Learning Hierarchical Structures with Autoregressive Language Modelling

Luca Sfragara

June 13, 2025

Abstract

I study how a vanilla Transformer learns *hierarchical* structure by training GPT-2 with rotary position encodings on a family of synthetic context–free grammars and comparing it to an LSTM of similar size. After only 2×10^8 tokens, the Transformer achieves $> 99\%$ accuracy in both free generation and prefix-completion, while the LSTM remains below 10%. Attention visualizations reveal sharp head specialization: some heads copy the previous token, others jump to fixed offsets that mark non-terminal boundaries, and deeper layers forward this information with near-identity patterns. Targeted ablations that zero specific heads' outputs reduce accuracy exactly when the ablated head matches a grammar-critical role, suggesting functional causality. Although the model was never shown parse trees during training, its internal routing echoes the chart-based algorithms traditionally used to parse CFGs. The emergence of such structure parallels findings in cognitive neuroscience, where separate cortical rhythms track words, phrases, and sentences simultaneously. Together, these results suggest that multi-head self-attention provides the flexible memory required to discover compositional and hierarchical rules that recurrent networks struggle to capture.

Acknowledgments

To my parents.

Contents

1 Introduction

Transformers-based Large Language Models (LLMs) have rapidly become ubiquitous across research and industry, owing to their remarkable ability to model complex patterns in language, reason over long contexts, and generalize across a wide range of tasks. LLMs achieve state-of-the-art (SOTA) results in many NLP tasks without prior specific fine-tuning, exhibiting impressive few-shot, or zero-shot, and in-context learning abilities (Brown et al., 2020a).

Current state-of-the-art Transformer-based models comprise tens of billions of parameters distributed across multiple layers, rendering their inner workings highly complex and often opaque. Nonetheless, advancing interpretability remains crucial - not only for ensuring the safety and reliability of these systems, but also for expanding the frontier of what such models are capable of achieving.

There have been significant efforts in the field of interpretability, starting from induction heads and copy-paste mechanisms (Wang et al., 2022), to, more recently, unveiling the internal sequence of steps through which a model reaches the answer (Wang et al., 2025). For example, as shown by the work of Anthropic ((Wang et al., 2022), (Wang et al., 2025)) there are understood mechanisms and patterns through which GPT2 predicts the next token ”Mary” given the sentence ”When Mary and John went to the store, John gave a drink to ...”, as well as evidence towards the emergence of a planning ability when a transformer-based LLM is asked to write a rhyming poem.

Most of the current literature focuses on emergent abilities in the ”wild”, that is, through prompting models that are currently publicly available and trained on real text data. Thus, these attempts, while highly insightful, are limited by the noise embedded in the text, i.e. the training data, and the inability to model text in a fully controlled setting. In addition, most of the current mechanistic interpretability literature focuses on downstream and relatively complex tasks. For example, Anthropic’s seminal paper (Wang et al., 2025) studies circuits that lead to multistep-reasoning, arithmetic addition and so forth. However, all of these tasks first require a model to have the ability to *understand* text.

Intuitively, having a foundational understanding of natural language is subordinate to detecting sentiment, summarizing text, and many of the other emergent capabilities of LLMs. The notion of *understanding* is non-trivial and is explored in this thesis. I adopt a deliberately algorithmic notion of *understanding*: a model is said to understand a language if its internal computation can be mapped onto a well-defined procedure that decides if a string belongs to the language and if it can construct a valid derivation. This echoes the perspective of Allen-Zhu and Li (2023), who argue that genuine mechanistic insight is possible only when the task admits a ground-truth algorithm against which the inner workings of transformers can be compared. Natural text lacks such an oracle; by

contrast, context-free grammars (CFGs) supply both a generative process and textbook dynamic-programming (DP) parsers that certify grammaticality. CFGs consist of terminals and non-terminal symbols and production rules and, by definition, hierarchically produce complex strings. Working with CFGs therefore lets us pose a sharp question:

> *Can a purely autoregressive Transformer, trained only with next-token prediction, discover and internally implement the same DP reasoning that classical parsers use?*

To answer this, I generate artificial sentences from a Context-Free Grammar, whose validity hinges on deeply nested symbols. In fact, a single non-terminal introduced near the top of the derivation (e.g. at depth two) can expand into an arbitrarily long subtree, yielding hundreds of tokens in the surface string; thus correct generation or verification requires the model to propagate structural information across very long ranges rather than relying on local n-gram statistics. Indeed, if a model merely memorizes frequent n-grams it will fail; if it learns to propagate structural information in a DP-like fashion it will generalize perfectly, and the associated attention heads or feed-forward channels should exhibit algorithmic roles that we can probe and ablate.[1]

Thesis Structure. Section 2 introduces the inherent compositional and hierarchical nature of language. Section 3 introduces the data generation model used, Context-Free Grammars, and analyzes their properties and classical parsing algorithms. Section 4 and 5 present, respectively, the GPT model and the training procedure. Section 6 presents the experimental results, which are further analyzed in Section 7, which explores how transformers may learn hierarchical structures in the data. Lastly, Section 8 and Section 9 present limitations, next steps and conclusions.

Results. The main experimental results are:

- Result 1 - **Transformers learn CFGs while LSTMs do not**:
  A GPT2-small model with Rotary Positional Encoding is able to almost perfectly learn to generate and complete valid grammar strings.
- Result 2 - **Attention heads become highly specialized**:
  Attention heads, especially in the first layer, become highly specialized, each attending to different parts of the sequences, including long and short range dependencies.

[1]The complete code for the project is available on GitHub at `https://github.com/LucaSfragara/Thesis`.

- Result 3 - **Not all attention heads are equally important**:
  Pruning or ablating individual heads or a subset of them causes varying perplexity degradation depending on the function performed by the head, demonstrating a non-uniform contribution across heads.

- Result 4 - **Transformers' hidden states linearly encode NT-ancestor symbol**:
  A trained linear probe on a transformer layer's hidden state can almost perfectly recover NT ancestors symbols.

2 Compositional and Hierarchical Characteristics of Natural Language

Data, whether text, images, or other modalities, can be viewed as arising from an underlying data-generating process (DGP) that defines a hierarchy of compositional rules specifying how high-level constructs decompose into lower-level primitives (Cagnetta et al., 2023). For example, natural language is inherently hierarchical and compositional, building up meaning by recursively expanding higher-level concepts into multiple sub-parts (for example, a verb phrase expanded into the verb and its arguments, a word into its syllables and its syllables into characters). This conceptualization is particularly powerful: it allows us to understand why Deep Learning is so successful, as it exploits this hierarchical structure to learn the data - likely learning to "unfold" the space where such data resides (see next subsection).

2.1 Manifold Hypothesis

The Manifold Hypothesis is a widely accepted tenet of Deep Learning, which posits that (Cayton, 2005):

> *The dimensionality of many data sets is only artificially high; though while each data point consists of perhaps thousands of features, it may be described as a function of only a few underlying parameters. That is, the data points are actually samples from a low-dimensional manifold that is embedded in a high-dimensional space.*

Evidently, text data sits in a very high-dimensional space. Under the manifold hypothesis, however, high-dimensional observations concentrate near smooth, low-dimensional submanifolds embedded in the ambient space (the higher dimensional container in which a manifold "lives"),

reflecting the generative hierarchy of the data generation process (Whiteley et al., 2022). Contemporary deep architectures for language—most notably Transformer models—implicitly likely exploit this low-dimensional geometry by organizing linguistic features along hierarchical manifold branches. Lower layers tend to capture local syntactic and lexical regularities (e.g. word-level manifolds for part-of-speech or subword patterns), while deeper layers integrate these local charts into broader semantic abstractions, effectively "stitching together" multiple submanifolds into a coherent global representation. (Hewitt and Manning, 2019).

