UNRAVELING SYNTAX: HOW LANGUAGE MODELS LEARN CONTEXT-FREE GRAMMARS

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce a new framework for understanding how language models acquire syntax. While large models achieve impressive results, little is known about their learning dynamics. Our approach starts with the observation that most domains of interest – such as natural language syntax, coding languages, arithmetic problems – are captured by probabilistic context-free grammars (PCFGs). We study the learning dynamics of small models trained on synthetic languages generated from PCFGs, enabling precise control over grammar complexity, recursion depth, and subgrammar structure. We prove several general, recursive formulae for the training loss and Kullback-Leibler divergence over the *subgrammar* structure of a PCFG. Empirically, we find that unlike children– who first master simple substructures before progressing to more complex constructions– transformers reduce loss across all subgrammars in parallel. We further show that subgrammar pretraining can improve the final loss for smaller models, and that pretrained models develop internal representations more aligned with the grammar's substructure. Finally, we demonstrate that models struggle with deeper recursive structures (a limitation even of large language models), revealing fundamental challenges in how neural networks represent hierarchical syntax. Overall, our work initiates the study of the learning dynamics of transformers on PCFGs as a versatile testbed for probing learning in language models, opening a research direction with many open questions.

1 INTRODUCTION

Large language models (LLMs) have stunned the world by achieving sophisticated language abilities in the past few years, yet we still do not know *how* they reach such high levels of performance.

A field of research has emerged that aims to reveal the dynamics of deep neural network training by restricting attention to well-understood hypothesis classes such as polynomials, XOR, or modular counting. In these domains experiments are feasible and theory is tractable and well understood analytically, unlike the space of all images, or even the English language. Language modeling, by contrast, remains poorly understood. Mainly because training high-quality LLMs require hundreds of billions of parameters and billions of dollars to train (Grattafiori et al., 2024; Cottier et al., 2024). Training "own's one" state-of-the-art LLM simply to study its training dynamics is unfeasible. While prior work has analyzed the static behavior of trained models (probing internal representations, in-context learning, etc.), little is known about the process of language acquisition. Do LLMs, for example, master simpler substructures before progressing to more complex syntax, as children do?

Because natural language syntax is largely governed by *context-free grammars* (CFGs), we propose a novel surrogate approach: study how small neural models acquire synthetic CFGs. In addition to feasibility, we can precisely control the complexity of such grammars, including the vocabulary size, sentence length, and arguably most interestingly, the amount or depth of embedding and recursive structure. With this approach, we can ask a stylized version of our question above regarding whether a language model trained on a CFG G first achieves competence on a simpler subgrammar $H \subset G$.

Our contributions are threefold. In Section 4, we establish several fundamental theoretical properties of the CFG substructure and their relationship to learning. We prove that the KL Divergence between a CFG and a trained model decomposes into a sum over its *subgrammars* – a concept we define and

explore rigorously in the same section. This implies that models learn all subgrammars in parallel rather than through stages, a prediction confirmed empirically. In Section 5, show that pretraining on subgrammars can improve final loss for small architectures, while larger models perform similarly with or without pretraining. However, in both cases, we present evidence using Centered Kernel Alignment (CKA) that pretrained models have a more structured representation of subgrammar structure than directly trained models. Finally, in Section 6, we examine generalization to deeper recursion, finding that models handle long contexts at fixed depth but fail sharply when recursion depth increases. We observe a similar behavior in state-of-the art models, which reliably solve long but shallow arithmetic chains yet fail on deeply nested arithmetic expressions. A final contribution will be an open source library for training models on CFGs.

More broadly, this work initiates the theoretical and empirical study of training dynamics over CFGs – a natural, structured, and fundamental class—towards a theory of language learning in neural networks, analogous to those developed for Boolean and arithmetic function families.

2 Related Work

Transformers (Vaswani et al., 2017), and language models more broadly, have been studied in two predominant research directions: improving training methods (Bubeck et al., 2023; Jaech et al., 2024; Guo et al., 2025) and probing trained models to analyze how knowledge is stored and activated during inference (Meng et al., 2022; Geva et al., 2021; Dar et al., 2022; Ferrando & Voita, 2024). Much less is known about how such models acquire language. However, one remarkable study Evanson et al. (2023) showed that GPT-2 displayed developmental stages *reminiscent* of child language learning, from simple subject–verb constructions to wh-questions and relative clauses.

Our work initiates the theoretical and empirical study of the learning dynamics of neural networks on CFGs, a highly structured mathematical formalism, intending to grow a subfield analogous to those for Boolean and arithmetic functions. We approach this problem via the surrogate (and theoretically significant in its own right) approach of studying the dynamics of *language models acquiring formal languages*. Prior families such as juntas, parities, and modular counting have highlighted optimization challenges ranging from variable selection to hierarchical dependencies (Klivans & Kothari, 2014; Telgarsky, 2016; Abbe et al., 2024; Daniely & Malach, 2020). CFGs provide a linguistically motivated setting where recursive structure is explicit, and formal language theory offers a well-developed foundation (e.g. see (Cotterell et al., 2023) for a survey).

Formal languages have been used to test neural models, with mixed success. RNNs and LSTMs often fail to learn subregular grammars despite theoretical capacity (Avcu et al., 2017), and transformers perform well on many formal languages but struggle with recursion and counter-based mechanisms (Bhattamishra et al., 2020). Other studies confirm that transformers often fail on deeply nested grammatical structures (Lampinen, 2024). Results consistently show that gradient descent, rather than model expressivity, is the limiting factor. Similar findings arise for LSTMs, where data distribution and length generalization strongly affect performance (Suzgun et al., 2018).

On the theoretical side, Hahn (2020) established limitations of self-attention in capturing long-range dependencies, even though transformers are known to be Turing-complete (Pérez et al., 2021) and universal approximators of sequence functions (Yun et al., 2019); see Strobl et al. (2024) for a survey. Probing studies have also revealed internal stack-like representations in models trained on counter languages (Tiwari et al., 2025).

3 Preliminaries and Definitions

3.1 Formal Languages

Definition 3.1 (CFG). *A Context-Free Grammar (CFG) is a tuple $G = (\Sigma, \mathcal{N}, \mathcal{S}, \mathcal{P})$ where Σ is a finite set of terminal symbols, $\mathcal{N}$ is a finite set of non-terminal symbols (disjoint from Σ), $S \in \mathcal{N}$ is the designated start symbol, and $\mathcal{P}$ is a finite set of production rules of the form*

$$A \to \alpha$$

where $A \in \mathcal{N}$ and $\alpha \in (\mathcal{N} \cup \Sigma)^*$ is a string of terminals and non-terminals (α can be the empty string which we designate with ϵ).

The language $L_G \subseteq \Sigma^*$ associated with a CFG G is the set of all strings over the terminals that can be derived from S via successive applications of rules in $\mathcal{P}$. A language generated by a CFG is a Context-Free Language *(CFL).*

Definition 3.2 (PCFG)**.** *A Probabilistic Context-Free Grammar (PCFG) is a context-free grammar $G = (\Sigma, \mathcal{N}, \mathcal{S}, \mathcal{P})$ augmented with a probability function $\mathcal{W}$ that assigns to each rule $(A \to \alpha) \in \mathcal{P}$ a non-negative probability $\mathcal{W}(A \to \alpha)$ such that for each $A \in \mathcal{N}$, $\sum_{\{(A\to\alpha)\ \in\ \mathcal{P}\}} \mathcal{W}(A \to \alpha) = 1$.*

Brief history. CFGs were originally defined in the context of linguistics (Chomsky, 1956), as the vast majority of the syntax of natural languages, as well as the syntax of programming languages and mathematics, are CFGs (Shieber, 1985; Pullum & Gazdar, 1982). CFGs occupy a position of intermediate complexity in the "Chomsky hierarchy" of computational models, strictly stronger than the finite-state automata which compute the regular languages, and weaker than the Turing machine, which can compute (or recognize) any language that is computable[1]. Since CFGs capture languages with recursion and embedded structure, there intuitively exists a notion of a *sub*grammar within a grammar. However, several subtleties crop up when attempting to define a subgrammar. There are at least two interesting definitions: one of *substrings* of CFG sentences that can be generated from a non-terminal, and the other as a subset of the CFG language generated by a subset of the rules. We term these *inner* and *outer* subgrammars respectively. We will sometimes say *supergrammar* for a bigger grammar containing a subgrammar.

