HOW LANGUAGE MODELS LEARN CONTEXT-FREE GRAMMARS

ABSTRACT

We design experiments to study *how* generative language models, such as GPT, learn context-free grammars (CFGs) — complex language systems with tree-like structures that encapsulate aspects of human logic, natural languages, and programs. CFGs, comparable in difficulty to pushdown automata, can be ambiguous, usually requiring dynamic programming for rule verification. We create synthetic data to show that pre-trained transformers can learn to generate sentences with near-perfect accuracy and impressive diversity, even for quite challenging CFGs. Crucially, we uncover the *mechanisms* behind transformers learning such CFGs. We find that the hidden states implicitly encode the CFG structure (such as putting tree node info exactly on the subtree boundary), and that the transformer can form "boundary to boundary" attentions that mimic dynamic programming. We also discuss CFG extensions and transformer robustness against grammar errors.

1 INTRODUCTION

Language models (OpenAI, 2023) are neural networks designed to learn the probability distribution of natural language and generate text. Models like GPT (Radford et al., 2018) can accurately follow language structures (Shen et al., 2017; Tenney et al., 2019), even in smaller models (Black et al., 2021). However, the mechanisms and representations these models use to capture language rules and patterns remain unclear. Despite recent theoretical advances in understanding language models (Bhattamishra et al., 2020; Jelassi et al., 2022; Li et al., 2023; Liu et al., 2022; Yao et al., 2021), most are limited to simple settings and fail to account for the complex structure of languages.

In this paper, we explore the **mechanisms** behind generative language models learning probabilistic context-free grammars (CFGs) (Lee, 1996). CFGs, capable of generating a diverse set of *highly structured* expressions, consist of terminal (T) and nonterminal (NT) symbols, a root symbol, and production rules. 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 instance, the CFG below generates the language of balanced parentheses:

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

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

Many structures in languages can be viewed as CFGs, including *grammars, structures of the codes, mathematical expressions, music patterns, article formats* (for poems, instructions, legal documents), etc. We use transformer (Vaswani et al., 2017) as the generative language model and study how it learns the CFGs. Transformers can encode some CFGs, especially those that correspond to the grammar of natural languages (Arps et al., 2022; Hewitt & Manning, 2019; Manning et al., 2020; Maudslay & Cotterell, 2021; Shi et al., 2022; Vilares et al., 2020; Wu et al., 2020; Zhao et al., 2023). However, the *mechanism* behind how such CFGs can be efficiently learned by transformers remains unclear. Previous works (Deletang et al., 2023) studied transformer's learnability on a few languages in the Chomsky hierarchy (which includes CFGs) but the inner mechanisms regarding how transformer can or cannot solve those tasks remain 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.

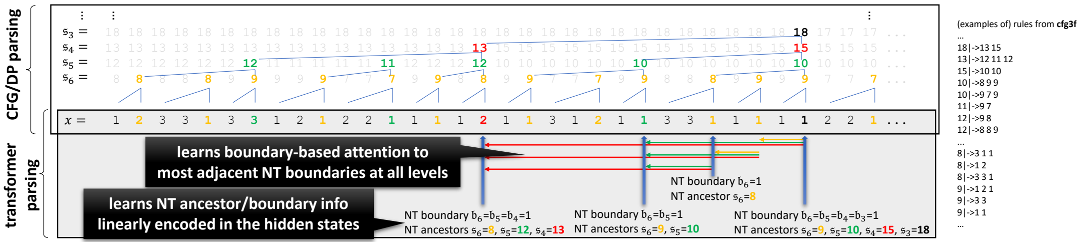

CFG/DP parsing
transformer parsing
$\mathfrak{s}_3$ =
$\mathfrak{s}_4$ =
$\mathfrak{s}_5$ =
$\mathfrak{s}_6$ =
x = 1 2 3 3 1 3 3 1 2 1 2 2 1 1 1 1 2 1 1 3 1 2 1 1 3 3 1 1 1 1 1 2 2 1 ...
learns boundary-based attention to most adjacent NT boundaries at all levels
learns NT ancestor/boundary info linearly encoded in the hidden states
NT boundary $\mathfrak{b}_6$=$\mathfrak{b}_5$=$\mathfrak{b}_4$=1
NT ancestors $\mathfrak{s}_6$=8, $\mathfrak{s}_5$=12, $\mathfrak{s}_4$=13
NT boundary $\mathfrak{b}_6$=$\mathfrak{b}_5$=1
NT ancestors $\mathfrak{s}_6$=9, $\mathfrak{s}_5$=10
NT boundary $\mathfrak{b}_6$=1
NT ancestor $\mathfrak{s}_6$=8
NT boundary $\mathfrak{b}_6$=$\mathfrak{b}_5$=$\mathfrak{b}_4$=$\mathfrak{b}_3$=1
NT ancestors $\mathfrak{s}_6$=9, $\mathfrak{s}_5$=10, $\mathfrak{s}_4$=15, $\mathfrak{s}_3$=18
(examples of) rules from cfg3f
...
18|->13 15
13|->12 11 12
15|->10 10
10|->8 9 9
10|->9 7 9
11|->9 7
12|->9 8
12|->8 8 9
...
8|->3 1 1
8|->1 2
8|->3 3 1
9|->1 2 1
9|->3 3
9|->1 1
...

Figure 1: An example string x from $\mathcal{G} = \mathsf{cfg3f}$. Though formally defined in Section 2, bold symbols in color represent *NT boundaries* which marks the ending positions of the parsed CFG subtrees at various levels ℓ: we denote by $\mathfrak{b}_\ell(i) = 1$ if position i is at the NT boundary for level ℓ. The *NT ancestor* $\mathfrak{s}_\ell(i)$ represents the tree node's name at level ℓ for a symbol at position i.

Remark. In this paper, we analyze the transformer's ability to learn highly ambiguous CFGs. Even if the CFG rules are given, typically one uses dynamic programming (DP) to decide if $x \in L(\mathcal{G})$.

In this study, we pre-train GPT-2 (Radford et al., 2019) on a language modeling task using 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 and Figure 9 in the appendix for more. We test the model's accuracy and *diversity* by feeding it *prefixes* from the CFG and observing if it can generate accurate completions.

- We show the model can achieve near-perfect CFG generation accuracies.
- We check the model's output distribution / diversity show it is close to that of the true CFG.

Our paper's **key contribution** is an analysis of *how transformers recover the structures of the underlying CFG*, examining attention patterns and hidden states. Specifically, we:

- Develop a probing method to verify that the model's hidden states linearly encode NT information almost perfectly, a significant finding as pre-training does not expose the CFG structure.
- Introduce methods to visualize and quantify attention patterns, demonstrating that GPT learns position-based and boundary-based attentions, contributing to understanding the CFG's regularity, periodicity, and hierarchical structure.
- Suggest that GPT models learn CFGs by *implementing a dynamic programming-like algorithm*. We find that boundary-based attention allows a token to attend to its closest NT symbols in the CFG tree, even when separated by hundreds of tokens. 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 1 for an illustration.

We also explore *implicit CFGs* (Post & Bergsma, 2013), where each T symbol is a bag of tokens, and data is generated by randomly sampling tokens. This allows capturing additional structure, like word categories. We demonstrate that the model learns implicit CFGs by encoding the T symbol information in its token embedding layer. We also investigate *model robustness* using CFGs, testing the model's ability to correct errors and generate valid CFGs from a corrupted prefix.

2 CONTEXT-FREE GRAMMARS

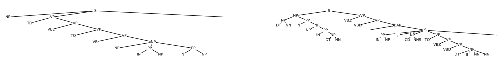
A probabilistic context-free grammar (CFG) is a formal system defining a string distribution using production rules. It comprises four components: terminal symbols ($\mathbf{T}$), nonterminal symbols ($\mathbf{NT}$), a root symbol ($root \in \mathbf{NT}$), and production rules ($\mathcal{R}$). We represent a CFG as $\mathcal{G} = (\mathbf{T}, \mathbf{NT}, \mathcal{R})$, with $L(\mathcal{G})$ denoting the string distribution generated by $\mathcal{G}$.

We mostly focus on L-level CFGs where each level $\ell \in [L]$ corresponds to a set of symbols $\mathbf{NT}_\ell$ with $\mathbf{NT}_\ell \subseteq \mathbf{NT}$ for $\ell < L$, $\mathbf{NT}_L = \mathbf{T}$, and $\mathbf{NT}_1 = \{root\}$. Symbols at different levels are disjoint: $\mathbf{NT}_i \cap \mathbf{NT}_j = \varnothing$ for $i \neq j$. 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 rules in the form:

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

Given a non-terminal symbol $a \in \mathbf{NT}$ and any rule $r = (a \mapsto \star)$, we say $a \in r$. For each $a \in \mathbf{NT}$, its associated set of rules is $\mathcal{R}(a) := \big\{ r \mid r \in \mathcal{R}_\ell \wedge a \in r \big\}$, its *degree* is $|\mathcal{R}(a)|$, and the CFG's *size* is $(|\mathbf{NT}_1|, |\mathbf{NT}_2|, \dots, |\mathbf{NT}_L|)$.

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



(a) real-life English CFG derived from Penn Treebank, short and simple

(b) a family of max-depth 11 CFGs where rules have length 1 or 2 that GPT can learn, see cfg0 in Appendix H

Figure 2: CFG visual comparisons: *left* is a medium-length sample, and *right* is a 80%-percentile-length sample

1. Start with the *root* symbol $\mathbf{NT}_1$.
2. For each layer $\ell < L$, keep a sequence of symbols $s_\ell = \big(s_{\ell,1}, \cdots, s_{\ell,m_\ell}\big)$.
3. For the next layer, randomly sample a rule $r \in \mathcal{R}(s_{\ell,i})$ for each $s_{\ell,i}$ with uniform probability.[1] Replace $s_{\ell,i}$ with b, c, d if $r = (s_{\ell,i} \mapsto b, c, d)$, or with b, c if $r = (s_{\ell,i} \mapsto b, c)$. Let the resulting sequence be $s_\ell = \big(s_{\ell+1,1}, \cdots, s_{\ell+1,m_{\ell+1}}\big)$.
4. During generation, when a rule $s_{\ell,i} \mapsto s_{\ell+1,j}, s_{\ell+1,j+1}$ is applied, define the parent $\mathsf{par}_{\ell+1}(j) = \mathsf{par}_{\ell+1}(j+1) := i$ (and similarly if the rule of $s_{\ell,i}$ is of length 3).
5. Define **NT ancestor indices** $\mathfrak{p} = (\mathfrak{p}_1(i), \ldots, \mathfrak{p}_L(i))_{i\in[m_L]}$ and **NT ancestor symbols** $\mathfrak{s} = (\mathfrak{s}_1(i), \ldots, \mathfrak{s}_L(i))_{i\in[m_L]}$ as shown in Figure 1:

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

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

Definition 2.1. *A symbol x_i in a sample $(x, \mathfrak{p}, \mathfrak{s}) \sim L(\mathcal{G})$ is the **NT boundary / NT end** at level $\ell \in [L-1]$ if $\mathfrak{p}_\ell(i) \neq \mathfrak{p}_\ell(i+1)$ or $i = \mathbf{len}(x)$. We denote $\mathfrak{b}_\ell(i) := \mathbb{1}_{x_i \text{ is the NT boundary at level } \ell}$ as the **NT-end boundary** indicator function. The **deepest NT-end** of i is*

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

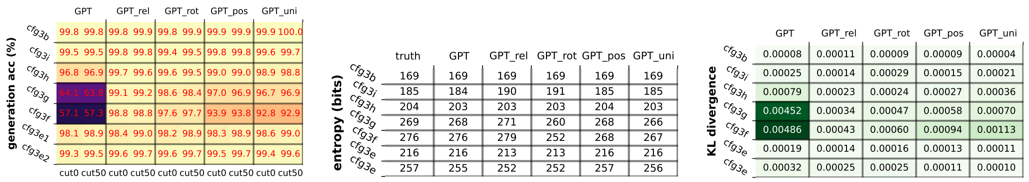
The cfg3 synthetic CFG family. We focus on seven synthetic CFGs of depth $L = 7$ detailed in Section B.1. The hard datasets cfg3b, cfg3i, cfg3h, cfg3g, cfg3f have sizes $(1, 3, 3, 3, 3, 3, 3)$ and increasing difficulties cfg3b < cfg3i < cfg3h < cfg3g < cfg3f. The easy datasets cfg3e1 and cfg3e2 have sizes $(1, 3, 9, 27, 81, 27, 9)$ and $(1, 3, 9, 27, 27, 9, 4)$ respectively. The sequences generated by these CFGs are up to $3^6 = 729$ in length. Typically, the learning difficulty of CFGs *inversely scales* with the number of NT/T symbols or CFG rules, assuming other factors remain constant (see Figure 3 and more in Appendix H). We thus primarily focus on cfg3b, cfg3i, cfg3h, cfg3g, cfg3f.

Why Such CFGs. In this paper, we use CFG as a proxy to study some rich, recursive structure in languages, which can cover some logics, grammars, formats, expressions, patterns, etc. Those structures are diverse yet strict (for example, Section 3.1 should be only followed by Section 3.1.1, Section 4 or Section 3.2, not others). We create a synthetic CFG to approximate such richness and structure. The CFGs we consider are non-trivial, with likely over $2^{270} > 10^{80}$ strings in cfg3f among a total of over $3^{300} > 10^{140}$ possible strings of length 300 or more (see the entropy estimation in Figure 3). The probability of a random string belonging to this language is nearly zero, and a random completion of a valid prefix is unlikely to satisfy the CFG.

Moreover, to probe the *inner workings* of the transformer, we choose a CFG family with a "canonical representation" and show a high correlation between this representation and the hidden states in the learned transformer. Such a *controlled experiment* allows us to better understand the learning process. We also construct additional CFG families to study "not-so-canonical" CFG trees, with results deferred to Appendix H. We do not claim our result captures all CFGs, however, we view our work as a promising starting point: our CFG is already quite challenging for a transformer to learn — for example, in Appendix H, we show that a CFG derived from English Penn TreeBank can be learned well using small models (like GPTs with $\sim$ 100k parameters), whereas our cfg3 family requires GPT2 with 100M parameters — yet we can still identify how transformer learns it.

---

[1] For simplicity, we consider the uniform case, eliminating rules with extremely low probability. Such rules complicate the learning of the CFG and the investigation of a transformer's inner workings. Our results can easily extend to non-uniform cases, provided the distributions are not heavily unbalanced.



generation acc (%)

	GPT		GPT_rel		GPT_rot		GPT_pos		GPT_uni	
	cut0	cut50	cut0	cut50	cut0	cut50	cut0	cut50	cut0	cut50
cfg3b	99.8	99.8	99.8	99.9	99.8	99.9	99.9	99.9	99.9	100.0
cfg3i	99.5	99.5	99.8	99.8	99.4	99.5	99.8	99.8	99.6	99.7
cfg3h	96.8	96.9	99.7	99.6	99.6	99.5	99.0	99.0	98.9	98.8
cfg3g	64.1	63.8	99.1	99.2	98.6	98.4	97.0	96.9	96.7	96.9
cfg3f	57.1	57.3	98.8	98.8	97.6	97.7	93.9	93.8	92.8	92.9
cfg3e1	98.1	98.9	98.4	99.0	98.2	98.9	98.3	98.9	98.6	99.0
cfg3e2	99.3	99.5	99.6	99.7	99.6	99.7	99.5	99.7	99.4	99.6

entropy (bits)

	truth	GPT	GPT_rel	GPT_rot	GPT_pos	GPT_uni
cfg3b	169	169	169	169	169	169
cfg3i	185	184	190	191	185	185
cfg3h	204	203	203	203	204	203
cfg3g	269	268	271	260	268	266
cfg3f	276	276	279	252	268	267
cfg3e	216	216	213	213	216	216
cfg3e	257	255	252	252	257	256

KL divergence

	GPT	GPT_rel	GPT_rot	GPT_pos	GPT_uni
cfg3b	0.00008	0.00011	0.00009	0.00009	0.00004
cfg3i	0.00025	0.00014	0.00029	0.00015	0.00021
cfg3h	0.00079	0.00023	0.00024	0.00027	0.00036
cfg3g	0.00452	0.00034	0.00047	0.00058	0.00070
cfg3f	0.00486	0.00043	0.00060	0.00094	0.00113
cfg3e	0.00019	0.00014	0.00016	0.00013	0.00011
cfg3e	0.00032	0.00025	0.00025	0.00011	0.00010

Figure 3: Generation accuracy (left), entropy (middle), KL-divergence (right) across multiple CFG datasets. **Observation:** Less ambiguous CFGs (cfg3e1, cfg3e2, as they have fewer NT/T symbols) are easier to learn. Modern transformer variants using relative positional embedding ($\texttt{GPT}_{\textsf{rel}}$ or $\texttt{GPT}_{\textsf{pos}}$) are better for learning complex CFGs. We also present weaker variants $\texttt{GPT}_{\textsf{pos}}$ and $\texttt{GPT}_{\textsf{uni}}$ that base their attention matrices solely on token positions (serving specific purposes in Section 5.1).

3 TRANSFORMER LEARNS SUCH CFGS

In this section, we evaluate the generative capability of the transformer by testing its accuracy in completing sequences from prefixes of strings in $L(\mathcal{G})$. We also evaluate the diversity of the generated outputs and verify if the distribution of these strings aligns with the ground truth $L(\mathcal{G})$.

Models. We denote the vanilla GPT2 small architecture (12-layer, 12-head, 768-dimensions) as `GPT` (Radford et al., 2019). Given GPT2's weak performance due to its absolute positional embedding, we implemented two modern variants. We denote `GPT` with relative positional attention (He et al., 2020) as $\texttt{GPT}_{\textsf{rel}}$, and `GPT` with rotary positional embedding (Black et al., 2022; Su et al., 2021) as $\texttt{GPT}_{\textsf{rot}}$. For specific purposes in later sections, we introduce two weaker variants of `GPT`. $\texttt{GPT}_{\textsf{pos}}$ replaces the attention matrix with a matrix based solely on tokens' relative positions, while $\texttt{GPT}_{\textsf{uni}}$ uses a constant, uniform average of past tokens from various window lengths as the attention matrix. Detailed explanations of these variants are in Section B.2.

Completion accuracy. We generate a large corpus $\{x^{(i)}\}_{i\in[N]}$ from a synthetic CFG $\mathcal{G}$ as described in Section 2. A model F is pretrained on this corpus, treating each terminal symbol as a separate token, using an auto-regressive task (Section B.3 for details). For evaluation, F generates completions for prefixes $x_{:c} = (x_1, x_2, \cdots, x_c)$ from strings x freshly generated from $L(\mathcal{G})$. The *generation accuracy* is measured as $\mathbf{Pr}_{x\sim L(G)+\text{randomness of }F}[(x_{:c}, F(x_{:c})) \in L(\mathcal{G})]$. We use multinomial sampling without beam search for generation.[2]

Figure 3 (left) shows the generation accuracies for cuts $c = 0$ and $c = 50$. The $c = 0$ result tests the transformer's ability to generate a sentence in the CFG, while $c = 50$ tests its ability to complete a sentence.[3] The results show that the pretrained transformers can generate near-perfect strings that adhere to the CFG rules for the cfg3 data family.

Generation diversity. Could it be possible that the trained transformer only memorized a small subset of strings from the CFG? We evaluate its learning capability by measuring the diversity of its generated strings. High diversity suggests a better understanding of the CFG rules.

Diversity can be estimated through entropy. Given a distribution p over strings and a sampled subset $S = \{x^{(i)}\}_{i\in[M]}$ from p, for any string $x \in S$, denote by $\mathbf{len}(x)$ its length so $x = (x_1, \ldots, x_{\mathbf{len}(x)})$, and denote by $x_{\mathbf{len}(x)+1} = \mathsf{eos}$. The entropy in bits for p can be estimated by

$$-\tfrac{1}{|S|}\sum_{x\in S}\sum_{i\in[\mathbf{len}(x)+1]}\log_2 \mathbf{Pr}_p\left[x_i \mid x_1, \ldots, x_{i-1}\right]$$

We compare the entropy of the true CFG distribution and the transformer's output distribution using $M = 20000$ samples in Figure 3 (middle).

Diversity can also be estimated using the birthday paradox to lower bound the support size of a distribution (Arora & Zhang, 2017). Given a distribution p over strings and a sampled subset $S = \{x^{(i)}\}_{i\in[M]}$ from p, if every pair of samples in S are distinct, then with good probability the support of p is of size at least $\Omega(M^2)$. In Appendix C.1, we conducted an experiment with $M = 20000$. We performed a birthday paradox experiment from every symbol $a \in \mathbf{NT}_{\ell_1}$ to some other level

---

[2] The last softmax layer converts the model outputs into a probability distribution over (next) symbols. We follow this distribution to generate the next symbol, reflecting the unaltered distribution learned by the transformer. This is the source of the "randomness of F" and is often referred to as using "temperature $\tau = 1$."

[3] Our cfg3 family is large enough to ensure a negligible chance of a freshly sampled prefix of length 50 being seen during pretraining.

