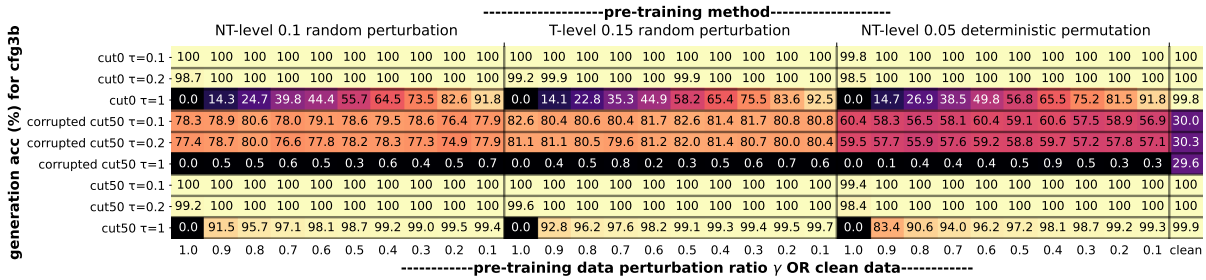


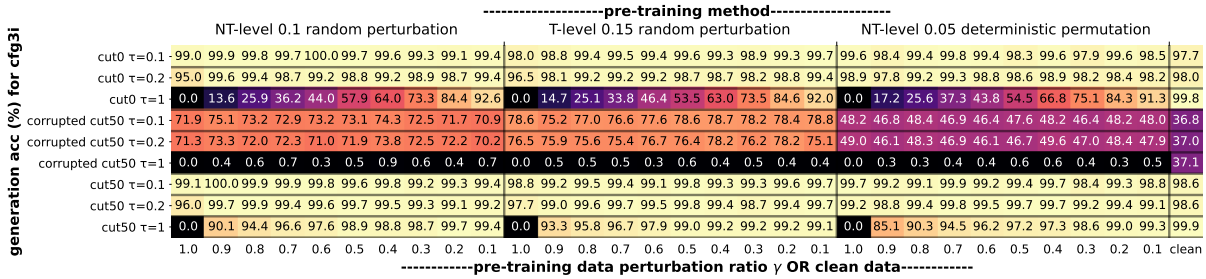
Figure 24: Generation accuracies on eight implicit CFG variants from pre-trained language models.

F More Experiments on Robustness

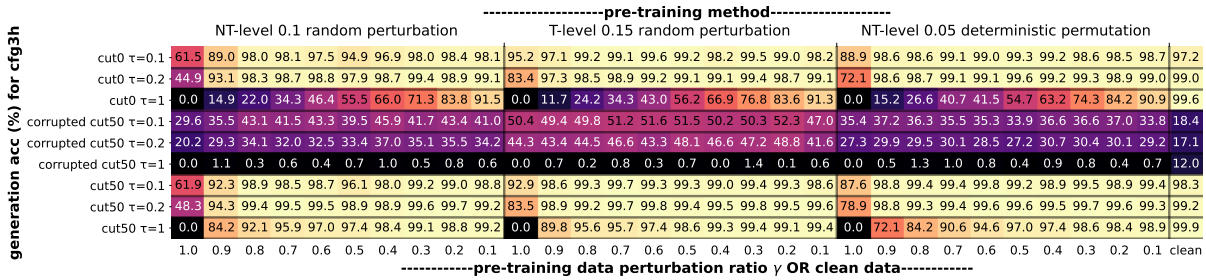
Recall that in Figure 10, we have compared clean training vs training over three types of perturbed data, for their generation accuracies given both clean prefixes and corrupted prefixes. We now include more experiments with respect to more datasets in Figure 25. For each entry of the figure, we have generated 2000 samples to evaluate the generation accuracy.



(a) cfg3b dataset



(b) cfg3i dataset



(c) cfg3h dataset

Figure 25: Generation accuracies for models pre-trained cleanly VS pre-trained over perturbed data, on clean or corrupted prefixes with cuts $c = 0$ or $c = 50$, using generation temperatures $\tau = 0.1, 0.2, 1.0$.

Observation 1. In Rows 4/5, by comparing against the last column, we see it is *beneficial* to include low-quality data (e.g. grammar mistakes) during pre-training. The amount of low-quality data could be little ($\gamma = 0.1$ fraction) or large (*every training sentence may have grammar mistake*).

Observation 2. In Rows 3/6/9 of Figure 10 we see pre-training teaches the model a *mode switch*. When given a correct prefix it is in the *correct mode* and completes with correct strings (Row 9); given corrupted prefixes it *always* completes sentences with grammar mistakes (Row 6); given no prefix it generates corrupted strings with probability γ (Row 3).

Observation 3. Comparing Rows 4/5 to Row 6 in Figure 10 we see that high robust accuracy is achieved only when generating using low temperatures τ. Using low temperature encourages the model to, for each next token, pick a more probable solution. This allows it to achieve good robust accuracy *even when* the model is trained totally on corrupted data ($\gamma = 1.0$).

References

[1] Zeyuan Allen-Zhu and Yuanzhi Li. Backward feature correction: How deep learning performs deep learning. In *COLT*, 2023. Full version available at `http://arxiv.org/abs/2001.04413`.

[2] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *ICML*, 2019. Full version available at `http://arxiv.org/abs/1811.03962`.