By contrast, generating synthetic text via a PCFG provides us with ground-truth manifold coordinate—each nonterminal expansion corresponds to a known submanifold—thus enabling controlled mechanistic studies of how Transformers traverse and compose along these nested geometric structures.

2.2 Understanding Natural Language and Grammars

Evidence for strong performance in downstream tasks as well as seminal work by Anthropic (Wang et al., 2025) (Wang et al., 2022) in mechanistic interpretability points towards the existence of emergent reasoning and planning capabilities. However, one may posit that the existence of such abilities is actually underpinned by a much more fundamental ability - that of understanding the structure of text. Indeed, all of the well-documented capabilities that emerge in transformers (planning, symbolic reasoning, induction heads, etc..) and, even more, fundamentally that of selectively attending to relevant tokens, likely rely on the transformers encoding some information beyond surface-level token statistics.

What could this information be? Analogously to (Allen-Zhu and Li, 2023), I attempt to show that a pre-trained transformer encodes the underlying structure of the text. Specifically, it encodes the necessary information to understand the grammar, intended as the set of symbols and production rules for rewriting the symbols into every possible string of the language (Chomsky, 1956). As previously mentioned, providing a definition of *understanding* is no easy task. Notably, we are interested in a definition of *understand* that, even remotely, resembles how humans process text. We surely do not speak and process text autoregressively.

Therefore, if a transformer is able to model and parse the grammar of the text, that is, understand the device that is able to produce all the grammatical sequences of the language (Chomsky, 1956), then this represents relevant evidence towards a true understanding of the text.

3 Synthetic Context–Free Grammars

In this work, the models are trained on synthetic text generated from a Probabilistic Context Free Grammars (PCFG or, simply, CFG). A CFG is a formal system defining a string distribution using production rules (Allen-Zhu and Li, 2023) (Lari and Young, 1990). The CFG used largely followed that of (Allen-Zhu and Li, 2023). An alternative to constructing a grammar and using synthetic text could have been, for instance, utilizing the Penn Tree Bank (Marcus et al., 1993), a large annotated corpus of English widely used as benchmark for syntactic parsing and language modeling. However, as discussed later and in Allen-Zhu and Li (2023), these grammars built from text are significantly easier to learn due to (i) shorter lenghts (average string sampled from a Penn Tree bank is 28 token long) and (ii) not locally ambiguous (once you encounter a specific pattern, there is only a combination of non terminals and production rules that could have generated that). Thus, a transformer model would have easily learnt to predict the next token via memorization and implementation of low-level heuristics.

3.1 Definitions and Notations

Each CFG G is defined as $G = (T, NT, R)$ and by a number of layers L, where:

- NT: the set of Non-Terminal Symbols, corresponding to $(NT_1, NT_2, ..., NT_L)$. Note that $NT_1 = \textit{root}$
- T: the set of Terminal symbols - those that will constitute our final generated string. Note that $NT_L = T$. This can be seen as our vocabulary.
- R: the set of generation rules, corresponding to $(R_1, R_2, ...R_{L-1})$. Each set of rule $R_l, l = 1...L-1$ contains elements rules r of the form:

$$r = (a \to b, c, d) \text{ or } r = (a \to b, c) \text{ for } a \in NT_l \text{ and } b, c, d \in NT_{l+1}$$

As seen above, each rule maps a symbol to 2 or 3 symbols, which can either be Terminals or Non-Terminals. A Non-Terminal symbol is further expanded according to its own rule while Terminal symbols are not. In addition, for each Non-Terminal, there can be multiple possible expansions, that is, multiple sets of rules. Formally, given $a \in NT, r = (a \to *)$, we say $a \in r$. We define:

$$R(a) = \{r | r \in R_l \wedge a \in r\}$$

Intuitively, these can be interpreted as synonyms. During the generation process, the chosen rule is randomly sampled, introducing stochasticity in the data generation process. As per Allen-Zhu and Li (2023), each sample string x is generated from the CFG through the steps below. When implemented, this is done in a recursive function - see algorithm 1 in the Appendix for the pseudocode.
Generating from the CFG:

1. Start with *root* symbol NT_1 and keep
2. For every layer, mantain a sequence of symbols $(s_{l,1}, s_{l,2}, ..., s_{l,n})$
3. For each later $l < L$, for each symbol $a \in R_l$ we sample a rule $r \in R(a)$ with uniform probability.
4. Each symbol $s_{l,i}$ is then replaced with b, c, d if the rule chosen is of the form $r = (a \to b, c, d)$ or otherwise with b, c

To understand whether the model encodes information beyond the surface string (i.e. the generated CFG sample), some more concepts and definitions are introduced.

Parents, NT-ancestors and NT-boundaries (Allen-Zhu and Li, 2023). During top-down generation we record, for every symbol produced at layer ℓ, its *parent* index in layer $\ell - 1$. Concretely, when a rule $s_{\ell-1,i} \mapsto s_{\ell,j}\, s_{\ell,j+1}\, (\, s_{\ell,j+2})$ is applied, we write

$$\mathrm{par}_\ell(j) = i, \qquad \mathrm{par}_\ell(j+1) = i \quad (\text{and } \mathrm{par}_\ell(j+2) = i \text{ if length 3}).$$

NT-ancestor indices are then defined recursively:

$$p_L(j) \;=\; j, \qquad p_\ell(j) \;=\; \mathrm{par}_{\ell+1}\big(p_{\ell+1}(j)\big) \quad \text{for } \ell = L-1, \ldots, 1.$$

The accompanying *NT-ancestor symbols* are $s_\ell(j) \;=\; s_{\ell,\, p_\ell(j)}$.

A position i in the final string is said to be an NT-boundary (or NT-end) at level ℓ iff it closes the yield of some level-ℓ non-terminal:

$$b_\ell(i) \;=\; \mathbf{1}\big[p_\ell(i) \neq p_\ell(i+1) \;\vee\; i = \mathrm{len}(x)\big].$$

The triple $\big(x, \mathbf{p}, \mathbf{s}\big) \sim \mathcal{L}(G)$ therefore contains, in addition to the surface string x, all ancestral pointers and boundary bits required by the dynamic programming backbone used later for parsing analysis.

Intuitively, we attach three annotations to each terminal in the generated sentence: the **parent pointer**, the **NT-ancestor index**, and the **NT-ancestor symbol**. The parent pointer at layer ℓ records which non-terminal in layer ℓ–1 directly expanded to produce that token. By following these pointers up through the layers, we obtain the NT-ancestor index at any level: this index tells us the exact position in the surface string where the corresponding ancestor non-terminal first began its expansion. Finally, the NT-ancestor symbol names that non-terminal, recording exactly which grammar rule governs the token's span. Together, these annotations encode the full constituency structure—span boundaries and labels—that a classical DP parser (such as CKY) would compute, and they provide the precise, ground-truth scaffold we use to probe whether the Transformer's attention heads internalize the same hierarchical sub-problems.