$\ell_2 > \ell_1$, comparing that with the ground truth. For instance, we confirmed for the cfg3f dataset, there are at least $\Omega(M^2)$ distinct sequences to level 5 generated from a symbol $a \in \mathbf{NT}_2$ — not to mention from the root in $\mathbf{NT}_1$ to the leaf at level 7. In particular, M^2 is already more than the number of parameters in the model. From both experiments, we conclude that the pre-trained model **does not rely on simply memorizing** a small set of patterns to learn the CFGs.

Distribution comparison. To fully learn a CFG, it is crucial to learn the distribution of generating probabilities. However, comparing distributions of exponential support size can be challenging. A naive approach is to compare 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 C.2.

Another approach is to compute the KL-divergence between the per-symbol conditional distributions. Let p^* be the distribution over strings in the true CFG and p be that from the transformer model. Let $S = \left\{x^{(i)}\right\}_{i\in[M]}$ be samples from the true CFG distribution. Then, the KL-divergence can be estimated as follows:[4]

$$\frac{1}{|S|}\sum_{x\in S}\frac{1}{\mathbf{len}(x)+1}\sum_{i\in[\mathbf{len}(x)+1]}\sum_{t\in\mathbf{T}\cup\{\mathsf{eos}\}}\mathbf{Pr}_{p^*}[t \mid x_1,\dots,x_{i-1}]\log\frac{\mathbf{Pr}_{p^*}[t|x_1,\dots,x_{i-1}]}{\mathbf{Pr}_{p}[t|x_1,\dots,x_{i-1}]}$$

In Figure 3 (right) we compare the KL-divergence between the true CFG distribution and the transformer's output distribution using $M = 20000$ samples.

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.

Recall classical way to solve CFGs. Given CFG $\mathcal{G}$, the classical way to verify if a sequence x satisfies $L(\mathcal{G})$ is to use dynamic programming (DP) (Sakai, 1961; Sipser, 2012). One possible implementation of DP involves using the function $\mathsf{DP}(i, j, a)$, which determines whether or not $x_i, x_{i+1}\dots, x_j$ can be generated from symbol a following the CFG rules. From this DP representation, a DP recurrent formula can be easily derived.[5]

In the context of this paper, any sequence $x \sim L(\mathcal{G})$ that satisfies the CFG must satisfy the following conditions (recall the NT-boundary $\mathfrak{b}_\ell$ and the NT-ancestor $\mathfrak{s}_\ell$ notions from Section 2):

$$\mathfrak{b}_\ell(i-1) = 1, \mathfrak{b}_\ell(j) = 1, \forall k \in [i, j), \mathfrak{b}_\ell(k) = 0 \text{ and } \mathfrak{s}_\ell(i) = a \implies \mathsf{DP}(i, j, a) = 1 \qquad (4.1)$$

Note that (4.1) is not an "if and only if" condition because there may be a subproblem $\mathsf{DP}(i, j, a) = 1$ that does not lie on the final CFG parsing tree but is still locally parsable by some valid CFG subtree. However, (4.1) provides a "backbone" of subproblems, where verifying their $\mathsf{DP}(i, j, a) = 1$ values *certifies* that the sentence x is a valid string from $L(\mathcal{G})$. It is worth mentioning that ***depending on the implementation of a DP program*** (e.g., different orders on pruning or binarization), ***not all*** (i, j, a) tuples need to be computed in $\mathsf{DP}(i, j, a)$. Only those in the "backbone" are necessary.

Connecting to transformer. In this section, we investigate whether pre-trained transformer F not only generates grammatically correct sequences, but also implicitly encodes the NT ancestor and boundary information. If it does, this suggests that the transformer contains sufficient information to support all the $\mathsf{DP}(i, j, a)$ values in the backbone. This is a significant finding, considering that transformer F is trained solely on the auto-regressive task without any exposure to NT information. If it does encode the NT information after pretraining, it means that the model can both generate and certify sentences in the CFG language.

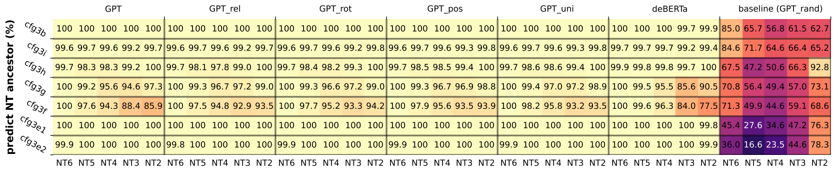
4.1 FINDING 1: TRANSFORMER'S HIDDEN STATES ENCODE NT ANCESTORS AND BOUNDARIES

Let l be the *last layer* of the transformer (other layers are considered in Appendix D.2). Given an input string x, the hidden state of the transformer at layer l and position i is denoted as

---

[4] A nearly identical formula was also used in DuSell & Chiang (2022).

[5] For example, one can compute $\mathsf{DP}(i, j, a) = 1$ if and only if there exists $i = i_1 < i_2 < \cdots < i_k = j + 1$ such that $\mathsf{DP}(i_r, i_{r+1}-1, b_r) = 1$ for all $r \in [k-1]$ and $a \to b_1, b_2, \dots, b_k$ is a rule of the CFG. Implementing this naively would result in a $O(\mathbf{len}^4)$ algorithm for CFGs with a maximum rule length of 3. However, it can be implemented more efficiently with $O(\mathbf{len}^3)$ time by introducing auxiliary nodes (e.g., via binarization).



predict NT ancestor (%)	GPT					GPT_rel					GPT_rot					GPT_pos					GPT_uni					deBERTa					baseline (GPT_rand)				
	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2
cfg3b	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	99.7	99.9	85.0	65.7	56.8	61.5	62.7
cfg3i	99.6	99.7	99.6	99.2	99.7	99.6	99.7	99.6	99.2	99.7	99.6	99.7	99.6	99.2	99.8	99.6	99.7	99.6	99.3	99.8	99.6	99.7	99.6	99.3	99.8	99.7	99.7	99.7	99.2	99.4	84.6	71.7	64.6	66.4	65.2
cfg3h	99.7	98.3	98.3	99.2	100	99.7	98.1	97.8	99.0	100	99.7	98.4	98.2	99.3	100	99.7	98.5	98.5	99.4	100	99.7	98.6	98.6	99.4	100	99.9	99.8	99.8	99.7	100	67.5	47.2	50.6	66.3	92.8
cfg3g	100	99.2	95.6	94.6	97.3	100	99.3	96.7	97.2	99.0	100	99.3	96.6	97.2	99.0	100	99.3	96.7	96.9	98.8	100	99.4	97.0	97.2	98.9	100	99.5	95.5	85.6	90.5	70.8	56.4	49.4	57.0	73.1
cfg3f	100	97.6	94.3	88.4	85.9	100	97.5	94.8	92.9	93.5	100	97.7	95.2	93.3	94.2	100	97.9	95.6	93.5	93.9	100	98.2	95.8	93.2	93.5	100	99.6	96.3	84.0	77.5	71.3	49.9	44.6	59.1	68.6
cfg3e1	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	99.8	45.4	27.6	34.6	47.2	76.3
cfg3e2	99.9	100	100	100	100	99.8	100	100	100	100	99.9	100	100	100	100	99.9	100	100	100	100	99.9	100	100	100	100	100	100	100	100	99.9	36.0	16.6	23.5	44.6	78.3

Figure 4: After pre-training, hidden states of generative models implicitly encode the NT ancestors information. The NT_ℓ column represents the accuracy of predicting $\mathfrak{s}_\ell$, the NT ancestors at level ℓ.

It also encodes NT boundaries, see Appendix D.1; and such information is discovered gradually and *hierarchically*, across layers and training epochs, see Appendix D.2 and D.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.

$E_i(x) \in \mathbb{R}^d$. We investigate whether a linear function can predict $\big(\mathfrak{b}_1(i), \dots, \mathfrak{b}_L(i)\big)_{i \in [\mathbf{len}(x)]}$ and $\big(\mathfrak{s}_1(i), \dots, \mathfrak{s}_L(i)\big)_{i \in [\mathbf{len}(x)]}$ using only $\big(E_i(x)\big)_{i \in [\mathbf{len}(x)]}$. If possible, it implies that the last-layer hidden states *encode the CFG's structural information up to a linear transformation*.

Our multi-head linear function. Due to the high dimensionality of this linear function (e.g., $\mathbf{len}(x) = 300$ and $d = 768$ yield 300×768 dimensions) and *variable string lengths*, we propose a multi-head linear function for efficient learning. We consider a set of linear functions $f_r \colon \mathbb{R}^d \to \mathbb{R}^{|\mathbf{NT}|}$, where $r \in [H]$ and H is the number of "heads". To predict any $\mathfrak{s}_\ell(i)$, we apply:

$$G_i(x) = \textstyle\sum_{r \in [H], k \in [\mathbf{len}(x)]} w_{r, i \to k} \cdot f_r(E_k(x)) \in \mathbb{R}^{|\mathbf{NT}|} \tag{4.2}$$

where $w_{r,i\to k} := \frac{\exp(\langle P_{i,r}, P_{k,r} \rangle)}{\sum_{k' \in [\mathbf{len}(x)]} \exp(\langle P_{i,r}, P_{k',r} \rangle)}$ for trainable parameters $P_{i,r} \in \mathbb{R}^{d'}$. G_i can be seen as a "multi-head attention" over linear functions. We train $G_i(x) \in \mathbb{R}^{|\mathbf{NT}|}$ using the cross-entropy loss to predict $\big(\mathfrak{s}_\ell(i)\big)_{\ell \in [L]}$. Despite having multiple heads,

$$G_i(x) \textbf{ is still a linear function over } \big(E_k(x)\big)_{k \in [\mathbf{len}(x)]}$$

as the linear weights $w_{r,i\to k}$ depend only on positions i and k, not on x. Similarly, we train $G'_i(x) \in \mathbb{R}^L$ using the logistic loss to predict the values $\big(\mathfrak{b}_\ell(i)\big)_{\ell \in [L]}$. Details are in Section B.4.

Results. Our experiments (Figure 4) suggest that pre-training allows the generative models to *almost perfectly encode* the NT ancestor and NT boundary information in the last transformer layer's hidden states, up to a *linear* transformation.

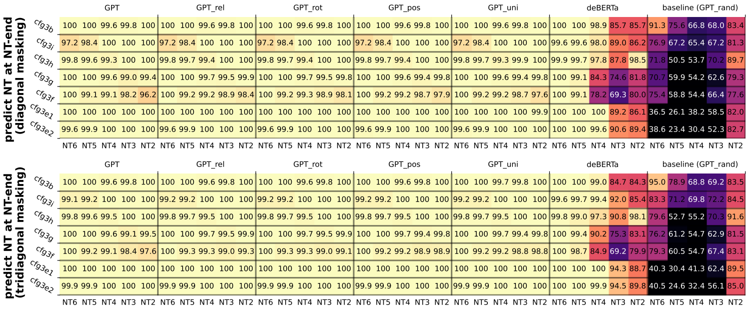
4.2 FINDING 2: TRANSFORMER'S HIDDEN STATES ENCODE NT ANCESTORS AT NT BOUNDARIES

We previously used the *entire* hidden state layer, $\big(E_i(x)\big)_{i \in [\mathbf{len}(x)]}$, to predict $\big(\mathfrak{s}_\ell(i)\big)_{\ell \in [L]}$ for *each* position i. This is essential for a generative/decoder model as it's impossible to extract i's NT ancestors by only examining $E_i(x)$ or the hidden states to its *left*, especially if a token x_i is near the string's start or a subtree's starting token in the CFG.

However, if we only consider a neighborhood of position i in the hidden states, say $E_{i\pm1}(x)$, what can we infer from it through linear probing? We can replace $w_{r,i\to k}$ in (4.2) with a replace $w_{r,i\to k}$ with zeros for $|i-k| > 1$ (tridiagonal masking), or with zeros for $i \neq k$ (diagonal masking).

Results. We observe two key points. First, diagonal or tridiagonal masking is sufficient for predicting NT boundaries, i.e., $\mathfrak{b}_\ell(i)$, with decent accuracy (deferred to Figure 15 in Appendix D.1). More importantly, at NT boundaries (i.e., $\mathfrak{b}_\ell(x) = 1$), such masking is adequate for accurately predicting the NT ancestors $\mathfrak{s}_\ell(x)$ (see Figure 5). Hence, we conclude that the information of position i's NT ancestors is *locally encoded around position i when i is on the NT boundary*.

Related work. Our probing approach is akin to the seminal work by Hewitt & Manning (2019), which uses linear probing to examine the correlation between BERT's hidden states and the parse tree distance metric (similar to NT-distance in our language). Subsequent studies (Arps et al., 2022; Manning et al., 2020; Maudslay & Cotterell, 2021; Shi et al., 2022; Vilares et al., 2020; Wu et al., 2020; Zhao et al., 2023) have explored various probing techniques to suggest that BERT-like transformers can approximate CFGs from natural languages.

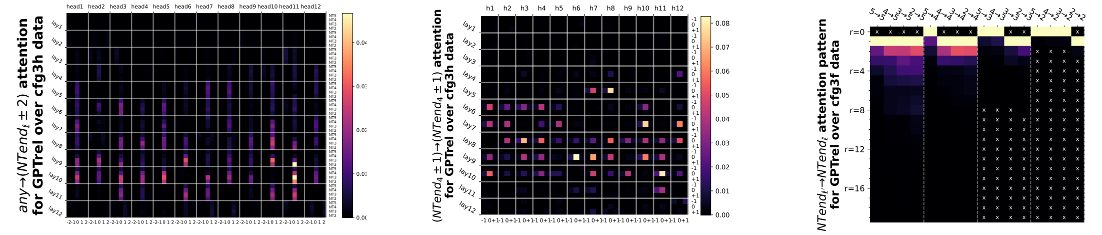


predict NT at NT-end (diagonal masking)	GPT					GPT_rel					GPT_rot					GPT_pos					GPT_uni					deBERTa					baseline (GPT_rand)				
	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2
cfg3b	100	100	99.6	99.8	100	100	100	99.6	99.8	100	100	100	99.6	99.8	100	100	100	99.6	99.8	100	100	100	99.6	99.8	100	100	100	98.9	85.7	85.7	91.3	75.6	66.8	68.0	83.4
cfg3i	97.2	98.4	100	100	100	97.2	98.4	100	100	100	97.2	98.4	100	100	100	97.2	98.4	100	100	100	97.2	98.4	100	100	100	99.6	99.6	98.0	89.0	86.2	76.9	67.2	65.4	67.2	81.3
cfg3h	99.8	99.6	99.3	100	100	99.8	99.7	99.4	100	100	99.8	99.7	99.4	100	100	99.8	99.7	99.4	100	100	99.8	99.7	99.3	99.9	100	99.9	99.7	97.8	87.8	98.5	71.8	50.5	53.7	70.2	89.7
cfg3g	100	100	99.6	99.0	99.4	100	100	99.7	99.5	99.9	100	100	99.7	99.5	99.8	100	100	99.6	99.4	99.8	100	100	99.6	99.4	99.8	100	99.1	84.3	74.6	81.8	70.7	59.9	54.2	62.6	79.3
cfg3f	100	99.1	99.1	98.2	96.2	100	99.2	99.2	98.9	98.4	100	99.2	99.3	98.9	98.1	100	99.2	99.2	98.7	97.9	100	99.2	99.2	98.7	97.6	100	99.1	78.2	69.3	80.0	75.4	58.8	54.4	66.4	77.6
cfg3e1	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	99.9	100	100	100	89.2	86.1	36.5	26.1	38.2	58.5	82.0
cfg3e2	99.6	99.9	100	100	100	99.6	99.9	100	100	100	99.6	99.9	100	100	100	99.6	99.9	100	100	100	99.6	99.9	100	100	100	100	100	99.6	90.6	89.4	38.6	23.4	30.4	52.3	82.7

predict NT at NT-end (tridiagonal masking)	GPT					GPT_rel					GPT_rot					GPT_pos					GPT_uni					deBERTa					baseline (GPT_rand)				
	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2
cfg3b	100	100	99.6	99.8	100	100	100	99.6	99.8	100	100	100	99.6	99.8	100	100	100	99.6	99.8	100	100	100	99.7	99.8	100	100	100	99.0	84.7	84.3	95.0	78.9	68.8	69.2	83.5
cfg3i	99.1	99.2	100	100	100	99.2	99.2	100	100	100	99.2	99.2	100	100	100	99.2	99.2	100	100	100	99.2	99.2	100	100	100	99.6	99.7	99.4	92.0	85.4	83.3	71.2	69.8	72.2	84.5
cfg3h	99.8	99.6	99.5	100	100	99.8	99.7	99.5	100	100	99.8	99.7	99.5	100	100	99.8	99.7	99.5	100	100	99.8	99.7	99.5	100	100	99.8	99.0	97.3	90.8	98.1	79.6	52.7	55.2	70.3	91.6
cfg3g	100	100	99.6	99.1	99.5	100	100	99.7	99.5	99.9	100	100	99.7	99.5	99.9	100	100	99.7	99.4	99.8	100	100	99.7	99.4	99.8	100	99.4	90.2	75.3	83.1	76.2	61.2	54.7	62.9	81.5
cfg3f	100	99.2	99.1	98.4	97.6	100	99.3	99.3	99.0	99.3	100	99.3	99.3	99.0	99.1	100	99.2	99.2	98.9	98.9	100	99.2	99.2	98.8	98.8	100	98.7	84.9	69.2	79.9	79.3	60.5	54.7	67.4	83.1
cfg3e1	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	94.3	88.7	40.3	30.4	41.3	62.4	89.5
cfg3e2	99.9	99.9	100	100	100	99.9	99.9	100	100	100	99.9	99.9	100	100	100	99.9	99.9	100	100	100	99.9	99.9	100	100	100	100	100	99.9	94.5	89.8	40.5	24.6	32.4	56.1	85.0

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 5: Generative pre-trained transformer encodes NT ancestors almost exactly *at* NT boundaries. The NT_ℓ column represents the linear-probing accuracy of predicting $\mathfrak{s}_\ell(i)$ at locations i with $\mathfrak{b}_\ell(i) = 1$.



(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 6: 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 E.2, E.3 and E.4.

Our approach differs in that we use synthetic data to demonstrate that linear probing can *almost perfectly* recover NT ancestors and boundaries, even for complex CFGs that generate strings exceeding hundreds of tokens. We focus on pre-training *generative* language models. For a non-generative, BERT-like model pre-trained via language-modeling (MLM), such as the contemporary variant DeBERTa (He et al., 2020), learning *deep* NT information (i.e., close to the CFG root) is less effective, as shown in Figure 4. This is expected, as the MLM task may only require the transformer to learn NT rules for, say, 20 neighboring tokens. Crucially, BERT-like models do *not* store deep NT information at the NT boundaries (see Figure 5).

Our results, along with Section 5, provide evidence that generative language models like GPT-2 employ a dynamic-programming-like approach to generate CFGs, while encoder-based models, typically trained via MLM, struggle to learn more complex/deeper CFGs.

5 HOW DO TRANSFORMERS LEARN NTS?

We now delve into the attention patterns. We demonstrate that these patterns mirror the CFG's syntactic structure and rules, with the transformer employing different attention heads to learn NTs at different CFG levels.

5.1 POSITION-BASED ATTENTION

We first note that the transformer's attention weights are primarily influenced by the tokens' relative distance. This holds true even when *trained on the CFG data* with *absolute positional embedding*. This implies that the transformer learns the CFG's regularity and periodicity through positional information, which it then uses for generation. (We defer the figures to Appendix E.1 as this finding may not surprise some readers.)

Motivated by this, we explore whether position-based attention *alone* can learn CFGs. In Figure 3, we find that $\texttt{GPT}_{\textsf{pos}}$ (or even $\texttt{GPT}_{\textsf{uni}}$) performs well, surpassing the vanilla `GPT`, but not reaching the full potential of $\texttt{GPT}_{\textsf{rel}}$. This supports the superior practical performance of relative-position based transformer variants (such as $\texttt{GPT}_{\textsf{rel}}$, $\texttt{GPT}_{\textsf{rot}}$, DeBERTa) over their base models (`GPT` or BERT). **On this other hand, this also indicates that position attention along is not enough for transformers to learn CFGs.**

5.2 BOUNDARY-BASED ATTENTION

Next, we *remove* the position-bias from the attention matrix to examine the remaining part. We find that the transformer also learns a strong boundary-based attention pattern, where tokens on the NT-end boundaries typically **attend to the "most adjacent" NT-end boundaries**, similar to standard dynamic programming for parsing CFGs (see Figure 1). This attention pattern enables the transformer to effectively learn the hierarchical and recursive structure of the CFG, and generate output tokens based on the NT symbols and rules.

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 of the transformer, on input sequence x. Given a sample pool $\{x^{(n)}\}_{n\in[N]} \in L(\mathcal{G})$, we compute for each layer l, head h,[6]

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

which represents the average attention between any token pairs of distance p over the sample pool. To remove position-bias, we focus on $B_{l,h,j\to i}(x) := A_{l,h,j\to i}(x) - \overline{A}_{l,h,j-i}$ in this subsection. Our observation can be broken down into three steps.

