Between Circuits and Chomsky: Pre-pretraining on Formal Languages Imparts Linguistic Biases

Michael Y. Hu[1] **Jackson Petty**[2] **Chuan Shi**[1] **William Merrill**[1] **Tal Linzen**[1,2]

[1]Center for Data Science [2]Department of Linguistics
New York University
{michael.hu, petty, cs5526, willm, linzen}@nyu.edu

Abstract

Pretraining language models on formal language can improve their acquisition of natural language. Which features of the formal language impart an inductive bias that leads to effective transfer? Drawing on insights from linguistics and complexity theory, we hypothesize that effective transfer occurs when two conditions are met: the formal language should capture the dependency structures present in natural language, and it should remain within the computational limitations of the model architecture. We experiment with pre-pretraining (training on formal language before natural languages) on transformers and find that formal languages capturing hierarchical dependencies indeed enable language models to achieve lower loss on natural language and better linguistic generalization compared to other formal languages. We also find modest support for the hypothesis that the formal language should fall within the computational limitations of the architecture. Strikingly, pre-pretraining reduces loss more efficiently than training on a matched amount of natural language. For a 1B-parameter language model trained on roughly 1.6B tokens of natural language, pre-pretraining achieves the same loss and better linguistic generalization with a 33% smaller token budget. Finally, we also give mechanistic evidence of transfer from formal to natural language: attention heads acquired during pre-pretraining remain crucial for the model's performance on syntactic evaluations.[1]

1 Introduction

Language models have achieved impressive performance on many tasks, but they remain data-hungry, requiring five to six orders of magnitude more data than humans to achieve human-level performance (Paul, 2017; Warstadt et al., 2023). This high data requirement presents challenges for training models in low-resource settings (Zhong et al., 2024; Hettiarachchi et al., 2025), understanding how language models can serve as cognitive models of language acquisition with human-like data constraints (Wilcox et al., 2025), and continuing to improve models even after most of the existing natural language data has been used for pretraining (Villalobos et al., 2024). Thus, data efficiency during training is an important frontier for language models.

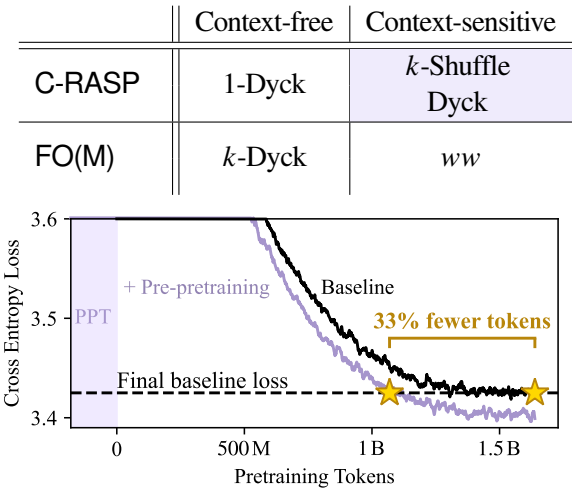

Figure 1: The intersection of Chomsky and circuit hierarchies (top), where C-RASP ⊂ FO(M) and context-free ⊂ context-sensitive. Within this 2 × 2, we find that pre-pretraining on k-Shuffle Dyck, a context-sensitive language definable in C-RASP, lets 1B-parameter models match the final baseline performance of no pre-pretraining with 33% fewer training tokens (bottom). See §3.2.

A recently-explored approach for increasing data efficiency teaches models useful inductive biases by first training them on formal languages before training on natural language (Papadimitriou and Jurafsky, 2020; Chiang and Lee, 2022; McCoy and Griffiths, 2025). We refer to this paradigm as pre-pretraining. What features of formal languages make transfer to natural language effective? Papadimitriou and Jurafsky (2023) show that within the Chomsky hierarchy, context-sensitive languages transfer best to natural language compared to simpler classes of languages. We

[1]Code is available at https://github.com/michahu/pre-pretraining.

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 9691–9709

expand on their investigation and explore an additional factor: the computational limitations of the language model's architecture. In particular, transformers—the architecture that underlies most popular language models—cannot learn all context-sensitive languages, both in theory and practice (Merrill and Sabharwal, 2023; Strobl et al., 2024). In fact, within all levels of the Chomsky hierarchy, some languages are harder for transformers to learn than others, and many are impossible for them to learn (Merrill et al., 2023, 2024). Can a formal language give rise to positive transfer even when it cannot be fully learned by a transformer?

In this work, we hypothesize that optimal transfer from formal to natural language in transformer language models occurs at the intersection of two theoretical hierarchies: the Chomsky hierarchy of formal languages and the circuit complexity hierarchy that bounds transformer computational power (see §3). Specifically, we hypothesize that effective pre-pretraining languages should be:

1. expressive enough to capture hierarchical natural language dependencies, and
2. learnable by transformers in a way that generalizes to longer strings than observed in training.

To satisfy the second condition, we define our formal languages in C-RASP (Yang and Chiang, 2024), a restricted programming language whose functions allow transformers to exhibit length generalization (Huang et al., 2025).

Our empirical results support the first part of the hypothesis and provide some support for the second part (§4). Pre-pretraining on languages with hierarchical dependencies outperforms pre-pretraining on any of the other formal languages that we tested—moreover, it outperforms pre-pretraining on a matched amount of natural language. Of the formal languages with hierarchical dependencies, those that are definable in C-RASP generally achieve equal or better performance, but they are only clearly superior on some of the tasks we evaluated.

Next, we show that when positive transfer occurs, the model reuses attention heads it learned during pre-pretraining, suggesting that mechanisms from pre-pretraining transfer to natural language (§5). Finally, we scale up our experiments to a 1B-parameter language model, and show that in pre-pretraining is effective in that size as well, increasing token efficiency by 33% (§6). Overall, we conclude that formal language pre-pretraining is an effective way to improve generalization and data efficiency, and take a step towards characterizing the formal languages that are most promising for this purpose.

2 Background

2.1 The Chomsky Hierarchy

The Chomsky hierarchy (Chomsky, 1959) is a nested classification of increasingly-complex formal languages. This classification is based on the kinds of computations needed to process formal structures resembling those found in human language. For example, regular languages, the least complex, can be recognized by finite-state automata. While regular languages can capture most phenomena in natural language phonology and morphology, they are insufficient for syntax: representing the hierarchical structure of natural language syntax with a finite-state automaton would require infinitely many states (Chomsky, 1956). Subsequent works showed that modeling some syntactic phenomena requires not only context-free but also context-sensitive grammars (Shieber, 1985), though the prevalence of such phenomena may be limited.

Dyck languages. A classic context-free language is k-Dyck: the language of well-balanced parentheses with k bracket types. For example, `([])[]` is a valid 2-Dyck string, where rounded and square parentheses are the two bracket types. k-Dyck is often taken as a canonical example of context-free hierarchical structure because any context-free language can be reduced to Dyck via a single transformation (inverse homomorphism) and intersection with a regular language (Chomsky and Schützenberger, 1959).

Shuffle Dyck. Removing the constraint that Dyck braces must be well-nested, but maintaining the constraint that every opening brace must be closed and vice versa, yields k-Shuffle Dyck,[2] a minimal relaxation of k-Dyck that is strictly context-sensitive rather than context-free (Suzgun et al., 2019; Strobl et al., 2024). Crossing braces in k-Shuffle Dyck can be thought of as a formal model of the cross-serial dependencies underlying aspects of language argued to be context-sensitive (Papadimitriou and Jurafsky, 2023).

[2]Despite what its name might suggest, k-Shuffle Dyck does not randomly shuffle strings from k-Dyck. Every opening brace in k-Shuffle Dyck must still be closed by a matching closing brace later in the string; this constraint would not in general be satisfied by randomly shuffled k-Dyck strings. Instead, k-Shuffle Dyck can be defined by interleaving k 1-Dyck strings with different braces (Suzgun et al., 2019), as if by riffle shuffling. We use the terminology "Shuffle Dyck" for consistency with prior work.

2.2 The Circuit Hierarchy

We focus in this work on transformer language models. There are languages at each level of the Chomsky hierarchy that a transformer cannot recognize (Merrill and Sabharwal, 2023; Liu et al., 2024; Strobl et al., 2024). Thus, the Chomsky hierarchy alone does not precisely capture how difficult a language is for transformers to learn: for instance, transformers can learn some context-free languages (Butoi et al., 2025) and yet fail to learn other regular languages (Merrill et al., 2024). To better understand the expressive power of transformers, recent work has analyzed formal languages within a different hierarchy: the circuit complexity hierarchy, which better captures the computations performed by transformers (Hao et al., 2022; Yang et al., 2024). Here, we will focus on two logics that emerge from the circuit complexity viewpoint: FO(M) (Merrill and Sabharwal, 2023) and C-RASP (Yang and Chiang, 2024).