Definition 3.3 (Inner Subgrammar)**.** *An inner subgrammar of a PCFG $G = (\Sigma, \mathcal{N}, \mathcal{S}, \mathcal{P}, \mathcal{W})$ is itself a PCFG $G' = (\Sigma', \mathcal{N}', \mathcal{S}', \mathcal{P}', \mathcal{W}')$ such that $\Sigma' \subseteq \Sigma$, $\mathcal{N}' \subseteq \mathcal{N}$, $\mathcal{S}' \in \mathcal{N}'$ is the start symbol of the subgrammar, and $\mathcal{P}'$ is the set of all rules with non-terminals in $\mathcal{N}'$. Finally, W' is the restriction of $\mathcal{W}$ to $\mathcal{P}'$, renormalized so that for every $A \in \mathcal{N}'$, $\sum_{\{(A\to\alpha)\in\mathcal{P}'\}} \mathcal{W}'(A \to \alpha) = 1$.*

Definition 3.4 (Proper Subgrammar)**.** *A* proper *subgrammar is an inner subgrammar G' of a CFG G which does not contain G itself.*

Definition 3.5 (Outer Subgrammar)**.** *An outer subgrammar of a PCFG $(\Sigma, \mathcal{N}, \mathcal{S}, \mathcal{P}, \mathcal{W})$ is a PCFG $G' = (\Sigma', \mathcal{N}', \mathcal{S}, \mathcal{P}', \mathcal{W}')$, with $\Sigma' \subseteq \Sigma$, $\mathcal{N}' \subseteq \mathcal{N}$, $\mathcal{P}' \subseteq \mathcal{P}$, and W' is the renormalized restriction of $\mathcal{W}$ to $\mathcal{P}'$. In particular, to be a valid outer subgrammar, $\mathcal{P}'$ must contain at least one rule from $\mathcal{P}$ where the left-hand side is S, and for each of its non-terminals.*

An outer subgrammar captures the notion of a subset of the *language* generated by a PCFG obtained by keeping a subset of expansions of various non-terminals (starting from S). Every string generated by an outer subgrammar is a valid string of the supergrammar. An outer subgrammar more closely corresponds to the notion of a "simple" version of a language– for instance, how children produce language during acquisition, whereas inner subgrammars are the inherent *compositional* substructures of a CFG.

3.2 Language Modeling

In this work, all distributions are assumed to be over strings of a finite alphabet Σ, although many of the definitions apply to arbitrary domains.

Definition 3.6 (Kullback-Leibler Divergence)**.** *Given distributions P and Q over Σ^*, the Kullback-Leibler (KL) Divergence of Q from P is*

$$\mathrm{KL}(P \parallel Q) = \sum_{s \in \Sigma^*} P(s) \log \frac{P(s)}{Q(s)}$$

A *language model* Q_θ is a function family parametrized by θ, such that $Q_\theta(x)$ yields a probability distribution over $x \in \Sigma^*$. In this work one can think of all Q_θ as auto-regressive (though for several theoretical results this is not strictly necessary), meaning Q_θ explicitly models the next token distribution, and $Q_\theta(x_1, \ldots, x_n) = \Pi_{i=1}^{n} Q_\theta(x_i | x_1, \ldots, x_{i-1})$.

In Language Modeling, Q_θ is optimized with Maximum Likelihood Estimation:

[1] If one accepts the Church-Turing thesis, which states that any physically-realizable computational system can be simulated by a Turing Machine.

Definition 3.7 (Maximum Likelihood Estimation)**.** *Given a model family Q_θ and target distribution P, the Maximum Likelihood Estimator $Q_{\hat{\theta}}$ is parametrized by*

$$\hat{\theta} = \arg\max_\theta \mathcal{L}(\theta)$$

where

$$\mathcal{L}(\theta) = \mathbb{E}_{s\sim P}\left[-\log Q_\theta(s)\right]$$

Practically, this is done by maximizing the combined likelihood under Q_θ of a set of samples, or equivalently (by monotonicity of $\log$) minimizing the sum of negative log-likelihoods; in the limit, this exactly approaches $\mathcal{L}(\theta)$.

Definition 3.8 (Shannon Entropy)**.** *The Shannon Entropy of a probability distribution is*

$$H(P) = \mathbb{E}_{s\sim P}\left[\log P(s)\right]$$

Proposition 3.9. *Given a true distribution P and model Q_θ parametrized by θ,*

$$\mathcal{L}(\theta) = D_{\mathrm{KL}}(P \parallel Q_\theta) + H(P)$$

The proof (given in the Appendix A) is a straightforward application of the linearity of expectation. The theorem states that loss of a model equals its KL-divergence from the true distribution, plus an entropy term that depends only on the underlying distribution (i.e. is independent of θ). In particular this implies that $\hat{\theta}$ minimizes θ if and only if it minimizes $D_{\mathrm{KL}}(P \parallel Q_\theta)$.

4 Loss and the Compositionality of Context-Free Grammars

4.1 Decomposition of PCFG into subgrammars

Theorem 4.1 (Unique decomposition of PCFG into inner subgrammars)**.** *Every (probabilistic) context-free grammar G can be uniquely decomposed into a hierarchy of its inner subgrammars.*

This hierarchical structure can be represented as a directed acyclic graph (DAG) with self-loops (that is, the graph is acyclic except *that edges from a node v to itself are permitted). Each node is labeled by the set of non-terminals that generate the corresponding subgrammar.*

The proof recursively constructs the DAG by first identifying the "top-level" subgrammars of G; see Appendix A. While to our knowledge, the theorem in this particular formulation is our own, the nodes of the DAG decomposition correspond to the "grammatical levels" of a CFG in Gruska's classical work on CFG theory Gruska (1971).

4.2 Subgrammar structure and Language Modeling

We now study the connection between the subgrammar structure of CFGs and training language models on the corresponding CFL. Let $G = (\Sigma, \mathcal{N}, S, \mathcal{P}, \mathcal{W})$ be a PCFG that induces a distribution P_G over Σ^*, and Q_θ an autoregressive language model trained to approximate P_G (that is, given a partial sentence over Σ^* it outputs a terminal, or EOS).

We first consider a very simple case, where the only expansion of S is $S \to \alpha A \beta$, where A is some proper subgrammar (does not generate S), and $\alpha, \beta \in \Sigma^*$ are strings of terminals:

$$\begin{aligned}
D_{\mathrm{KL}}(P \parallel Q) &= \sum_{a\in\Sigma^*} P_G(\alpha a\beta)\log\frac{P_G(\alpha a\beta)}{Q_\theta(\alpha a\beta)} \\
&= \sum_{a\in\Sigma^*} P_G(\alpha a\beta)[\log P_G(\alpha\dots) + \log P_G(a|\alpha) + \log P_G(\beta|a\alpha) - \log Q_\theta(\alpha\dots) \\
&\qquad + \log Q_\theta(a|\alpha) + \log Q_\theta(\beta|a\alpha)] \\
&= \frac{\log P_G(\alpha\dots)}{\log Q_\theta(\alpha\dots)} + \sum_a P_A(a)\frac{\log P_A(a)}{\log Q_\theta(a|\alpha)} + \sum_a P(a)\frac{\log P_G(\beta|\alpha a)}{\log Q_\theta(\beta|\alpha a)}
\end{aligned}$$

In an abuse of notation, above $P_G(\alpha|\epsilon)$ denotes the probability of a partial sequence beginning with α, $P_G(\alpha)$ the probability of a following α (in a partial sequence), and so on; similarly $Q_\theta(\alpha|\epsilon)$ is the Q_θ outputs the prefix α (starting with the empty context), etc. The decomposition of P_G and Q_θ in the second line follows from the subgrammar structure of G in the case of P_G, and from the fact that Q_θ is an autoregressive model (generating from left to right) for the Q_θ terms. In short, *the KL-divergence evaluates to a sum of conditioned KL-divergences* corresponding to the subgrammar A, of prefix α, and suffix β. The latter can themselves be thought of as simple subgrammars; indeed, we can rewrite G to include two new non-terminals that evaluate to α and β respectively (with prob. 1), and we would then have a sum over three "sub"-divergences.