- Firstly, $B_{l,h,j\to i}(x)$ exhibits a strong bias towards *tokens i at NT ends*. As shown in Figure 6(a), we present the average value of $B_{l,h,j\to i}(x)$ over data x and pairs i, j where $i + \delta$ is the deepest NT-end at level ℓ (symbolically, $\mathfrak{b}^\sharp(i+\delta) = \ell$). The attention weights are highest when $\delta = 0$ and decrease rapidly for surrounding tokens.
- Secondly, $B_{l,h,j\to i}(x)$ also favors pairs i, j *both at NT ends* at some level ℓ. In Figure 6(b), we show the average value of $B_{l,h,j\to i}(x)$ over data x and pairs i, j where $\mathfrak{b}_\ell(i+\delta_1) = \mathfrak{b}_\ell(j+\delta_2) = 1$ for $\delta_1, \delta_2 \in \{-1, 0, 1\}$.
- Thirdly, $B_{l,h,j\to i}(x)$ favors *"adjacent" NT-end token pairs i, j*. We define "adjacency" as follows: We introduce $B^{\text{end}\to\text{end}}_{l,h,\ell'\to\ell,r}$ to represent the average value of $B_{l,h,j\to i}(x)$ over samples x and token pairs i, j that are at the deepest NT-ends on levels ℓ, ℓ' respectively (symbolically, $\mathfrak{b}^\sharp(i) = \ell \wedge \mathfrak{b}^\sharp(j) = \ell'$), and are at a distance r based on the ancestor indices at level ℓ (symbolically, $\mathfrak{p}_\ell(j) - \mathfrak{p}_\ell(i) = r$). In Figure 6(c), we observe that $B^{\text{end}\to\text{end}}_{l,h,\ell'\to\ell,r}$ decreases as r increases, and is highest when $r = 0$ (or $r = 1$ for pairs $\ell' \to \ell$ without an $r = 0$ entry).[7]

In conclusion, tokens corresponding to NT-ends at level ℓ' statistically have higher attention weights to their *most adjacent* NT-ends at every level ℓ, *even after removing position-bias*.[8]

Connection to DP. Recall that dynamic programming (DP) comprises two components: *storage* and *recurrent formula*. While it's impractical to identify a specific DP implementation that the transformer follows since there are countless many ways to implement a DP, we can highlight *common elements* in DP implementations and their correlation with the transformer. In Section 4, we demonstrated that the generative transformer can encode the DP's *storage* "backbone", encompassing all necessary $\mathsf{DP}(i, j, a)$ on the correct CFG parsing tree of a given string.

For the *recurrent formula*, consider a CFG rule $a \mapsto b, c, d$ in the correct CFG parsing tree. If non-terminal (NT) b spans positions 21-30, c spans 31-40, and d spans 41-50, the DP must establish "memory links" between positions 30-40 and 40-50. This can be achieved by storing the $[bc]$ information at position 40 and merging it with $[d]$ at position 50, or by storing $[cd]$ at position 50 and merging it with $[b]$ at position 30. Regardless of the method, a common feature is the memory link from 30 to 40 and from 40 to 50. Hence, we have been examining such NT-end to NT-end attention links among adjacent NTs in this section.

The transformer is not only a parsing algorithm but also a generative one. Suppose $a \mapsto b, c$ and $c \mapsto d, e, f$ are on the correct parsing tree. When generating symbol e, the model, not having finished reading def, must access the precomputed knowledge from the uncle node b. This is why we also visualized those attentions from an NT-end to its most adjacent NT-end at a different level.

---

[6] 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 set average.

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

[8] Without removing position-bias, such a statement might be meaningless as the position-bias may favor "adjacent" anything, including NT-end pairs.

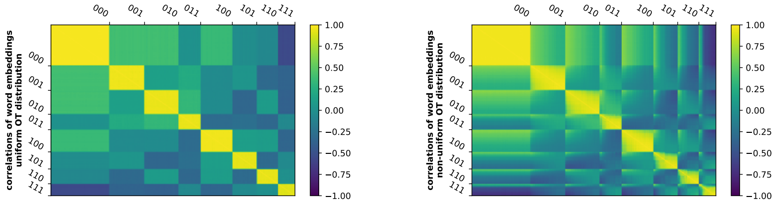
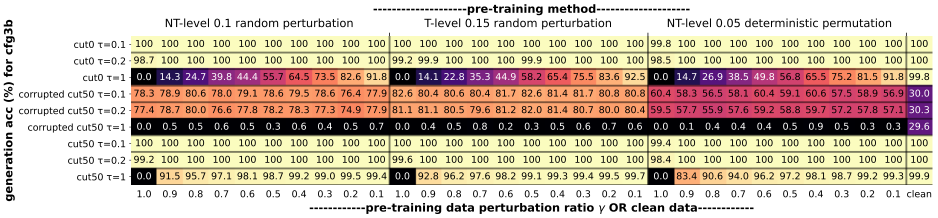


Figure 7: Language models learn implicit CFGs by using word embeddings to encode terminal symbol.

In implicit CFGs, the terminal symbols $t \in \mathbf{T}$ are associated with bags of tokens $\mathbf{OT}_t$ from which observable tokens are sampled. We present word embedding correlations pre-trained on an implicit CFG with $|\mathbf{T}| = 3$ and vocabulary size 300. Details are in Section A.1.


--------------------pre-training method--------------------
generation acc (%) for cfg3b
------------pre-training data perturbation ratio γ OR clean data------------

	NT-level 0.1 random perturbation									
	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
cut0 τ=0.1	100	100	100	100	100	100	100	100	100	100
cut0 τ=0.2	98.7	100	100	100	100	100	100	100	100	100
cut0 τ=1	0.0	14.3	24.7	39.8	44.4	55.7	64.5	73.5	82.6	91.8
corrupted cut50 τ=0.1	78.3	78.9	80.6	78.0	79.1	78.6	79.5	78.6	76.4	77.9
corrupted cut50 τ=0.2	77.4	78.7	80.0	76.6	77.8	78.2	78.3	77.3	74.9	77.9
corrupted cut50 τ=1	0.0	0.5	0.5	0.6	0.5	0.3	0.6	0.4	0.5	0.7
cut50 τ=0.1	100	100	100	100	100	100	100	100	100	100
cut50 τ=0.2	99.2	100	100	100	100	100	100	100	100	100
cut50 τ=1	0.0	91.5	95.7	97.1	98.1	98.7	99.2	99.0	99.5	99.4

	T-level 0.15 random perturbation									
	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
cut0 τ=0.1	100	100	100	100	100	100	100	100	100	100
cut0 τ=0.2	99.2	99.9	100	100	100	99.9	100	100	100	100
cut0 τ=1	0.0	14.1	22.8	35.3	44.9	58.2	65.4	75.5	83.6	92.5
corrupted cut50 τ=0.1	82.6	80.4	80.6	80.4	81.7	82.6	81.4	81.7	80.8	80.8
corrupted cut50 τ=0.2	81.1	81.1	80.5	79.6	81.2	82.0	81.4	80.7	80.0	80.4
corrupted cut50 τ=1	0.0	0.4	0.5	0.8	0.2	0.3	0.5	0.6	0.7	0.6
cut50 τ=0.1	100	100	100	100	100	100	100	100	100	100
cut50 τ=0.2	99.6	100	100	100	100	100	100	100	100	100
cut50 τ=1	0.0	92.8	96.2	97.6	98.2	99.1	99.3	99.4	99.5	99.7

	NT-level 0.05 deterministic permutation										
	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	clean
cut0 τ=0.1	99.8	100	100	100	100	100	100	100	100	100	100
cut0 τ=0.2	98.5	100	100	100	100	100	100	100	100	100	100
cut0 τ=1	0.0	14.7	26.9	38.5	49.8	56.8	65.5	75.2	81.5	91.8	99.8
corrupted cut50 τ=0.1	60.4	58.3	56.5	58.1	60.4	59.1	60.6	57.5	58.9	56.9	30.0
corrupted cut50 τ=0.2	59.5	57.7	55.9	57.6	59.2	58.8	59.7	57.2	57.8	57.1	30.3
corrupted cut50 τ=1	0.0	0.1	0.4	0.4	0.4	0.5	0.9	0.5	0.3	0.3	29.6
cut50 τ=0.1	99.4	100	100	100	100	100	100	100	100	100	100
cut50 τ=0.2	98.4	100	100	100	100	100	100	100	100	100	100
cut50 τ=1	0.0	83.4	90.6	94.0	96.2	97.2	98.1	98.7	99.2	99.3	99.9

Figure 8: 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 A.2.

In sum, while defining a good backbone for the DP recurrent formula may be challenging, we have demonstrated several attention patterns in this section that largely mimic dynamic programming regardless of the DP implementations.

6 CONCLUSION

Extensions. We defer *implicit CFGs* and *robust CFGs* to Appendix A, but briefly showcase the main discoveries in Figure 7 and 8.

Other related works. Numerous studies aim to uncover the inner workings of pretrained transformers. Some have observed attention heads that pair closing brackets with open ones, as noted in a concurrent study Zhang et al. (2023). Some have investigated induction heads applying logic operations to the input Olsson et al. (2022). Wang et al. (2022) explored many different types of attention heads, including “copy head” and “name mover head”. While our paper differs from these studies due to the distinct tasks we examine, we highlight that CFG is a *deep, recursive* task. Nevertheless, we still manage to reveal that the inner layers execute attentions in a complex, recursive, dynamic-programming-like manner, not immediately evident at the input level.

On the other hand, some studies can precisely determine each neuron’s function after training, typically on a simpler task using simpler architecture. For instance, Nanda et al. (2023) examined 1- or 2-layer transformers with a context length of 3 for the arithmetic addition. Our analysis focuses on the inner workings of GPT2-small, which has 12 layers and a context length exceeding 300. While we cannot precisely determine each neuron’s function, we have identified the roles of some heads and some hidden states, which correlate with dynamic programming.

Conclusion. In this paper, we studied how a transformer learns the CFGs structures in pretraining. CFGs in a language can include grammar, format, expressions, patterns, etc. We consider a synthetic, yet quite challenging family of CFGs to show how the inner workings of trained language models on these CFGs are highly correlated with the internal states of dynamic programming algorithms to parse those CFGs.

REFERENCES

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`.

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`.

Sanjeev Arora and Yi Zhang. Do gans actually learn the distribution? an empirical study. *arXiv preprint arXiv:1706.08224*, 2017.

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

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

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`.

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`.

Gregoire Deletang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, et al. Neural networks and the chomsky hierarchy. In *ICLR*, 2023.

Brian DuSell and David Chiang. Learning hierarchical structures with differentiable nondeterministic stacks. In *ICLR*, 2022.

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

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)*, pp. 4129–4138, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1419. URL `https://aclanthology.org/N19-1419`.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pp. 8571–8580, 2018.

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

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*, pp. 4171–4186, 2019.

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

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

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

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.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993. URL `https://aclanthology.org/J93-2004`.

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

Neel Nanda, Lawrence Chan, Tom Liberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. *arXiv preprint arXiv:2301.05217*, 2023.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

OpenAI. Gpt-4 technical report, 2023.

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)*, pp. 866–872, 2013.

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

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

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

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.

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, pp. 8267–8276, 2022.

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

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

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.

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.

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

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*, 2022.

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*, pp. 4166–4176, 2020.

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

Shizhuo Dylan Zhang, Curt Tigges, Stella Biderman, Maxim Raginsky, and Talia Ringer. Can transformers learn to solve problems recursively? *arXiv preprint arXiv:2305.14699*, 2023.

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

APPENDIX

A EXTENSIONS OF CFGS

A.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 $x = (x_1, x_2, \dots, x_m) \sim L(\mathcal{G})$, 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,

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

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 7), indicating that transformer learns implicit CFGs by using its token embedding layer to encode the hidden terminal symbol information. Details are in Appendix F.

A.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).

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: $\mathbf{Pr}_{x\sim L(\mathcal{G}),\, F}[(x_{:c}, F(\widetilde{x}_{:c})) \in L(\mathcal{G})]$.

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.[9]

- (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 $s_\ell = \left(s_{\ell,1}, s_{\ell,2}, \dots, s_{\ell,m_{L-1}}\right)$ 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.

---

[9] 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.

We focus on $\rho = 0.15$ with a wide range of perturbation rate $\tau = 0.0, 0.1, \dots, 0.9, 1.0$. We present our findings in Figure 8. Noticeable observations include:

- Rows 4/5 of Figure 8 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 8). 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 8 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).
- Comparing Rows 4/5 to Row 6 in Figure 8 we see that high robust accuracy is achieved when generating using low temperatures τ.[10] 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.

B EXPERIMENT SETUPS

B.1 DATASET DETAILS

We construct seven synthetic CFGs of depth $L = 7$ with varying levels of learning difficulty. It can be inferred that the greater the number of T/NT symbols, the more challenging it is to learn the CFG. For this reason, to push the capabilities of language models to their limits, we primarily focus on $\mathsf{cfg3b}, \mathsf{cfg3i}, \mathsf{cfg3h}, \mathsf{cfg3g}, \mathsf{cfg3f}$, which are of sizes $(1, 3, 3, 3, 3, 3, 3)$ and present increasing levels of difficulty. Detailed information about these CFGs is provided in Figure 9:

- In cfg3b, we construct the CFG such that the degree $|\mathcal{R}(a)| = 2$ for every NT a. We also ensure that in any generation rule, consecutive pairs of T/NT symbols are distinct.

  The 25%, 50%, 75%, and 95% percentile string lengths are $251, 278, 308, 342$ respectively.
- In cfg3i, we set $|\mathcal{R}(a)| = 2$ for every NT a. We remove the requirement for distinctness to make the data more challenging than cfg3b.

  The 25%, 50%, 75%, and 95% percentile string lengths are $276, 307, 340, 386$ respectively.
- In cfg3h, we set $|\mathcal{R}(a)| \in \{2, 3\}$ for every NT a to make the data more challenging than cfg3i.

  The 25%, 50%, 75%, and 95% percentile string lengths are $202, 238, 270, 300$ respectively.
- In cfg3g, we set $|\mathcal{R}(a)| = 3$ for every NT a to make the data more challenging than cfg3h.

  The 25%, 50%, 75%, and 95% percentile string lengths are $212, 258, 294, 341$ respectively.
- In cfg3f, we set $|\mathcal{R}(a)| \in \{3, 4\}$ for every NT a to make the data more challenging than cfg3g.

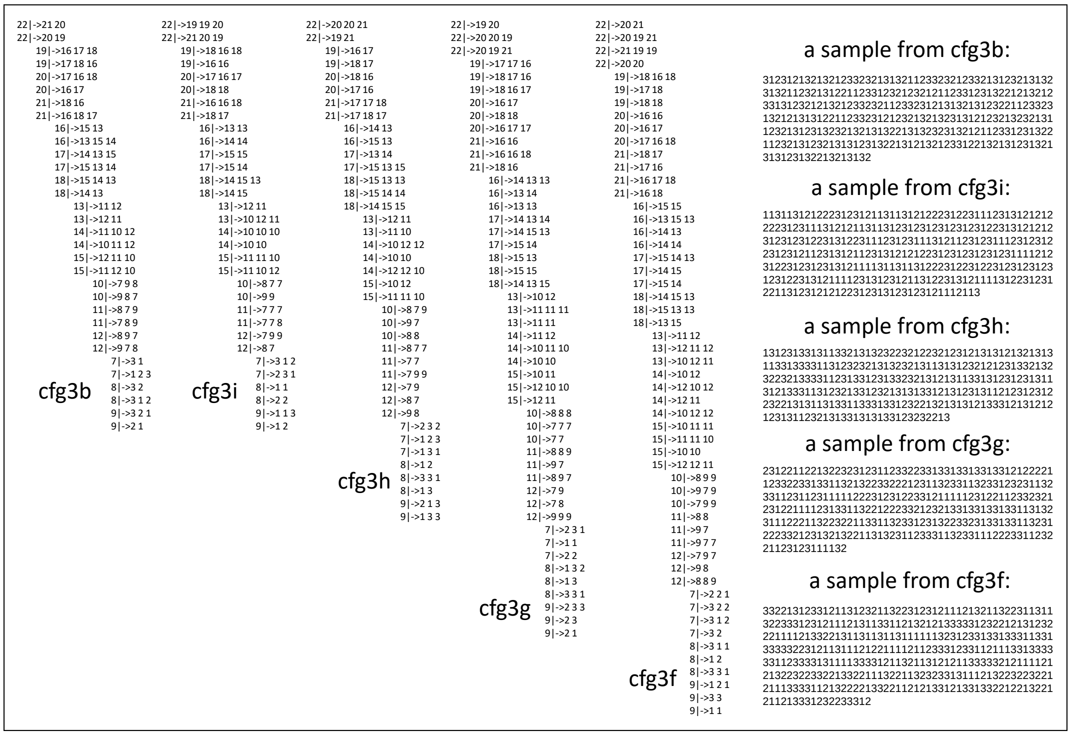
  The 25%, 50%, 75%, and 95% percentile string lengths are $191, 247, 302, 364$ respectively.

Remark B.1. From the examples in Figure 9, it becomes evident that for grammars $\mathcal{G}$ of depth 7, proving that a string x belongs to $L(\mathcal{G})$ is highly non-trivial, even for a human being, and even when the CFG rules are known. The standard method of demonstrating $x \in L(\mathcal{G})$ is through dynamic programming. We further discuss what we mean by a CFG's "difficulty" in Appendix H, and provide additional experiments beyond the cfg3 data family.

Remark B.2. cfg3f is a dataset that sits right on the boundary of difficulty at which GPT2-small is capable of learning (refer to subsequent subsections for training parameters). While it is certainly

---

[10] 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.



cfg3b

```
22|->21 20
22|->20 19
  19|->16 17 18
  19|->17 18 16
  20|->17 16 18
  20|->16 17
  21|->18 16
  21|->16 18 17
    16|->15 13
    16|->13 15 14
    17|->14 13 15
    17|->15 13 14
    18|->15 14 13
    18|->14 13
      13|->11 12
      13|->12 11
      14|->11 10 12
      14|->10 11 12
      15|->12 11 10
      15|->11 12 10
        10|->7 9 8
        10|->9 8 7
        11|->8 7 9
        11|->7 8 9
        12|->8 9 7
        12|->9 7 8
          7|->3 1
          7|->1 2 3
          8|->3 2
          8|->3 1 2
          9|->3 2 1
          9|->2 1
```

cfg3i

```
22|->19 19 20
22|->21 20 19
  19|->18 16 18
  19|->16 16
  20|->17 16 17
  20|->18 18
  21|->16 16 18
  21|->18 17
    16|->13 13
    16|->14 14
    17|->15 15
    17|->15 14
    18|->14 15 13
    18|->14 15
      13|->12 11
      13|->10 12 11
      14|->10 10 10
      14|->10 10
      15|->11 11 10
      15|->11 10 12
        10|->8 7 7
        10|->9 9
        11|->7 7 7
        11|->7 7 8
        12|->7 9 9
        12|->8 7
          7|->3 1 2
          7|->2 3 1
          8|->1 1
          8|->2 2
          9|->1 1 3
          9|->1 2
```

cfg3h

```
22|->20 20 21
22|->19 21
  19|->16 17
  19|->18 17
  20|->18 16
  20|->17 16
  21|->17 17 18
  21|->17 18 17
    16|->14 13
    16|->15 13
    17|->13 14
    17|->15 13 15
    18|->15 13 13
    18|->15 14 14
    18|->14 15 15
      13|->12 11
      13|->11 10
      14|->10 12 12
      14|->10 10
      14|->12 12 10
      15|->10 12
      15|->11 11 10
        10|->8 7 9
        10|->9 7
        10|->8 8
        11|->8 7 7
        11|->7 7
        11|->7 9 9
        12|->7 9
        12|->8 7
        12|->9 8
          7|->2 3 2
          7|->1 2 3
          7|->1 3 1
          8|->1 2
          8|->3 3 1
          8|->1 3
          9|->2 1 3
          9|->1 3 3
```

cfg3g

```
22|->19 20
22|->20 20 19
22|->20 19 21
  19|->17 17 16
  19|->18 17 16
  19|->18 16 17
  20|->16 17
  20|->18 18
  20|->16 17 17
  21|->16 16
  21|->16 16 18
  21|->18 16
    16|->14 13 13
    16|->13 14
    16|->13 13
    17|->14 13 14
    17|->14 15 13
    17|->15 14
    18|->15 13
    18|->15 15
    18|->14 13 15
      13|->10 12
      13|->11 11 11
      13|->11 11
      14|->11 12
      14|->10 11 10
      14|->10 10
      15|->10 11
      15|->12 10 10
      15|->12 11
        10|->8 8 8
        10|->7 7 7
        10|->7 7
        11|->8 8 9
        11|->9 7
        11|->8 9 7
        12|->7 9
        12|->7 8
        12|->9 9 9
          7|->2 3 1
          7|->1 1
          7|->2 2
          8|->1 3 2
          8|->1 3
          8|->3 3 1
          9|->2 3 3
          9|->2 3
          9|->2 1
```

cfg3f

```
22|->20 21
22|->20 19 21
22|->21 19 19
22|->20 20
  19|->18 16 18
  19|->17 18
  19|->18 18
  20|->16 16
  20|->16 17
  20|->17 16 18
  21|->18 17
  21|->17 16
  21|->16 17 18
  21|->16 18
    16|->15 15
    16|->13 15 13
    16|->14 13
    16|->14 14
    17|->15 14 13
    17|->14 15
    17|->15 14
    18|->14 15 13
    18|->15 13 13
    18|->13 15
      13|->11 12
      13|->12 11 12
      13|->10 12 11
      14|->10 12
      14|->12 10 12
      14|->12 11
      14|->10 12 12
      15|->10 11 11
      15|->11 11 10
      15|->10 10
      15|->12 12 11
        10|->8 9 9
        10|->9 7 9
        10|->7 9 9
        11|->8 8
        11|->9 7
        11|->9 7 7
        12|->7 9 7
        12|->9 8
        12|->8 8 9
          7|->2 2 1
          7|->3 2 2
          7|->3 1 2
          7|->3 2
          8|->3 1 1
          8|->1 2
          8|->3 3 1
          9|->1 2 1
          9|->3 3
          9|->1 1
```

a sample from cfg3b:

```
312312132132123323213132112332321233213123213132
313211232131221123312321232121123312313221213212
331312321213212332321123323121313213123221123323
132121313122112332312123213213231312123213232131
123213123132321321313221313232313212112331231322
112321312321313123132213121321233122132131231321
313123132213213132
```

a sample from cfg3i:

```
113113121222312312113113121222312231112313121212
222312311131212113113123123123123123122313121212
312312312231312231112312311131211231231112312312
231231211231312112313121212231231231231231111212
312231231231312111131131131222312231223123123123
123122313121111231312312113122313121111312231231
221131231212122312313123123121112113
```

a sample from cfg3h:

```
131231331311332131323223212232123121313121321313
113313333113123232131323213113131232121231332132
322321333311231331231332321312131133131231231311
312133311312321331232131313312131231311212312312
232213131131331133313312322132131312133312131212
1231311232131331313133123232213
```

a sample from cfg3g:

```
231221122132232312311233223313313313312122221
123322331331132132233222123113233113233123231132
33112311231111122231231223312111123122112332321
231221111231331132212223321232133133133133113132
311122211322322113311323312313223323133133113231
222332123132132211313231123331132331112223311232
21123123111132
```

a sample from cfg3f:

```
332213123312113123211322312312111213211322311311
322333123121112131133112132121333331232212131232
221111213322131131131131111113231233133133311331
333332231211311121221111211233312331121113313333
331123333131111333312113211312121133333212111121
213223223322133221113221132323313111213223223221
211133331121322221332211212133121331332212213221
211213331232233312
```

Figure 9: 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 typical way to decide if a string x belongs to the CFG language $x \in L(\mathcal{G})$ may require dynamic programming.

possible to consider deeper and more complex CFGs, this would necessitate training a larger network for a longer period. We choose not to do this as our findings are sufficiently convincing at the level of cfg3f.