FO(M). First-order logic with majority, or FO(M), is a provable *upper bound* on the languages that transformers can express: that is, any transformer that recognizes a language can be converted into an FO(M) program that defines (or recognizes) the same language (Merrill and Sabharwal, 2023). FO(M) programs operate by computing counts over the number of indices in an input string that satisfy certain predicates. For example, $Q_a(i)$ is a basic predicate that checks whether input token i is an a. The following FO(M) program uses $Q_a(i)$ to define the language of strings with exactly 3 a's:

$$\#i \leq n[Q_a(i)] = 3 \tag{1}$$

Beyond this example, FO(M) can implement a rich variety of programs by nesting quantifiers and building complex predicates out of logical $(\wedge, \vee, \neg)$ and arithmetic operators $(+, =, <)$. In particular, FO(M) can define the k-Dyck language for any $k \geq 1$ (Proposition A.6 in the Appendix). For example, the following program defines 1-Dyck:

$$\begin{aligned} \mathsf{depth}(i) &\equiv \#j \leq i[Q_(i)] - \#j \leq i[Q_)(i)] \\ [\mathsf{depth}(n) = 0] &\wedge \#i \leq n[\mathsf{depth}(i) < 0] = 0 \end{aligned} \tag{2}$$

To define 2-Dyck, this can be extended by modifying depth to track two bracket types and computing the following depth index:

$$\mathsf{dindex}(i) \equiv \#j \leq i[\mathsf{depth}(i) = \mathsf{depth}(j)] \tag{3}$$

To finish the definition, we add a condition to enforce that any open and close brace paired by depth and dindex also match in their type (i.e., both are parentheses or both are square braces). See Proposition A.7 for further details.

C-RASP. While any transformer can be compiled into FO(M), it is not necessarily the case that any FO(M) program can be implemented by a transformer. C-RASP is a restriction of FO(M) designed to be a *lower bound* on what transformers can express: that is, if a language is definable in C-RASP, then there exists a transformer that recognizes it (Yang and Chiang, 2024).[3] The most crucial restriction for our purposes is that each C-RASP predicate can only refer to one index variable i, whereas in FO(M) predicates can refer to two (or more) indices i, j introduced by different quantifiers (for more detail, see Yang and Chiang, 2024). This means C-RASP can define (1) or (2) above, but not k-Dyck for $k \geq 2$, as C-RASP cannot express the function dindex in (3), which compares the depth of two different indices.

Recent work has also suggested a connection between C-RASP and transformers' ability to generalize to strings longer than those observed in training (Zhou et al., 2024; Huang et al., 2025): the definability of a language L in C-RASP predicts whether transformers can reliably length-generalize when trained on strings from L. One interpretation of this finding is that mechanisms expressible in C-RASP may be more robustly learnable by transformers. We thus hypothesize that we will observe more reliable transfer from pre-pretraining transformers on formal languages that can be defined in C-RASP compared to languages that cannot.

3 Methods

3.1 Defining Pre-pretraining

We train a language model using an optimizer $\mathcal{A}(\mathcal{D}, t, \theta_{\text{init}})$ which returns parameters θ_t after t timesteps (gradient updates). We apply $\mathcal{A}$ sequentially:

1. Pre-pretrain for t_0 steps on dataset $\mathcal{D}_{\text{ppt}}$ to obtain model parameters θ_{t_0}.
2. Pretrain for t_1 steps on dataset $\mathcal{D}_{\text{pt}}$ to obtain θ_{t_1}.

Our objective is to minimize the expected loss on the pretraining dataset, i.e. to find $\arg\min_{\theta_{t_1}} \mathbb{E}[\ell(\mathcal{D}_{\text{pt}}, \theta_{t_1})]$. We hold $\mathcal{A}$'s hyperparameters, t_1, and $\mathcal{D}_{\text{pt}}$ fixed, and we transfer model parameters directly from pre-pretraining to pretraining.

[3] C-RASP is a well-defined variant of the Restricted Access Sequence Processing programming language (RASP; Weiss et al., 2021; Lindner et al., 2023).

In other words, to minimize $\ell(\mathcal{D}_{\text{pt}},\theta_{t_1})$, we can only change the pre-pretraining dataset $\mathcal{D}_{\text{ppt}}$ and duration t_0. We compare pre-pretraining on our proposed $\mathcal{D}_{\text{ppt}}$ datasets (§3.2) against several baselines:

- No pre-pretraining ($t_0 = 0$).
- Pre-pretraining on random binary strings.
- Pre-pretraining on random strings of k integers.
- Pre-pretraining on unseen natural language data $\mathcal{D}^*_{\text{pt}}$ drawn from the same distribution as $\mathcal{D}_{\text{pt}}$.

Aside from the no-pre-pretraining baseline, we pre-pretrained the baselines for t_0 = 500 steps, the optimal number of steps for k-Shuffle Dyck (see §4). We note that the natural language pre-pretraining baseline is not equivalent to training on the union of $\mathcal{D}^*_{\text{pt}}$ and $\mathcal{D}_{\text{pt}}$ for longer, since pre-pretraining on natural language uses learning rate warmup twice, once in pre-pretraining and once in pretraining.

Lower validation loss compared to the no-pre-pretraining baseline would indicate that pre-pretraining on formal languages is beneficial. The random string baselines help establish whether this effect is specific to the particular formal languages we study. Finally, outperforming pre-pretraining on $\mathcal{D}^*_{\text{pt}}$ would suggest that formal languages provide a better initialization for pretraining than the pretraining data itself.

Evaluation. In addition to measuring validation loss, we perform targeted evaluations for grammaticality and verbatim retrieval. For grammaticality judgments, we compare the likelihood assigned by the model to minimal pairs of sentences that differ only in their grammaticality (e.g., *Only Bill would ever complain* is grammatical, but *Even Bill would ever complain* is not). Accuracy is measured as the proportion of pairs where the grammatical sentence is assigned higher likelihood than the ungrammatical one (Marvin and Linzen, 2018). We use the BLiMP grammaticality judgment dataset (Warstadt et al., 2020a). Verbatim retrieval tests language modeling on text passages with repeated lists (Armeni et al., 2022, 2024); the model is expected to assign a very high likelihood to the words in the second repetition of the list, such that lower loss indicates better performance. Both evaluations assess models' ability to learn and apply consistent patterns—a capability that could benefit from pre-pretraining on formal languages might strengthen. For examples of these evaluations, see Tables 3 and 4 in the Appendix.

Efficiency. In the regime with plentiful pretraining data, an ideal pre-pretraining language should minimize the number of pre-pretraining steps t_0 required: if a formal language requires very large t_0 for effective transfer, then simply pretraining on natural language, without any pre-pretraining, would be more practical in terms of total compute (though even in this case pre-pretraining may still be beneficial when the amount of data available for pretraining is small, for example in low-resource languages). We quantify efficiency using the marginal rate of substitution (MRS) between formal and natural language at 10,000 steps of natural language pretraining. In other words, we ask: if we train on 500 steps of the formal language, how many more steps does it take for the natural-language-only baseline to catch up?

For example, let x be the number of pre-pretraining steps and y be the number of pretraining steps, and suppose the following two pairs (x, y) of training steps achieve the same final loss: (0, 10,000) and (500, 6,000). Then the marginal rate of substitution is

$$\frac{|y_1 - y_2|}{|x_1 - x_2|} = \frac{|10{,}000 - 6{,}000|}{|0 - 500|} = 8.$$

The gain in token efficiency would be

$$1 - \frac{6{,}000 + 500}{1{,}0000} = 35\%$$

For a visualization, see Figure 7.

In our setting, a good pre-pretraining language would (1) minimize the amount of pre-pretraining steps t_0 (**efficiency**), and (2) increase the evaluation **performance** of the language model.

3.2 Between Circuits and Chomsky

We hypothesize that a good pre-pretraining language should both mimic particular aspects of the complexity of natural language and be robustly learnable by transformers in a way that generalizes to longer strings than observed in training. Because, as we discussed in §2, natural language is hierarchically structured and C-RASP is a formal model of what transformers can learn robustly, this motivates the following hypothesis:

> **Expressivity hypothesis:** A formal language that confers a helpful inductive bias should be hierarchically structured (either context-free or context-sensitive) and definable in C-RASP.

To test this hypothesis, we pre-pretrain transformer language models on the following formal languages:

1. 1-Dyck: the nested parentheses language. This language is context-free and in C-RASP.
2. k-Dyck: contains k different types of parentheses. The language is context-free and in FO(M) but not in C-RASP.

3. k-Shuffle Dyck: k-Dyck with cross-serial dependencies. This language is context-sensitive and in C-RASP.[4]
4. ww: The copy language. This language is context-sensitive and in FO(M) but not in C-RASP.

Language	Example
1-Dyck	((( )))
k-Dyck	([{ }])
k-Shuffle Dyck	([{]) }
ww	1 2 3 1 2 3

Table 1: Examples of our pre-pretraining languages.

The three variants of Dyck languages model hierarchical structure, while ww has a fixed dependency structure that maps the first half of the string onto the second half (Table 1). Proofs of where these languages lie on the Chomsky and circuit hierarchies can be found in Appendix A.[5]

We deliberately chose languages that are similar to each other. k-Dyck and k-Shuffle Dyck can be seen as different extensions of 1-Dyck: k-Dyck swaps out paired parentheses in valid 1-Dyck strings with new parentheses pairs, while k-Shuffle Dyck effectively interleaves several 1-Dyck sequences (Suzgun et al., 2019). Finally, ww contrasts with k-Shuffle Dyck as a maximally context-sensitive language, since *all* the dependencies in ww are cross-serial (i.e. none are nested within one another).

We construct 1-Dyck, k-Dyck, and k-Shuffle Dyck corpora with matching depth distributions by randomly opening or closing parentheses with probability $p = 0.5$, which yields a harmonic distribution over depths. We truncate the length of the sequences at 2048. We also match the vocabulary size: k-Dyck, k-Shuffle Dyck, and ww corpora each have 128 unique vocabulary items, or 64 unique parentheses pairs ($k = 64$) for the Dyck languages (we explore the effect of this hyperparameter in §6). All models are pre-pretrained on the same number of tokens with sequence packing.