[3] David Arps, Younes Samih, Laura Kallmeyer, and Hassan Sajjad. Probing for constituency structure in neural language models. *arXiv preprint arXiv:2204.06201*, 2022.

[4] Nisha Bhalse and Vivek Gupta. Learning cfg using improved tbl algorithm. *Computer Science & Engineering*, 2(1):25, 2012.

[5] Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the ability and limitations of transformers to recognize formal languages. *arXiv preprint arXiv:2009.11264*, 2020.

[6] Sid Black, Leo Gao, Phil Wang, Connor Leahy, and Stella Biderman. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, March 2021. URL `https://doi.org/10.5281/zenodo.5297715`.

[7] Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. GPT-NeoX-20B: An open-source autoregressive language model. In *Proceedings of the ACL Workshop on Challenges & Perspectives in Creating Large Language Models*, 2022. URL `https://arxiv.org/abs/2204.06745`.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[9] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[10] Alexander Clark. Computational learning of syntax. *Annual Review of Linguistics*, 3:107–123, 2017.

[11] Jose M Font, Tobias Mahlmann, Daniel Manrique, and Julian Togelius. A card game description language. In *Applications of Evolutionary Computation: 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3-5, 2013. Proceedings 16*, pages 254–263. Springer, 2013.

[12] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*, 2020.

[13] John Hewitt and Christopher D. Manning. A structural probe for finding syntax in word representations. 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 4129–4138, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1419. URL `https://aclanthology.org/N19-1419`.

[14] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[15] Samy Jelassi, Michael Sander, and Yuanzhi Li. Vision transformers provably learn spatial structure. *Advances in Neural Information Processing Systems*, 35:37822–37836, 2022.

[16] Aravind K Joshi, K Vijay Shanker, and David Weir. The convergence of mildly context-sensitive grammar formalisms. *Technical Reports (CIS)*, page 539, 1990.

[17] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186, 2019.

[18] Lillian Lee. Learning of context-free languages: A survey of the literature. *Techn. Rep. TR-12-96, Harvard University*, 1996.

[19] Yuchen Li, Yuanzhi Li, and Andrej Risteski. How do transformers learn topic structure: Towards a mechanistic understanding. *arXiv preprint arXiv:2303.04245*, 2023.

[20] Bingbin Liu, Jordan T Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. *arXiv preprint arXiv:2210.10749*, 2022.

[21] Christopher D Manning, Kevin Clark, John Hewitt, Urvashi Khandelwal, and Omer Levy. Emergent linguistic structure in artificial neural networks trained by self-supervision. *Proceedings of the National Academy of Sciences*, 117(48):30046–30054, 2020.

[22] Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Probabilistic cfg with latent annotations. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 75–82, 2005.

[23] Rowan Hall Maudslay and Ryan Cotterell. Do syntactic probes probe syntax? experiments with jabberwocky probing. *arXiv preprint arXiv:2106.02559*, 2021.

[24] Darnell Moore and Irfan Essa. Recognizing multitasked activities from video using stochastic context-free grammar. In *AAAI/IAAI*, pages 770–776, 2002.

[25] Milad Moradi and Matthias Samwald. Evaluating the robustness of neural language models to input perturbations. *arXiv preprint arXiv:2108.12237*, 2021.

[26] OpenAI. Gpt-4 technical report, 2023.

[27] Matt Post and Shane Bergsma. Explicit and implicit syntactic features for text classification. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 866–872, 2013.

[28] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[29] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[30] Itiroo Sakai. Syntax in universal translation. In *Proceedings of the International Conference on Machine Translation and Applied Language Analysis*, 1961.

[31] Yasubumi Sakakibara. Learning context-free grammars using tabular representations. *Pattern Recognition*, 38(9):1372–1383, 2005.

[32] Yikang Shen, Zhouhan Lin, Chin-Wei Huang, and Aaron Courville. Neural language modeling by jointly learning syntax and lexicon. *arXiv preprint arXiv:1711.02013*, 2017.

[33] Hui Shi, Sicun Gao, Yuandong Tian, Xinyun Chen, and Jishen Zhao. Learning bounded context-free-grammar via lstm and the transformer: Difference and the explanations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8267–8276, 2022.

[34] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning, 2012.

[35] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2021.

[36] Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R Bowman, Dipanjan Das, et al. What do you learn from context? probing for sentence structure in contextualized word representations. *arXiv preprint arXiv:1905.06316*, 2019.

[37] Lifu Tu, Garima Lalwani, Spandana Gella, and He He. An empirical study on robustness to spurious correlations using pre-trained language models. *Transactions of the Association for Computational Linguistics*, 8:621–633, 2020.

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

[39] David Vilares, Michalina Strzyz, Anders Søgaard, and Carlos Gómez-Rodríguez. Parsing as pretraining. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9114–9121, 2020.

[40] David Jeremy Weir. *Characterizing mildly context-sensitive grammar formalisms*. University of Pennsylvania, 1988.

[41] Zhiyong Wu, Yun Chen, Ben Kao, and Qun Liu. Perturbed masking: Parameter-free probing for analyzing and interpreting bert. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4166–4176, 2020.

[42] Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. Self-attention networks can process bounded hierarchical languages. *arXiv preprint arXiv:2105.11115*, 2021.

[43] Haoyu Zhao, Abhishek Panigrahi, Rong Ge, and Sanjeev Arora. Do transformers parse while predicting the masked word? *arXiv preprint arXiv:2303.08117*, 2023.