Simultaneously, to illustrate that transformers can learn CFGs with larger $|\mathbf{NT}|$ or $|\mathbf{T}|$, we construct datasets cfg3e1 and cfg3e2 respectively of sizes $(1, 3, 9, 27, 81, 27, 9)$ and $(1, 3, 9, 27, 27, 9, 4)$. They are too lengthy to describe so only included in the supplementary materials.

B.2 MODEL ARCHITECTURE DETAILS

We define `GPT` as the standard GPT2-small architecture (Radford et al., 2019), which consists of 12 layers, 12 attention heads per layer, and 768 (=12 × 64) hidden dimensions. We pre-train `GPT` on the aforementioned datasets, starting from random initialization. For a baseline comparison, we also implement DeBERTa (He et al., 2020), resizing it to match the dimensions of GPT2 — thus also comprising 12 layers, 12 attention heads, and 768 dimensions.

Architecture size. We have experimented with models of varying sizes and observed that their learning capabilities scale with the complexity of the CFGs. To ensure a fair comparison and enhance reproducibility, we primarily focus on models with 12 layers, 12 attention heads, and 768 dimensions. The transformers constructed in this manner consist of 86M parameters.

Modern GPTs with relative attention. Recent research (Black et al., 2022; He et al., 2020; Su et al., 2021) has demonstrated that transformers can significantly improve performance by using attention mechanisms based on the *relative* position differences of tokens, as opposed to the absolute positions used in the original GPT2 (Radford et al., 2019) or BERT (Kenton & Toutanova, 2019). There are two main approaches to achieve this. The first is to use a "relative positional embedding layer" on $|j - i|$ when calculating the attention from j to i (or a bucket embedding to save space). This approach is the most effective but tends to train slower. The second approach is to apply a rotary positional embedding (RoPE) transformation (Su et al., 2021) on the hidden states; this is known to be slightly less effective than the relative approach, but it can be trained much faster.

We have implemented both approaches. We adopted the RoPE implementation from the GPT-NeoX-20B project (along with the default parameters), but downsized it to fit the GPT2 small model. We refer to this architecture as $\texttt{GPT}_{\mathsf{rot}}$. Since we could not find a standard implementation of GPT using relative attention, we re-implemented GPT2 using the relative attention framework from DeBERTa (He et al., 2020). (Recall, DeBERTa is a variant of BERT that effectively utilizes relative positional embeddings.) We refer to this architecture as $\texttt{GPT}_{\mathsf{rel}}$.

Weaker GPTs utilizing only position-based attention. For the purpose of analysis, we also consider two significantly weaker variants of GPT, where the attention matrix *exclusively depends* on the token positions, and not on the input sequences or hidden embeddings. In other words, the attention pattern remains *constant* for all input sequences.

We implement $\texttt{GPT}_{\mathsf{pos}}$, a variant of $\texttt{GPT}_{\mathsf{rel}}$ that restricts the attention matrix to be computed solely using the (trainable) relative positional embedding. This can be perceived as a GPT variant that *maximizes the use of position-based attention*. We also implement $\texttt{GPT}_{\mathsf{uni}}$, a 12-layer, 8-head, 1024-dimension transformer, where the attention matrix is *fixed*; for each $h \in [8]$, the h-th head *consistently* uses a fixed, uniform attention over the previous $2^h - 1$ tokens. This can be perceived as a GPT variant that *employs the simplest form of position-based attention*.

Remark B.3. It should not be surprising that $\texttt{GPT}_{\mathsf{pos}}$ or $\texttt{GPT}_{\mathsf{uni}}$ 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 $\texttt{GPT}_{\mathsf{rel}}$, 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 (Allen-Zhu et al., 2019; Jacot et al., 2018). We call this $\texttt{GPT}_{\mathsf{rand}}$.

B.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.[11] For DeBERTa, we use learning rate 0.0001 which is better and 2000 steps of learning rate linear warmup.

Throughout the experiments, for both pre-training and testing, we only use **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 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. Please note, given that our CFG language is very large (e.g., length 300 tree of length-2/3 rules and degree 4 would have at least $4^{300/3}$ possibility), there is *almost no chance that training/testing hit the same sentence*.

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, we find it more interesting to use the resources to run it against different datasets and against different

---

[11] 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.

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 10.

B.4 PREDICT NT ANCESTOR AND NT BOUNDARY

Recall from Section 4.1 that we have proposed to use a multi-head 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 can be of dimension 512×768 — when having a context length 512 and hidden dimension 768 — recall in (4.2), we have proposed to use a multi-head attention to construct such linear function for efficient learning purpose. This significantly reduces sample complexity and makes it much easier to find the linear function.

In our implementation, we choose $H = 16$ heads and hidden dimension $d' = 1024$ when constructing this position-based attention in (4.2). 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.

Once again we use *fresh new samples* when training such linear functions. When evaluating the accuracies on predicting the NT ancester / boundary information, we also use fresh new samples. Recall our CFG language is sufficiently large so there is negligible chance that the model has seen such a string during training.

C MORE EXPERIMENTS ON GENERATION

C.1 GENERATION DIVERSITY VIA BIRTHDAY PARADOX

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 C.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:*[12]

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

Definition C.2. *For symbol* $a \in \mathbf{NT}_{\ell_1}$ *and some layer* $\ell_2 \in \{\ell_1, \ell_1+1, \dots, L\}$*, define multi-set*[13]

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

and we define the multi-set union $\mathcal{S}_{a\to\ell_2} = \bigcup_{i\in[M]} \mathcal{S}_{a\to\ell_2}(x^{(i)})$*, which is* ***the multiset of all sentential forms that can be derived from NT symbol*** a ***to depth*** ℓ_2.

(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. 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 $x^{(1)}, \dots, x^{(M)}$ are i.i.d. sampled from the real distribution $L(\mathcal{G})$, and denote by

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

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

---

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

[13] 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.

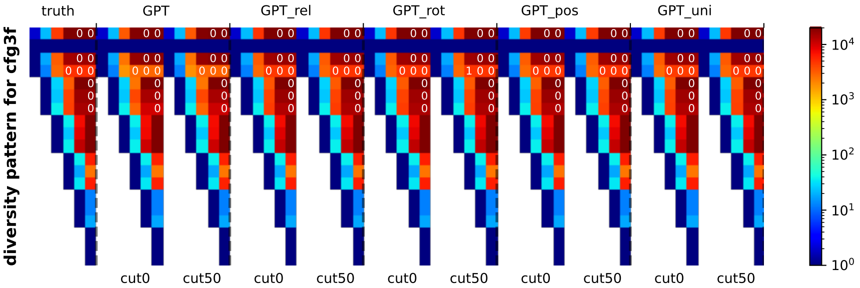


Figure 10: Comparing the generation diversity $\mathcal{S}^{\text{truth}}_{a\to\ell_2}$ and $\mathcal{S}^{F}_{a\to\ell_2}$ across different learned GPT models ($c = 0$ or $c = 50$). Rows correspond to NT symbols a and columns correspond to $\ell_2 = 2, 3, \dots, 7$. Colors represent the number of distinct 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 Figure 11, 12, and 13

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.

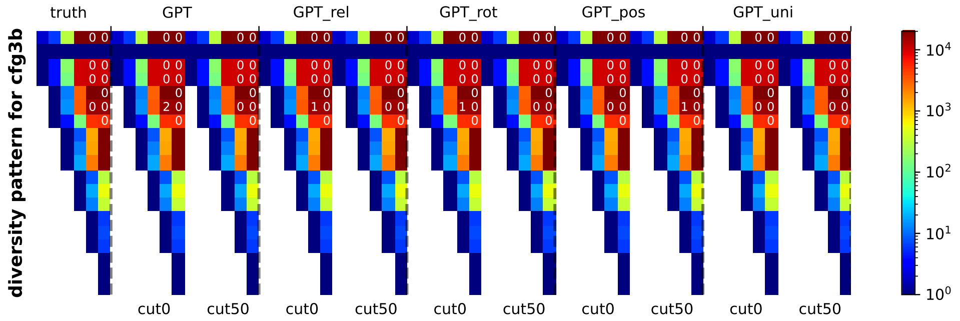
that nearly-perfectly satisfy the CFG rules.

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. 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.[14] We present our findings in Figure 10 with $M = 20000$ samples for the cfg3f dataset.

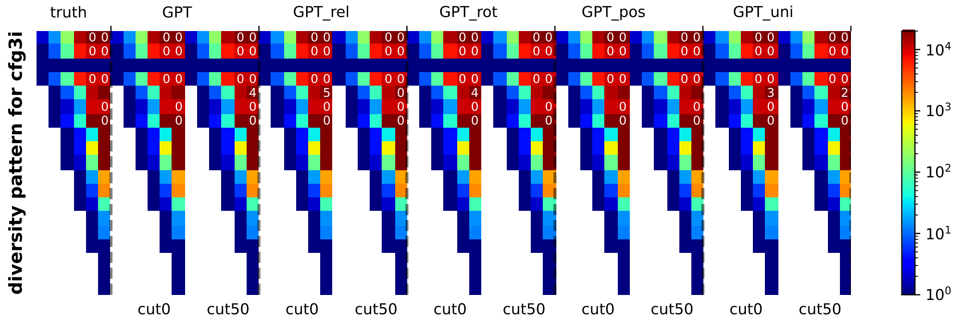
In Figure 11 we present that for cfg3b, cfg3i, cfg3h, cfg3g, in Figure 12 for cfg3e1, and in Figure 13 for cfg3e2. We note that not only for hard, ambiguous datasets, also for those less ambiguous (cfg3e1, cfg3e2) datasets, language models are capable of generating very diverse outputs.

---

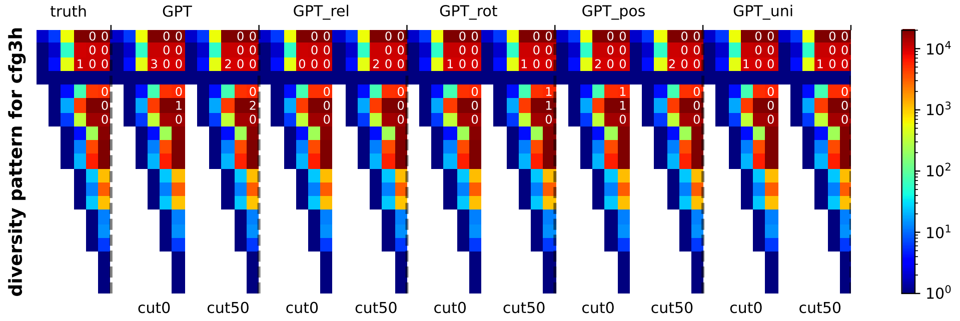
[14] A model might generate a same NT symbol sequence s_{L-1}, and then generate different Ts randomly from each NT. In this way, the model still generates strings x's with large diversity, but $\mathcal{S}^{F}_{a\to L-1}(x)$ is small. If $\mathcal{S}^{F}_{a\to\ell_2}$ is large for every ℓ_2 and a, then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



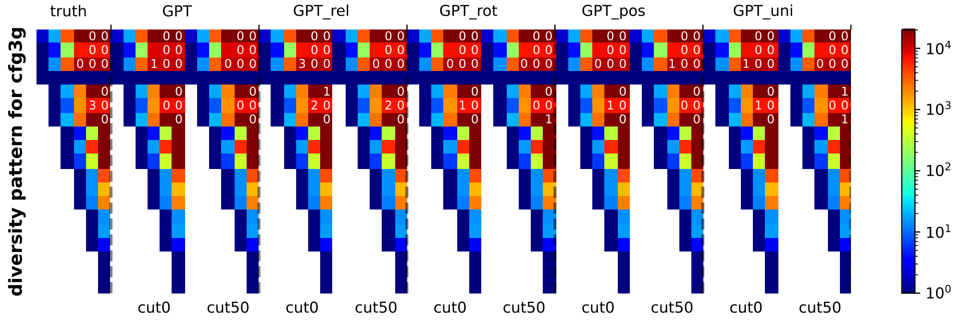
(a) cfg3b dataset



(b) cfg3i dataset



(c) cfg3h dataset



(d) cfg3g dataset

Figure 11: 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).

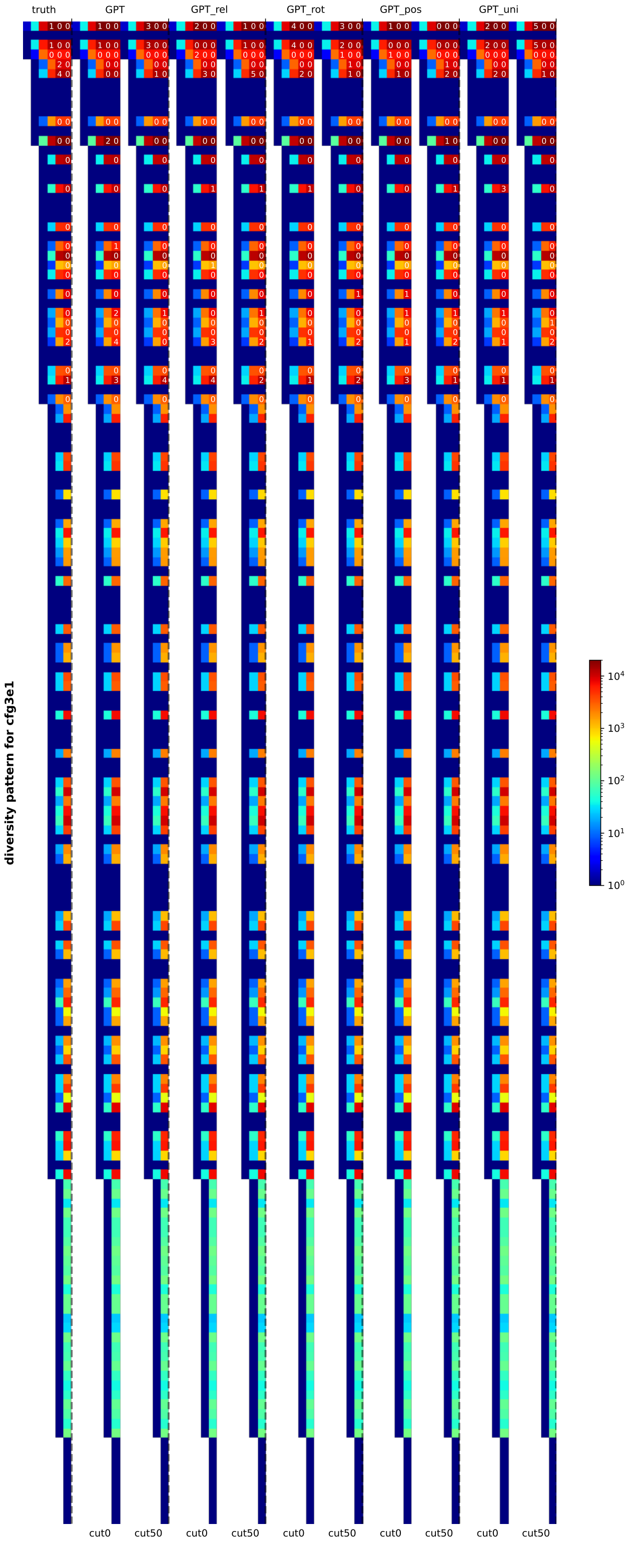


Figure 12: Comparing the generation diversity $\mathcal{S}^{\text{truth}}_{a\to\ell_2}$ and $\mathcal{S}^{F}_{a\to\ell_2}$ across different learned GPT models (and for $c = 0$ or $c = 50$). Rows correspond to NT symbols a and columns correspond to $\ell_2 = 2, 3, \ldots, 7$. Colors represent the number 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 `cfg3e1` dataset.