4 Testing the Expressivity Hypothesis

For natural language ($\mathcal{D}_{\text{pt}}$), we trained Pythia 160M models (Biderman et al., 2023) for 10,000 steps, or roughly 665 million tokens. We use C4 as the natural language dataset (Raffel et al., 2019). For training hyperparameters, see Appendix B.

Efficiency. We find that the optimal amount of pre-pretraining t_0^* differs between formal languages. To estimate t_0^*, we sweep four pre-pretraining durations t_0. Figure 3 shows validation loss on natural language after pre-pretraining for 30 to 260 million tokens of formal language (500 to 4000 gradient updates).

While both k-Shuffle Dyck and k-Dyck outperform natural language pre-pretraining, k-Shuffle Dyck is more efficient with $t_0^* = 500$ compared to $t_0^* = 1000$ for k-Dyck. Pre-pretraining on ww is unhelpful at all durations. For each of the languages where pre-pretraining is effective, there is an optimal duration after which additional formal language pre-pretraining leads to less effective transfer overall. k-Shuffle Dyck has the highest MRS, indicating that it replaces tokens on natural language most efficiently (see Table 5 in the Appendix). Furthermore, the MRS for 1-Dyck, k-Dyck, and k-Shuffle Dyck are all greater than 1, indicating that exchanging natural language for these formal languages is **compute-optimal** in our setting.

Performance. k-Shuffle Dyck is the best-performing formal language on the natural language validation set from C4, followed by k-Dyck (Figure 2). Interestingly, pre-pretraining on all four formal languages improves accuracy in grammaticality, but pre-pretraining on natural language does not (for grammaticality accuracies by category, see Figure 8 in the Appendix). This indicates that formal language pre-pretraining also changes models' generalization properties, in addition to driving the language modeling loss lower. We hypothesize this is because pre-pretraining induces representations useful for modeling hierarchical structure; we revisit this point in §5.

Pre-pretraining on either random binary strings or k-integer strings has a negative effect: it results in higher validation loss than no pre-pretraining. This rules out the hypothesis that any pre-pretraining is helpful, regardless of the data being pre-pretrained on.

Summary. Hierarchical dependencies, which both k-Dyck and k-Shuffle Dyck have, appear to be crucial for positive transfer from formal to natural language. Although of these two languages only k-Shuffle Dyck is expressible by C-RASP, it only significantly outperforms k-Dyck on verbatim retrieval. That being said, k-Shuffle Dyck is more efficient than k-Dyck, achieving its optimal amount of pre-pretraining 500 steps earlier. Taken together, we find modest evidence

[4] Figure 10 shows a minimal code snippet for C-RASP.

[5] The languages NEST and CROSS from Papadimitriou and Jurafsky (2023) are instances of k-Dyck and k-Shuffle Dyck, respectively. Their results align with our hypothesis.

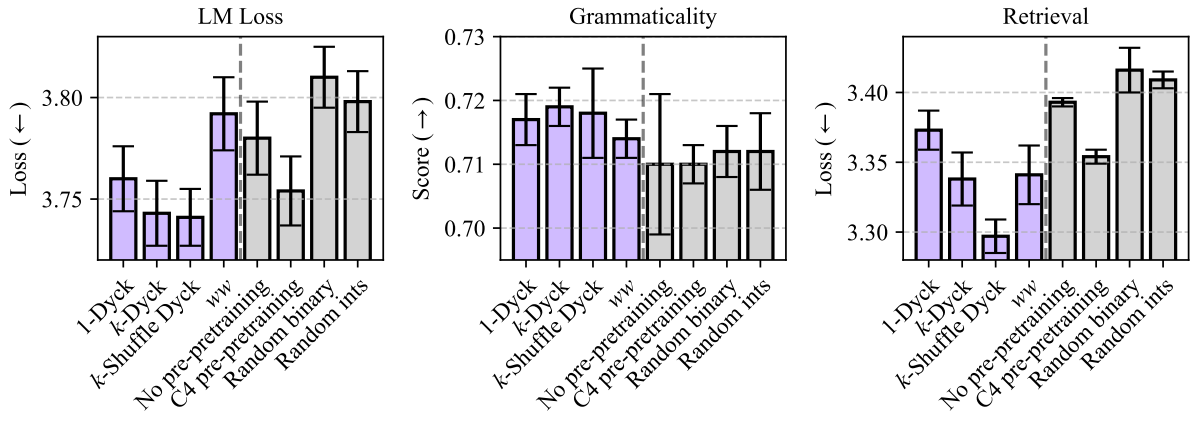

Figure 2: Evaluating models on overall language modeling loss, grammaticality and retrieval, at the optimal amount of pre-pretraining t_0^* for each formal language (the value of t_0^* for each language is determined based on Figure 3).

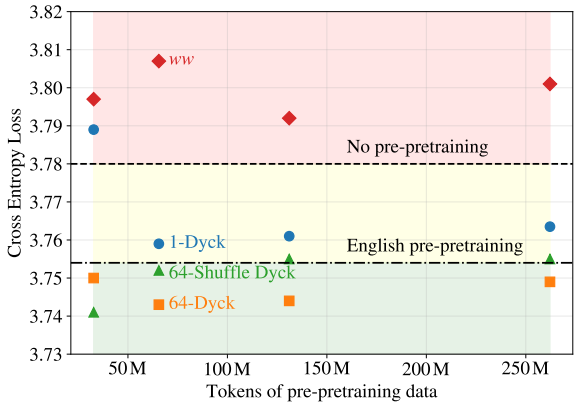

Figure 3: C4 validation loss as a function of pre-pretraining tokens. For the formal languages that improve validation loss over no pre-pretraining, there is an optimal training duration after which additional pre-pretraining is harmful.

supporting the importance of the expressibility of the language in C-RASP.

5 Mechanistic Analysis: Subnetworks

What is the mechanism by which pre-pretraining facilitates the learning of natural language? We hypothesize that the model implements next-token prediction on $\mathcal{D}_{\text{ppt}}$ using a sparse subnetwork, or some subset of the total parameters $\mathcal{M}(\theta_{t_0}) \subset \theta_{t_0}$ ($\mathcal{M}$ for short). Once we transfer θ_{t_0} to learn $\mathcal{D}_{\text{pt}}$, this subnetwork $\mathcal{M}$ continues to drive the performance of language modeling on $\mathcal{D}_{\text{pt}}$.

Subnetworks hypothesis: Attention heads established during formal language pre-pretraining are later used to represent the hierarchical structure of natural language.

We test this hypothesis by ablating attention heads of the pre-pretraining subnetwork and comparing the drop in performance against random attention head ablations. Concretely, we pre-pretrain on $\mathcal{D}_{\text{ppt}}$ and prune the model to find the sparse subnetwork $\mathcal{M}(\theta_{t_0})$. We use the heuristic core pruning algorithm from Bhaskar et al. (2024), which iteratively removes attention heads from the transformer using structured pruning (Xia et al., 2022) while minimizing the tradeoff between sparsity and language modeling loss on $\mathcal{D}_{\text{ppt}}$. After transfer and training on $\mathcal{D}_{\text{pt}}$, we evaluate the masked model $\mathcal{M}(\theta_{t_1})$ against a model $\mathcal{M}_{\text{null}}(\theta_{t_1})$ where a subnetwork with the same number of randomly chosen attention heads was ablated.

Positive transfer from $\mathcal{D}_{\text{ppt}}$ to $\mathcal{D}_{\text{pt}}$ could occur for reasons unrelated to subnetworks (e.g., computations are distributed across all heads or in other components of the model). In this case, the masked model $\mathcal{M}$ should perform no better than random masks $\mathcal{M}_{\text{null}}$ when applied to θ_{t_1}. However, if pre-pretraining does induce useful inductive biases, we would expect $\mathcal{M}$ to be an important subnetwork even after training on $\mathcal{D}_{\text{pt}}$. So in the alternative hypothesis, $\mathcal{M}$ should significantly outperform $\mathcal{M}_{\text{null}}$.

Results. After pre-pretraining on k-Shuffle Dyck, we ablate 50% of the attention heads. Following previous work (Bhaskar et al., 2024; Zhang and Nanda, 2024), we replace an ablated head with its mean activation. We compare $\mathcal{M}$ to the random subnetwork $\mathcal{M}_{\text{null}}$ and to $\mathcal{M}$'s complement subnetwork $\mathcal{M}^c$.

We find that $\mathcal{M}$ outperforms $\mathcal{M}_{\text{null}}$ and $\mathcal{M}^c$ in both language modeling and grammaticality (Figure 4). We reject the null hypothesis that the subnetwork $\mathcal{M}$ established during pre-pretraining has the same performance as a randomly sampled subnetwork ($p \ll 0.001$). Further supporting the role of the heads in $\mathcal{M}$, we find that $\mathcal{M}^c$, which excludes all of the heads identified by the pruning

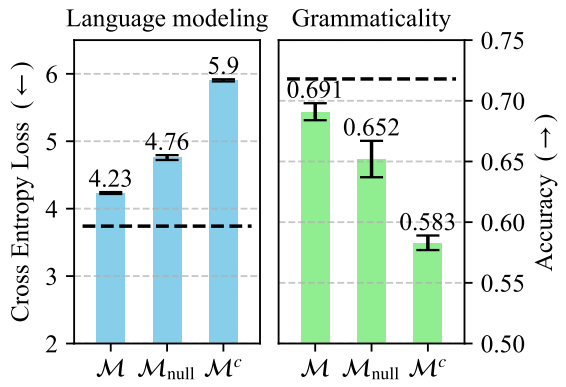

Figure 4: Language modeling and grammaticality performance for the learned subnetwork $\mathcal{M}$, its complement $\mathcal{M}^c$, and randomly sampled masks $\mathcal{M}_{\text{null}}$. $\mathcal{M}$ outperforms $\mathcal{M}^c$ and $\mathcal{M}_{\text{null}}$, indicating that the subnetwork learned during pre-pretraining continues to play a critical role after training on natural language. Dashed lines indicate performance of the base model without pruning.

procedure, performs much more poorly than $\mathcal{M}_{\text{null}}$, that only includes a random subset of them. That being said, while the performance $\mathcal{M}$ is close to the performance of the full network, it does not quite match it, indicating that attention heads outside of $\mathcal{M}$ also play a role in processing natural language.