Definition 4.2. *Given PCFG distribution P_G and arbitrary distribution Q over Σ^*, and top-level subgrammar A of G, we denote by*

$$D_{\mathrm{KL}}(P_G \parallel Q)_A = \sum_{s\in\Sigma^*} P(s|\epsilon)P_G(A|s) \sum_{a\in\Sigma^*} D_{\mathrm{KL}}(P_A \parallel Q(\cdot|s))$$

That is, $D_{\mathrm{KL}}(R \parallel Q)_A$ can be seen as the "restriction" of the KL-divergence to substrings from the subgrammar A (by summing over all contexts that can begin A). In the case of a fixed string $\alpha \in \Sigma^*$ we will write $D_{\mathrm{KL}}(P_G \parallel Q)_\alpha$ where the second sum is replaced with a single term for α (equiv. one can view α as a subgrammar of one string).

Then we have, from our previous example

$$D_{\mathrm{KL}}(P \parallel Q) = D_{\mathrm{KL}}(P \parallel Q)_\alpha + D_{\mathrm{KL}}(P \parallel Q)_A + D_{\mathrm{KL}}(P \parallel Q)_\beta$$

The same decomposition holds more generally. Let the *top-level* subgrammars denote the children of the root node in a CFG's subgrammar decomposition.

Theorem 4.3 (KL loss as a recursive function over subgrammars)**.** *Let G be a PCFG with top-level subgrammars $A_1, \ldots, A_k$. Let $C \subset \Sigma^*$ be the set of (fixed) substrings of terminals that occur between non-terminals of G. Then*

$$D_{\mathrm{KL}}(P_G \parallel Q_\theta) = \sum_{i=1}^{k} D_{\mathrm{KL}}(P_G \parallel Q_\theta)_{A_i} \; + \sum_{\alpha\in C} D_{\mathrm{KL}}(P_G \parallel Q_\theta)_\alpha$$

Corollary 4.4. *If we rewrite G as an equivalent PCFG with additional non-terminals such that S maps to strings only non-terminals (correpsonding to subgrammars $A_1, \ldots, A_k$); then, the right sum of Theorem 4.3 can be removed:*

$$D_{\mathrm{KL}}(P_G \parallel Q_\theta) = \sum_{i=1}^{k} D_{\mathrm{KL}}(P_G \parallel Q_\theta)_{A_i}$$

The full proof of Theorem 4.3 and Corollary 4.4 is in Appendix A. Upon closer inspection, the recursive formula actually applies to any *subgrammar*; that is, for subgrammar A with subgrammars $B_1, \ldots, B_l$, $D_{\mathrm{KL}}(P_G \parallel Q_\theta)_A = \sum_j D_{\mathrm{KL}}(P_G \parallel Q_\theta)_{B_j}$ (indeed, we could have states Theorem 4.3 with respect to subgrammars, as $D_{\mathrm{KL}}(P_G \parallel Q_\theta) = D_{\mathrm{KL}}(P_G \parallel Q_\theta)_G$). Hence, this formula can be expanded recursively over each of the *subgrammars* A_i by repeated applications of the same theorem, resulting in a sum over all the *leaves* of the DAG decomposition of G into its subgrammars; see Corollary A.1 in the Appendix for the precise statement.

Now, suppose each top-level subgrammar A_i occurs with probability p_i over the top-level rules that expand S; it is tempting to conclude that the recursive formula simplifies to $KL(P_G \parallel Q_\theta) = \sum_{i=1}^{k} p_i D_{\mathrm{KL}}(P_A \parallel Q_\theta)$ (where the KL terms are no longer restrictions, but bona-fide divergences between the distribution P_A and Q_θ as a language model for A). However, this works only if Q_θ is excellent and models P_A identically under any context where the subgrammar A can occur, which may not be the case!

Corollary 4.5. *Let G be a PCFG where S evaluates to rules with only non-terminals (correspondingly, subgrammars) $A_1, \ldots, A_k$ each of which occurs with prob. p_i.*

*Assume Q_θ "*understands composition*": for any subgrammar A_i and two contexts s, s' for which $P_G(A|s)P_G(A|s') > 0$, $Q_\theta(A_i|s) = Q_\theta(A_i|s')$ (the restrictions of Q_θ to strings from A_i given*

possible contexts s or s', are identical). Then

$$D_{\mathrm{KL}}(P_G \parallel Q_\theta) = \sum_{i=1}^{k} p_i D_{\mathrm{KL}}(P_{A_i} \parallel Q_\theta(A_i))$$

Where $Q_\theta(A_i) = Q_\theta(A_i|s)$ for arbitrary context s s.t. $P_G(A_i|s) > 0$.

The condition that the language model "understands composition" is strong, but yields a particularly elegant decomposition for the case of a very good language model that models a subgrammar identically wherever it can occur (hence the condition that $P_G(A|s)$ and $P_G(A|s')$ are both positive). In this case, we can say even more about the recurrent structure of KL-divergence. In Theorem 4.3 and its corollaries, any of the top-level subgrammars A_i could have been the grammar G itself (if G has a self-loop). For a language model that understands recursion, we can say even more about the KL-divergence as a function of the *degree* of "self-loopiness", or recursion.

Theorem 4.6 (KL-divergence with expected recurrence)**.** *Let G have* proper *top-level subgrammars $A_1, \ldots, A_k$, each occurring with probability p_k over the rules expanding S, and let Q_θ be a language model for P_G that understands composition.*

Let the recursion *R be the number of times S occurs in the top-level rule chosen to expand S. Then,*

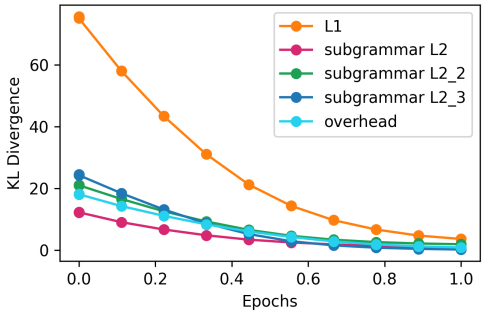
$$D_{\mathrm{KL}}(P_G \parallel Q_\theta) = \frac{\sum_{i=1}^{k} D_{\mathrm{KL}}(P_A \parallel Q_\theta(A_i))}{1 - \mathbb{E}[R]}$$

If $1 - \mathbb{E}[R] < 0$, then the KL-divergence is unbounded *if $D_{\mathrm{KL}}(P_A \parallel Q_\theta(A_i)) > 0$ for any A_i.*

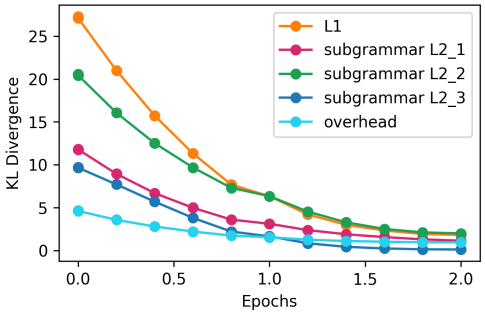
See 4.5 for a full proof. Theorem 4.6 can be seen as the equation for the "base case" in the recursive formula for KL-divergence, since an irreducible (leaf) subgrammar evaluates only to strings of terminals and itself. This equation shows that the expected recursion in such a (sub)grammar must be less than 1 (and the closer it is to 1, the greater the "blow-up" of its divergence to a language model); indeed, if the expected recursion is 1 or greater, the PCFG sampling process that recursively expands the root symbol will in expectation never terminate.

Finally, Theorem A.2 in the Appendix gives a similar additive decomposition for *outer* subgrammars.

To visualize these recurrence relations, we train a small transformer on several synthetic CFGs with varied subgrammar structure, and plot the KL-divergence over training in 4. These plots show visually how, throughout all stages of learning, the KL divergence (loss) is the sum over the corresponding loss for each subgrammar.



(a) A grammar with inner subgrammars, each occurs with 100% probability. Overhead refers to constant strings in between subgrammar roots.



(b) L2_1 and L2_2 occur with 30% probability; L2_3 with 40% probability.

Figure 1: KL-divergence decomposition in a two-layer Transformer. Grammar definitions are given in the appendix.