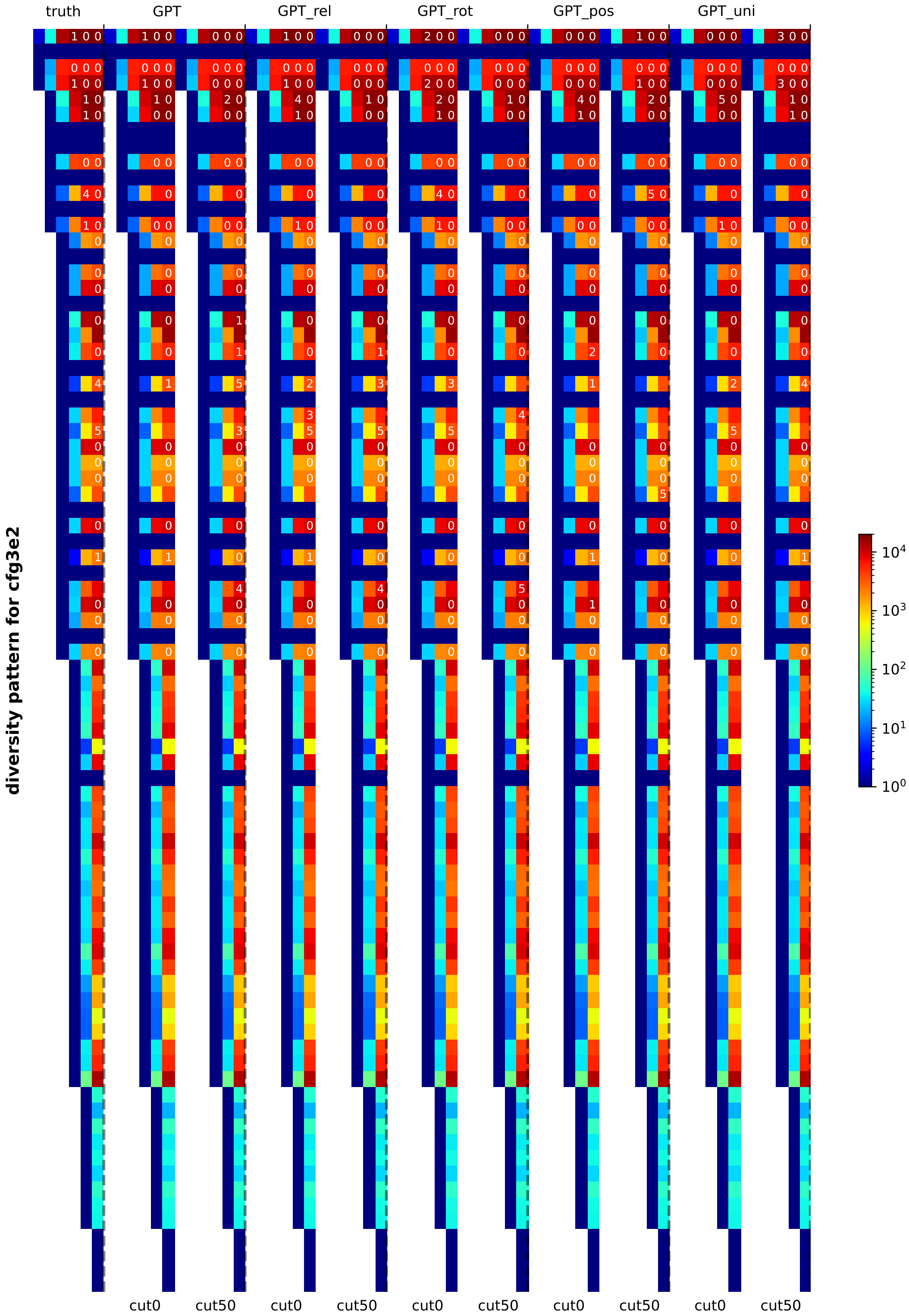
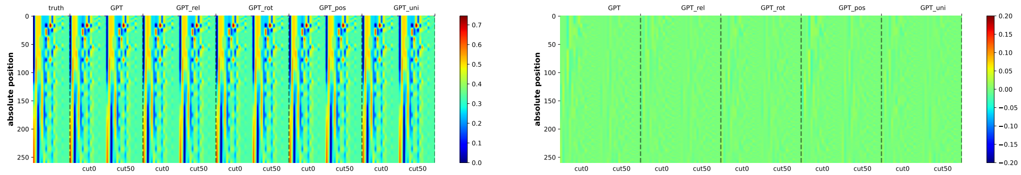


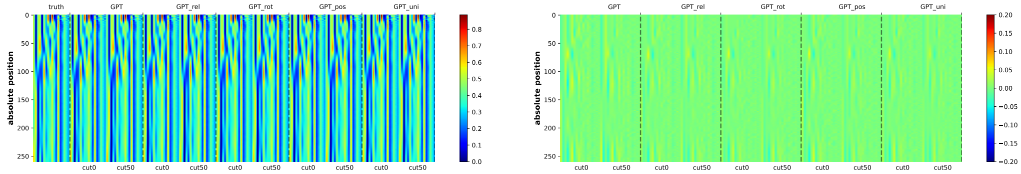
Figure 13: Comparing the generation diversity $\mathcal{S}^{\text{truth}}_{a\to\ell_2}$ and $\mathcal{S}^{F}_{a\to\ell_2}$ across different learned GPT models (and for $c = 0$ or $c = 50$). Rows correspond to NT symbols a and columns correspond to $\ell_2 = 2, 3, \ldots, 7$. Colors represent the number 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.

C.2 MARGINAL DISTRIBUTION COMPARISON

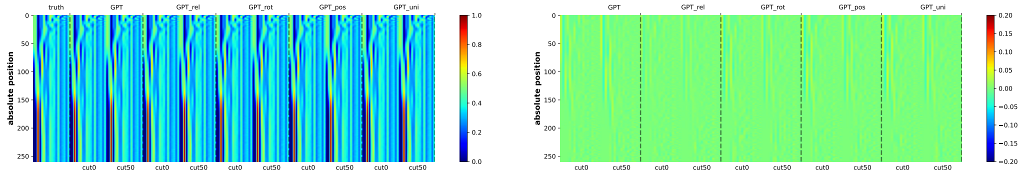
In order to effectively learn a CFG, it is also important to match the distribution of generating probabilities. While measuring this can be challenging, we have conducted at least a simple test on the marginal distributions $p(a, i)$, which represent the probability of symbol $a \in \mathbf{NT}_\ell$ 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 14.



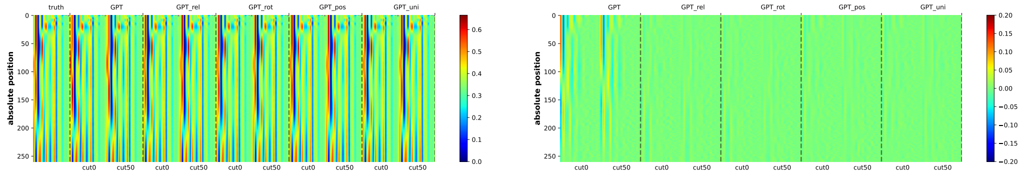
(a) cfg3b dataset; marginal distribution (b) cfg3b dataset; marginal distribution - ground truth



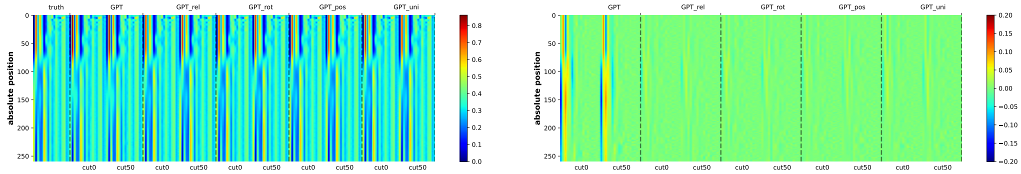
(c) cfg3i dataset; marginal distribution (d) cfg3i dataset; marginal distribution - ground truth



(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

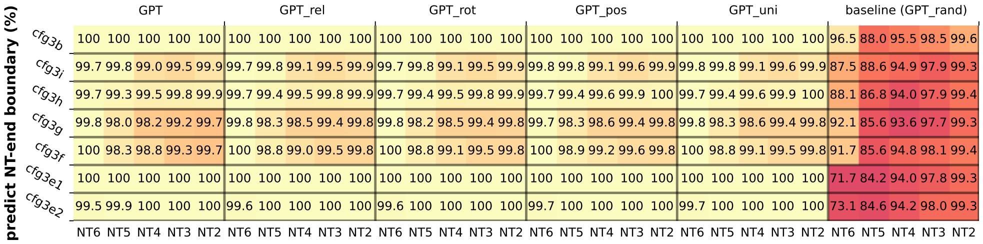
Figure 14: Marginal distribution $p(a, i)$ difference between a trained model and the ground-truth, for an NT/T symbol a (column) at position i (row). Figures on the left compare the marginal distribution of the ground-truth against those generated from 5 models $\times$ 2 cut positions ($c = 0/c = 50$). 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.

D MORE EXPERIMENTS ON NT ANCESTOR AND NT BOUNDARY PREDICTIONS

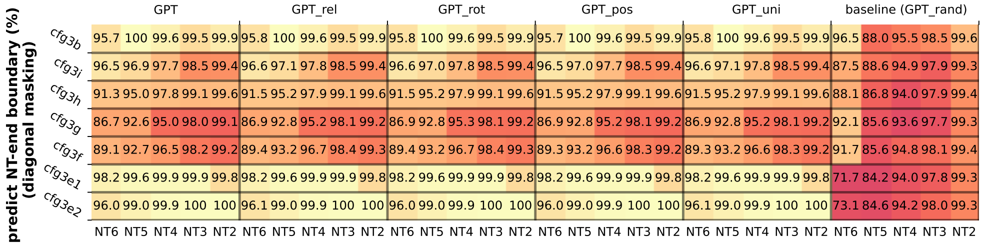
D.1 NT ANCESTOR AND NT BOUNDARY PREDICTIONS

Earlier, as confirmed in Figure 4, we established that the hidden states (of the final transformer layer) have implicitly encoded the NT ancestor symbols $\mathfrak{s}_\ell(i)$ for each CFG level ℓ and token position i using a linear transformation. In Figure 15(a), we also demonstrated that the same conclusion applies to the NT-end boundary information $\mathfrak{b}_\ell(i)$. More importantly, for $\mathfrak{b}_\ell(i)$, we showed that this information is *stored locally*, very close to position i (such as at $i \pm 1$). Detailed information can be found in Figure 15.

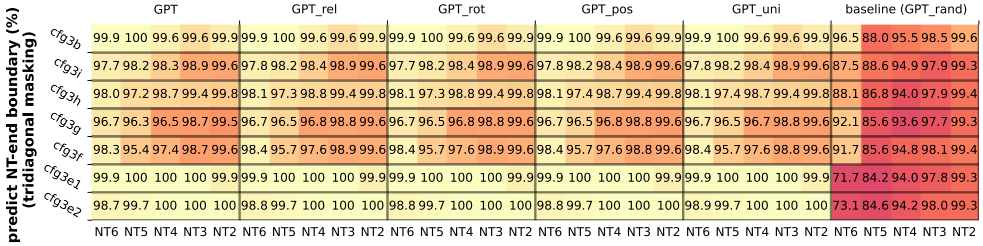
Furthermore, as recalled in Figure 5, we confirmed that at any NT boundary where $\mathfrak{b}_\ell(i) = 1$, the transformer has also locally encoded clear information about the NT ancestor symbol $\mathfrak{s}_\ell(i)$, either exactly at i or at $i \pm 1$. To be precise, this is a conditional statement — given that it is an NT boundary, NT ancestors can be predicted. Therefore, in principle, one must also verify that the prediction task for the NT boundary is successful to begin with. Such missing experiments are, in fact, included in Figure 15(b) and Figure 15(c).



(a) Predicting NT boundaries: the column NT_ℓ for $\ell = 2, 3, 4, 5, 6$ represents the accuracy of predicting $\mathfrak{b}_\ell$ using the multi-head linear probing function described in (4.2).



(b) Predicting NT boundaries with diagonal masking: the column NT_ℓ for $\ell = 2, 3, 4, 5, 6$ represents the accuracy of predicting $\mathfrak{b}_\ell$ using (4.2) but setting $w_{r,i \to k} = 0$ for $i \neq k$.



(c) Predicting NT boundaries with tridiagonal masking: the column NT_ℓ for $\ell = 2, 3, 4, 5, 6$ represents the accuracy of predicting $\mathfrak{b}_\ell$ using (4.2) but setting $w_{r,i \to k} = 0$ for $|i - k| > 1$.

Figure 15: After pre-training, the NT-end boundary information — i.e., $\mathfrak{b}_\ell(i)$ for position i and NT level ℓ — is largely stored *locally* near the hidden state at position $i \pm 1$, up to a linear transformation. This can be compared with the prediction accuracy of the NT ancestor $\mathfrak{s}_\ell(i)$ in Figure 4.

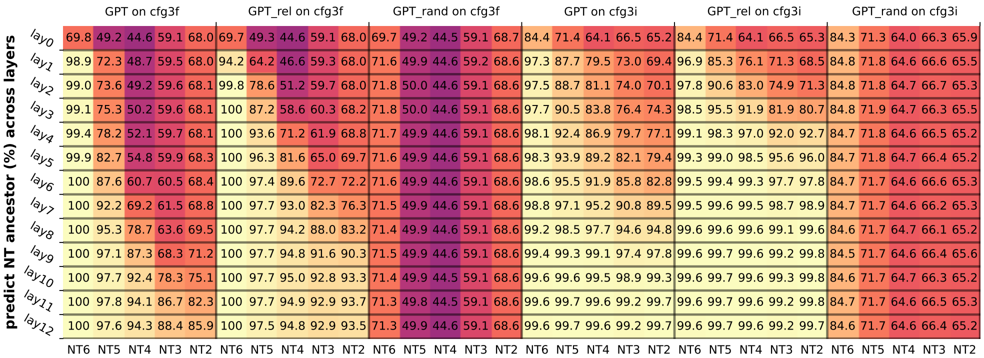
Observation. This implies, the transformer actually *knows*, with a very good accuracy, that "position i is already the end of NT on level ℓ", by just reading all the texts until this position (possibly peeking one more to its right).

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 1). 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.

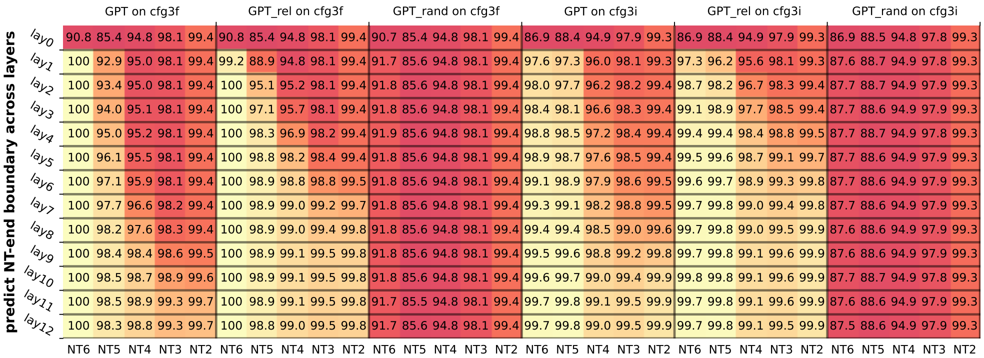
D.2 NT PREDICTIONS ACROSS TRANSFORMER'S LAYERS

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 16, we present this finding by calculating the *linear* encoding accuracies with respect to all the 12 transformer layers in `GPT` and $\texttt{GPT}_{\textsf{rel}}$. We confirm that generative models discover such information *hierarchically*.



predict NT ancestor (%) across layers	GPT on cfg3f					GPT_rel on cfg3f					GPT_rand on cfg3f					GPT on cfg3i					GPT_rel on cfg3i					GPT_rand on cfg3i				
	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2
lay_0	69.8	49.2	44.6	59.1	68.0	69.7	49.3	44.6	59.1	68.0	69.7	49.2	44.5	59.1	68.7	84.4	71.4	64.1	66.5	65.2	84.4	71.4	64.1	66.5	65.3	84.3	71.3	64.0	66.3	65.9
lay_1	98.9	72.3	48.7	59.5	68.0	94.2	64.2	46.6	59.3	68.0	71.6	49.9	44.6	59.2	68.6	97.3	87.7	79.5	73.0	69.4	96.9	85.3	76.1	71.3	68.5	84.8	71.8	64.6	66.6	65.5
lay_2	99.0	73.6	49.2	59.6	68.1	99.8	78.6	51.2	59.7	68.0	71.8	50.0	44.6	59.1	68.6	97.5	88.7	81.1	74.0	70.1	97.8	90.6	83.0	74.9	71.3	84.8	71.8	64.7	66.7	65.3
lay_3	99.1	75.3	50.2	59.6	68.1	100	87.2	58.6	60.3	68.2	71.8	50.0	44.6	59.1	68.6	97.7	90.5	83.8	76.4	74.3	98.5	95.5	91.9	81.9	80.7	84.8	71.9	64.7	66.3	65.5
lay_4	99.4	78.2	52.1	59.7	68.1	100	93.6	71.2	61.9	68.8	71.7	49.9	44.6	59.1	68.6	98.1	92.4	86.9	79.7	77.1	99.1	98.3	97.0	92.0	92.7	84.7	71.8	64.6	66.5	65.2
lay_5	99.9	82.7	54.8	59.9	68.3	100	96.3	81.6	65.0	69.7	71.6	49.9	44.6	59.1	68.6	98.3	93.9	89.2	82.1	79.4	99.3	99.0	98.5	95.6	96.0	84.7	71.8	64.7	66.4	65.2
lay_6	100	87.6	60.7	60.5	68.4	100	97.4	89.6	72.7	72.2	71.6	49.9	44.6	59.1	68.6	98.6	95.5	91.9	85.8	82.8	99.5	99.4	99.3	97.7	97.8	84.7	71.7	64.6	66.6	65.3
lay_7	100	92.2	69.2	61.5	68.8	100	97.7	93.0	82.3	76.3	71.5	49.9	44.6	59.1	68.6	98.8	97.1	95.2	90.8	89.5	99.5	99.6	99.5	98.7	98.9	84.7	71.7	64.6	66.2	65.3
lay_8	100	95.3	78.7	63.6	69.5	100	97.7	94.2	88.0	83.2	71.4	49.9	44.6	59.1	68.6	99.2	98.5	97.7	94.6	94.8	99.6	99.6	99.6	99.1	99.6	84.6	71.7	64.7	66.1	65.2
lay_9	100	97.1	87.3	68.3	71.2	100	97.7	94.8	91.6	90.3	71.5	49.9	44.6	59.1	68.6	99.4	99.3	99.1	97.4	97.8	99.6	99.7	99.6	99.2	99.8	84.5	71.7	64.6	66.4	65.6
lay_{10}	100	97.7	92.4	78.3	75.1	100	97.7	95.0	92.8	93.3	71.4	49.9	44.5	59.1	68.6	99.6	99.6	99.5	98.9	99.3	99.6	99.7	99.6	99.3	99.8	84.6	71.7	64.7	66.3	65.2
lay_{11}	100	97.8	94.1	86.7	82.3	100	97.7	94.9	92.9	93.7	71.3	49.8	44.5	59.1	68.6	99.6	99.7	99.6	99.2	99.7	99.6	99.7	99.6	99.2	99.8	84.7	71.7	64.6	66.5	65.3
lay_{12}	100	97.6	94.3	88.4	85.9	100	97.5	94.8	92.9	93.5	71.3	49.9	44.6	59.1	68.6	99.6	99.7	99.6	99.2	99.7	99.6	99.7	99.6	99.2	99.7	84.6	71.7	64.6	66.4	65.2

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



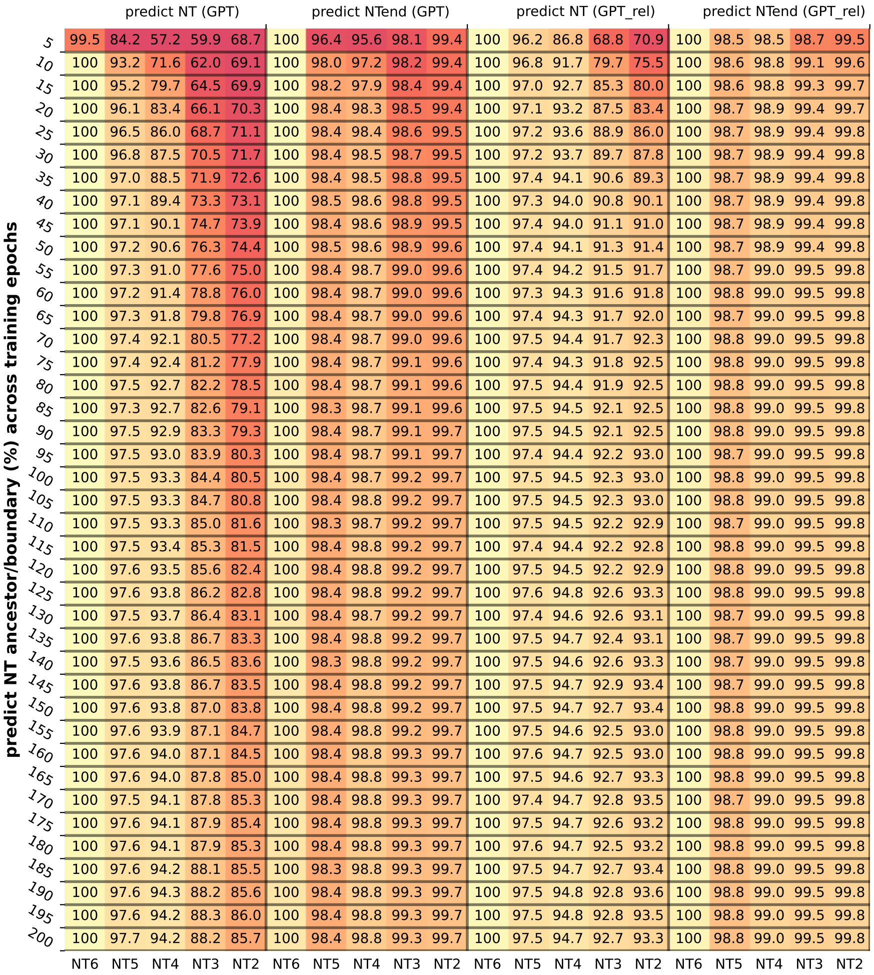
predict NT-end boundary across layers	GPT on cfg3f					GPT_rel on cfg3f					GPT_rand on cfg3f					GPT on cfg3i					GPT_rel on cfg3i					GPT_rand on cfg3i				
	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2
lay_0	90.8	85.4	94.8	98.1	99.4	90.8	85.4	94.8	98.1	99.4	90.7	85.4	94.8	98.1	99.4	86.9	88.4	94.9	97.9	99.3	86.9	88.4	94.9	97.9	99.3	86.9	88.5	94.8	97.8	99.3
lay_1	100	92.9	95.0	98.1	99.4	99.2	88.9	94.8	98.1	99.4	91.7	85.6	94.8	98.1	99.4	97.6	97.3	96.0	98.1	99.3	97.3	96.2	95.6	98.1	99.3	87.6	88.7	94.9	97.8	99.3
lay_2	100	93.4	95.0	98.1	99.4	100	95.1	95.2	98.1	99.4	91.8	85.6	94.8	98.1	99.4	98.0	97.7	96.2	98.2	99.4	98.7	98.2	96.7	98.3	99.4	87.7	88.7	94.9	97.9	99.3
lay_3	100	94.0	95.1	98.1	99.4	100	97.1	95.7	98.1	99.4	91.8	85.6	94.8	98.1	99.4	98.4	98.1	96.6	98.3	99.4	99.1	98.9	97.7	98.5	99.4	87.7	88.6	94.9	97.9	99.3
lay_4	100	95.0	95.2	98.1	99.4	100	98.3	96.9	98.2	99.4	91.9	85.6	94.8	98.1	99.4	98.8	98.5	97.2	98.4	99.4	99.4	99.4	98.4	98.8	99.5	87.7	88.7	94.9	97.8	99.3
lay_5	100	96.1	95.5	98.1	99.4	100	98.8	98.2	98.4	99.4	91.8	85.6	94.8	98.1	99.4	98.9	98.7	97.6	98.5	99.4	99.5	99.6	98.7	99.1	99.7	87.7	88.6	94.9	97.9	99.3
lay_6	100	97.1	95.9	98.1	99.4	100	98.9	98.8	98.8	99.5	91.8	85.6	94.8	98.1	99.4	99.1	98.9	97.9	98.6	99.5	99.6	99.7	98.9	99.3	99.8	87.7	88.6	94.9	97.9	99.3
lay_7	100	97.7	96.6	98.2	99.4	100	98.9	99.0	99.2	99.7	91.8	85.6	94.8	98.1	99.4	99.3	99.1	98.2	98.8	99.5	99.7	99.8	99.0	99.4	99.8	87.7	88.6	94.9	97.9	99.3
lay_8	100	98.2	97.6	98.3	99.4	100	98.9	99.0	99.4	99.8	91.8	85.6	94.8	98.1	99.4	99.4	99.4	98.5	99.0	99.6	99.7	99.8	99.0	99.5	99.9	87.6	88.6	94.9	97.9	99.3
lay_9	100	98.4	98.4	98.6	99.5	100	98.9	99.1	99.5	99.8	91.8	85.6	94.8	98.1	99.4	99.5	99.6	98.8	99.2	99.8	99.7	99.8	99.1	99.6	99.9	87.6	88.6	94.9	97.9	99.3
lay_{10}	100	98.5	98.7	98.9	99.6	100	98.9	99.1	99.5	99.8	91.8	85.6	94.8	98.1	99.4	99.6	99.7	99.0	99.4	99.9	99.8	99.8	99.1	99.6	99.9	87.7	88.7	94.9	97.8	99.3
lay_{11}	100	98.5	98.9	99.3	99.7	100	98.9	99.1	99.5	99.8	91.7	85.5	94.8	98.1	99.4	99.7	99.8	99.1	99.5	99.9	99.7	99.8	99.1	99.6	99.9	87.6	88.6	94.9	97.9	99.3
lay_{12}	100	98.3	98.8	99.3	99.7	100	98.8	99.0	99.5	99.8	91.7	85.6	94.8	98.1	99.4	99.7	99.8	99.0	99.5	99.9	99.7	99.8	99.1	99.5	99.9	87.5	88.6	94.9	97.9	99.3

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

Figure 16: Generative models discover NT ancestors and NT boundaries hierarchically.