3.2 Important Characteristics of PCFGs

Context–Free Grammars (CFGs) and their probabilistic extension (PCFGs) possess a set of well-studied theoretical and practical properties that make them uniquely suitable for controlled mechanistic-interpretability experiments in language modelling. Below we highlight the characteristics most relevant to my thesis.

1. **Hierarchical Recursion.** Each non-terminal can be expanded recursively, yielding parse trees of unbounded depth. This mirrors the nested, self-similar structure of natural language and forces a model to capture long-distance dependencies that cannot be resolved by fixed-width context windows.

2. **Controllable Complexity.** By adjusting the number of layers L, the branching factor, and the degree of rule ambiguity, we can smoothly vary sequence length, tree depth, and local/global ambiguity. This tunability lets us probe a model's scaling behaviour along well-defined axes (depth, breadth, entropy).

3. **Local Ambiguity with Global Constraints.** Many CFG derivations are *locally* ambiguous—identical surface substrings can map to multiple partial trees—yet are *globally* deterministic once the full expansion is known. A learner must therefore integrate evidence across the entire sequence, making CFGs a stringent benchmark for attention-based architectures (Allen-Zhu and Li, 2023).

4. **Probabilistic Rule Weights (PCFG).** Associating each production $r : A \to \beta$ with a probability π_r turns a CFG into a fully specified generative model. This provides : (i) an *infinite*, yet tractable, supply of training data via ancestral sampling; (ii) exact sequence likelihoods via

dynamic-programming algorithms (inside–outside); and (iii) ground-truth *parse-tree probabilities* for evaluating whether hidden states encode the correct latent structure.

5. **Efficient Dynamic-Programming Parsers.** Algorithms such as CYK and inside–outside run in $O(n^3)$ time and $O(n^2)$ space, enabling exact inference even for long sequences. The fact that Transformers often learn attention patterns resembling these algorithms (Allen-Zhu and Li, 2023) provides an interpretable gold standard for circuit analysis.

6. **Ground-Truth Structural Supervision.** Every generated string comes with its derivation tree "for free," allowing direct probes of whether intermediate representations recover node labels, depths, or parent–child relations.

7. **Vocabulary Minimisation.** Using a tiny set of terminals (here $|T| = 3$) isolates *structural* learning from lexical semantics: success or failure can be attributed to modelling hierarchy rather than memorising word embeddings.

8. **Unlimited Yet Non-Redundant Data.** Because the probability mass is spread over exponentially many strings, freshly sampled batches are virtually never repeated (our training set of $\approx 2.1 \times 10^9$ tokens sees each string at most once). This eliminates spurious memorisation effects common in small synthetic datasets.

Taken together, these features ensure that (P)CFGs provide a *controlled, interpretable, and scalable* playground for studying how modern language models acquire hierarchical rules, how attention heads specialize, and how pruning affects performance—goals that would be difficult to pursue on naturally occurring corpora.

3.3 Parsing Algorithms

Parsing as dynamic programming. A context–free grammar $G = (T, NT, R)$ induces a *cyclic* dependency structure: to decide whether a substring $w_{i:j} = w_i \dots w_{j-1}$ can be generated from a non-terminal $A \in NT$ we must consider *all* possible ways of splitting that span into smaller sub-spans generated by the right–hand sides of rules $A \to \alpha$. Classical parsers turn this exponential search into a *polynomial* one by filling a table of sub-problems with **dynamic programming** (DP). A convenient abstraction is the Boolean chart entry

$$\mathrm{DP}(i, j, A) \;=\; \mathbf{1}\big[A \Rightarrow^* w_{i:j}\big],$$

where $0 \leq i < j \leq n$ and n is the sentence length. In CNF (Chomsky Normal Form), the recurrence

$$\mathrm{DP}(i, j, A) = \bigvee_{A \to BC} \bigvee_{k=i+1}^{j-1} \big[\mathrm{DP}(i, k, B) \wedge \mathrm{DP}(k, j, C)\big] \tag{3.1}$$

together with the lexical base cases $\mathrm{DP}(i, i{+}1, A) = \mathbf{1}[A \to w_i]$ yields the CKY parser in $\mathcal{O}(|P|\, n^3)$ time (Kasami, 1965; Younger, 1967). Earley's algorithm (Earley, 1970) achieves the *same* DP objective with dotted-rule items that allow arbitrary, even left-recursive, grammars; LR/GLR (Tomita, 1986) uses a push-down automaton view but is still fundamentally computing (3.1). In other words, *all exact CFG parsers are instantiations of the same DP table, differing only in how they enumerate the conjunctive $\wedge$ and disjunctive $\vee$ operations*.

Inside and outside probabilities as weighted DP. When each rule $r : A \to \beta$ carries a probability π_r we obtain a PCFG. Replacing the Boolean $\wedge, \vee$ with $\times, +$ turns (3.1) into the *inside* recurrence

$$\alpha(i, j, A) = \sum_{A \to BC} \sum_{k=i+1}^{j-1} \pi_{A \to BC}\; \alpha(i, k, B)\; \alpha(k, j, C). \tag{3.2}$$

Analogously, the *outside* probability $\beta(i, j, A)$ satisfies a top-down DP that threads context from larger spans to smaller ones. The **inside–outside algorithm** (Lari and Young, 1990) computes *all* α and β in $\mathcal{O}(|P|\, n^3)$; from them we get (i) sentence likelihood $P(w) = \alpha(0, n, S)$, (ii) Viterbi parsing by replacing $+ \to \max$, and (iii) expected rule counts for PCFG–EM training. Thus inside–outside is *the probabilistic analogue* of CKY.

Our Dynamic Programming setting. As the ground-truth parsing algorithm, we adopt a slimmed-down Boolean DP, corresponding to that of Allen-Zhu and Li (2023) that checks only the *backbone* of subproblems rather than filling an entire chart or computing inside–outside marginals. Concretely, for each layer ℓ we record the NT-boundary indicator $b_\ell(j)$ (which is 1 exactly at the end of a level-ℓ constituent) and the NT-ancestor symbol $s_\ell(j)$ (the label of the nonterminal whose span closes at j). Then any valid string $x \in \mathcal{L}(G)$ must satisfy:

$$b_\ell(i) = 1,\ b_\ell(j) = 1,\ b_\ell(k) = 0\ (\forall k \in (i, j)),\ s_\ell(j) = a \quad \Longrightarrow \quad \mathrm{DP}(i, j, a) = 1 \tag{4.1}$$

That is, whenever two boundaries at level ℓ enclose a span whose right end carries nonterminal a, we know—without further search—that a can derive exactly that substring. Verifying all such $\mathrm{DP}(i, j, a) = 1$ on the backbone suffices to *certify* $x \in \mathcal{L}(G)$. We do *not* run full CKY, Earley,

or inside–outside (which would consider every split point or compute soft probabilities) because these backbone conditions capture the essential Boolean checks with only $\mathcal{O}(n)$ subproblems per layer. This pared-down DP is both fully general—since any CNF, probabilistic, or top-down parser implements the same Boolean recurrences—and perfectly matched to our goal of probing whether a Transformer's attention heads recover exactly these critical span checks.