A breakdown of grammaticality judgment accuracy by grammatical phenomenon shows that only a handful of phenomena are unaffected by masking, some substantially (e.g., the accuracy on subject-verb agreement drops 12 percentage points; see 9 in the Appendix). These phenomena appear to be ones that are syntactically simple but diagnose sensitivity to word structure (morphology), e.g., the distinction between *broke* and *broken*; we hypothesize that this aspect of linguistic knowledge is less likely to be mechanistically related to the processing of dependencies in a formal language.

6 Additional Analyses

This section reports three additional experiments. Due to the computational cost of pretraining, we focus on k-Shuffle Dyck, which performed well in our main experiments. First, we test and rule out the hypothesis that pre-pretraining on k-Shuffle Dyck is only effective because of its local statistical properties, and conclude that its effectiveness stems from its structural properties. Next, we study the impact of the vocabulary size hyperparameter k on the effectiveness of transfer from k-Shuffle Dyck. Finally, we perform a larger scale training run with Pythia 1B and find that pre-pretraining on k-Shuffle Dyck helps in this setting as well. In all of these experiments, we used the optimal number of pre-pretraining gradient updates $t_0^* = 500$ found in our main experiment (equivalent to 30 million tokens).

Transfer is not only due to local statistical properties. Could the successful transfer from k-Shuffle Dyck to natural language be due to the local statistical properties of k-Shuffle Dyck, rather than its dependency structure? Learning local statistical regularities is consistent with the finding that neural networks can exhibit **distributional simplicity bias** (DSB)—they learn simpler statistical patterns, such as the mean and covariance of their representations, before progressing to higher-order relationships (Saxe et al., 2014; LeCun et al., 1991); in particular, transformer language models learn n-gram statistics in order of increasing complexity (Belrose et al., 2024).

To test this hypothesis, we create variants of k-Shuffle Dyck that share its local statistics but not its global, rule-based structure. Concretely, we train unigram, bigram, and trigram models on the pre-pretraining corpus we generated from k-Shuffle Dyck for our main experiment, and, using these n-gram models, we generate "metamer datasets" (Kumar et al., 2022) of equivalent size.

We find that pre-pretraining on metamer datasets is strictly less effective than pre-pretraining on k-Shuffle Dyck, ruling out the hypothesis that the benefit of pre-pretraining on k-Shuffle Dyck is due to local statistics. That being said, pre-pretraining on the unigram metamer performs the worst, followed by bigram and trigram, suggesting that local statistics may explain part of the success of pre-pretraining on structured languages.

Larger vocabulary size may be beneficial. To check whether better hyperparameters exist for k-Shuffle Dyck, we sweep its vocabulary size, trying $k = 32, 128$ and 256 in addition to our previous experiments with $k = 64$. We find that $k = 128$ has the best performance across all metrics instead of $k = 64$, suggesting there likely do exist better hyperparameters.

Finding good ways to optimize these hyperparameters is an interesting area for future work. The hyperparameter tuning process for pre-pretraining is expensive, as evaluating the hyperparameters requires pretraining a language model. Nevertheless, various approximations such as early truncation exist in the hyperparameter tuning literature (Li et al., 2017; Swersky et al., 2013), and one can also use scaling laws to experiment at a smaller scale first (Yang et al., 2022).

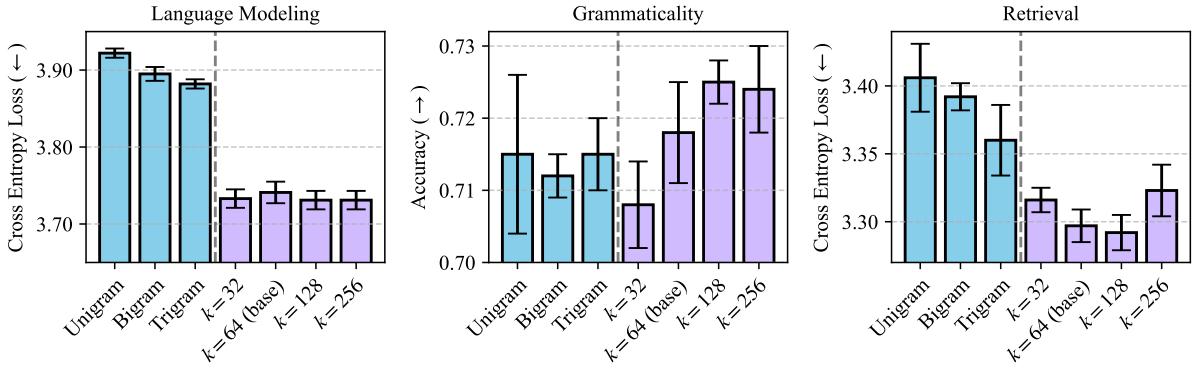

Figure 5: **Blue bars**: Testing if the benefit of pre-pretraining on k-Shuffle Dyck can be reduced to learning local statistics. We find that pre-pretraining on n-gram metamers (Kumar et al., 2022) of k-Shuffle Dyck performs worse than pre-pretraining on k-Shuffle Dyck itself. **Purple bar**: Vocabulary size manipulation. The best vocabulary size for k-Shuffle Dyck is $k=128$.

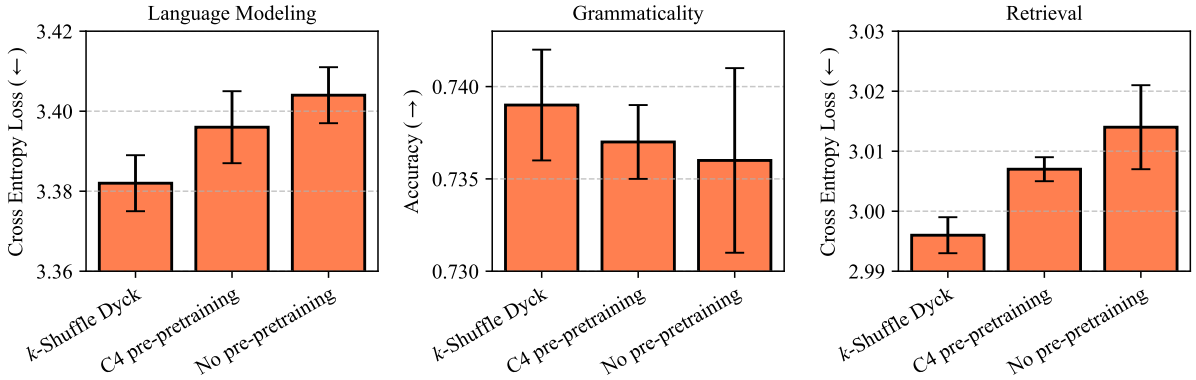

Figure 6: Pre-pretraining Pythia-1B on 1.6B tokens of k-Shuffle Dyck improves over the baselines, especially on language modeling and the retrieval evaluation.

Pre-pretraining is effective at the 1B scale too. Finally, we examine whether our results generalize to larger settings by training Pythia-1B on 1.63B tokens from C4 (25,000 steps). In this setting, pre-pretraining on k-Shuffle Dyck continues to outperform the baselines on all evaluation metrics (Figure 6) and achieves the final loss of the no-pre-pretraining baseline after training for only 1.10B total tokens. This equates to a token efficiency gain of 33% $\left(1-\frac{1.10\text{B}}{1.63\text{B}}\right)$, or an MRS of 17.3 $\left(\frac{1.63\text{B}-1.10\text{B}}{0.03\text{B}}\right)$. k-Shuffle Dyck's MRS is $\gg 1$ for both 160M and 1B training runs, suggesting that pre-pretraining could increase the efficiency of large-scale pretraining as well.

7 Related Work

The goal of pre-pretraining is similar to that of optimization-based meta-learning, which aims to create a weight initialization that allows the model to rapidly learn new tasks (Finn et al., 2017; Nichol et al., 2018) and languages (McCoy et al., 2020a; McCoy and Griffiths, 2025). The beneficial effect of pre-pretraining on formal language is consistent with the evidence of transfer from source code to natural language, especially for structured tasks (Petty et al., 2025; Aryabumi et al., 2025; Kim et al., 2024). In the vision domain, Nakamura et al. (2024) show that a thousand synthetically generated images can replace a million images from ImageNet-1k, in a similar spirit to our work.