D.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 17. 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" (Allen-Zhu & Li, 2023).



predict NT ancestor/boundary (%) across training epochs	predict NT (GPT)					predict NTend (GPT)					predict NT (GPT_rel)					predict NTend (GPT_rel)				
	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2	NT6	NT5	NT4	NT3	NT2
5	99.5	84.2	57.2	59.9	68.7	100	96.4	95.6	98.1	99.4	100	96.2	86.8	68.8	70.9	100	98.5	98.5	98.7	99.5
10	100	93.2	71.6	62.0	69.1	100	98.0	97.2	98.2	99.4	100	96.8	91.7	79.7	75.5	100	98.6	98.8	99.1	99.6
15	100	95.2	79.7	64.5	69.9	100	98.2	97.9	98.4	99.4	100	97.0	92.7	85.3	80.0	100	98.6	98.8	99.3	99.7
20	100	96.1	83.4	66.1	70.3	100	98.4	98.3	98.5	99.4	100	97.1	93.2	87.5	83.4	100	98.7	98.9	99.4	99.7
25	100	96.5	86.0	68.7	71.1	100	98.4	98.4	98.6	99.5	100	97.2	93.6	88.9	86.0	100	98.7	98.9	99.4	99.8
30	100	96.8	87.5	70.5	71.7	100	98.4	98.5	98.7	99.5	100	97.2	93.7	89.7	87.8	100	98.7	98.9	99.4	99.8
35	100	97.0	88.5	71.9	72.6	100	98.4	98.5	98.8	99.5	100	97.4	94.1	90.6	89.3	100	98.7	98.9	99.4	99.8
40	100	97.1	89.4	73.3	73.1	100	98.5	98.6	98.8	99.5	100	97.3	94.0	90.8	90.1	100	98.7	98.9	99.4	99.8
45	100	97.1	90.1	74.7	73.9	100	98.4	98.6	98.9	99.5	100	97.4	94.0	91.1	91.0	100	98.7	98.9	99.4	99.8
50	100	97.2	90.6	76.3	74.4	100	98.5	98.6	98.9	99.6	100	97.4	94.1	91.3	91.4	100	98.7	98.9	99.4	99.8
55	100	97.3	91.0	77.6	75.0	100	98.4	98.7	99.0	99.6	100	97.4	94.2	91.5	91.7	100	98.7	99.0	99.5	99.8
60	100	97.2	91.4	78.8	76.0	100	98.4	98.7	99.0	99.6	100	97.3	94.3	91.6	91.8	100	98.8	99.0	99.5	99.8
65	100	97.3	91.8	79.8	76.9	100	98.4	98.7	99.0	99.6	100	97.4	94.3	91.7	92.0	100	98.7	99.0	99.5	99.8
70	100	97.4	92.1	80.5	77.2	100	98.4	98.7	99.0	99.6	100	97.5	94.4	91.7	92.3	100	98.8	99.0	99.5	99.8
75	100	97.4	92.4	81.2	77.9	100	98.4	98.7	99.1	99.6	100	97.4	94.3	91.8	92.5	100	98.8	99.0	99.5	99.8
80	100	97.5	92.7	82.2	78.5	100	98.4	98.7	99.1	99.6	100	97.5	94.4	91.9	92.5	100	98.8	99.0	99.5	99.8
85	100	97.3	92.7	82.6	79.1	100	98.3	98.7	99.1	99.6	100	97.5	94.5	92.1	92.5	100	98.8	99.0	99.5	99.8
90	100	97.5	92.9	83.3	79.3	100	98.4	98.7	99.1	99.7	100	97.5	94.5	92.1	92.5	100	98.8	99.0	99.5	99.8
95	100	97.5	93.0	83.9	80.3	100	98.4	98.7	99.1	99.7	100	97.4	94.4	92.2	93.0	100	98.7	99.0	99.5	99.8
100	100	97.5	93.3	84.4	80.5	100	98.4	98.7	99.2	99.7	100	97.5	94.5	92.3	93.0	100	98.8	99.0	99.5	99.8
105	100	97.5	93.3	84.7	80.8	100	98.4	98.8	99.2	99.7	100	97.5	94.5	92.3	93.0	100	98.8	99.0	99.5	99.8
110	100	97.5	93.3	85.0	81.6	100	98.3	98.7	99.2	99.7	100	97.5	94.5	92.2	92.9	100	98.7	99.0	99.5	99.8
115	100	97.5	93.4	85.3	81.5	100	98.4	98.8	99.2	99.7	100	97.4	94.4	92.2	92.8	100	98.8	99.0	99.5	99.8
120	100	97.6	93.5	85.6	82.4	100	98.4	98.8	99.2	99.7	100	97.5	94.5	92.2	92.9	100	98.8	99.0	99.5	99.8
125	100	97.6	93.8	86.2	82.8	100	98.4	98.8	99.2	99.7	100	97.6	94.8	92.6	93.3	100	98.8	99.0	99.5	99.8
130	100	97.5	93.7	86.4	83.1	100	98.4	98.7	99.2	99.7	100	97.4	94.6	92.6	93.1	100	98.7	99.0	99.5	99.8
135	100	97.6	93.8	86.7	83.3	100	98.4	98.8	99.2	99.7	100	97.5	94.7	92.4	93.1	100	98.7	99.0	99.5	99.8
140	100	97.5	93.6	86.5	83.6	100	98.3	98.8	99.2	99.7	100	97.5	94.6	92.6	93.3	100	98.7	99.0	99.5	99.8
145	100	97.6	93.8	86.7	83.5	100	98.4	98.8	99.2	99.7	100	97.5	94.7	92.9	93.4	100	98.7	99.0	99.5	99.8
150	100	97.6	93.8	87.0	83.8	100	98.4	98.8	99.2	99.7	100	97.5	94.7	92.7	93.4	100	98.8	99.0	99.5	99.8
155	100	97.6	93.9	87.1	84.7	100	98.4	98.8	99.2	99.7	100	97.5	94.6	92.5	93.0	100	98.8	99.0	99.5	99.8
160	100	97.6	94.0	87.1	84.5	100	98.4	98.8	99.3	99.7	100	97.6	94.7	92.5	93.0	100	98.8	99.0	99.5	99.8
165	100	97.6	94.0	87.8	85.0	100	98.4	98.8	99.3	99.7	100	97.5	94.6	92.7	93.3	100	98.8	99.0	99.5	99.8
170	100	97.5	94.1	87.8	85.3	100	98.4	98.8	99.3	99.7	100	97.4	94.7	92.8	93.5	100	98.7	99.0	99.5	99.8
175	100	97.6	94.1	87.9	85.4	100	98.4	98.8	99.3	99.7	100	97.5	94.7	92.6	93.2	100	98.8	99.0	99.5	99.8
180	100	97.6	94.1	87.9	85.3	100	98.4	98.8	99.3	99.7	100	97.6	94.7	92.5	93.2	100	98.8	99.0	99.5	99.8
185	100	97.6	94.2	88.1	85.5	100	98.3	98.8	99.3	99.7	100	97.5	94.7	92.7	93.4	100	98.8	99.0	99.5	99.8
190	100	97.6	94.3	88.2	85.6	100	98.4	98.8	99.3	99.7	100	97.5	94.8	92.8	93.6	100	98.8	99.0	99.5	99.8
195	100	97.6	94.2	88.3	86.0	100	98.4	98.8	99.3	99.7	100	97.5	94.8	92.8	93.5	100	98.8	99.0	99.5	99.8
200	100	97.7	94.2	88.2	85.7	100	98.4	98.8	99.3	99.7	100	97.5	94.7	92.7	93.3	100	98.8	99.0	99.5	99.8

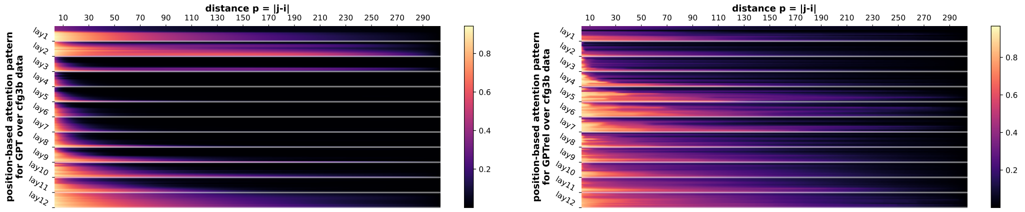
Figure 17: Generative models discover NT ancestors and NT boundaries gradually across training epochs (here 1 epoch equals 500 training steps). CFG levels closer to the leaves are learned faster, and their accuracies continue to increase as deeper levels are being learned, following a principle called "backward feature correction" in deep hierarchical learning (Allen-Zhu & Li, 2023).

E MORE EXPERIMENTS ON ATTENTION PATTERNS

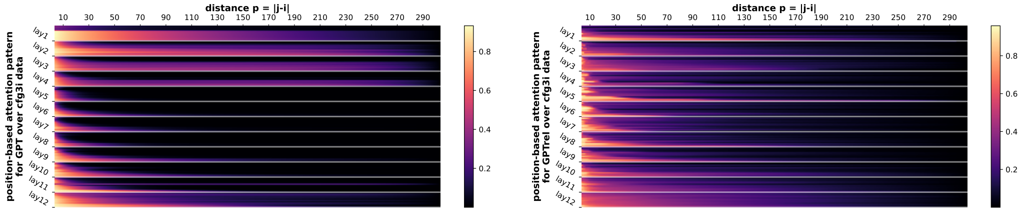
E.1 POSITION-BASED ATTENTION PATTERN

Recall from Section 5.1 that we asserted the transformer's attention weights are primarily influenced by the relative distance of the tokens. This remains true even when *trained on the CFG data* with *absolute positional embedding*. We omitted the details in the main body due to space constraints, but we will provide them now.

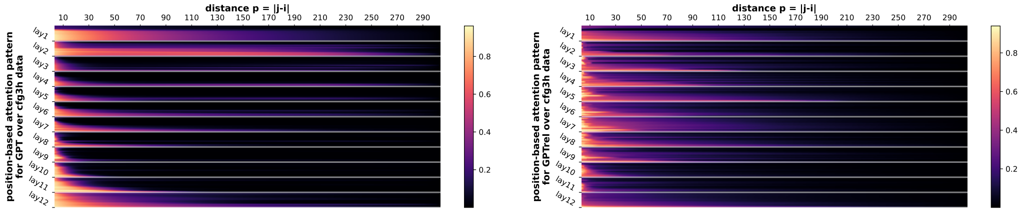
Formally, let $A_{l,h,j\to i}(x)$ for $j \geq i$ represent the attention weight for positions $j \to i$ at layer l and head h of the transformer, on input sequence x. For each layer l, head h, and distance $p \geq 0$, we compute the average of the partial sum $\sum_{1\leq i'\leq i} A_{l,h,j\to i'}(x)$ over all data x and pairs i, j with $j - i = p$. We observe a strong correlation between the attention pattern and the relative distance $p = j - i$. The attention pattern is also *multi-scale*, with some attention heads focusing on shorter distances and others on longer ones. We plot this cumulative sum for different l, h, p in Figure 18 in both `GPT`/`GPT`$_{\text{rel}}$ for various datasets.



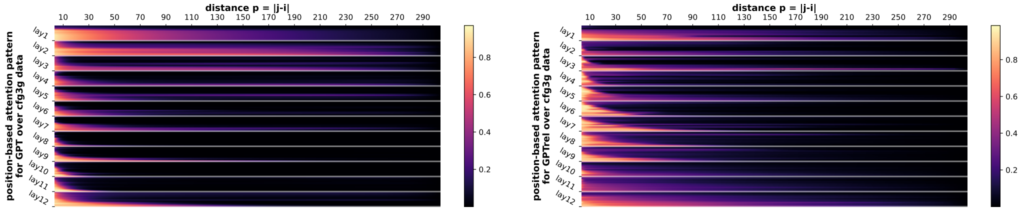
(a) `GPT` on cfg3b (b) $\texttt{GPT}_{\text{rel}}$ on cfg3b



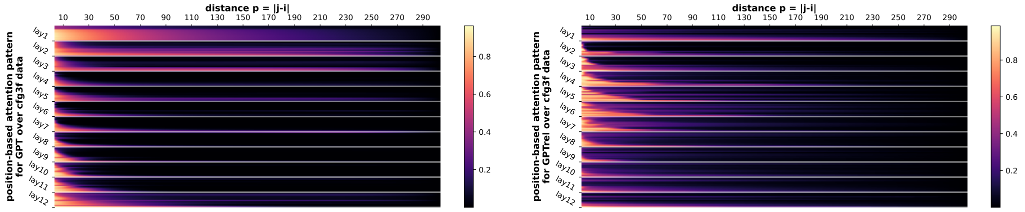
(c) `GPT` on cfg3i (d) $\texttt{GPT}_{\text{rel}}$ on cfg3i



(e) `GPT` on cfg3h (f) $\texttt{GPT}_{\text{rel}}$ on cfg3h



(g) `GPT` on cfg3g (h) $\texttt{GPT}_{\text{rel}}$ on cfg3g

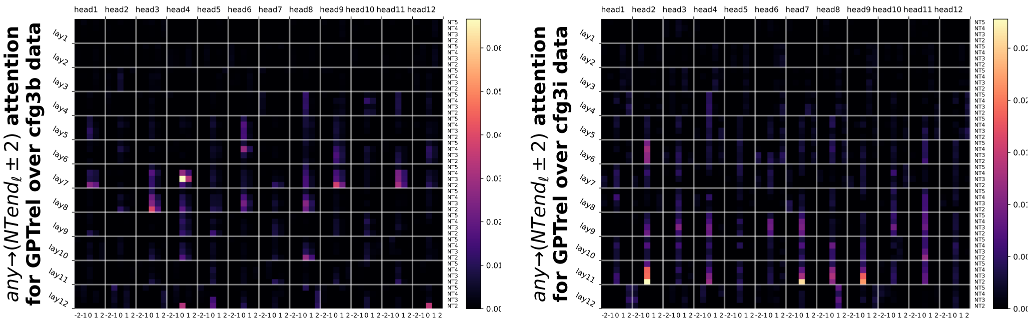


(i) `GPT` on cfg3f (j) $\texttt{GPT}_{\text{rel}}$ on cfg3f

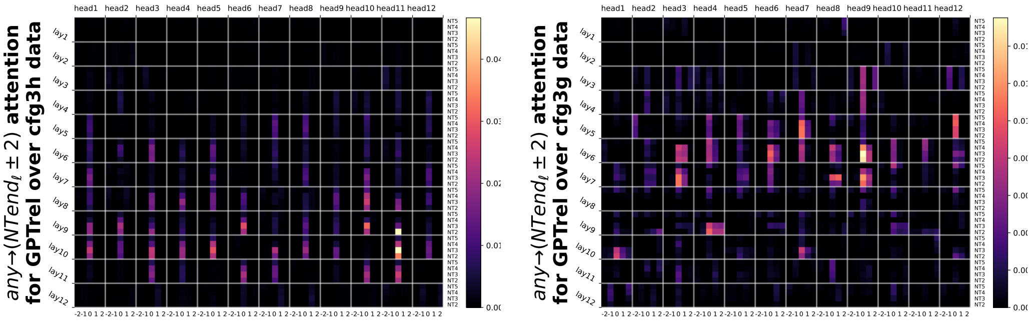
Figure 18: Position-based attention pattern. The 12 rows in each layer represent 12 heads. **Observations.** The attention pattern is multi-scale: different heads or layers have different dependencies on p.

E.2 FROM ANYWHERE TO NT-ENDS

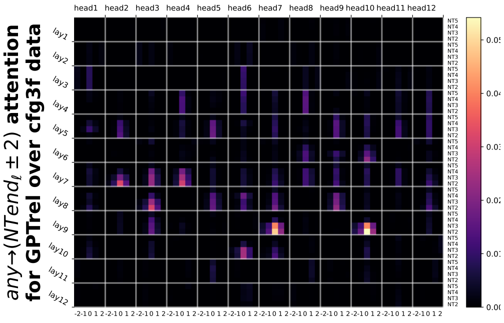
Recall from Figure 6(a), we showed that after removing the position-bias $B_{l,h,j\to i}(x) := A_{l,h,j\to i}(x) - \overline{A}_{l,h,j-i}$, the attention weights have a very strong bias towards *tokens i that are at NT ends*. In Figure 19 we complement this experiment with more datasets.



(a) cfg3b dataset (b) cfg3i dataset



(c) cfg3h dataset (d) cfg3g dataset



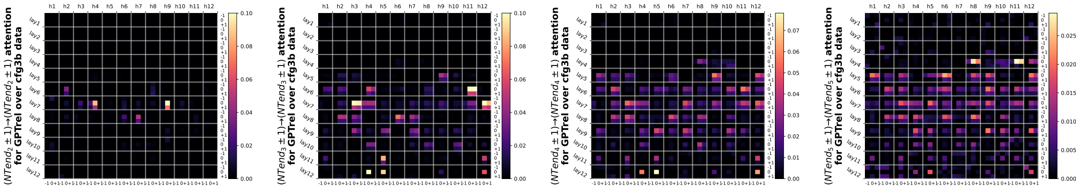
(e) cfg3f dataset

Figure 19: 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 $\delta = -2, -1, 0, +1, +2$.

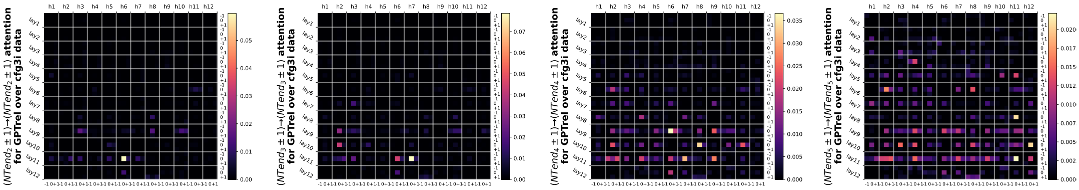
Observation. Attention is largest when $\delta = 0$ and drops rapidly to the surrounding tokens of i.