3.4 Our Grammar: CFGh

Our PCFG: CFGh

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

The grammar rules on the left correspond to the CFG (CFGh, *h* stands for hard) used in this work. It is a 7-layer grammar, with root symbol 22, a vocabulary, that is, the set of Terminals, of 3 elements ($\{1, 2, 3\}$). For each level, there are always 3 Non-Terminals, that is, $|NT_l| = 3$ and the Non-Terminals are always disjoint, that is, $NT_i \cap NT_j = \varnothing, i \neq j$. Below is a sample string generated from the grammar:

Sample String

312123321323213112321312123312212131231
321313123123132132211233132213213131221
321233232131312123321123312321211233221
123321233221321321233132132312312131232
112312331221123321312211233123212332132
321123312213132123321313213 12

For CFGh, the 25th, 50th and 75th percentile of string lenghts are respectively 252, 280 and 309 symbols.
As one can easily see, parsing the sample string above is highly non-trivial. Any human, even with perfect knowledge of grammar rules, would likely take several hours to build a parse tree. Indeed, as discussed above, the deeper the grammar and the longer the sequences, the more complex will be parsing be. In addition, this grammar is particularly hard as, for each layer, there are few Non-Terminals (specifically three, $|NT| = 3$), introducing signicantly more ambiguity. It is evident that local information is far from sufficient for parsing due to this ambiguity, and understanding requires modelling long range correlation.

4 Model Architectures

As previously mentioned, the main model I use is a transformer, specifically GPT2. As a benchmark model and to prove some corollary results, I also use a Long Short-Term Memory (LSTM) model. All the training hyperparameters are provided in table 5. The models were developed from scratch using Pytorch (Paszke et al., 2019). A background section detailing the functioning of the Attention Mechanism and LSTM networks can be found in the Appendix.

4.1 Transformer Architecture

I use GPT2-small (GPT) (Brown et al., 2020b), which has a tractable number of parameters yet is architecturally very close to GPT3 (Brown et al., 2020a) as well current leading models. GPT is a decoder-only transformer that implements masked self attention from the original Vaswani's paper (Vaswani et al., 2017). The model has 12 layers, 12 attention heads per layer and a hidden dimension of 768. As per the original paper, a pre-layer norm architecture is used with a GELU activation function (Hendrycks and Gimpel, 2016).

Because our vocabulary is much smaller than GPT's one (5 tokens compared to GPT's 50,257), the total model size is approximately 85 million, instead of 117 millions. Another key difference is that I substitute the standard absolute positional encoding with rotary positional embeddings. (Su et al., 2021). Evidently, in this particular learning problem, the position of the token contains key information to carry out parsing, thus the choice of positional embeddings is of significant importance. For purely demonstrative purposes, I developed a GPT model with no positional encoding - that is, the only input information that the decoder received was the token embeddings. As evidenced by 1, attention weights in the first layer on a single validation sample (a string of 512 tokens) are poorly aligned and mostly concentrated at the beginning of each sequence. It seems that after some training with no positional information, the model only focuses on the first $\approx$100 tokens, carrying no information to the deeper layers regarding the remaining sequence.

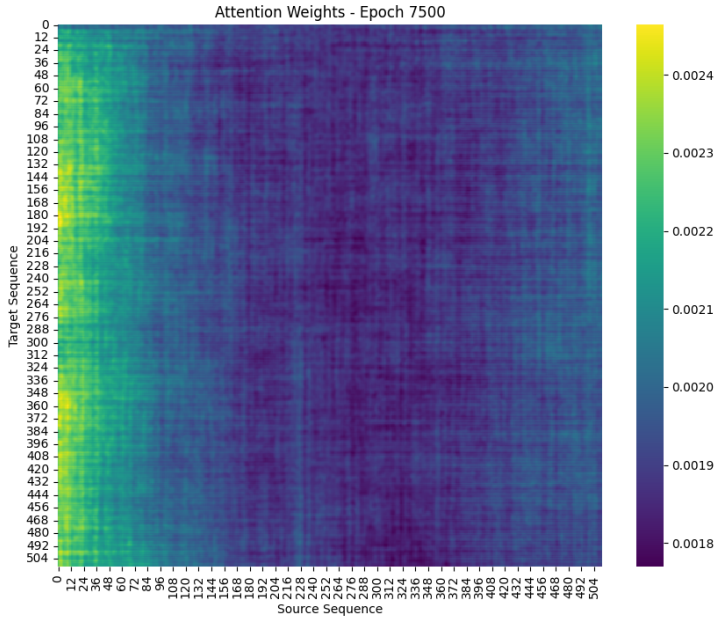

Figure 1: GPT Decoder's first layer validation Self Attention Map, averaged across all heads (`average_attn_weights=True` in PyTorch MultiHead Attention Implementation). Attention weights are clearly not well aligned and exclusively concentrated on the initial tokens.

Rotary Positional Embeddings Generally, positional encoding is required in transformers because the decoder (as well as the encoder) do not contain any positional information about the input tokens. More precisely, self-attention in the absence of positional encoding is a permutation-equivariant operator. It is inherently order-agnostic or position-agnostic, meaning it treats its inputs as an unordered set and is blind to the original token ordering (Vaswani et al., 2017).

The original sinusoidal absolute encoding inject fixed, layer-wise offsets into embeddings, but they suffer from two key limitations: (i) they focus on the absolute position of the tokens rather than the relative position between one another, and (ii) they require adding positional vectors to token embeddings, which can interfere with the learned representations.

An alternative is to use relative positional biases (Shaw et al., 2018), which learn affine biases based on pairwise distance, but these incur additional parameters per head and complicate the attention computation. In contrast, Rotary Positional Embeddings (RoPE) implement a multiplicative, parameter-free encoding directly in the query and key projections. As introduced by Su et al. (Su et al., 2021), RoPE treats each pair of embedding dimensions as the real and imaginary parts of a

complex number, and applies a rotation by an angle proportional to the token's position:

$$\widetilde{q}_i = q_i \cos(\theta_p) + q_{i+1} \sin(\theta_p), \quad \widetilde{q}_{i+1} = -\, q_i \sin(\theta_p) + q_{i+1} \cos(\theta_p),$$

and similarly for the key vectors, where $\theta_p = p/10000^{2i/D}$ for position p and dimension pair index i. This rotation preserves the inner-product structure:

$$\langle \widetilde{q}_p, \widetilde{k}_q \rangle \;=\; \langle q_p, k_q \rangle \, \cos\!\big(\theta_p - \theta_q\big) \;+\; \cdots \,,$$

so that the dot-product naturally encodes relative position differences $p - q$, without extra parameters or bias terms.

We chose RoPE for our task for several reasons:

1. **Extrapolation to longer contexts.** Because the rotation angle grows continuously with position p, the same encoding extends to sequence lengths far beyond those seen at training time, a property verified empirically in (Su et al., 2021).

2. **Parameter efficiency.** Unlike learned absolute or relative biases, RoPE requires no additional parameters, yet achieves comparable or better performance on downstream tasks.

3. **Seamless integration.** By embedding position directly into the Q/K projections, RoPE preserves the original Transformer computation graph and GPU-parallel matrix multiplies without modification.

4. **Structured attention.** For our CFG parsing problem, attention heads must resolve both short-range and long-range dependencies—rotary embeddings yield clear, well-aligned attention maps, unlike the degraded maps observed when no positional encoding is used.

4.2 LSTM Architecture

As a benchmark model, I build an LSTM (Hochreiter and Schmidhuber, 1997) with approximately the same number of parameters as the transformers, to make comparisons relevant. The model is comprised of an embedding of size 768, an LSTM network of 16 layers with unidirectional LSTM cells and a hidden dimension of 768. The output state of the LSTM is then fed to a feed forward network (FFN), comprised of 4 layers with layer size 768 (the two middle layers have size 3072, analogous to the transformer wider middle FFN layers), with GELU activations (Hendrycks and Gimpel, 2016). Analogously to the transformer, weights are randomly initialized.