Transfer in NLP has also been studied across different languages and domains (Ruder et al., 2019; Pruksachatkun et al., 2020; Deshpande et al., 2022). Most relevant to our work, Mueller and Linzen (2023) show that pretraining on child-directed speech gives a better inductive bias for learning hierarchical syntactic features than standard pretraining corpora. Furthermore, introducing a small amount of synthetic, disambiguating data into pretraining can induce a language model to change its generalization strategy (Warstadt et al., 2020b). Related but distinct from our approach are studies that use synthetic data

sampled from a formal language to evaluate models' generalization behavior in a controlled way (McCoy et al., 2019; Kim and Linzen, 2020; Li et al., 2023).

Pre-pretraining is a form of curriculum learning (Bengio et al., 2009), the approach of actively adjusting properties of the data during training. Recent work has developed algorithms that automate the discovery of language modeling curricula (Chen et al., 2025; Jiang et al., 2025), and many language model training recipes introduce different data mixtures at different stages (Allal et al., 2025; OLMo et al., 2025; Ouyang et al., 2022). The positive results of our experiments contrast with the largely negative results of the attempts to improve language models' data efficiency via linguistics-inspired curriculum learning, as part of the BabyLM challenge (Warstadt et al., 2023; Hu et al., 2024), pointing to the crucial effect of the particular data presented as part of the curriculum.

8 Discussion

We have found that pre-pretraining on formal languages can improve the language modeling loss and linguistic generalization abilities of transformer language models. In fact, pre-pretraining on some formal languages was more effective than increasing the amount of natural language training data by the same amount: the inductive bias conferred by the formal language was more helpful than additional in-distribution data. While most of the experiments in this paper were with 160M-parameter models, we also found benefits from pre-pretraining in the 1B-parameter setting.

We hypothesized that the languages that are most effective in this paradigm are those that, first, feature hierarchical dependencies, and second, are representable in C-RASP, and therefore readily learnable by transformers. The first part of this hypothesis is supposed by the superior performance of k-Dyck and k-Shuffle Dyck, the languages with hierarchical dependencies, relative to other languages. Our evidence for the importance of expressibility in C-RASP is less clear: k-Shuffle Dyck clearly outperformed k-Dyck, which is not expressible in C-RASP, on one of the evaluations, and also required fewer steps of pre-pretraining than the other languages (§4). While natural language is arguably context-sensitive, in the Chomsky hierarchy sense, not every context-sensitive language was beneficial in pre-pretraining: in fact, pre-pretraining was harmful when we used the copy language ww, which, while context-sensitive since it contains cross-serial dependencies, does not illustrate the notion of hierarchy and is not definable in C-RASP. That being said, since k-Dyck performed almost as well as k-Shuffle Dyck, there may exist a sharper characterization of the class of formal languages that confer a helpful inductive bias than defined by our expressivity hypothesis; experiments with a larger sample of formal languages would be needed to progress towards such a characterization.

The marginal rate of substitution between formal and natural language is greater than one (Table 5), meaning that one token of formal language in pre-pretraining substitutes for more than one token of natural language in pretraining. This is a surprising result from the perspective of statistical learning theory (Vapnik, 2000), in that we observe faster convergence by swapping in data from a *different distribution*. We hypothesize that initialization can have a critical effect on learning dynamics (McCoy et al., 2020b; Sellam et al., 2022), and pre-pretraining on formal language produces a initialization that is favorable to natural language learning.

9 Limitations

In this work, we considered blocked training, where we first train on formal language and then on natural language. While blocked training has the advantage that the initialization produced by formal language pre-pretraining can then be distributed and easily plugged into existing pretraining pipelines, it is possible that the optimal training regimen involves mixing formal and natural language during training (Korbak et al., 2023). We also evaluated the effectiveness of pre-pretraining in a setting where natural language pretraining data is plentiful, as it is for English, such that it is possible to train the model for a considerable number of tokens without processing the same data multiple times over several epochs. We hypothesize that pre-pretraining will be even more effective for low-resource natural languages, and may yield different scaling properties with respect to pre-pretraining data (Muennighoff et al., 2023). Relatedly, a natural extension to this project is establishing scaling laws for pre-pretraining; the benefit of pre-pretraining beyond 1 billion parameters and 1.6 billion tokens is currently unknown. Finally, our work only considers transformers. Circuit complexity has also quantified the expressive power of neural networks like RNNs and state-space models (Merrill et al., 2020, 2024), and it would be interesting to extend our results to these architectures.

Acknowledgments

Many thanks to Andy Yang, Angelica Chen, Dan Friedman, Isabel Papadimitriou, Lindia Tjuatja, Mayee Chen, Qingyang Zhu, and the NYU Computation and Psycholinguistics Lab for feedback and discussion. This work was supported in part through the NYU IT High Performance Computing resources, services, and staff expertise. This project is supported by the National Science Foundation (NSF) under grant NRT-HDR: FUTURE as well as Grant No. IIS-2239862. MYH is supported by the NSF Graduate Research Fellowship. WM is supported by the NSF Graduate Research Fellowship as well as the Two Sigma PhD Fellowship.

References

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. 2025. SmolLM2: When Smol goes big – data-centric training of a small language model. *Preprint*, arXiv:2502.02737.

Kristijan Armeni, Christopher Honey, and Tal Linzen. 2022. Characterizing verbatim short-term memory in neural language models. In *Proceedings of the 26th Conference on Computational Natural Language Learning (CoNLL)*, pages 405–424, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.

Kristijan Armeni, Marko Pranjić, and Senja Pollak. 2024. Transformer verbatim in-context retrieval across time and scale. In *Proceedings of the 28th Conference on Computational Natural Language Learning*, pages 56–68, Miami, FL, USA. Association for Computational Linguistics.

Viraat Aryabumi, Yixuan Su, Raymond Ma, Adrien Morisot, Ivan Zhang, Acyr Locatelli, Marzieh Fadaee, Ahmet Üstün, and Sara Hooker. 2025. To Code or Not To Code? Exploring Impact of Code in Pre-training. In *The Thirteenth International Conference on Learning Representations*.

Y Bar-Hillel, M Perles, and E Shamir. 1961. On formal properties of simple phrase structure grammars. *Language Typology and Universals*, 14:143–172.

Nora Belrose, Quintin Pope, Lucia Quirke, Alex Troy Mallen, and Xiaoli Fern. 2024. Neural Networks Learn Statistics of Increasing Complexity. In *Forty-first International Conference on Machine Learning*.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA. Association for Computing Machinery.

Adithya Bhaskar, Dan Friedman, and Danqi Chen. 2024. The heuristic core: Understanding subnetwork generalization in pretrained language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14351–14368, Bangkok, Thailand. Association for Computational Linguistics.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. 2023. Pythia: a suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning*, ICML 2023. JMLR.org.

Alexandra Butoi, Ghazal Khalighinejad, Anej Svete, Josef Valvoda, Ryan Cotterell, and Brian DuSell. 2025. Training Neural Networks as Recognizers of Formal Languages. In *The Thirteenth International Conference on Learning Representations*.

Mayee F Chen, Michael Y. Hu, Nicholas Lourie, Kyunghyun Cho, and Christopher Re. 2025. Aioli: A Unified Optimization Framework for Language Model Data Mixing. In *The Thirteenth International Conference on Learning Representations*.

Cheng-Han Chiang and Hungyi Lee. 2022. On the Transferability of Pre-trained Language Models: A Study from Artificial Datasets. In *The Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI-22)*.

Noam Chomsky. 1956. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124.

Noam Chomsky. 1959. On Certain Formal Properties of Grammars. *Information and Control*, 2(2):137–167.

Noam Chomsky and M.P. Schützenberger. 1959. The Algebraic Theory of Context-Free Languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, volume 26 of *Studies in Logic and the Foundations of Mathematics*, pages 118–161. Elsevier.

Ameet Deshpande, Partha Talukdar, and Karthik Narasimhan. 2022. When is BERT multilingual? isolating crucial ingredients for cross-lingual transfer. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3610–3623, Seattle, United States. Association for Computational Linguistics.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1126–1135. JMLR.org.

Yiding Hao, Dana Angluin, and Robert Frank. 2022. Formal Language Recognition by Hard Attention Transformers: Perspectives from Circuit Complexity. *Transactions of the Association for Computational Linguistics*, 10:800–810.

Hansi Hettiarachchi, Tharindu Ranasinghe, Paul Rayson, Ruslan Mitkov, Mohamed Gaber, Damith Premasiri, Fiona Anting Tan, and Lasitha Randunu Chandrakantha Uyangodage. 2025. Overview of the First Workshop on Language Models for Low-Resource Languages (LoResLM 2025). In *Proceedings of the First Workshop on Language Models for Low-Resource Languages*, pages 1–8, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. 2006. *Introduction to automata theory, languages, and computation*, 3 edition. Pearson.

Michael Y. Hu, Aaron Mueller, Candace Ross, Adina Williams, Tal Linzen, Chengxu Zhuang, Ryan Cotterell, Leshem Choshen, Alex Warstadt, and Ethan Gotlieb Wilcox. 2024. Findings of the Second BabyLM Challenge: Sample-Efficient Pretraining on Developmentally Plausible Corpora. In *The 2nd BabyLM Challenge at the 28th Conference on Computational Natural Language Learning*, pages 1–21, Miami, FL, USA. Association for Computational Linguistics.

Xinting Huang, Andy Yang, Satwik Bhattamishra, Yash Sarrof, Andreas Krebs, Hattie Zhou, Preetum Nakkiran, and Michael Hahn. 2025. A Formal Framework for Understanding Length Generalization in Transformers. In *The Thirteenth International Conference on Learning Representations*.