E.3 FROM NT-ENDS TO NT-ENDS

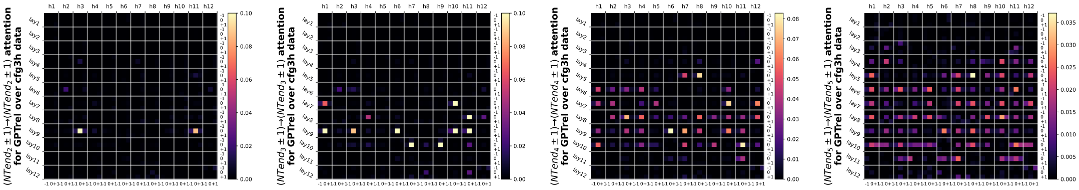
As mentioned in Section 5.2 and Figure 6(b), not only do tokens generally attend more to NT-ends, but among those attentions, *NT-ends* are also *more likely* to attend to NT-ends. We include this full experiment in Figure 20 for every different level $\ell = 2, 3, 4, 5$, between any two pairs $j \to i$ that are both at NT-ends for level ℓ, for the cfg3 datasets.



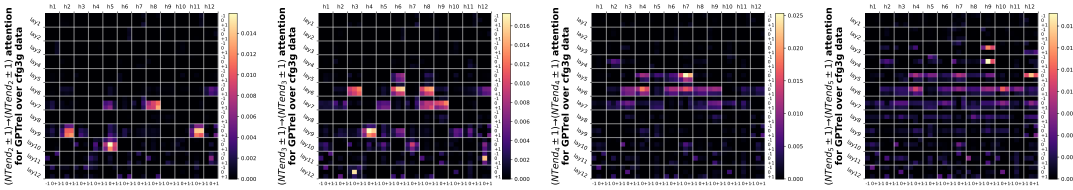
(a) cfg3b at level $\ell = 2$ (b) cfg3b at level $\ell = 3$ (c) cfg3b at level $\ell = 4$ (d) cfg3b at level $\ell = 5$



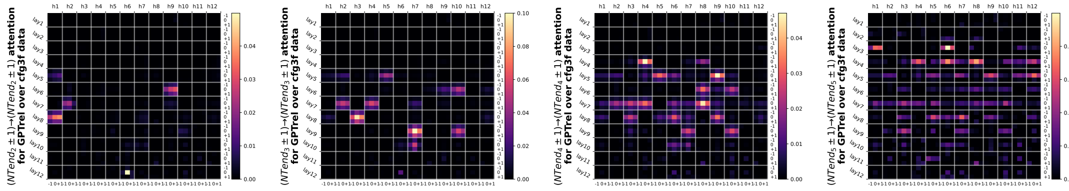
(e) cfg3i at level $\ell = 2$ (f) cfg3i at level $\ell = 3$ (g) cfg3i at level $\ell = 4$ (h) cfg3i at level $\ell = 5$



(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 $\ell = 5$

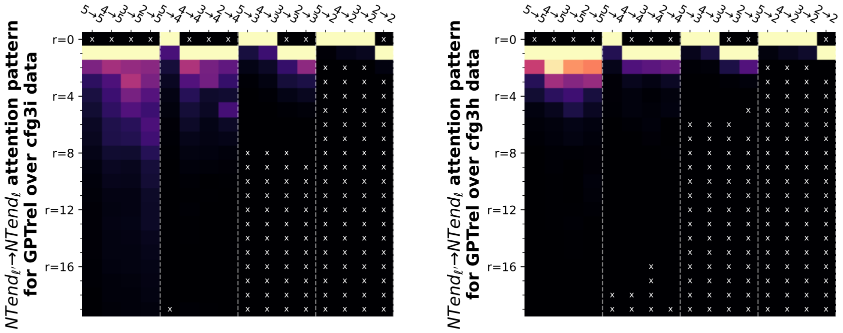
Figure 20: Attention pattern $B_{l,h,j\to i}(x)$ averaged over data x and pairs i, j such that $i + \delta_1$ and $j + \delta_2$ are at the NT-end boundaries in level ℓ of the CFG. In each block, the three rows correspond to $\delta_1 = -1, 0, +1$ and the three columns correspond to $\delta_2 = -1, 0, +1$.

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 15).

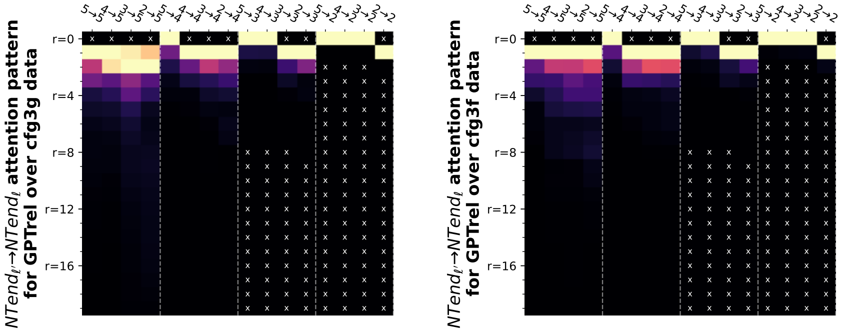
E.4 FROM NT-ENDS TO ADJACENT NT-ENDS

In Figure 6(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 6(c) for a single dataset, and averaged over all the transformer layers. In the full experiment Figure 21 we show that for more datasets, and Figure 22 we show that for individual layers.



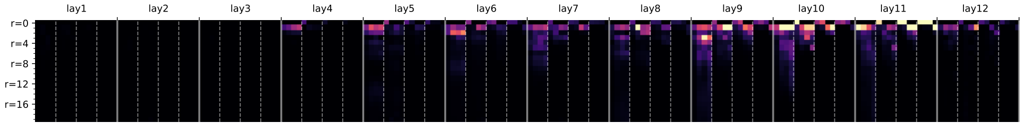
(a) cfg3i (b) cfg3h



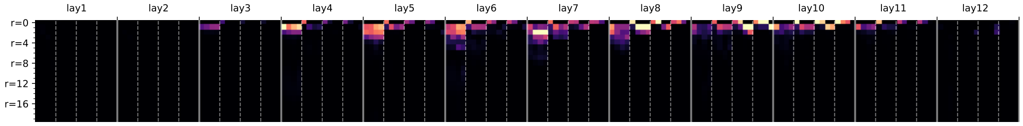
(c) cfg3g (d) cfg3f

Figure 21: 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.

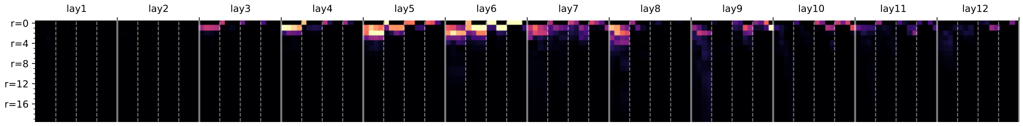
Remark. We present this boundary bias by looking at how close NT boundaries at level ℓ' attend to any other NT boundary at level ℓ. For some distances r, this "distance" that we have defined may be non-existing. For instance, when $\ell \geq \ell'$ one must have $r > 0$. Nevertheless, we see that the 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.



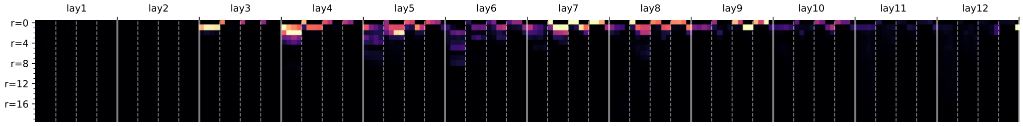
(a) cfg3i



(b) cfg3h



(c) cfg3g



(d) cfg3f

Figure 22: 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 21.

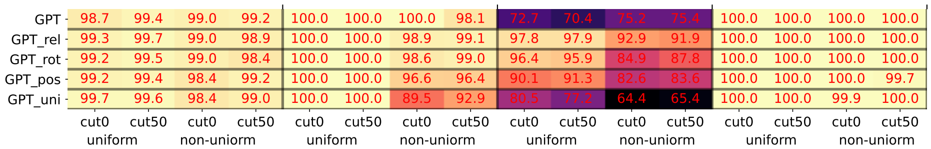
Observation. Different transformer layers are responsible for learning "NT-end to most adjacent NT-end" at different CFG levels.

F MORE EXPERIMENTS ON IMPLICIT 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 B.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 23, where in each cell we evaluate accuracy using 2000 generation samples. We also present the correlation matrix of the word embedding layer in Figure 7 for the $\texttt{GPT}_{\text{rel}}$ model (the correlation will be similar if we use other models).

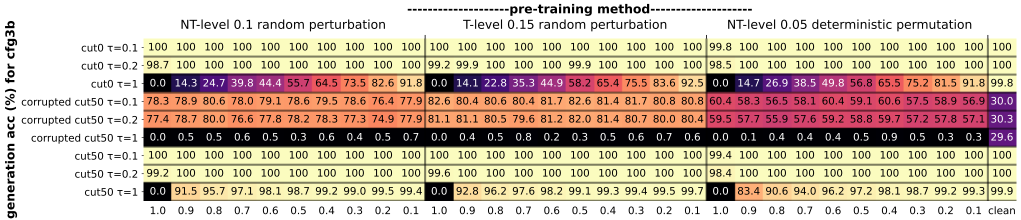


	disjoint |vocab|=90				disjoint |vocab|=300				overlap |vocab|=90				overlap |vocab|=300			
	uniform		non-uniorm		uniform		non-uniorm		uniform		non-uniorm		uniform		non-uniorm	
	cut0	cut50	cut0	cut50	cut0	cut50	cut0	cut50	cut0	cut50	cut0	cut50	cut0	cut50	cut0	cut50
GPT	98.7	99.4	99.0	99.2	100.0	100.0	100.0	98.1	72.7	70.4	75.2	75.4	100.0	100.0	100.0	100.0
GPT_rel	99.3	99.7	99.0	98.9	100.0	100.0	98.9	99.1	97.8	97.9	92.9	91.9	100.0	100.0	100.0	100.0
GPT_rot	99.2	99.5	99.0	98.4	100.0	100.0	98.6	99.0	96.4	95.9	84.9	87.8	100.0	100.0	100.0	100.0
GPT_pos	99.2	99.4	98.4	99.2	100.0	100.0	96.6	96.4	90.1	91.3	82.6	83.6	100.0	100.0	100.0	99.7
GPT_uni	99.7	99.6	98.4	99.0	100.0	100.0	89.5	92.9	80.5	77.2	64.4	65.4	100.0	100.0	99.9	100.0

Figure 23: Generation accuracies on eight implicit CFG variants from pre-trained language models.

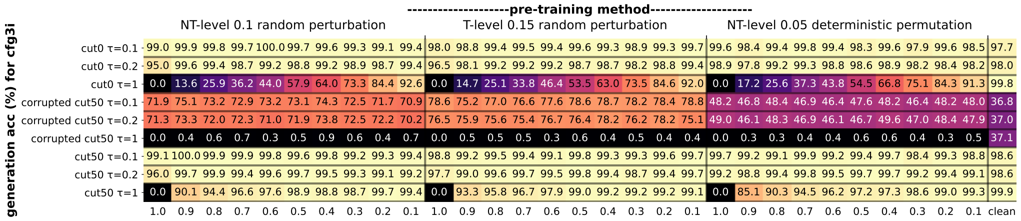
G MORE EXPERIMENTS ON ROBUSTNESS

Recall that in Figure 8, 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 24. For each entry of the figure, we have generated 2000 samples to evaluate the generation accuracy.



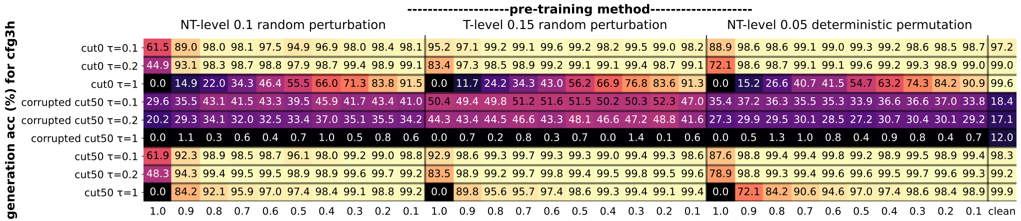
	NT-level 0.1 random perturbation										T-level 0.15 random perturbation										NT-level 0.05 deterministic permutation										
	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	clean
cut0 τ=0.1	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	99.8	100	100	100	100	100	100	100	100	100	100
cut0 τ=0.2	98.7	100	100	100	100	100	100	100	100	100	99.2	99.9	100	100	100	99.9	100	100	100	100	98.5	100	100	100	100	100	100	100	100	100	100
cut0 τ=1	0.0	14.3	24.7	39.8	44.4	55.7	64.5	73.5	82.6	91.8	0.0	14.1	22.8	35.3	44.9	58.2	65.4	75.5	83.6	92.5	0.0	14.7	26.9	38.5	49.8	56.8	65.5	75.2	81.5	91.8	99.8
corrupted cut50 τ=0.1	78.3	78.9	80.6	78.0	79.1	78.6	79.5	78.6	76.4	77.9	82.6	80.4	80.6	80.4	81.7	82.6	81.4	81.7	80.8	80.8	60.4	58.3	56.5	58.1	60.4	59.1	60.6	57.5	58.9	56.9	30.0
corrupted cut50 τ=0.2	77.4	78.7	80.0	76.6	77.8	78.2	78.3	77.3	74.9	77.9	81.1	81.1	80.5	79.6	81.2	82.0	81.4	80.7	80.0	80.4	59.5	57.7	55.9	57.6	59.2	58.8	59.7	57.2	57.8	57.1	30.3
corrupted cut50 τ=1	0.0	0.5	0.5	0.6	0.5	0.3	0.6	0.4	0.5	0.7	0.0	0.4	0.5	0.8	0.2	0.3	0.5	0.6	0.7	0.6	0.0	0.1	0.4	0.4	0.4	0.5	0.9	0.5	0.3	0.3	29.6
cut50 τ=0.1	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	99.4	100	100	100	100	100	100	100	100	100	100
cut50 τ=0.2	99.2	100	100	100	100	100	100	100	100	100	99.6	100	100	100	100	100	100	100	100	100	98.4	100	100	100	100	100	100	100	100	100	100
cut50 τ=1	0.0	91.5	95.7	97.1	98.1	98.7	99.2	99.0	99.5	99.4	0.0	92.8	96.2	97.6	98.2	99.1	99.3	99.4	99.5	99.7	0.0	83.4	90.6	94.0	96.2	97.2	98.1	98.7	99.2	99.3	99.9

(a) cfg3b dataset



	NT-level 0.1 random perturbation										T-level 0.15 random perturbation										NT-level 0.05 deterministic permutation										
	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	clean
cut0 τ=0.1	99.0	99.9	99.8	99.7	100.0	99.7	99.6	99.3	99.1	99.4	98.0	98.8	99.4	99.5	99.4	99.6	99.3	98.9	99.3	99.7	99.6	98.4	99.4	99.8	99.4	98.3	99.6	97.9	99.6	98.5	97.7
cut0 τ=0.2	95.0	99.6	99.4	98.7	99.2	98.8	99.2	98.9	98.7	99.4	96.5	98.1	99.2	99.2	99.2	98.7	98.7	98.2	98.8	99.4	98.9	97.8	99.2	99.3	98.8	98.6	98.9	98.2	98.4	98.2	98.0
cut0 τ=1	0.0	13.6	25.9	36.2	44.0	57.9	64.0	73.3	84.4	92.6	0.0	14.7	25.1	33.8	46.4	53.5	63.0	73.5	84.6	92.0	0.0	17.2	25.6	37.3	43.8	54.5	66.8	75.1	84.3	91.3	99.8
corrupted cut50 τ=0.1	71.9	75.1	73.2	72.9	73.2	73.1	74.3	72.5	71.7	70.9	78.6	75.2	77.0	76.6	77.6	78.6	78.7	78.2	78.4	78.8	48.2	46.8	48.4	46.9	46.4	47.6	48.2	46.4	48.2	48.0	36.8
corrupted cut50 τ=0.2	71.3	73.3	72.0	72.3	71.0	71.9	73.8	72.5	72.2	70.2	76.5	75.9	75.6	75.4	76.7	76.4	78.2	76.2	78.2	75.1	49.0	46.1	48.3	46.9	46.1	46.7	49.6	47.0	48.4	47.9	37.0
corrupted cut50 τ=1	0.0	0.4	0.6	0.7	0.3	0.5	0.9	0.6	0.4	0.7	0.0	0.5	0.5	0.5	0.3	0.6	0.4	0.5	0.4	0.4	0.0	0.3	0.3	0.4	0.4	0.6	0.6	0.4	0.3	0.5	37.1
cut50 τ=0.1	99.1	100.0	99.9	99.9	99.8	99.6	99.8	99.2	99.3	99.4	98.8	99.2	99.5	99.4	99.1	99.8	99.3	99.3	99.6	99.7	99.7	99.2	99.1	99.9	99.2	99.4	99.7	98.4	99.3	98.8	98.6
cut50 τ=0.2	96.0	99.7	99.9	99.4	99.6	99.7	99.5	99.3	99.1	99.2	97.7	99.0	99.6	99.7	99.5	99.8	99.4	98.7	99.4	99.7	99.2	98.8	99.4	99.8	99.5	99.7	99.7	99.2	99.4	99.1	98.6
cut50 τ=1	0.0	90.1	94.4	96.6	97.6	98.9	98.8	98.7	99.7	99.4	0.0	93.3	95.8	96.7	97.9	99.0	99.2	99.2	99.2	99.1	0.0	85.1	90.3	94.5	96.2	97.2	97.3	98.6	99.0	99.3	99.9

(b) cfg3i dataset

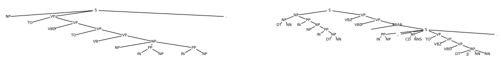


	NT-level 0.1 random perturbation										T-level 0.15 random perturbation										NT-level 0.05 deterministic permutation										
	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	1.0	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1	clean
cut0 τ=0.1	61.5	89.0	98.0	98.1	97.5	94.9	96.9	98.0	98.4	98.1	95.2	97.1	99.2	99.1	99.6	99.2	98.2	99.5	99.0	98.2	88.9	98.6	98.6	99.1	99.0	99.3	99.2	98.6	98.5	98.7	97.2
cut0 τ=0.2	44.9	93.1	98.3	98.7	98.8	97.9	98.7	99.4	98.9	99.1	83.4	97.3	98.5	98.9	99.2	99.1	99.1	99.4	98.7	99.1	72.1	98.6	98.7	99.1	99.1	99.6	99.2	99.3	98.9	99.0	99.0
cut0 τ=1	0.0	14.9	22.0	34.3	46.4	55.5	66.0	71.3	83.8	91.5	0.0	11.7	24.2	34.3	43.0	56.2	66.9	76.8	83.6	91.3	0.0	15.2	26.6	40.7	41.5	54.7	63.2	74.3	84.2	90.9	99.6
corrupted cut50 τ=0.1	29.6	35.5	43.1	41.5	43.3	39.5	45.9	41.7	43.4	41.0	50.4	49.4	49.8	51.2	51.6	51.5	50.2	50.3	52.3	47.0	35.4	37.2	36.3	35.5	35.3	33.9	36.6	36.6	37.0	33.8	18.4
corrupted cut50 τ=0.2	20.2	29.3	34.1	32.0	32.5	33.4	37.0	35.1	35.5	34.2	44.3	43.4	44.5	46.6	43.3	48.1	46.6	47.2	48.8	41.6	27.3	29.9	29.5	30.1	28.5	27.2	30.7	30.4	30.1	29.2	17.1
corrupted cut50 τ=1	0.0	1.1	0.3	0.6	0.4	0.7	1.0	0.5	0.8	0.6	0.0	0.7	0.2	0.8	0.3	0.7	0.0	1.4	0.1	0.6	0.0	0.5	1.3	1.0	0.8	0.4	0.9	0.8	0.4	0.7	12.0
cut50 τ=0.1	61.9	92.3	98.9	98.5	98.7	96.1	98.0	99.2	99.0	98.8	92.9	98.6	99.3	99.7	99.3	99.3	99.0	99.4	99.3	98.6	87.6	98.8	99.4	99.4	99.8	99.2	98.9	99.5	98.9	99.4	98.3
cut50 τ=0.2	48.3	94.3	99.4	99.5	99.5	98.9	98.9	99.6	99.7	99.2	83.5	98.9	99.2	99.7	99.8	99.4	99.8	99.5	99.5	99.6	78.9	98.8	99.3	99.4	99.6	99.6	99.5	99.7	99.6	99.3	99.2
cut50 τ=1	0.0	84.2	92.1	95.9	97.0	97.4	98.4	99.1	98.8	99.2	0.0	89.8	95.6	95.7	97.4	98.6	99.3	99.4	99.1	99.4	0.0	72.1	84.2	90.6	94.6	97.0	97.4	98.6	98.4	98.9	99.9

(c) cfg3h dataset

Figure 24: 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 8 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 8 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 ($\gamma = 1.0$).