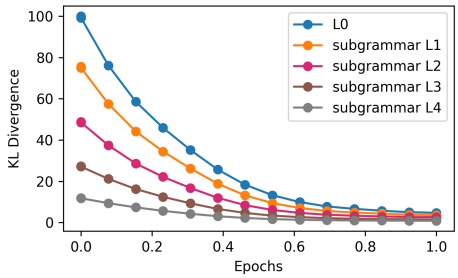
To illustrate Theorem 4.6, consider a simple CFG with two rules:

$$S \to x \;\; (p), \quad S \to (S \text{ and } S) \;\; (1-p)$$

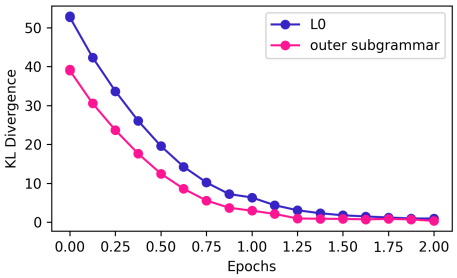
The expected recursion is $\mathbb{E}[R] = 2(1-p)$. Assuming the language model understands composition, we then have that the KL-divergence is $C/(2p-1)$ where C is some constant. We train a small transformer over this language with increasing $p \in (0.5, 1]$, demonstrating qualitatively the non-linear (inverse proportional) growth of KL-divergence as p (the probability of *not* recursing) approaches 0.5). A visual representation can be found in Appendix 4.

4.3 How Language Models learn

This framework enables the construction of explicitly hierarchical grammars, where one might intuitively expect a model to first master a simpler subgrammar before progressing to the encompassing supergrammar. Surprisingly, our experiments reveal the opposite: all subgrammars exhibit simultaneous decreases in loss, even when they are large or shared across many supergrammars and hence central to the grammar as a whole.



(a) Deeper Recursion: a language with an inner subgrammar DAG of depth 4.



(b) KL decomposition for an *outer* subgrammar using most of the rules (see Theorem A.2)

Figure 2: Two examples of the learning behavior of subgrammars.

5 Impact of Pretraining With Subgrammars

While the previous section establishes a mathematical relationship between training loss and subgrammar structure, it is natural to consider whether the structure of CFGs could be exploited in training; e.g. is pretraining on a subgrammar helpful? Perhaps mastering simpler components first facilitates learning of more complex structures later. Such approaches are studied in *curriculum learning* (Bengio et al., 2009; Wang et al., 2021) and modular pretraining strategies (Andreas et al., 2016; Kaiser et al., 2017).

5.1 Robustness to Subgrammar Location

One might expect the choice of subgrammar to influence learning, given the autoregressive nature of transformers. In particular, a *prefix subgrammar*, an inner subgrammar always occurring at the beginning of sequences of G, might be easier to retain, whereas the results from pretraining on a *suffix subgrammar* or an *infix subgrammar* (appearing in the middle and disconnected from sentence endpoints) might be overwritten when training on the full grammar begins. However, our results show this is not the case: the model reliably retains modeling performance on *any* subgrammar, regardless of its position. This robustness is illustrated in Figure 5. As the experiments of the following section suggest, it appears that training on a subgrammar ferries the model into a distinct area of weight space in which the subgrammar is internally represented, and further optimization (on the whole language) remains in this subspace.

5.2 Activation-space analysis

We examine how subgrammar pretraining affects internal representations by comparing models trained from scratch to those pretrained on a subgrammar and then continued on the full grammar. Similarity is measured with Centered Kernel Alignment (CKA) (Kornblith et al., 2019) across 30 random seeds.

Much to our surprise, we also found that for smaller models, subgrammar pretraining can even help achieve a *lower final loss* (Figure 6). This effect diminishes as the model size and representational complexity increase (for instance, this occurs for 2-layer transformers but not 4-layers). As expected, larger models consistently reach lower losses regardless of pretraining.

CKA analysis reveals that pretrained models exhibit *higher alignment across attention layers than models trained from scratch*, both when computed over full-grammar sequences, and (less surprisingly) subgrammar sequences (Table 1). A longer pretraining phase *further increases alignment*, although excessive pretraining can eventually reduce gains in final loss (see same Table).

Why are the pretrained models more "aligned" to one another (that is, represent sequences more similarly?) To probe this, we compare the representational similarity of the top quantile of seeds via cosine similarity of embeddings of three types of sequences: (i) sequences consisting solely of the subgrammar, (ii) sequences with no occurrence of the subgrammar, and (iii) sequences with both the subgrammar and other subsequences. We also compute (iv) the similarity between embedded pairs of a subgrammar sequence and a subgrammar-free sequence. For (i) and (ii), the attention-layers of pretrained models cluster subgrammar sequences (resp. no subgrammar sequences) significantly closer together than directly-trained models. This suggests that substructures learned during pretraining are retained after exposure to the full grammar. Finally, the gap between (iv) and (i), and between (iv) and (ii) is greater in pretrained models, suggesting pretrained models are better at internally segregating sequences with and without subgrammar subsequences (Table 2).

Our experiments are not exhaustive, and we leave open the question of *how to train a model to consistently converge to the best optima*, given the rather strong prior of the subgrammar structure of the target CFG. Too little pretraining may not provide a strong enough inductive bias, while too much may over-specialize the model to the subgrammar and hinder transfer. This trade-off mirrors classical insights from curriculum learning, where an optimal "window" of pretraining exposure exists (Bengio et al., 2009; Weinshall et al., 2018).

	Two-layer Transformer				Four-layer Transformer	
	Pretraining 10 epochs		Pretraining 20 epochs		Pretraining 10 epochs	
	Attention	MLP	Attention	MLP	Attention	MLP
Full grammar sequences						
From Scratch	0.258	0.535	0.249	0.535	0.249	0.469
With Pretraining	0.281	0.534	0.303	0.511	0.323	0.491
Percentage change (%)	*+8.9*	*-0.2*	*+21.7*	*-4.7*	*+8.3*	*+1.0*
Subgrammar sequences						
From Scratch	0.298	0.561	0.288	0.558	0.295	0.513
With Pretraining	0.339	0.566	0.348	0.544	0.347	0.525
Percentage change (%)	*+13.8*	*-0.1*	*+20.8*	*-2.6*	*+10.7*	*+1.9*
Subgrammar pretraining only	0.288	0.558	0.288	0.558	0.295	0.523

Table 1: Average CKA similarity (0–1) across attention and MLP layers of a different Transformers when pretraining for 10 vs. 20 epochs. *Percentage change* indicates the relative difference between models trained from scratch and with pretraining.

6 Generalization: Do LMs "Know Syntax"?

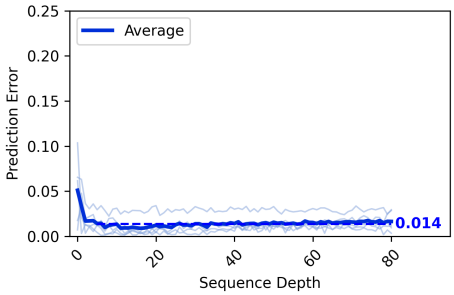
With language models achieving low training loss, it is natural to ask whether they genuinely internalize and can generalize the rules of the PCFG. This question connects to the broader debate about whether language models exhibit intelligence in terms of structure and composition, or whether they are best understood as extraordinarily powerful pattern-matchers.

To probe this, we evaluate a small transformer trained on an especially simple PCFG: `Nested Parentheses` (Appendix C). The model achieves very low loss statistically. We test generalization to probabilistically unlikely (but grammatically valid) sequences with increasing length in two ways: (i) extending the context at the same depth of recursion, feeding in $(a)^i$, and (ii) growing sequences through repeatedly applying the recursive rule, resulting in contexts at increasingly deeper

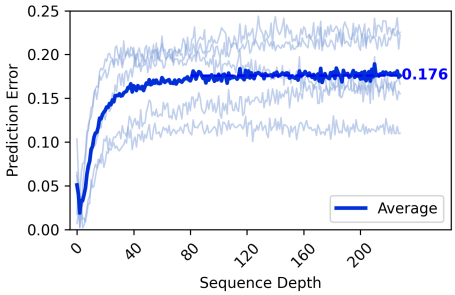
depths of recursion, of the form $)^i$. We then compare the model's output logits (its output distribution) against the ground-truth next-token distribution. The next-token distribution is identical for all test contexts, even between cases (i) and (ii).