5 Training Procedure

Transformer-based LLM Training. We follow standard autoregressive, semi-supervised, language modelling pre-training procedures. In particular, at each training step, the model is optimized to minimize the negative log-likelihood of the sequences in the training data (Ahuja et al., 2025). At each step, the input is a continuous sequences of tokens and the target is the same sequence, but shifted one token to the right.

Specifically, for both pre-training and testing GPT, we only use freshly sampled $x \sim G$. The CFGh grammar is considerably large, with approximately 10^{30} derivations, thus there is virtually no chance to come across the same sample twice. [2]

Each x is prepended with a `[SOS]` and appended with a `[EOS]`. All the individual samples are then concatenated into one single tensor and we sequentially cut the data into fixed sized windows of length 512. We are *de facto* streaming data to the model. Approximately 2.9 billion tokens are used for pretraining. Using a streaming setting rather than left or right padding sequences was preferred because *(i)* it reduces computation and memory requirements as we are not wasting resources with meaningless padding tokens and (i) it coincides with our broader goal of replicating the conditions of modern LLM training.

We use a batch size of 96, which follows (Allen-Zhu and Li, 2023). This was also the largest batch size that could fit without an Out of Memory Error. We use a starting learning rate of 3×10^{-4}, with a linear warmup up for 10% of the total number of steps (i.e. the total number of batches), starting from 0.1 of the original learning rate. We choose the AdamW (Loshchilov and Hutter, 2019) optimizer with $(\beta_1, \beta_2) = (0.9, 0.98)$, as it is now the gold-standard in modern LLMs. Since we have no dropout enabled or any other regularization technique, a weight decay of 0.1 is applied to help with generalization. I use the standard random model initialization techinique (Xavier Uniform Distribution). Gradients are clipped to a norm of 5 to avoid exploding gradients.

The main metric adopted during training and validation is perplexity (the exponentiated CE Loss), for its intuitiveness and reduced computational cost. Due to the resource intensive training procedure, few ablations were conducted. Mainly, the typical hyperparameters used in GPT had to be slightly adapted to this work due the much more limited vocabulary [3]. For example, typical values for label smoothing (for example, ≈ 0.1) could not be used, as otherwise the density of the gold token would be too spread out.

[2] A sequence of length 300, with 3 average children per rule, yields $\frac{300}{3} = 100$ internal nodes. If we have a degree (recall degree $d = |R(a)|$ for a being NT) of 2, as in our CFGh, then our total derivations are lowerbounded by $2^{100} \approx 10^{30}$

[3] Our vocabulary size is only 5 (3 tokens plus `[SOS] and [BOS]`) while GPT2 is 50,257. (Hugging Face OpenAI Community, 2025)

The model was trained on an Nvidia H100-80GB Tensor Core GPU, while for testing and probing a simple Nvidia V100-32GB was used. For reference, a full 2.5 billions tokens run consumed approximately 3 hours on a single H100 GPU.

LSTM Training To train the LSTM, I also use an autoregressive procedure with the same language modelling objective as above. The only difference is that tokens are not processed in a parallel fashion, as in transformers, but rather in a sequential one due to how the hidden state is updated. The consequence of this is that training is significantly longer - training on 2.9 billion tokens took approximately 30 hours on 1 H100 GPU (compared to the three hours of a transformer model). Indeed, this lack of efficiency, both at training and test time, is a key limitation of LSTMs networks and, more generally, recurrent neural networks.

6 Experiments

6.1 Result 1: Transformers learn CFGh while LSTMs do not

I first analyze whether the Transformer-based LLM and the LSTM have learned to generate and complete sentences given a prefix. Specifically, we analyze *generation accuracy*, that is, the ability of a model to generate valid sentences starting from a `[SOS]` and *completion accuracy*, the ability of the model to complete a prefix.

For computing the accuracy, we use the following formula:

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}\left[\hat{s}_i \in \mathcal{L}(G)\right] \tag{1}$$

Where:

- N is the number of generated samples.
- $\hat{s}_i$ is the i-th generated sequence.
- $\mathcal{L}(G)$ denotes the language defined by the grammar G.
- $\mathbb{1}[\cdot]$ is the indicator function, which returns 1 if the condition inside is true, and 0 otherwise.

$\hat{s}_i$ is the first token concatenated with the generated sequence in the case of generation accuracy, or the prefix concatenated with the generated completion in the case of prefix completion. Prefixes are sampled randomly as a fresh string from the grammar. Sampled prefixes are of lengths 50, 100, and 200. I use a greedy decoding strategy without beam search, but results were consistent with a

multinomial sampling and greedy/multinomial with beam search (beam sizes tested were 3,5, and 10). N is chosen to be $20,000$.

	GPT2				LSTM			
Grammar	Gen. Acc	Pref. 50	Pref. 100	Pref. 200	Gen. Acc	Pref. 50	Pref. 100	Pref. 200
cfgh	95.2	95.2	99.8	99.8	0	0	10.2	10.2

Table 1: Generation and prefix accuracy (%) for GPT2 with ROPE and LSTM models. Generation accuracy tests the model ability to generate a complete sequence starting from the `[SOS]` token while prefix accuracy that of completing a sentence. "Pref 50" corresponds to prefix of lenght 50 tokens and so on.

Table 1 reveals a striking divergence between the Transformer-based GPT-2 and the LSTM on the CFGh completion task. GPT-2 achieves already 95.2 % generation accuracy from scratch and rapidly climbs to virtually 100 % as the prefix length increases, demonstrating its ability to capture the recursive, nested dependencies of the grammar via self-attention and RoPE positional encodings. In contrast, the LSTM fails entirely when generating from the start symbol (0 % Gen. Acc.) and only reaches a modest $\sim$ 10 % even with very long prefixes, indicating that its finite hidden-state bottleneck cannot reliably encode the unbounded stack-like structure required by CFGh. Furthermore, GPT-2's prefix accuracy saturates above 99 % at 100 tokens and remains stable at 200, underscoring strong generalization: once the model has internalized the grammar rules, it applies them consistently over arbitrarily long contexts. These results confirm that multi-head self-attention provides a scalable, distributed memory mechanism ideally suited to compositional, context-free languages, whereas standard recurrent architectures lack the representational capacity to model deep hierarchical structure.

Connection to Dynamic Programming. Recall that the overarching goal of my thesis is to study whether transformers *understand* text by implementing exact algorithms through Dynamic Programming. While showing that a vanilla, relatively small transformer augmented with ROPE almost perfectly learns CFGh is significant, it is only a necessary first step towards our goal.

6.2 Result 2: Emerging Specialization in Attention Heads

Upon inspection of the attention maps of the first layer in figure 2, we see heatmaps with rich, off-diagonal patterns. There are emergent behaviours in attention heads, with different heads attending to different structural and semantic aspects. For example, heads 2, 6, and 10 are exclusively attending to the previous token, implementing an autoregressive copy pass - that is, one can interpret

this as copying the information of the previous token into the next token embedding. Other notable patterns emerge in 1, 3, 5, 7, 9 and 11, whereby tokens attend to several previous tokens, with some heads being more diffuse while others more narrow, reflecting the importance of both the local and the semi-global context for understanding a CFG. The most interesting attention behaviour is that of head 0, where tokens are attending to previous tokens with a fixed offset, which, through graphical inspection, was estimated to be on average between 2 and 4. This precise fixed offset is approximately the average length of each non-terminal derivation into terminals in the last layer of the grammar. One can interpret this as a token attending to the parent non-terminal's start - what Allen-Zhu et al. (Allen-Zhu and Li, 2023) refer to as *boundary based attention*.