(a) the real-life CFG derived from Penn Treebank, short and simple

(b) the cfg3 family we used in the main body of this paper has rule lengths 2 or 3 (cfg3f in this figure)

(c) the cfg8 family has rule lengths 1, 2, or 3 (cfg8e in this figure)

(d) the cfg9 family has rule lengths 1, 2, or 3 (cfg9e in this figure)

(e) the cfg0 family has max-depth 11 and rule lengths 1 or 2 (cfg0e in this figure)

Figure 25: CFG comparisons: *left* is a medium-length sample and *right* is a 80%-percentile-length sample

H Beyond the CFG3 Data Family

The primary focus of this paper is on the cfg3 data family, introduced in Section B.1. This paper does not delve into how GPTs parse English or other natural languages. In fact, our CFGs are more "difficult" than, for instance, the English CFGs derived from the Penn TreeBank (PTB) (Marcus et al., 1993). By "difficult", we refer to the ease with which a human can parse them. For example, in the PTB CFG, if one encounters `RB JJ` or `JJ PP` consecutively, their parent must be `ADJP`. In contrast, given a string

3322131233121131232113223123121112132113223113113223331231211121311331121321213333312322121312322211112133221311311311
3111111323123313313331133133333223121131112122111121123331233112111331333333112333313111133331211321131212113333321211
1121213223223322133221113221132323313111213223223221211133333112132222133221121213312133133221221322121121333123223 3312

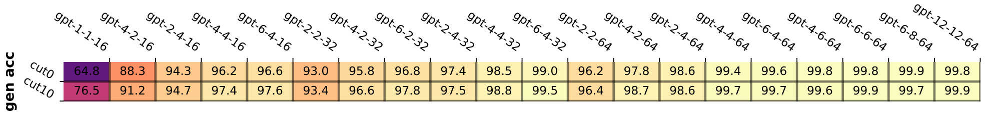
that is in cfg3f, even with all the CFG rules provided, one would likely need a large piece of scratch paper to perform dynamic programming by hand to determine the CFG tree used to generate it.

Generally, the difficulty of CFGs scales with the average length of the strings. For instance, the average length of a CFG in our cfg3 family is over 200, whereas in the English Penn Treebank (PTB), it is only 28. However, the difficulty of CFGs may *inversely scale* with the number of Non-Terminal/Terminal (NT/T) symbols. Having an excess of NT/T symbols can simplify the parsing of the string using a greedy approach (recall the `RB JJ` or `JJ PP` examples mentioned earlier). This is why we minimized the number of NT/T symbols per level in our cfg3b, cfg3i, cfg3h, cfg3g, cfg3f construction. For comparison, we also considered cfg3e1, cfg3e2, which have many NT/T symbols per level. Figure 3 shows that such CFGs are extremely easy to learn.

To broaden the scope of this paper, we also briefly present results for some other CFGs. We include the *real-life* CFG derived from the Penn Treebank, and *three new families* of synthetic CFGs (cfg8, cfg9, cfg0). Examples from these are provided in Figure 25 to allow readers to quickly compare their difficulty levels.

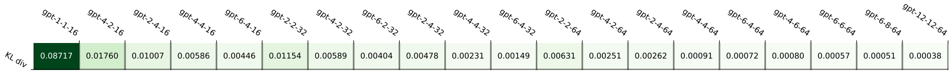
H.1 The Penn TreeBank CFG

We derive the English CFG from the Penn TreeBank (PTB) dataset (Marcus et al., 1993). To make our experiment run faster, we have removed all the CFG rules that have appeared fewer than 50 times



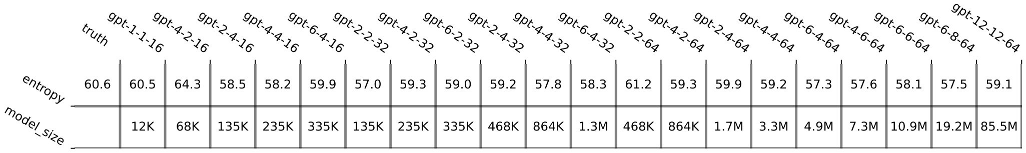
gen acc	gpt-1-1-16	gpt-4-2-16	gpt-2-4-16	gpt-4-4-16	gpt-6-4-16	gpt-2-2-32	gpt-4-2-32	gpt-6-2-32	gpt-2-4-32	gpt-4-4-32	gpt-6-4-32	gpt-2-2-64	gpt-4-2-64	gpt-2-4-64	gpt-4-4-64	gpt-6-4-64	gpt-4-6-64	gpt-6-6-64	gpt-6-8-64	gpt-12-12-64
cut0	64.8	88.3	94.3	96.2	96.6	93.0	95.8	96.8	97.4	98.5	99.0	96.2	97.8	98.6	99.4	99.6	99.8	99.8	99.9	99.8
cut10	76.5	91.2	94.7	97.4	97.6	93.4	96.6	97.8	97.5	98.8	99.5	96.4	98.7	98.6	99.7	99.7	99.6	99.9	99.7	99.9

(a) generation accuracies for cuts $c = 0$ and $c = 10$



	gpt-1-1-16	gpt-4-2-16	gpt-2-4-16	gpt-4-4-16	gpt-6-4-16	gpt-2-2-32	gpt-4-2-32	gpt-6-2-32	gpt-2-4-32	gpt-4-4-32	gpt-6-4-32	gpt-2-2-64	gpt-4-2-64	gpt-2-4-64	gpt-4-4-64	gpt-6-4-64	gpt-4-6-64	gpt-6-6-64	gpt-6-8-64	gpt-12-12-64
KL div	0.08717	0.01760	0.01007	0.00586	0.00446	0.01154	0.00589	0.00404	0.00478	0.00231	0.00149	0.00631	0.00251	0.00262	0.00091	0.00072	0.00080	0.00057	0.00051	0.00038

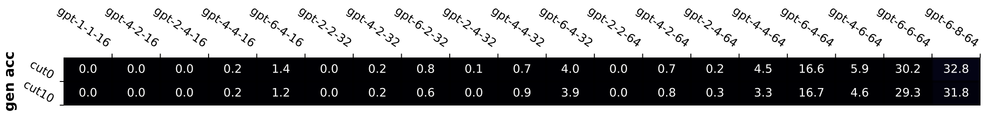
(b) KL-divergence



	truth	gpt-1-1-16	gpt-4-2-16	gpt-2-4-16	gpt-4-4-16	gpt-6-4-16	gpt-2-2-32	gpt-4-2-32	gpt-6-2-32	gpt-2-4-32	gpt-4-4-32	gpt-6-4-32	gpt-2-2-64	gpt-4-2-64	gpt-2-4-64	gpt-4-4-64	gpt-6-4-64	gpt-4-6-64	gpt-6-6-64	gpt-6-8-64	gpt-12-12-64
entropy	60.6	60.5	64.3	58.5	58.2	59.9	57.0	59.3	59.0	59.2	57.8	58.3	61.2	59.3	59.9	59.2	57.3	57.6	58.1	57.5	59.1
model_size		12K	68K	135K	235K	335K	135K	235K	335K	468K	864K	1.3M	468K	864K	1.7M	3.3M	4.9M	7.3M	10.9M	19.2M	85.5M

(c) entropy and model size

Figure 26: Real-life PTB CFG learned by $\texttt{GPT}_{\text{rot}}$ of different model sizes.



gen acc	gpt-1-1-16	gpt-4-2-16	gpt-2-4-16	gpt-4-4-16	gpt-6-4-16	gpt-2-2-32	gpt-4-2-32	gpt-6-2-32	gpt-2-4-32	gpt-4-4-32	gpt-6-4-32	gpt-2-2-64	gpt-4-2-64	gpt-2-4-64	gpt-4-4-64	gpt-6-4-64	gpt-4-6-64	gpt-6-6-64	gpt-6-8-64
cut0	0.0	0.0	0.0	0.2	1.4	0.0	0.2	0.8	0.1	0.7	4.0	0.0	0.7	0.2	4.5	16.6	5.9	30.2	32.8
cut10	0.0	0.0	0.0	0.2	1.2	0.0	0.2	0.6	0.0	0.9	3.9	0.0	0.8	0.3	3.3	16.7	4.6	29.3	31.8

Figure 27: By contrast, small $\texttt{GPT}_{\text{rot}}$ model sizes cannot learn the cfg3f data (generation accuracies).

in the data.[15] This results in 44 T+NT symbols and 156 CFG rules. The maximum node degree is 65 (for the non-terminal `NP`) and the maximum CFG rule length is 7 (for `S -> `` S , '' NP VP .`). If one performs binarization (to ensure all the CFG rules have a maximum length of 2), this results in 132 T+NT symbols and 288 rules.

Remark H.1. Following the notion of this paper, we treat those symbols such as `NNS` (common noun, plural), `NN` (common noun, singular) as *terminal symbols*. If one wishes to also take into consideration the bag of words (such as the word vocabulary of plural nouns), we have called it *implicit CFG* and studied it in Section A.1. In short, adding bag of words does not increase the learning difficult of a CFG; the (possibly overlapping) vocabulary words will be simply encoded in the embedding layer of a transformer.

For this PTB CFG, we also consider transformers of sizes *smaller* than GPT2-small. Recall GPT2-small has 12 layers, 12 heads, and 64 dimensions for each head. More generally, we let GPT-ℓ-h-d denote an ℓ-layer, h-head, d-dim-per-head $\texttt{GPT}_{\text{rot}}$ (so GPT2-small can be written as GPT-12-12-64).

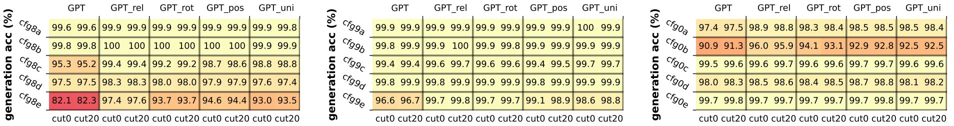
We use transformers of different sizes to pretrain on this PTB CFG. We repeat the experiments in Figure 3, that is, we compute the generation accuracy, completion accuracy (with cut $c = 10$), the output entropy and the KL-divergence. We report the findings in Figure 26. In particular:

- Even a 135K-sized GPT2 (GPT-2-4-16) can achieve generation accuracy 96+% and have a KL divergence on the order of 0.01. (Note the PTB CFG has 30 terminal symbols so its KL divergence may appear larger than that of cfg3 in Figure 3.)
- Even a 1.3M-sized GPT2 (GPT-6-4-32) can achieve generation accuracy 99% and have a KL divergence on the order of 0.001.
- Using $M = 10000$ samples, we estimate the entropy of the ground truth PTB CFG is around 60 bits, and the output entropy of those learned transformer models are also on this magnitude.
- By contrast, those small model sizes cannot learn the cfg3f data, see Figure 27.

---

[15] These are a large set of rare rules, each appearing with a probability $\leq 0.2\%$. We are evaluating whether the generated sentence belongs to the CFG, a process that requires CPU-intensive dynamic programming. To make the computation time tractable, we remove the set of rare rules.

Note that cfg3 does not contain rare rules either. Including such rules complicates the CFG learning process, necessitating a larger transformer and extended training time. It also complicates the investigation of a transformer's inner workings if these rare rules are not perfectly learned.



generation acc (%)	GPT		GPT_rel		GPT_rot		GPT_pos		GPT_uni	
	cut0	cut20	cut0	cut20	cut0	cut20	cut0	cut20	cut0	cut20
cfg8a	99.6	99.6	99.9	99.9	99.9	99.9	99.9	99.9	99.9	99.8
cfg8b	99.8	99.8	100	100	100	100	100	100	99.9	99.9
cfg8c	95.3	95.2	99.4	99.4	99.2	99.2	98.7	98.6	98.8	98.8
cfg8d	97.5	97.5	98.3	98.3	98.0	98.0	97.9	97.9	97.6	97.4
cfg8e	82.1	82.3	97.4	97.6	93.7	93.7	94.6	94.4	93.0	93.5

generation acc (%)	GPT		GPT_rel		GPT_rot		GPT_pos		GPT_uni	
	cut0	cut20	cut0	cut20	cut0	cut20	cut0	cut20	cut0	cut20
cfg9a	99.9	99.9	99.9	99.9	99.9	99.9	99.9	99.9	100	99.9
cfg9b	99.8	99.9	99.9	100	99.9	99.8	99.9	99.9	99.9	99.9
cfg9c	99.4	99.4	99.6	99.7	99.6	99.6	99.4	99.5	99.7	99.7
cfg9d	99.8	99.9	99.8	99.9	99.9	99.9	99.8	99.9	99.9	99.9
cfg9e	96.6	96.7	99.7	99.8	99.7	99.7	99.1	98.9	98.6	98.8

generation acc (%)	GPT		GPT_rel		GPT_rot		GPT_pos		GPT_uni	
	cut0	cut20	cut0	cut20	cut0	cut20	cut0	cut20	cut0	cut20
cfg0a	97.4	97.5	98.9	98.8	98.3	98.4	98.5	98.5	98.5	98.4
cfg0b	90.9	91.3	96.0	95.9	94.1	93.1	92.9	92.8	92.5	92.5
cfg0c	99.5	99.6	99.6	99.7	99.6	99.6	99.7	99.7	99.6	99.6
cfg0d	98.0	98.3	98.5	98.6	98.4	98.5	98.7	98.8	98.1	98.2
cfg0e	99.7	99.8	99.7	99.7	99.7	99.7	99.7	99.8	99.7	99.7

Figure 28: Generation accuracies for cfg8/9/0 data family; suggesting our results *also hold for unbalanced trees with len-1 rules.*

H.2 MORE SYNTHETIC CFGS

Remember that the cfg3 family appears "balanced" because all leaves are at the same depth and the non-terminal (NT) symbols at different levels are disjoint. This characteristic aids our investigation into the *inner workings* of a transformer learning such a language. We introduce three new synthetic data families, which we refer to as cfg8/9/0 (each with five datasets, totaling 15 datasets). These are all "unbalanced" CFGs, which support length-1 rules.[16] Specifically, the cfg0 family has a depth of 11 with rules of length 1 or 2, while the cfg8/9 family has depth 7 with rules of length 1/2/3. In all of these families, we demonstrate in Figure 28 that GPT can learn them with a satisfactory level of accuracy.

For this ICLR submission, we have included all the trees used in the supplementary materials. Below, we provide descriptions of how we selected them.

CFG8 family. The cfg8 family consists of five CFGs, namely cfg8a/b/c/d/e. They are constructed similarly to cfg3b/i/h/g/f, with the primary difference being that we sample rule lengths uniformly from $\{1, 2, 3\}$ instead of $\{2, 3\}$. Additionally,

- In cfg8a, we set the degree $|\mathcal{R}(a)| = 2$ for every NT a; we also ensure that in any generation rule, consecutive pairs of terminal/non-terminal symbols are distinct. The size is $(1, 3, 3, 3, 3, 3, 3)$.
- In cfg8b, we set $|\mathcal{R}(a)| = 2$ for every NT a; we remove the distinctness requirement to make the data more challenging than cfg8a. The size is $(1, 3, 3, 3, 3, 3, 3)$.
- In cfg8c, we set $|\mathcal{R}(a)| \in \{2, 3\}$ for every NT a to make the data more challenging than cfg8b. The size is $(1, 3, 3, 3, 3, 3, 3)$.
- In cfg8d, we set $|\mathcal{R}(a)| = 3$ for every NT a. We change the size to $(1, 3, 3, 3, 3, 3, 4)$ because otherwise a random string would be too close (in editing distance) to this language.
- In cfg8e, we set $|\mathcal{R}(a)| \in \{3, 4\}$ for every NT a. We change the size to $(1, 3, 3, 3, 3, 3, 4)$ because otherwise a random string would be too close to this language.

A notable feature of this data family is that, due to the introduction of length-1 rules, a string in this language $L(\mathcal{G})$ may be *globally ambiguous*. This means that there can be multiple ways to parse it by the same CFG, resulting in multiple solutions for its NT ancestor/boundary information *for most symbols*. Therefore, it is not meaningful to perform linear probing on this dataset, as the per-symbol NT information is mostly non-unique.[17]

CFG9 family. Given the ambiguity issues arising from the cfg8 data construction, our goal is to construct an unbalanced and yet challenging CFG data family where the non-terminal (NT) information is mostly unique, thereby enabling linear probing.

To accomplish this, we first adjust the size to $(1, 4, 4, 4, 4, 4, 4)$, then we permit only one NT per layer to have a rule of length 1. We construct five CFGs, denoted as cfg9a/b/c/d/e, and their degree configurations (i.e., $\mathcal{R}(a)$) are identical to those of the cfg8 family. We then employ rejection sampling by generating a few strings from these CFGs and checking if the dynamic programming (DP) solution is unique. If it is not, we continue to generate a new CFG until this condition is met.

Examples from cfg8e are illustrated in Figure 25. We will conduct linear probing experiments on this data family.

---

[16] When a length-1 CFG rule is applied, we can merge the two nodes at different levels, resulting in an "unbalanced" CFG.

[17] In contrast, the cfg3 data family is only *locally* ambiguous, meaning that it is difficult to determine its hidden NT information by locally examining a substring; however, when looking at the entire string as a whole, the NT information per symbol can be uniquely determined with a high probability (if using for instance dynamic programming).

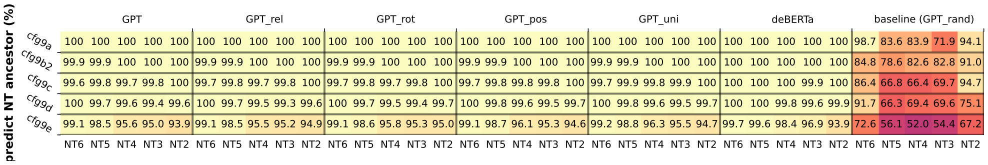


Figure 29: Same as Figure 4 but for the cfg9 family. After pre-training, hidden states of generative models implicitly encode the NT ancestors information. The NT_ℓ column represents the accuracy of predicting $\mathfrak{s}_\ell$, the NT ancestors at level ℓ. This suggests our probing technique applies more broadly.

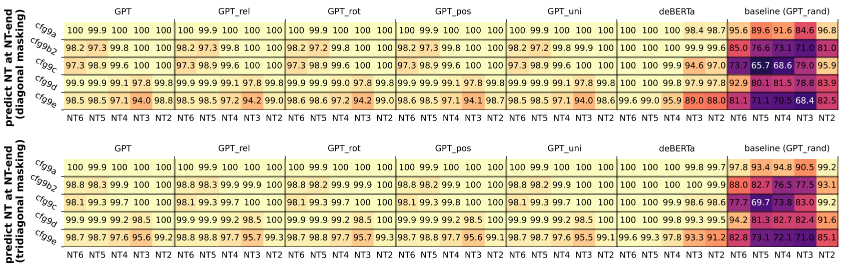


Figure 30: Same as Figure 5 but for the cfg9 data family. Generative pre-trained transformer encodes NT ancestors almost exactly _at_ NT boundaries. The NT_ℓ column represents the accuracy of predicting $\mathfrak{s}_\ell(i)$ at locations i with $\mathfrak{b}_\ell(i) = 1$. This suggests our probing technique applies more broadly.

CFG0 family. Since all the CFGs above support rules of length 3, we have focused on $L = 7$ to prevent the string length from becoming excessively long.[18] In the cfg0 family, we construct five CFGs, denoted as cfg9a/b/c/d/e. All of them have a depth of $L = 11$. Their rule lengths are randomly selected from $\{1, 2\}$ (compared to $\{2, 3\}$ for cfg3 or $\{1, 2, 3\}$ for cfg8/9). Their degree configurations (i.e., $\mathcal{R}(a)$) are identical to those of the cfg8 family. We have chosen their sizes as follows, noting that we have enlarged the sizes as otherwise a random string would be too close to this language:

- We use size $[1, 2, 3, 4, 4, 4, 4, 4, 4, 4, 4]$ for cfg0a/b.
- We use size $[1, 2, 3, 4, 5, 6, 6, 6, 6, 6, 6]$ for cfg0c.
- We use size $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]$ for cfg0d/e.

Once again, the CFGs generated in this manner are globally ambiguous like the cfg8 family, so we cannot perform linear probing on them. However, it would be interesting to demonstrate the ability of transformers to learn such CFGs.

Additional Experiments. We present the generation accuracies (or the complete accuracies for cut $c = 20$) for the three new data families in Figure 28. It is evident that the cfg8/9/0 families can be learned almost perfectly by GPT2-small, especially the relative/rotary embedding ones.

As previously mentioned, the cfg9 data family is predominantly globally ambiguous, making it an excellent synthetic data set for testing the encoding of the NT ancestor/boundary information, similar to what we did in Section 4. Indeed, we replicated all of our probing experiments in Figure 29 and Figure 30. This suggests that **our probing technique has broader applicability.**

---

[18] Naturally, a larger transformer would be capable of solving such CFG learning tasks when the string length exceeds 1000; we have briefly tested this and found it to be true. However, conducting comprehensive experiments of this length would be prohibitively expensive, so we have not included them in this paper.