Figure 3 shows a striking contrast. For case (i), the prediction error remains low throughout, while for case (ii) it grows as an inverse log curve. While the model appears to master the rules of the PCFG at shallow depth, this does not translate into robust handling of deeper recursive dependencies.

We also test the effect of prepending various valid prefixes before the sequence of increasing depth. Prefixes of type (i), of max depth 1, barely affect the plateau behavior. Prefixes of type (ii), a closed deep subsequence, increase the plateau error by nearly 0.14 – an even larger degradation than that induced by a faulty (non-grammatical) prefix. In other words, valid but deeply recursive prefixes can be more destabilizing to the model than outright malformed structures.



(a) Contexts of the form $(a)^i$ (depth 0)



(b) Contexts of the form $(^i$ (depth i)

Figure 3: LM error vs. longer context, with or without recursion

Anecdotally, we find similar behavior even in state-of-the-art frontier models. We test ChatGPT-5-Instant model on arithmetic expressions generated by a PCFG, presenting two kinds of long expressions: a chain composed of non-deep arithmetic operations, and a single deep arithmetic expression (depth 7)[2]. These experiments show that even LLMs, similar to our small LMs, struggle with depth and not length, correctly answering 5/5 non-deep arithmetic expressions but only 2/5 for a deep arithmetic expression. Note that for the not-deep arithmetic expressions (type 1), the LM in fact has to solve more terms than with the deeper recursion, but still solves them correctly.

7 Discussion

With this work, we initiate the study of the learning dynamics of language models on probabilistic CFGs, both in their own right as a theoretical formalism of interest, and as a testbed for probing the learning dynamics of natural language. Here, we propose several open problems and future directions. First, we conjecture that despite the results of Section 6 *there exists* a setting of the weights of, say, a 2-layer, 2-head transformer (as in our experiments) that *does* correctly model the PCFG (at least up to some very high bound on depth). This would show that gradient descent is not able to find such ideal solutions, analogous to work showing that while neural networks can in principle represent functions like parity, modular counting, or compositional rules, gradient descent often fails to find these solutions without strong inductive bias or curricula (Telgarsky, 2016; Abbe et al., 2024). Just as we considered CFGs, a theory of deep learning dynamics can be developed for other classes in the Chomsky hierarchy, including regular languages, mildly context-sensitive languages, etc. As a first step, how much harder is it for a fixed model architecture to learn synthetic languages from these classes (controlled for average sentence length, vocabulary size, etc)? How does "difficulty of depth" compare to other kinds of dependent structure? Finally, our work does not explore the question of *grammar induction*, the learning task of determining the CFG underlying the input data.

[2]We do not find the same discrepancies for ChatGPT-5-Thinking, which solves all of our examples within 3-4 minutes for each expression. The Thinking model may pass arithmetic expressions to a calculator or program, and/or uses an externally prompted or engineered chain-of-thought process; in any case, this departs from language modeling in the strict sense, and as considered in this work.

REFERENCES

Emmanuel Abbe, Elisabetta Cornacchia, Jan Hazła, and Donald Kougang-Yombi. Learning high-degree parities: The crucial role of the initialization. *arXiv preprint arXiv:2412.04910*, 2024.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 39–48, 2016.

Enes Avcu, Chihiro Shibata, and Jeffrey Heinz. Subregular complexity and deep learning. *arXiv preprint arXiv:1705.05940*, 2017.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 41–48, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585161. doi: 10.1145/1553374.1553380. URL https://doi.org/10.1145/1553374.1553380.

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

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

N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956. doi: 10.1109/TIT.1956.1056813.

Ryan Cotterell, Anej Svete, Clara Meister, Tianyu Liu, and Li Du. Formal aspects of language modeling. *arXiv preprint arXiv:2311.04329*, 2023.

Ben Cottier, Robi Rahman, Loredana Fattorini, Nestor Maslej, Tamay Besiroglu, and David Owen. The rising costs of training frontier ai models. *arXiv preprint arXiv:2405.21015*, 2024.

Amit Daniely and Eran Malach. Learning parities with neural networks. *Advances in Neural Information Processing Systems*, 33:20356–20365, 2020.

Guy Dar, Mor Geva, Ankit Gupta, and Jonathan Berant. Analyzing transformers in embedding space. *arXiv preprint arXiv:2209.02535*, 2022.

Linnea Evanson, Yair Lakretz, and Jean-Rémi King. Language acquisition: do children and language models follow similar learning stages? *arXiv preprint arXiv:2306.03586*, 2023.

Javier Ferrando and Elena Voita. Information flow routes: Automatically interpreting language models at scale. *arXiv preprint arXiv:2403.00824*, 2024.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361, 2021.

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

J. Gruska. Complexity and unambiguity of context-free grammars and languages. *Information and Control*, 18(5):502–519, 1971. ISSN 0019-9958. doi: 10.1016/S0019-9958(71)90519-5. URL https://www.sciencedirect.com/science/article/pii/S0019995871905195.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

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

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Lukasz Kaiser, Aidan N Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. One model to learn them all. *arXiv preprint arXiv:1706.05137*, 2017.

Adam Klivans and Pravesh Kothari. Embedding Hard Learning Problems Into Gaussian Space. In Klaus Jansen, José Rolim, Nikhil R. Devanur, and Cristopher Moore (eds.), *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*, volume 28 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 793–809, Dagstuhl, Germany, 2014. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-939897-74-3. doi: 10.4230/LIPIcs.APPROX-RANDOM.2014.793. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.APPROX-RANDOM.2014.793.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International conference on machine learning*, pp. 3519–3529. PMlR, 2019.

Andrew Lampinen. Can language models handle recursively nested grammatical structures? a case study on comparing models and humans. *Computational Linguistics*, 50(4):1441–1476, 2024.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372, 2022.

Geoffrey K. Pullum and Gerald Gazdar. Natural languages and context-free languages. *Linguistics and Philosophy*, 4(4):471–504, 1982. ISSN 01650157, 15730549. URL http://www.jstor.org/stable/25001071.

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. Attention is turing-complete. *Journal of Machine Learning Research*, 22(75):1–35, 2021. URL http://jmlr.org/papers/v22/20-302.html.

Stuart M. Shieber. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8(3):333–343, 1985. ISSN 01650157, 15730549. URL http://www.jstor.org/stable/25001210.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? a survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, 2024.

Mirac Suzgun, Yonatan Belinkov, and Stuart M Shieber. On evaluating the generalization of lstm models in formal languages. *arXiv preprint arXiv:1811.01001*, 2018.

Matus Telgarsky. Benefits of depth in neural networks. In *Conference on learning theory*, pp. 1517–1539. PMLR, 2016.

Utkarsh Tiwari, Aviral Gupta, and Michael Hahn. Emergent stack representations in modeling counter languages using transformers. *arXiv preprint arXiv:2502.01432*, 2025.

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.

Xin Wang, Yudong Chen, and Wenwu Zhu. A survey on curriculum learning. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):4555–4576, 2021.

Daphna Weinshall, Gad Cohen, and Dan Amir. Curriculum learning by transfer learning: Theory and experiments with deep networks. In *International conference on machine learning*, pp. 5238–5246. PMLR, 2018.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J Reddi, and Sanjiv Kumar. Are transformers universal approximators of sequence-to-sequence functions? *arXiv preprint arXiv:1912.10077*, 2019.

A Additional Proofs and Theorems

Proof of Proposition 3.9.

$$\begin{aligned}\mathcal{L}(\theta) &= \sum_{x\in\Sigma^*} P(x)(-\log Q_\theta(x)) \\ &= \sum_{x\in\Sigma^*} P(x)(\log P(x) - \log P(x) - \log Q_\theta(x)) \\ &= \sum_{x\in\Sigma^*} P(x)\log\frac{P(x)}{Q(x)} + \sum_{x\in\Sigma^*} P(x)\log P(x) \\ &= D_{\mathrm{KL}}(P \parallel Q_\theta) + H(P)\end{aligned}$$

□

Proof of Theorem 4.1. The decomposition can be constructed recursively. Given CFG $G = (\Sigma, \mathcal{N}, \mathcal{S}, \mathcal{P}, \mathcal{W})$, the root node of the DAG – initially labelled with only S – represents the entire grammar. If S can generate itself through successive applications of rules of G, we add a self-loop from S to itself.