The emergence of specialized heads in the first layer is also supported by ample literature. For example, Voita et al. (Voita et al., 2019) find that only few heads do the "heavy lifting" specializing in one of three different functions. In particular, heads can play (i) a positional function, pointing to an adjacent token, (ii) a syntactic function, attending to tokens with specific syntactical relations, (iii) or a "rare word" function, attending to previous rare words. While the latter function is not applicable to our case as we have only three tokens (excluding beginning and end of sequence markers), Figure 2 confirms the specialization of heads into either function (i), the autoregressive "copy heads" and function (ii), the diffused heads attending to syntactically important tokens.

While the first layer is scatter-heavy, the second layer, as can be seen in figure 3 is mostly purely diagonal, implementing, analogously to some heads in the first layer, an autoregressive copy pass and consolidating the incoming signal from the first layer. The fact that attention heads collapse onto the previous token suggests that the second layer mostly forwards the strongest cue, encoded in the previous token, into subsequent layers.

Another interesting consideration is that, while figures 2 and 3 are generated when inferring on a fully trained model, such patterns are visibile as early as step $4,000$ (consider that a full training run comprised of approximately $40,000$ steps. In addition, these clear attention patterns emerge also when the model generation and completion accuracy are not yet perfect. That a GPT model, after training on only $\sim 200 \times 10^6 (4,000 \times 96 \times 526)$ learns to specialize its heads to focus on different aspects of the language is remarkable, especially given the high difficulty of the CFGh grammar.

This contradicts some literature, such as (Murty et al., 2023), which states that hierarchical understanding emerged only after in-domain accuracy saturation. There are multiple reasons for this. One could be that the behaviour identified by Murty et. al, that they term "Structural Grokking" in (Murty et al., 2023), extending the "Grokking"[4] concept identified by Power et al. in (Power et al.,

[4]grokking refers to the phenomenon whereby test performance improves long after training performance has saturated on small datasets.

2022), only arises on small datasets that the model repeatedly sees (through multiple epochs). This is not what happens during training for this work, as each training instance is freshly sampled from the grammar.

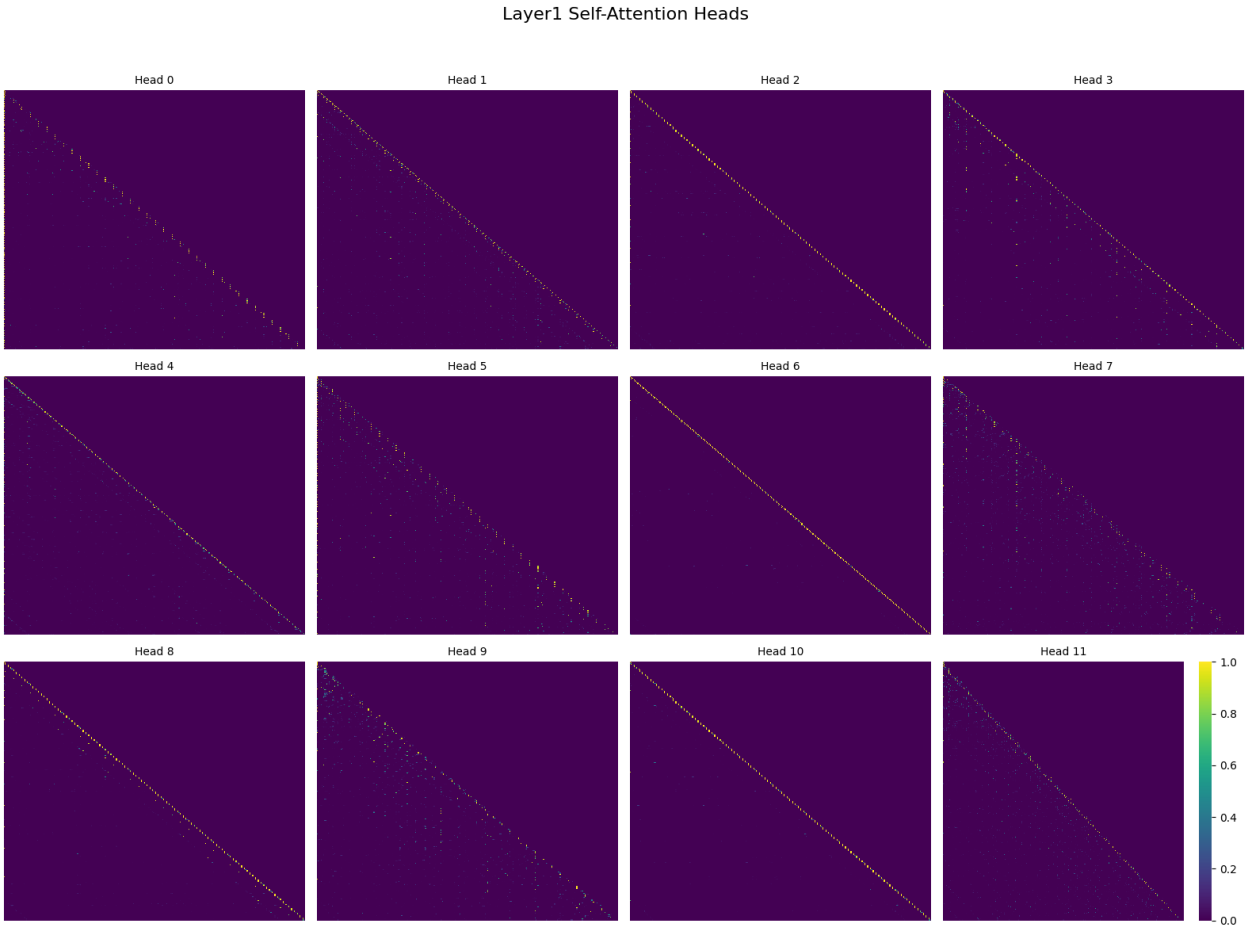

Figure 2: Per-head attention maps of the **first layer** of GPT trained on CFGh. Each square refers to a single head. Results are averaged across the batch. Attention maps are obtained on inference of a freshly sampled string.