Yiding Jiang, Allan Zhou, Zhili Feng, Sadhika Malladi, and J Zico Kolter. 2025. Adaptive Data Optimization: Dynamic Sample Selection with Scaling Laws. In *The Thirteenth International Conference on Learning Representations*.

Najoung Kim and Tal Linzen. 2020. COGS: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9087–9105, Online. Association for Computational Linguistics.

Najoung Kim, Sebastian Schuster, and Shubham Toshniwal. 2024. Code Pretraining Improves Entity Tracking Abilities of Language Models. *Preprint*, arXiv:2405.21068.

Tomasz Korbak, Kejian Shi, Angelica Chen, Rasika Bhalerao, Christopher L. Buckley, Jason Phang, Samuel R. Bowman, and Ethan Perez. 2023. Pretraining language models with human preferences. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Sreejan Kumar, Ishita Dasgupta, Raja Marjieh, Nathaniel D. Daw, Jonathan D. Cohen, and Thomas L. Griffiths. 2022. Disentangling Abstraction from Statistical Pattern Matching in Human and Machine Learning. *PLOS Computational Biology*, 19.

S.-Y. Kuroda. 1964. Classes of languages and Linear-Bounded Automata. *Information and Control*, 7(2):207–223.

Yann LeCun, Ido Kanter, and Sara A. Solla. 1991. Eigenvalues of covariance matrices: Application to neural-network learning. *Physical Review Letters*, 66:2396–2399.

Bingzhi Li, Lucia Donatelli, Alexander Koller, Tal Linzen, Yuekun Yao, and Najoung Kim. 2023. SLOG: A structural generalization benchmark for semantic parsing. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3213–3232, Singapore. Association for Computational Linguistics.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. 2017. Hyperband: a novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(1):6765–6816.

David Lindner, Janos Kramar, Sebastian Farquhar, Matthew Rahtz, Thomas McGrath, and Vladimir Mikulik. 2023. Tracr: Compiled Transformers as a Laboratory for Interpretability. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Qin Liu, Fei Wang, Chaowei Xiao, and Muhao Chen. 2024. From shortcuts to triggers: Backdoor defense with denoised PoE. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 483–496, Mexico City, Mexico. Association for Computational Linguistics.

Rebecca Marvin and Tal Linzen. 2018. Targeted syntactic evaluation of language models. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1192–1202, Brussels, Belgium. Association for Computational Linguistics.

R Thomas McCoy, Erin Grant, Paul Smolensky, Thomas L Griffiths, and Tal Linzen. 2020a. Universal linguistic inductive biases via meta-learning. In *Proceedings of the 42nd Annual Conference of the Cognitive Science Society*, pages 737–743.

R. Thomas McCoy and Thomas L. Griffiths. 2025. Modeling rapid language learning by distilling Bayesian priors into artificial neural networks. *Nature Communications*.

R. Thomas McCoy, Junghyun Min, and Tal Linzen. 2020b. BERTs of a feather do not generalize together: Large variability in generalization across models with similar test set performance. In *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 217–227, Online. Association for Computational Linguistics.

Tom McCoy, Ellie Pavlick, and Tal Linzen. 2019. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3428–3448, Florence, Italy. Association for Computational Linguistics.

William Merrill, Jackson Petty, and Ashish Sabharwal. 2024. The Illusion of State in State-Space Models. In *Forty-first International Conference on Machine Learning*.

William Merrill and Ashish Sabharwal. 2023. A Logic for Expressing Log-Precision Transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*.

William Merrill, Nikolaos Tsilivis, and Aman Shukla. 2023. A Tale of Two Circuits: Grokking as Competition of Sparse and Dense Subnetworks. *Preprint*, arXiv:2303.11873.

William Merrill, Gail Weiss, Yoav Goldberg, Roy Schwartz, Noah A. Smith, and Eran Yahav. 2020. A formal hierarchy of RNN architectures. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 443–459, Online. Association for Computational Linguistics.

David A. Mix Barrington, Neil Immerman, and Howard Straubing. 1990. On uniformity within NC1. *Journal of Computer and System Sciences*, 41(3):274–306.

Aaron Mueller and Tal Linzen. 2023. How to plant trees in language models: Data and architectural effects on the emergence of syntactic inductive biases. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11237–11252, Toronto, Canada. Association for Computational Linguistics.

Niklas Muennighoff, Alexander M. Rush, Boaz Barak, Teven Le Scao, Aleksandra Piktus, Nouamane Tazi, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. 2023. Scaling Data-Constrained Language Models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NeurIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Ryo Nakamura, Ryu Tadokoro, Ryosuke Yamada, Yuki M. Asano, Iro Laina, Christian Rupprecht, Nakamasa Inoue, Rio Yokota, and Hirokatsu Kataoka. 2024. Scaling Backwards: Minimal Synthetic Pre-Training? In *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part XV*, page 153–171, Berlin, Heidelberg. Springer-Verlag.

Alex Nichol, Joshua Achiam, and John Schulman. 2018. On First-Order Meta-Learning Algorithms. *ArXiv*, abs/1803.02999.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2025. 2 OLMo 2 Furious. *Preprint*, arXiv:2501.00656.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NeurIPS 2022, Red Hook, NY, USA. Curran Associates Inc.

Isabel Papadimitriou and Dan Jurafsky. 2020. Learning Music Helps You Read: Using transfer to study linguistic structure in language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6829–6839, Online. Association for Computational Linguistics.

Isabel Papadimitriou and Dan Jurafsky. 2023. Injecting structural hints: Using language models to study inductive biases in language learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 8402–8413, Singapore. Association for Computational Linguistics.

Jill Gilkersonand Jeffrey A. Richardsand Steven F. Warrenand Judith K. Montgomeryand Charles R. Greenwoodand D. Kimbrough Ollerand John H. L. Hansenand Terrance D. Paul. 2017. Mapping the Early Language Environment Using All-Day Recordings and Automated Analysis. *American Journal of Speech-Language Pathology*, 26(2):248–265.

Jackson Petty, Sjoerd van Steenkiste, and Tal Linzen. 2025. How Does Code Pretraining Affect Language Model Task Performance? *Transactions on Machine Learning Research*.

Yada Pruksachatkun, Jason Phang, Haokun Liu, Phu Mon Htut, Xiaoyi Zhang, Richard Yuanzhe Pang, Clara Vania, Katharina Kann, and Samuel R. Bowman. 2020. Intermediate-task transfer learning with pretrained language models: When and why does it work? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5231–5247, Online. Association for Computational Linguistics.

Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21:140:1–140:67.

Sebastian Ruder, Matthew E. Peters, Swabha Swayamdipta, and Thomas Wolf. 2019. Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pages 15–18, Minneapolis, Minnesota. Association for Computational Linguistics.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. 2014. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations*.

Thibault Sellam, Steve Yadlowsky, Ian Tenney, Jason Wei, Naomi Saphra, Alexander D'Amour, Tal Linzen, Jasmijn Bastings, Iulia Raluca Turc, Jacob Eisenstein, Dipanjan Das, and Ellie Pavlick. 2022. The MultiBERTs: BERT Reproductions for Robustness Analysis. In *International Conference on Learning Representations*.

Stuart M. Shieber. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343.

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

Mirac Suzgun, Yonatan Belinkov, Stuart Shieber, and Sebastian Gehrmann. 2019. LSTM networks can perform dynamic counting. In *Proceedings of the Workshop on Deep Learning and Formal Languages: Building Bridges*, pages 44–54, Florence. Association for Computational Linguistics.

Kevin Swersky, Jasper Snoek, and Ryan P. Adams. 2013. Multi-task Bayesian Optimization. In *Neural Information Processing Systems*.

Vladimir Naumovich Vapnik. 2000. The Nature of Statistical Learning Theory. In *Statistics for Engineering and Information Science*.

Pablo Villalobos, Anson Ho, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, and Marius Hobbhahn. 2024. Will we run out of data? Limits of LLM scaling based on human-generated data. *Preprint*, arXiv:2211.04325.

Alex Warstadt, Aaron Mueller, Leshem Choshen, Ethan Wilcox, Chengxu Zhuang, Juan Ciro, Rafael Mosquera, Bhargavi Paranjabe, Adina Williams, Tal Linzen, and Ryan Cotterell. 2023. Findings of the BabyLM challenge: Sample-efficient pretraining on developmentally plausible corpora. In *Proceedings of the BabyLM Challenge at the 27th Conference on Computational Natural Language Learning*, pages 1–34, Singapore. Association for Computational Linguistics.

Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. 2020a. BLiMP: The benchmark of linguistic minimal pairs for English. *Transactions of the Association for Computational Linguistics*, 8:377–392.

Alex Warstadt, Yian Zhang, Xiaocheng Li, Haokun Liu, and Samuel R. Bowman. 2020b. Learning which features matter: RoBERTa acquires a preference for linguistic generalizations (eventually). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 217–235, Online. Association for Computational Linguistics.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2021. Thinking Like Transformers. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139, pages 11080–11090.

Ethan Gotlieb Wilcox, Michael Hu, Aaron Mueller, Alex Warstadt, Leshem Choshen, Chengxu Zhuang, Adina Williams, Ryan Cotterell, and Tal Linzen. 2025. Bigger is not always better: The importance of human-scale language modeling for psycholinguistics. *Journal of Memory and Language*.

Mengzhou Xia, Zexuan Zhong, and Danqi Chen. 2022. Structured pruning learns compact and accurate models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1513–1528, Dublin, Ireland. Association for Computational Linguistics.