Let $X \subseteq N$ be the subset of non-terminals on the right-hand side of any rule $S \to \alpha$. For each $A \in N$, let G_A be the inner subgrammar generated by taking the closure of A in $\mathcal{P}$ – that is, all the expansions $A \to \alpha$, all expansions of those non-terminals on the right-hand side of those rules, and so on. In the case that the result subgrammar is all of G, we can add A as an additional label to the root node. Otherwise, G_A is a proper inner subgrammar, in which case we assign it a node as a child of S. Inductively, this procedure is applied to each new subgrammar node (which by construction has strictly fewer non-terminals than its supergrammar). □

Proof of Theorem 4.3. Theorems 4.3 and Corollary 4.4 are equivalent, for simplicity we directly prove Corollary 4.4.

Let $A_1, \ldots, A_k$ be the top-level subgrammars of G. Let $S \to A_{i_{1,1},\ldots,i_{1,l_1}}$, ..., $S \to A_{i_{r,1},\ldots,i_{r,l_r}}$ be all rules expanding S in $\mathcal{P}$, with probabilities $p_1, \ldots, p_r$ respectively. As we are directly proving Corollary 4.4, we assume S expands only to non-terminals (by which we will also denote the top-level subgrammars; note that some of these may be S itself if they are not proper subgrammars).

Denoting $P = P_G$ and $Q = Q_\theta$,

$$\begin{aligned}D_{\mathrm{KL}}(P \parallel Q) &= \sum_{s\in\Sigma^*} P(s)\log\frac{P(s)}{Q(s)} \\ &= \sum_{j=1}^{r} p_j \sum_{a_{j,1},\ldots,a_{j,l_j}} P(a_{j,1}\cdots a_{j,l_j})\log\frac{P(a_{j,1}\cdots a_{j,l_j})}{Q(a_{j,1}\cdots a_{j,l_j})} \\ &= \sum_{j=1}^{r} p_j \sum_{a_{j,1},\ldots,a_{j,l_j}} P_{A_{j,1}}(a_{j,1})\cdots P_{A_{j,l_j}}(a_{j,l_j}) \sum_{i=1}^{l_j}\log\frac{P_{A_{j,i}}(a_{j,i})}{Q(a_{j,i}|a_{j,1}\cdots a_{j,i-1})} \\ &= \Big[\sum_{j=1}^{r} p_j \sum_{i=1}^{l_j} \sum_{a_{j,1},\ldots,a_{j,i-1}} P_{A_{j,1}}(a_{j,1})\cdots P_{A_{j,i-1}}(a_{j,i-1})\Big] \sum_a P_{A_{j,i}}(a)\log\frac{P_{A_{j,i}}(a)}{Q(a|a_{j,1}\cdots a_{j,i-1})} \\ &= \sum_{i=1}^{k}\sum_s P_G(s|\epsilon)P_G(A|s)\sum_a P_{A_i}\log\frac{P_{A_i}(a)}{Q(a|s)}\end{aligned}$$

□

Corollary A.1. *Suppose G has subgrammars $Z_1, \ldots, Z_l$ as irreducible "leaf" subgrammars in its DAG subgrammar decomposition, and all rules evaluate to strinsg of only non-terminals, or only-*

terminals. Then

$$D_{\mathrm{KL}}(P_G \parallel Q_\theta) = \sum_{i=1}^{l} D_{\mathrm{KL}}(P_G \parallel Q_\theta)_{Z_i}$$

Proof of Theorem 4.6. Let G be a PCFG with top-level, proper subgrammars $A_1, \ldots, A_k$. Summing over the top-level rules (expansions of S), suppose S maps to a rule with i recursive S's with probability p_i ($\sum_{i=0}^{N} p_i = 1$ for some $N < \infty$). Then, $\mathbb{E}[R] = \sum_{i=1}^{N} p_i \cdot i$. Then by Corollary 4.5 (treating both proper subgrammars and recursive S as top-level subgrammars), we have

$$\begin{aligned} D_{\mathrm{KL}}(P_G \parallel Q_\theta) &= \sum_{i=1}^{k} D_{\mathrm{KL}}(P_{A_i} \parallel Q_\theta(A_i)) + \sum_{i=1}^{N} p_i \cdot i D_{\mathrm{KL}}(P_G \parallel Q_\theta) \\ &= \sum_{i=1}^{k} D_{\mathrm{KL}}(P_{A_i} \parallel Q_\theta(A_i)) + \mathbb{E}[R] D_{\mathrm{KL}}(P_G \parallel Q_\theta) \\ \implies D_{\mathrm{KL}}(P_G \parallel Q_\theta) &= \frac{\sum_{i=1}^{k} D_{\mathrm{KL}}(P_A \parallel Q_\theta(A_i))}{1 - \mathbb{E}[R]} \end{aligned}$$

□

Theorem A.2. *For G with outer subgrammar A, let $\bar{A}$ be its complement. The KL-divergence splits as a weighted sum:*

$$D_{\mathrm{KL}}(P_G \parallel Q_\theta) = P_G(A) D_{\mathrm{KL}}(P_A \parallel Q_\theta|_A) + P_G(\bar{A}) D_{\mathrm{KL}}(P_G|_{\bar{A}} \parallel Q_\theta|_{\bar{A}}) + D_{\mathrm{KL}}(P_G^* \parallel Q_\theta^*)$$

Where D^*, for $D \in \{P_G, Q_\theta\}$ is the 2 valued distribution of whether D outputs a string in A or $\bar{A}$, P_A is the language from CFG A, and $D|_B$ indicates the marginal distrubution of D over strings of $B \in \{A, \bar{A}\}$.

Proof. Writing P for P_G and Q for Q_θ for legibility,

$$\begin{aligned} D_{\mathrm{KL}}(P \parallel Q) &= \sum_{s \in A} P(s) \log \frac{P(s)}{Q(s)} + \sum_{s \in \bar{A}} P(s) \log \frac{P(s)}{Q(s)} \\ &= P(A) \sum_{s \in A} P_A(s) [\log P(A) + \log P_A(s) - \log Q^*(A) - \log Q|_A(s)] \\ &+ P(\bar{A}) \sum_{s \in \bar{A}} P|_{\bar{A}}(s) [\log P(\bar{A}) + \log P|_{\bar{A}}(s) - \log Q^*(\bar{A}) - \log Q|_{\bar{A}}(s)] \end{aligned}$$

From which the final decomposition follows quite immediately by rearranging terms. □

B Additional Experimental results

B.1 ChatGPT-5 Instant Arithmetic Stress Test

We generate arithmetic expressions using integers uniformly sampled from 0–9 and the operators {+, -, *, / } are generated. Expression depth is defined as the maximum level of nested parentheses. *Non-deep chains* consist of 50 expressions of depth at most 2, concatenated by addition. *Deep chains* consist of single expressions with recursive nesting up to depth 7. Below are an example each:

Non-deep arithmetic expression:
((4∗4)∗(1−9))+((6/3)∗(5/1))+((2−8)∗(8/5))+((5/9)∗(7∗7))+((7−4)+(8/7))+((9−6)+(1−0))+((0/1)+(9−9))+((4/1)+(0+5))+((6+6)/(2/5))+((4/5)+(0−2))+((3∗1)+(5+3))+((1−0)−(7−6))+((2∗5)∗(5/3))+((6+9)−(6/1))+((1+4)/(6+9))+((9/7)−(6+2))+((6−7)/9)+((4+1)+(7−3))+((5−3)−(1∗3))+((5+6)+4)+((5∗2)+(0−0))+((6∗7)∗8)+((5/2)+(4+6))+((5/5)∗(9/6))+((4−3)∗(8∗7))+((7/3)∗(9+3))+((7−0)+(5/9))+((6/8)−(2+0))+((0+6)/4)+((9−5)−(3−9))+((0+1)+(9−4))+((7−7)∗(1−8))+((7−1)+9)+((4−