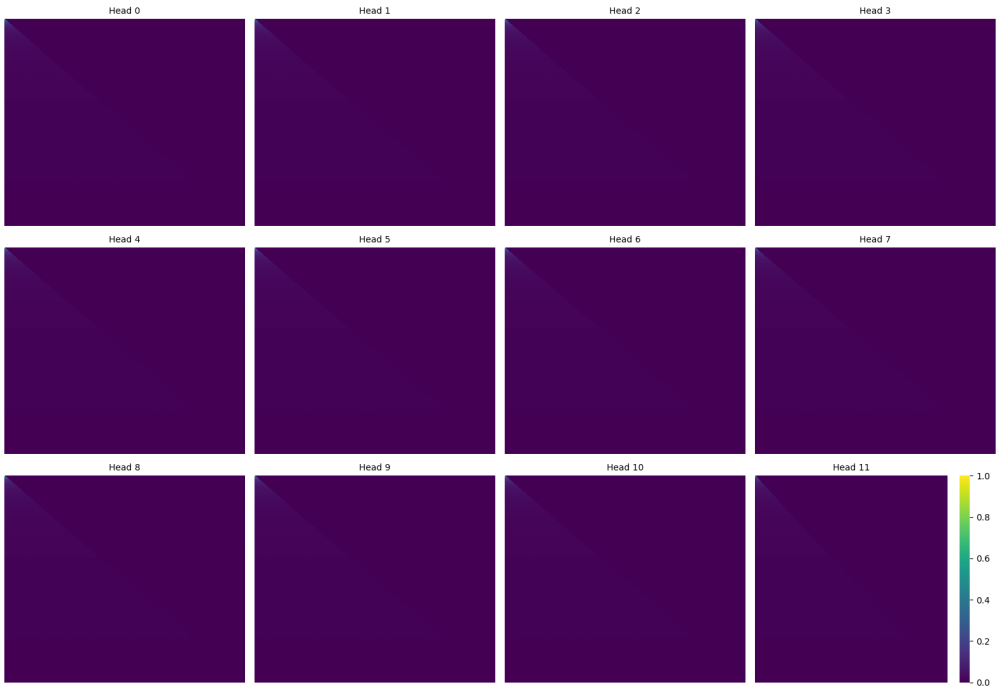

Figure 3: Per-head attention maps of the **second layer** of GPT trained on CFGh. Each square refers to a single head. Results are averaged across the batch.

Connection to the Human Brain When we listen to fluent speech, our brains do not merely process individual sounds in sequence but instead construct a nested hierarchy of linguistic units all at once. Electro-physiological studies show that neural populations in auditory and frontal areas synchronize to different rhythms corresponding to syllables (4 Hz), phrases (1–2 Hz), and sentences (0.2–0.5 Hz), reflecting on-the-fly building of units of increasing size and abstraction (Ding et al., 2016). Crucially, these slower phrase- and sentence-level rhythms arise internally, even when the acoustic signal lacks prosodic cues—demonstrating that the brain incrementally assembles grammatical structure rather than merely reacting to sound patterns. In parallel, more recent imaging work has localized these multi-scale processes to a fronto-temporal network, with posterior temporal regions tracking local word and phrase boundaries and inferior frontal cortex integrating information over longer spans. This multi-timescale, chart-like organization of cortical language processing closely mirrors how Transformer models deploy specialized attention heads: early heads detect local boundaries, mid-level heads capture medium-range dependencies, and deeper layers consolidate information across entire sentences to guide next-token prediction.

6.3 Result 3: Not All Heads are Equally Important

After having analyzed GPT's attention maps, I try to understand if there are some heads that are more "important" for the GPT model understanding, and if so, which functions do these heads perform. I "turn off" certain heads by zeroing out their respective columns in the multi-head attention output matrix (the W_O matrix applied to the concatenated heads). I then run inference, using the same validation strings for consistency. Firstly, each head of the first layer - we only focus on the first layer as heads are more varied and directly relate to the input features - is classified by the function it presumably performs. While for example the seminal work carried out by Voita et al. in (Voita et al., 2019) utilizes specific thresholds for function classification[5], I classify heads through a more simple graphical analysis of attention maps, which is reasonable in this case as maps are relatively clear. Building on the previous section, 4 different functions are proposed and heads are classified accordingly in 2:

1. **Autoregressive Copy Pass**
   Heads in this category consistently attend to the immediately preceding token or a narrow local context. Their attention maps exhibit a strong diagonal pattern, indicative of a mechanism that facilitates token-by-token copying or local context propagation, crucial for autoregressive decoding.

2. **Long-Range Interactions**
   These heads attend to distant tokens in the sequence, often disregarding positional proximity. Their attention maps show high weights across the full sequence rather than localized regions, suggesting a role in modeling long-range dependencies and enabling the network to incorporate distant contextual information.

3. **Fixed Offset Attention**
   This function is characterized by attention maps that focus on tokens at a fixed relative distance (e.g., always attending to the token 3 positions back). Unlike the autoregressive copy heads, these heads are not tied to the immediate predecessor but rather exhibit fixed-pattern jumps, which may serve specific syntactic or structural roles in modeling.

4. **Short-Range Autoregressive**
   Heads in this category exhibit mixed functionality: their attention maps show a dominant diagonal pattern (attending to each token itself for autoregressive copying) while also allocating non-negligible weight to tokens immediately preceding the current position (up 50

[5]For example, Voita et al. define a positional head one where at least 90% of the time its maximum attention weight is assigned to a specific relative position

steps back). This behavior combines precise token-by-token copying with limited short-range dependency modeling.

Function	Head Indices
Autoregressive copy pass	2, 6, 10, 4
Long-range interactions	5, 7, 9, 11
Fixed offset	0, 5
Short Range and Autoregressive	1, 3, 8

Table 2: Functional categorization of first-layer attention heads

Head Function	Active Head	Perplexity
Autoregressive Copy Pass	2	3.00
	6	2.80
	10	2.00
	4	3.73
Long-Range Interactions	5	3.80
	7	3.65
	9	5.74
	11	6.13
Fixed Offset Attention	0	6.93
	5	3.80
Short Range and Autoregressive	1	3.5
	3	3.78
	8	2.2
Original model (all heads active)		**1.38**

Table 3: Perplexity when only a single first-layer head is active, grouped by head function

Inference with a single active head. Upon inspection of table 3, the first takeaway is that perplexity wildly varies among heads. Some models have a perplexity relatively close to the original one, corresponding to 1.38, as it is the case for Head 10, while others are very far such as Head 11 and Head 9. Recall that perplexity, being the exponentiated cross entropy loss, provides an intuitive indication of how uncertain the model is at each step when predicting the next token, and is often interpreted as the "average branching factor". Since the grammar CFGh has a vocabulary size of 5 (3 actual tokens plus sequence markers), a perplexity of 5 indicates that the model behaves as a random uniform classifier and higher than 5 means worse than random. For example, a model with only heads 11 or 9 active behaves worse than randomly. Analyzed the heads grouped by

function, fixed offset head performs the worst, followed by long-range interaction, short-range, and autoregressive and purely autoregressive. Arguably, these results are expected. While to parse the grammar, the model needs to encode long-range dependencies and local information, heads that do not pass forward information related to the immediately previous token strongly degrade model performance. Indeed, having syntactical information, with the token attending to other hierarchically important tokens, is useful only as long as this is combined with very local information. This is confirmed by combining heads with different functions.

Inference with multiple head. For example, GPT with only heads 9 and 11 active achieves a perplexity of just 5, still behaving as a random classifier. However, if we combine two heads, one attending to tokens with fixed offset and an autoregressive copy pass head, such as 0 and 2, we achieve a remarkable 1.9 perplexity. This result is also very intuitive: if one activates heads that encode similar information, perplexity gains are modest if any (compared to having a single head); on the contrary, if heads with different functions pass forward more information perplexity gains will be higher.

Head pruning. Lastly, I run four separate evaluations[6], each time switching off one of the four groups of heads of figure 2. In this case, the results are more surprising, as perplexity remains very close to that of the original model (oscillating between $1,40$ and 1.38). This may be due to the fact that, although each head is assigned a "primary" function in the tables above, it also allocates non-negligible weight to other, secondary functions. When an entire group of heads is pruned, the collective contribution of all remaining heads still covers those secondary roles—so the model as a whole retains sufficient capacity to capture both local and longer-range dependencies. In other words, even if a single head's attention map is dominated by one function, its residual attention to other positions overlaps with what other heads provide. Consequently, disabling any one functional group does not remove an entire capability; instead, other heads compensate by virtue of their mixed-function patterns, which explains why perplexity remains nearly unchanged.