Andy Yang and David Chiang. 2024. Counting Like Transformers: Compiling Temporal Counting Logic Into Softmax Transformers. In *First Conference on Language Modeling*.

Andy Yang, David Chiang, and Dana Angluin. 2024. Masked Hard-Attention Transformers Recognize Exactly the Star-Free Languages. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Greg Yang, J. Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub W. Pachocki, Weizhu Chen, and Jianfeng Gao. 2022. Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer. *ArXiv*, abs/2203.03466.

Fred Zhang and Neel Nanda. 2024. Towards Best Practices of Activation Patching in Language Models: Metrics and Methods. In *The Twelfth International Conference on Learning Representations*.

Tianyang Zhong, Zhenyuan Yang, Zhengliang Liu, Ruidong Zhang, Yiheng Liu, Haiyang Sun, Yi Pan, Yiwei Li, Yifan Zhou, Hanqi Jiang, Junhao Chen, and Tianming Liu. 2024. Opportunities and Challenges of Large Language Models for Low-Resource Languages in Humanities Research. *Preprint*, arXiv:2412.04497.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Joshua M. Susskind, Samy Bengio, and Preetum Nakkiran. 2024. What Algorithms can Transformers Learn? A Study in Length Generalization. In *The Twelfth International Conference on Learning Representations*.

A Proofs

We make use of the following to establish that all languages we consider are context-sensitive.

Lemma A.1. *Any language definable in FO(M) can be recognized by a context-sensitive grammar.*

Proof. Mix Barrington et al. (1990) show that the class of languages definable in $\mathsf{FO(M)}$ is $\mathsf{LOGTIME}$-uniform TC^0, which is a subset of $\mathsf{L} = \mathsf{SPACE}(\log n)$. On the other hand, the context-sensitive languages are those languages recognizable by linear-bounded automata (Kuroda, 1964). That is, $\mathsf{CSL} = \mathsf{NSPACE}(n)$. Putting these characterizations together, we see that

$$\mathsf{TC}^0 \subseteq \mathsf{SPACE}(\log n) \subseteq \mathsf{NSPACE}(n) = \mathsf{CSL}.$$

Therefore we can conclude that any language definable in $\mathsf{FO(M)}$ is context-sensitive. □

We will make use of the classical pumping lemma to establish that some specific languages considered are *strictly* context-sensitive, i.e., not context-free.

Lemma A.2 (Pumping Lemma, Bar-Hillel et al., 1961)**.** *Let L be a context-free language. Then there exists a pumping constant $p > 0$ such that any string $s \in L$ of length $|s| \geq p$ can be written as $s = uvwxy$ where*

1. *$|vwx| \leq p$;*
2. *$|vx| \geq 1$; and*
3. *$uv^n wx^n y \in L$ for all $n \geq 0$.*

Additionally, we will leverage the following communication complexity result to prove that certain languages are undefinable in C-RASP:

Lemma A.3 (Huang et al., 2025, Theorem 12)**.** *Let L be a language definable in C-RASP. Fix $w \in L$ and $1 \leq i \leq |w|$. Let Alice have access to $w_{1:i}$ and Bob have access to $w_{i+1:|w|}$. Then there exists a protocol where Alice sends at most $O(\log n)$ bits to Bob and Bob can recognize whether $w \in L$.*

Crucially, if some L requires Alice to send Bob $\omega(\log n)$ bits, then it cannot be defined in C-RASP.

We will also use the equivalence between C-RASP and the Temporal Counting Logic $\mathsf{K}_t[\#]$ to show that languages are definable in C-RASP.

Lemma A.4 (Yang and Chiang, 2024, Theorem 4.3)**.** *A C-RASP program recognizes language L if and only if a $\mathsf{K}_t[\#]$ formula defines L.*

A.1 Language Characterizations

Proposition A.5. *1-Dyck is context-free and definable in C-RASP.*

Proof. That 1-Dyck is context-free follows from the fact that it can be generated by the following context-free grammar:

$$S \to (S)S,$$
$$S \to \varepsilon.$$

1-Dyck is defined by the following $\mathsf{K}_t[\#]$ formula

$$(\#[Q_(] = \#[Q_)]) \wedge (\#[\#[Q_)] > \#[Q_(]] = 0),$$

and so is implementable in C-RASP by Lemma A.4. □

Proposition A.6. *For $k \geq 2$, k-Dyck is context-free and not definable in C-RASP.*

Proof. That k-Dyck is context-free follows from the fact that it can be generated by a context-free grammar: for any fixed value of k, k-Dyck is generated by

$$S \to ({}_i S)_i S, \quad \text{where } i \in [k]$$
$$S \to \varepsilon$$

To see that k-Dyck is not definable in C-RASP, consider Dyck strings of length $2n$ where tokens 1 to n are opening braces, and tokens $n+1$ to $2n$ are closing braces. If Alice receives the first n tokens, she must send $\Omega(n)$ bits to Bob if Bob is to correctly recognize the input string, because each prefix has a different unique suffix that closes it. So k-Dyck is not in C-RASP by Lemma A.3. □

On the other hand, k-Dyck can be defined in $\mathsf{FO(M)}$.

Proposition A.7. *For $k \geq 1$, k-Dyck is definable in FO(M).*

Proof. Let $Q_(i)$ check whether token i is *any* of the k opening parentheses, and $Q_{(\kappa}(i)$ check whether token i is the κth opening parenthesis out of k. Continuing the definition from Section 2.2:

$$\mathsf{depth}(i) \equiv \#j \leq i[Q_(i)] - \#j \leq i[Q_)(i)]$$
$$\mathsf{dindex}(i) \equiv \#j \leq i[\mathsf{depth}(i) = \mathsf{depth}(j)]$$
$$\mathsf{paired}(j,i) \equiv [\mathsf{depth}(j) = \mathsf{depth}(i)+1] \wedge [\mathsf{dindex}(i) = \mathsf{dindex}(j)]$$
$$\mathsf{match}(j,i) \equiv \bigvee_\kappa [Q_{(\kappa}(j) \wedge Q_{)\kappa}(i)]$$
$$\mathsf{closed}(i) \equiv \exists j \leq i[\mathsf{paired}(j,i) \wedge \mathsf{match}(j,i)]$$

Having defined these macros, we are now ready to write the recognizer for k-Dyck:

$$[\mathsf{depth}(n) = 0] \wedge [\#i \leq n[\mathsf{depth}(i) < 0] = 0] \wedge \forall i \leq n[\mathsf{closed}(i)]$$

□

To understand why this construction cannot be implemented in C-RASP, observe that paired(j,i) and match(j,i) are binary predicates, which are not allowed in C-RASP.

Lemma A.8. *For $k \geq 2$, k-Shuffle Dyck is strictly context-sensitive and definable in C-RASP.*

Proof. See Ex. 7.20 in Hopcroft et al. (2006). Consider the case when $k=2$. Assume that 2-shuffle Dyck is context-free. Then $L = ({}^{n}[{}^{m})^{n}]^{m}$ is context-free since it is the intersection of k-Shuffle Dyck with $({}^{*}[{}^{*})^{*}]^{*}$ and CFLs are closed under intersection with regular languages.

Assume by way of contradiction that L is context-free and so has pumping constant p. Let $s = ({}^{p}[{}^{p})^{p}]^{p}$, whichy by hypothesis can be written as $uvwxy$. Since $|vwx| \leq p$, it either (a) lies entirely inside one of the blocks of p symbols or (b) lies partially in one block of p symbols and lies partially in at most one adjacent block. In the case of (a), suppose for clarity that vwx lies entirely in the $({}^{p}$ block. Since vx is not empty, $uv^0wx^0y \equiv uwy$ contains fewer (’s than)’s, and hence is not in L, a contradiction. In the case of (b), suppose for clarity that vwx straddles the $({}^{p}$ and $[{}^{p}$ blocks. Since vx is not empty, $uv^0wx^0y \equiv uwy$ contains either fewer (’s than)’s or fewer [’s than]’s, and hence is not in L, a contradiction. Since k-Shuffle Dyck for $k > 2$ contains 2-Shuffle Dyck, proving the $k=2$ case is sufficient to establish that k-Shuffle Dyck is not context free (but still context-sensitive by Lemma A.1).

Similar to the 1-Dyck case, we can exhibit a $\mathsf{K}_t[\#]$ formula to define k-Shuffle Dyck:

$$\bigwedge_{\kappa} \left(\#[Q_{(\kappa}] = \#[Q_{)\kappa}]\right) \wedge \left(\#[\#[Q_{)\kappa}] > \#[Q_{(\kappa}]] = 0\right)$$

So k-Shuffle Dyck is likewise definable in C-RASP by Lemma A.4. □

Proposition A.9. *ww is strictly context-sensitive and not definable in C-RASP.*

Proof. See Ex. 7.21 in Hopcroft et al. (2006). Suppose by way of contradiction that ww is context-free, and so has a pumping constant p. Let $s = \mathsf{a}^p\mathsf{b}^p\mathsf{a}^p\mathsf{b}^p$, which can be written as $uvwxy$ by hypothesis. Consider then the string $uv^0wx^0y \equiv uwy \in L$. We examine two cases depending on the position of vwx in s.

In the first case, suppose vx is contained entirely within the first block of a^p. If $|vx| = k$ then uwy has length $4p - k$ and begins with the substring $\mathsf{a}^{(p-k)}\mathsf{b}^p\ldots$ of length $2p-k$. By assumption $uwy = tt$ for some t of length $2p - k/2$, and since $k \geq 1$ it follows that $|t| > 2p - k$. Then the final symbol of t must lie within the *second* block of a’s; yet since s ends in b, tt must also end in b, a contradiction.