$0)+(0*8))+((6/9)*(2-2))+((5-6)-(8/4))+((3*5)/(4+2))+((3*4)-(5+2))+((7-1)+(8/8))+((4*0)-(9+7))+((3/6)-(4/3))+((0-2)-(1/9))+((0-8)*(8*0))+((0/1)*(2/8))+((9+5)*(8/3))+((1+8)/(4-9))+((0*6)*(2+4))+((5/6)+(2+0))+((2*7)-(2/2))+((8+8)*2)$

Result: $\frac{707449}{1260}$

Deep arithmetic expression:
$(((((((3+8)+(5-1))-((1-6)+(5+3)))+(((8-2)-(3-8))+((2*9)*(4+5))))*((((1/7)-(6*4))*((7+3)*(6+6)))-(((8*3)*(1+8))+((5-9)+(7/1)))))+(((((8*6)/(5-3))*((8*0)-(8-0)))+(((8-9)+(3-6))-((9/8)/(7*8))))/((((7-4)*(2+2))-((3-5)/(9-2)))/(((6/8)+(5*5))*((4-1)-(8+8))))))-((((((4-0)/(4-8))*((8-0)-(3-1)))+(((7*7)*(4/7))*((7*0)-(0/7))))-((((8*6)/(8+7))-((8/8)+(8/4)))-(((5+5)*(9*8))-((9/2)/(3-9))))))+((((((9-8)+(2*1))-((4+3)/(9-5)))/(((2*2)*(4*3))-((6-6)-(6+9))))*((((8/8)-(3*3))/((8+0)+(9/1)))*(((2*1)*6)*((1+5)/8))))))$

Result: $\frac{892410719}{448320600}$

B.2 Recursive Decomposition Experiments

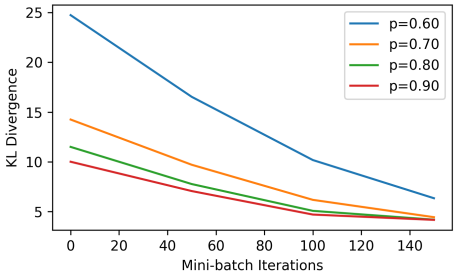
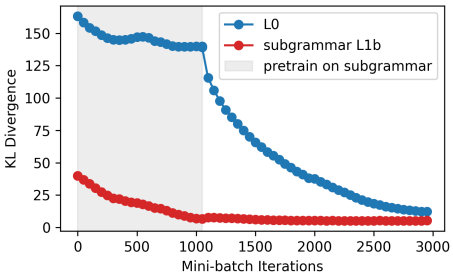


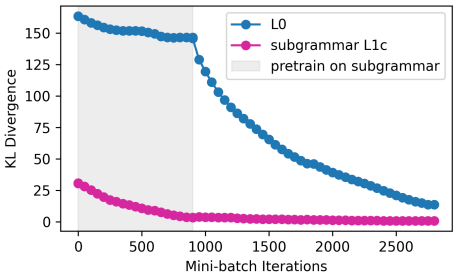
Figure 4: Two-layer Transformer showing the impact of the probability of recursion.

B.3 Pretraining Results

This appendix provides the detailed results referenced in the main text. All experiments compare transformers trained from scratch against those pretrained on a subgrammar before continuing on the full grammar. Figure 5 shows that no matter which subgrammar is chosen, when later training on the full grammar, it is not forgotten.



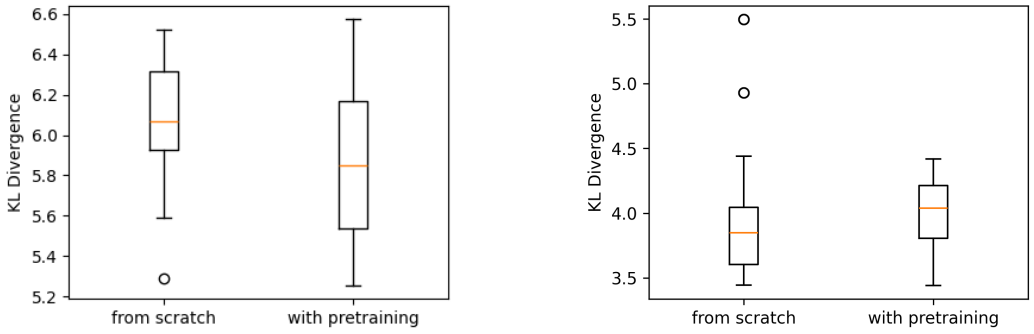
(a) Pretraining on an infix subgrammar



(b) Pretraining on a suffix subgrammar

Figure 5: Examples of pretraining on differently placed subgrammars using `ABC Grammar`.

Figure 6 illustrates the distribution of KL-divergences across 30 seeds when training directly versus with 10 epochs of subgrammar pretraining. Pretraining consistently shifts the distribution toward lower KL.



(a) Two-layer Transformer

(b) Four-layer Transformer

Figure 6: Distribution of final KL value of pretraining versus training from scratch

Table **??** reports average cosine similarity across attention and MLP layers, on three types of test sequences: (i) sequences consisting solely of subgrammar subsequences, (ii) sequences with no subgrammar subsequences, and (iii) sequences mixing subgrammar and other subsequences.

	Attention	MLP
Sequences with subgrammar only		
From Scratch	0.660	0.635
With Pretraining	0.743	0.611
Percentage change (%)	*+12.6*	*-3.9*
Sequences without subgrammar		
From Scratch	0.835	0.837
With Pretraining	0.876	0.841
Percentage change (%)	*+4.9*	*+0.5*
Sequences with subgrammar		
From Scratch	0.726	0.501
With Pretraining	0.687	0.543
Percentage change (%)	*-5.7*	*+8.4*

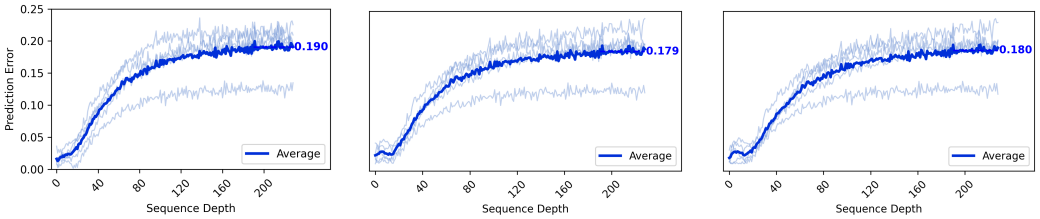
Table 2: Average cosine similarity [-1, +1] across attention and MLP layers of a two-layer Transformer when pretraining for 10 epochs. *Percentage change* indicates the relative difference between models trained from scratch and with pretraining.

B.4 Generalization and Prefix Experimentsn

This appendix provides the detailed figures referenced in the main text. They compare how different valid prefixes (shallow vs. deeply recursive) and malformed prefixes affect model stability, showing that deeply recursive but valid prefixes can degrade performance even more than ungrammatical ones.

C Definition of Grammars used for Experiments

In this section we properly introduce the PCFGs used for running the experiments.



(a) Prefix = $(((((((((a)))))))))$ (b) Prefix = $(a)(a)(a)(a)(a)(a)$ (c) Wrong prefix = $(a)(a)(a))(aa)(a)(a)$

Figure 7: Comparison of different prefixes for recursion type 2

KL DECOMPOSITION EXAMPLE 1

$$
\begin{aligned}
L1 &\rightarrow \texttt{sL2\_2}\ L2\_2\ \texttt{eL2\_2}\ \texttt{sL2\_1}\ L2\_1\ \texttt{eL2\_1}\ \texttt{sL2\_3}\ L2\_3\ \texttt{eL2\_3}\ [1.0] \\
L2\_1 &\rightarrow NUM\ [0.4]\ |\ L2\_1 \star L2\_1\ [0.15]\ |\ L2\_1 + L2\_1\ [0.15]\ |\ NUM\ NUM\ [0.3] \\
L2\_2 &\rightarrow \texttt{a}\ L2\_2\ \texttt{b}\ [0.6]\ |\ \texttt{c}\ [0.4] \\
L2\_3 &\rightarrow \texttt{x}\ L2\_3\ [0.8]\ |\ \texttt{x}\ [0.2] \\
NUM &\rightarrow \texttt{0}\ [0.2]\ |\ \texttt{1}\ [0.2]\ |\ \texttt{2}\ [0.2]\ |\ \texttt{3}\ [0.2]\ |\ \texttt{4}\ [0.1]\ |\ \texttt{5}\ [0.1]
\end{aligned}
$$

KL DECOMPOSITION EXAMPLE 2

$$
\begin{aligned}
L1 &\rightarrow \texttt{sL2\_1}\ L2\_1\ \texttt{eL2\_1}\ [0.3]\ |\ \texttt{sL2\_2}\ L2\_2\ \texttt{eL2\_2}\ [0.3]\ |\ \texttt{sL2\_3}\ L2\_3\ \texttt{eL2\_3}\ [0.4] \\
L2\_1 &\rightarrow NUM\ [0.4]\ |\ L2\_1 \star L2\_1\ [0.15]\ |\ L2\_1 + L2\_1\ [0.15]\ |\ NUM\ NUM\ [0.3] \\
L2\_2 &\rightarrow \texttt{a}\ L2\_2\ \texttt{b}\ [0.6]\ |\ \texttt{c}\ [0.4] \\
L2\_3 &\rightarrow \texttt{x}\ L2\_3\ [0.8]\ |\ \texttt{x}\ [0.2] \\
NUM &\rightarrow \texttt{0}\ [0.2]\ |\ \texttt{1}\ [0.2]\ |\ \texttt{2}\ [0.2]\ |\ \texttt{3}\ [0.2]\ |\ \texttt{4}\ [0.1]\ |\ \texttt{5}\ [0.1]
\end{aligned}
$$

DEEPER RECURSION

$$
\begin{aligned}
L0 &\rightarrow \texttt{sL1}\ L1\ \texttt{eL1}\ [0.7]\ |\ L0\ L0\ [0.3] \\
L1 &\rightarrow \texttt{sL2}\ L2\ \texttt{eL2a}\ [0.6]\ |\ L1\ L1\ [0.3]\ |\ V\ [0.1] \\
L2 &\rightarrow \texttt{sL3}\ L3\ \texttt{eL3}\ [0.6]\ |\ L2\ L2\ [0.3]\ |\ V\ [0.1] \\
L3 &\rightarrow \texttt{sL4}\ L4\ \texttt{eL4}\ [0.6]\ |\ L3\ L3\ [0.3]\ |\ V\ [0.1] \\
L4 &\rightarrow \texttt{(}\ V\ \texttt{)}\ [0.7]\ |\ V\ [0.3] \\
V &\rightarrow \texttt{a}\ [0.04]\ |\ \texttt{b}\ [0.04]\ |\ \texttt{c}\ [0.04]\ |\ \texttt{d}\ [0.04]\ |\ \texttt{e}\ [0.04]\ |\ \texttt{f}\ [0.04]\ |\ \texttt{g}\ [0.04] \\
&\quad\ \texttt{h}\ [0.04]\ |\ \texttt{i}\ [0.04]\ |\ \texttt{j}\ [0.04]\ |\ \texttt{k}\ [0.04]\ |\ \texttt{l}\ [0.04]\ |\ \texttt{m}\ [0.04]\ |\ \texttt{n}\ [0.04] \\
&\quad\ \texttt{o}\ [0.04]\ |\ \texttt{p}\ [0.04]\ |\ \texttt{q}\ [0.04]\ |\ \texttt{r}\ [0.04]\ |\ \texttt{s}\ [0.04]\ |\ \texttt{t}\ [0.04]\ |\ \texttt{u}\ [0.04] \\
&\quad\ \texttt{v}\ [0.04]\ |\ \texttt{w}\ [0.04]\ |\ \texttt{x}\ [0.04]\ |\ \texttt{y}\ [0.04]
\end{aligned}
$$

UNIFIED SUBGRAMMAR EXAMPLE

$$
\begin{aligned}
START &\rightarrow \texttt{sSUBJ}\ SUBJ\ \texttt{eSUBJ}\ \texttt{sVERB}\ VERB\ \texttt{eVERB}\ \texttt{sOBJ}\ OBJ\ \texttt{eOBJ}\ [1.0] \\
SUBJ &\rightarrow \textbf{NOUN}\ [0.2]\ |\ \texttt{a}\ NOUN\ [0.4]\ |\ \texttt{the}\ NOUN\ [0.4] \\
NOUN &\rightarrow \textbf{N}\ [0.7]\ |\ ADJ\ NOUN\ [0.3] \\
VERB &\rightarrow \textbf{V}\ [0.3]\ |\ V\ ADV\ [0.7] \\
OBJ &\rightarrow \textbf{blank}\ [0.5]\ |\ \texttt{with}\ SUBJ\ [0.5] \\
N &\rightarrow \texttt{dog}[0.2] \mid \texttt{cat}[0.2] \mid \texttt{fox}[0.1] \mid \texttt{parrot}[0.1] \mid \texttt{hamster}[0.1] \mid \texttt{turtle}[0.1] \mid \\
&\quad \texttt{horse}[0.1] \mid \texttt{pig}[0.1] \\
ADJ &\rightarrow \texttt{big}[0.2] \mid \texttt{poisonous}[0.2] \mid \texttt{cute}[0.2] \mid \texttt{lazy}[0.2] \mid \texttt{quick}[0.2] \\
V &\rightarrow \texttt{eats}[0.15] \mid \texttt{runs}[0.4] \mid \texttt{sleeps}[0.15] \mid \texttt{talks}[0.15] \mid \texttt{cleans itself}[0.15] \\
ADV &\rightarrow \texttt{quickly}[0.2] \mid \texttt{slowly}[0.3] \mid \texttt{happily}[0.3] \mid \texttt{excitedly}[0.1] \mid \texttt{lazily}[0.1]
\end{aligned}
$$

where the rules that are used for the unified subgrammar are highlighted in bold.

ABC GRAMMAR

$$
\begin{aligned}
L0 &\rightarrow \texttt{sL1a}\ L1a\ \texttt{eL1a}\ \texttt{sL1b}\ L1b\ \texttt{eL1b}\ \texttt{sL1c}\ L1c\ \texttt{eL1c}\ [1.0] \\
L1a &\rightarrow \texttt{sL2a}\ L2a\ \texttt{eL2a}\ L1a\ \texttt{sL2\_2a}\ L2\_2a\ \texttt{eL2\_2a}\ [0.4]\ |\ \texttt{sL2a}\ L2a\ \texttt{eL2a}\ L1a\ [0.2]\ | \\
&\quad \texttt{action}\ [0.4] \\
L1b &\rightarrow L1b\ \texttt{+}\ \texttt{sL2b}\ L2b\ \texttt{eL2b}\ [0.25]\ |\ \texttt{sL2b}\ L2b\ \texttt{eL2b}\ [0.75] \\
L1c &\rightarrow \texttt{xy}\ L1c\ [0.3]\ |\ \texttt{x}\ L1c\ [0.3]\ |\ \texttt{sL2c}\ L2c\ \texttt{eL2c}\ [0.4] \\
L2a &\rightarrow \texttt{sL3}\ L3\ \texttt{eL3}\ [0.5]\ |\ \texttt{not}\ L2a\ [0.25]\ |\ L2a\ \texttt{and}\ L2a\ [0.1]\ |\ L2a\ \texttt{or}\ L2a\ [0.15] \\
L2\_2a &\rightarrow \texttt{a}\ L2\_2a\ [0.8]\ |\ \texttt{a}\ [0.2] \\
L2b &\rightarrow \texttt{a}\ L2b\ \texttt{b}\ [0.6]\ |\ \texttt{c}\ [0.4] \\
L2c &\rightarrow \texttt{c}\ L2\_2ac\ [0.7]\ |\ \texttt{c}\ [0.6] \\
L3 &\rightarrow \texttt{==}\ [0.2]\ |\ \texttt{<=}\ [0.2]\ |\ \texttt{<}\ [0.2]\ |\ \texttt{>=}\ [0.2]\ |\ \texttt{>}\ [0.2]
\end{aligned}
$$

NESTED PARENTHESES

$$
\begin{aligned}
L0 &\rightarrow \texttt{(}\ L1\ \texttt{)}\ [0.7]\ |\ L0\ L0\ [0.3] \\
L1 &\rightarrow \texttt{(}\ L1\ \texttt{)}\ [0.8]\ |\ \texttt{a}\ [0.2]
\end{aligned}
$$