In the second case, suppose vx contains some a’s and some b’s. Since $|uvwxy| = 4p$ and $|vwx| \leq p$ it must be that $|uwy| \geq 3p$ and so $|t| = 3p/2$. Since vwx is too short to straddle more than two adjacent blocks of symbols and $3p/2 > p$ it must be the case that t must end in b^p. Yet there since $|vx| \geq 1$, there is only a single block of b^p within $|uwy|$, so the b^p block cannot be repeated, a contradiction.

By symmetry, these two cases straightforwardly extend to the cases when vx is contained entirely within the first block of b’s, the second block of a’s, or the second block of b’s (analogous to case 1); or when it is split between the blocks of a’s and b’s (case 2). Then ww is not context-free, but still context-sensitive by Lemma A.1.

From a communication complexity perspective, if Alice has the first half of some string, and Bob has the second half, Alice must send Bob $\Omega(n)$ bits to verify whether the string is of the form ww. Thus, by Lemma A.3, ww cannot be defined in C-RASP. □

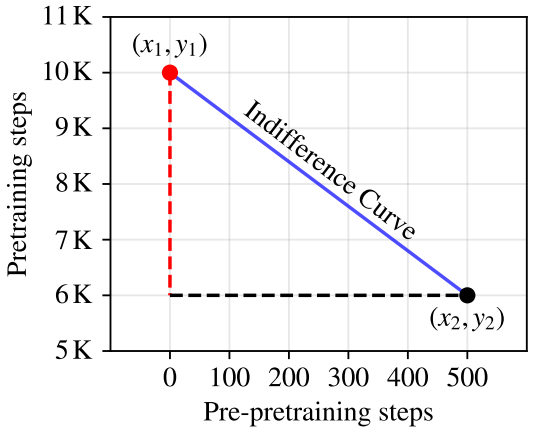

Figure 7: The indifference curve contains points with equal training loss. Marginal rate of substitution is the ratio between the red and black lines $\left(\frac{|y_1 - y_2|}{|x_1 - x_2|}\right)$. The token efficiency increase from applying pre-pretraining can be calculated as $1 - \frac{x_2 + y_2}{y_1}$.

B Hyperparameters

All experiments were done on NVIDIA A100 or H100 80GB GPUs. We warm up the learning rate both during pre-pretraining and pretraining. The below hyperparameters hold for both pre-pretraining and pretraining. That is, for simplicity, even if we only pre-pretrain for 500 steps, we still keep the

learning rate warmup at 1,000 steps. To achieve 50% attention head sparsity when pruning, we set the target sparsity to 70%. We used Huggingface `transformers==4.47.0` and `datasets==3.2.0`.

Hyperparameter	**Value**
Training Configuration	
Batch size	16
Gradient accumulation	2
Effective bsz	32
Sequence length	2048 tokens
Learning rate	5×10^{-4}
LR schedule	Cosine w/ warmup
Min. LR	5×10^{-5}
Warmup Steps	1000
Weight Decay	0.1
Gradient Clipping	1.0
Optimization	
Optimizer	AdamW
β_1,β_2	0.9, 0.999
ϵ	1e-6
Mixed Precision	bf16
Pruning (see Bhaskar et al. (2024))	
Learning rate	0.1
Regularization LR	1
Target sparsity	0.7
Warmup steps	1000

Table 2: Training hyperparameters.

Positive Example	Negative Example
✓ Only Bill would ever complain.	✗ Even Bill would ever complain.
✓ Diane watched Alan.	✗ Diane screamed Alan.
✓ Who should Derek hug after shocking Richard?	✗ Who should Derek hug Richard after shocking?

Table 3: Examples from the BLiMP dataset (Warstadt et al., 2020a): matched pairs of grammatical (positive, left) and ungrammatical (negative, right) sentences. We expect the language model to assign a higher probability to the grammatical sentence in each pair.

Examples
Before the meeting, Mary wrote down the following list of words: **window, door, roof**. After the meeting, she took a break and had a cup of coffee. When she got back, she read the list again: **window, door, roof**.
Before the meeting, John wrote down the following list of words: **nothing, riches, paper**. After the meeting, he took a break and had a cup of coffee. When he got back, he read the list again: **nothing, riches, paper**.

Table 4: Verbatim in-context retrieval (Armeni et al., 2022, 2024) examples. We expect a good language model to recognize based on the context that the list is repeated, retrieve the appropriate items from the first repetition of the list, and assign these items a very high probability.

Figure 8: Accuracy on BLiMP by grammatical phenomenon. The full names of the phenomena are: anaphor agreement, argument structure, binding, control/raising, determiner-noun agreement, ellipsis, filler-gap dependencies, irregular forms, island effects, negative polarity item licensing, quantifiers, and subject-verb agreement.

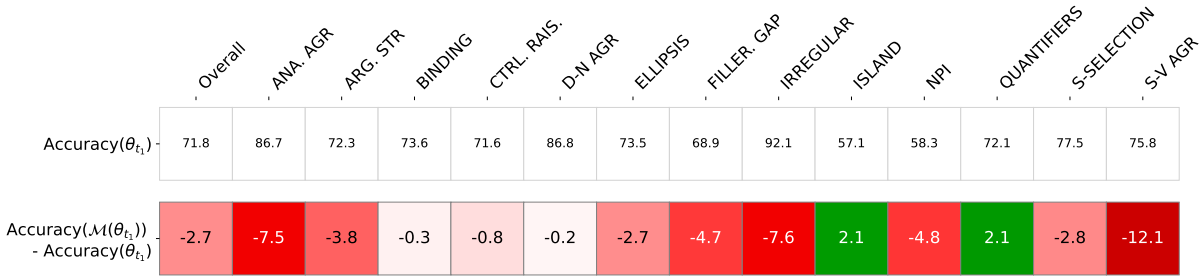

Figure 9: Performance changes on BLiMP after pruning half the attention heads from the model trained on k-Shuffle Dyck (see §5). The largest declines are on subject-verb agreement, irregular forms, and anaphor agreement. These categories require knowledge about word forms, and the sentences within these categories are generally simple (around 4 words).

	Language	LM Loss	Documents ↓	MRS	Grammaticality	Retrieval
Formal	1-Dyck	3.760 ±0.016	0.978 ±0.021	3.01	0.717 ±0.004	3.373 ±0.014
	k-Dyck	3.743 ±0.016	**0.998 ±0.001**	3.57	**0.719 ±0.003**	3.338 ±0.019
	k-Shuffle Dyck	**3.741 ±0.014**	**0.998 ±0.001**	**7.15**	0.718 ±0.007	**3.297 ±0.012**
	ww	3.792 ±0.018	0.557 ±0.247	−0.25	0.714 ±0.003	3.341 ±0.021
Controls	No pre-pretraining	3.780 ±0.018	—	—	0.710 ±0.011	3.393 ±0.003
	C4 pre-pretraining	3.754 ±0.017	0.992 ±0.007	6.65	0.710 ±0.003	3.354 ±0.005
	Random binary	3.810 ±0.015	0 ±0	−6.60	0.712 ±0.004	3.416 ±0.016
	Random ints	3.798 ±0.015	0.042 ±0.041	−5.97	0.712 ±0.006	3.409 ±0.006

Table 5: Evaluating models at the optimal amount of pre-pretraining t_0^* for each formal language (see §4). "Documents ↓" is the proportion of documents in the C4 validation set where the model has a lower loss than the model trained without pre-pretraining. 1-Dyck, k-Dyck, and k-Shuffle-Dyck all have marginal rates of substitution (MRS) greater than 1, indicating that pre-pretraining is more efficient than not pre-pretraining. k-Shuffle-Dyck performs the best overall on our evaluation metrics.

Figure 10: Implementation of a k-Shuffle Dyck sequence generator.

```python
import random


def generate_shuff_dyck(k, max_length=2048, p_open=0.5, max_depth=16):
    """
    Generate a k-shuffle Dyck sequence, truncated at max_length.
    When max depth is reached, close one bracket and continue.

    Args:
        k (int): Number of different types of brackets
        max_length (int): Target maximum length of the sequence
        p_open (float): Probability of opening a new bracket
        max_depth (int): Maximum nesting depth allowed

    Returns:
        list: Generated sequence where i represents opening bracket i
             and i+k represents closing bracket i

    Note: the final Dyck word may be invalid due to truncation, but
    we didn't find this to be an issue in practice.
    """
    sequence = []
    counts = [0] * k  # Track open brackets of each type

    while len(sequence) < max_length:
        depth = sum(counts)

        # Must open if all brackets are closed
        if depth == 0:
            bracket = random.randint(0, k - 1)
            sequence.append(bracket)
            counts[bracket] += 1
            continue

        # If at max depth, force a close
        if depth >= max_depth:
            open_brackets = [i for i, count in enumerate(counts) if count > 0]
            bracket = random.choice(open_brackets)
            sequence.append(bracket + k)
            counts[bracket] -= 1
            continue

        # Randomly choose to open or close
        if random.random() < p_open and depth < max_depth:
            bracket = random.randint(0, k - 1)
            sequence.append(bracket)
            counts[bracket] += 1
        else:
            # Close an existing bracket
            open_brackets = [i for i, count in enumerate(counts) if count > 0]
            bracket = random.choice(open_brackets)
            sequence.append(bracket + k)
            counts[bracket] -= 1

    return sequence
```