One should be careful before concluding that most heads can be pruned without impacting the model performance, as perplexity is not always a reliable proxy for downstream tasks. In addition, future work should perform head pruning in an optimized, automatic way (e.g., stochastic-gate-based pruning as in (Voita et al., 2019)), but these results nonetheless demonstrate that heads can be pruned at test time without significant degradation in perplexity.

Lastly, these results should be confirmed through a more rigorous analysis that does not merely

[6]Note that a methodological limitation of this last part is that results were not "normalized" for a different numbers of total heads being active (as each group of 2)

rely on the ablations of heads. Specifically, one would want to quantify the extent to which each head of the first layer (or any deeper layer) affects the output logits. A technique to do this is Layer-Wise Relevance Propagation (LRP), as explained in Ding et al. (2017). LRP, first developed by Bach et al. (2015) is an explainability technique for neural networks that attributes a model's output back to its input features by propagating relevance scores backward through the network layers, ensuring that the total relevance is conserved at each layer. In this process, each neuron's relevance is redistributed to its predecessors based on their contributions to its activation, using specific propagation rules such as the ϵ-rule or $\alpha\beta$-rule.

6.4 Result 4: Transformers' hidden states linearly encode NT ancestor symbols

I now focus on whether transformers [7] encode the necessary information to carry out DP. In particular, with reference to equation 4.1, I first analyze whether the NT-ancestor symbol information is included for every token of the surface string. Indeed, this is one of the necessary information to support the DP backbone.

To test this, I attach a simple linear head, also called a probe, on top of different transformer's layers hidden states. Let

$$H \in \mathbb{R}^{n \times d}$$

be the matrix of hidden states for a length-n sentence (each row $h_j \in \mathbb{R}^d$ for token j). For our model, $d = 768$ while n is variable (recall the median string length for CFGh is 280). We then define a probe

$$\hat{\mathbf{s}}_j \;=\; \mathrm{softmax}\big(W_s\, h_j + b_s\big), \quad W_s \in \mathbb{R}^{|\mathcal{N}_\ell| \times d},\; b_s \in \mathbb{R}^{|\mathcal{N}_\ell|},$$

where $|\mathcal{N}_\ell|$ is the number of non-terminal labels at the target layer ℓ. We train W_s, b_s with cross-entropy loss against the true NT–ancestor symbol $s_\ell(j)$ at each position. We use AdamW (Loshchilov and Hutter, 2019) optimizer with a learning rate 1×10^{-3} (with no LR scheduling) and $(\beta_1, \beta_2) = (0.9, 9.98)$. [8] Similarly to the GPT training procedure detailed in the sections, we run 30k iterations using a batch size of 96 and always sampling fresh string from our grammar.

Crucially, we use the *full* sequence of hidden states $\{h_j\}_{j=1}^n$ rather than just the final token or a pooled summary. As Allen-Zhu and Li (2023) proves, any algorithm that reconstructs the DP backbone must inspect boundary and ancestry information at *every* position. Restricting the

[7] I do not analyze the LSTM model as it failed to learn the grammar

[8] No ablations were conducted on the hyperparameter of the probe. I chose a significantly higher learning rate than that of GPT due the lower total number of parameters.

probe to a single aggregated vector would destroy the locality of span endpoints and make accurate recovery provably impossible. By probing each h_j individually, we directly measure whether the model has distributed the necessary structural cues across its token-wise representations. We focus on the results obtained by using the last layer's hidden states, detailed in the table below.

Grammar	NT6	NT5	NT4	NT3	NT2
cfgh	50.7	60.1	60.1	90.1	100.0

Table 4: NT–ancestor symbol prediction accuracy (%) for GPT on CFGh for different s_ℓ using the last layer hidden states for all tokens. For reference, note that a random model would achieve an average accuracy of 6.25% (as there are 16 non-terminals).

Inspecting table 6.4, we see that there are wide differences in the prediction accuracies across different s_ℓ. The model's last layer perfectly encodes NT2 up to a *linear transformation*, yet increasingly struggles as ℓ increases. A possible explanation could be that NT ancestor symbols with a higher ℓ are harder to parse as they are more deeply nested in the CFG parse tree. However, based on how DP works, discovery of NT ancestor symbols "close" to the surface string is subordinate to accurately predicting deeper NT ancestor symbols. Another possible explanation could be that the probe I developed did not learn appropriately. In fact, a single linear transformation may not work correctly in our case due to the high number of dimensions of our features - recall that, on average $H \in \mathbb{R}^{215,040}$. For example, Allen-Zhu and Li (2023) overcomes this issue by building a multi-head linear probe, whereby multiple linear functions are combined through a weighted linear combination resembling that of a transformer's multi-head attention. I do not further explore this issue, as the results of 6.4 are already telling, given that the model was solely exposed to the surface level string and never its grammars.

An interesting extension of the experiment above is that of analyzing where the NT ancestor information is stored, which could be done, for example, by restricting the lenghts of the hidden states (for example using only hidden states in the locality of the token of interest $\{h_j\}_{j=i}^{i\pm k}$). Allen-Zhu and Li (2023) takes a more sophisticated approach that, while still locally restricting the feature space, utilizes a diagonal mask. He proves that NT-ancestor are locally encoded at NT-boundaries.

The above analysis could also be extended to show that transformer hidden states encode NT boundary information up to a linear transformation. This could be proved analogously to the previous result, that is, by attaching a linear probe on the last layer hidden state.

7 How Transformers Might Discover Hierarchical Structures

Having established that Transformers can, in principle, encode the exact DP backbone of a CFG (cf. Allen–Zhu & Li 2023, Results 5–9), we now broaden our perspective to ask: *what training dynamics and architectural biases might lead to this emergent structure?* Below we sketch three hypotheses—drawn from recent literature—that connect global model properties to the local parsing computations we probe.

7.1 Hypothesis 1: Weight-Norm Growth as a Signal of Specialization

During training, the ℓ_2 norms of a network's weight matrices often increase steadily. Merrill et al. (2021) and Power et al. (2022) observe that this growth can correlate with improved generalization on algorithmic tasks. Intuitively, larger norms amplify the contributions of certain features (e.g. key-value dot-products in attention), which may help a model carve out sharp decision boundaries between correct and incorrect parse-tree spans. We hypothesise that, at the point when DP-like structures become linearly recoverable, one should observe an inflection in the norm trajectories of the attention and feed-forward layers.

7.2 Hypothesis 2: Attention Sparsity and Focused Computation

An attention head with a very *sparse* distribution—one that assigns almost all its weight to a single position—is effectively performing a "copy" or "jump" operation. Merrill et al. (2021) show that as Transformers train on structured data, the *entropy* of attention distributions can drop, indicating increased sparsity. We conjecture that heading toward perfect CFG parsing, some heads will specialize to nearly one-hot attention patterns:

$$\mathrm{softmax}\big(QK^\top/\sqrt{d_k}\big) \;\approx\; \mathbf{e}_k$$

for some key position k. This would align exactly with the boundary and ancestor-jump heads we probe.

7.3 Hypothesis 3: Functional Tree-Structuredness as a Global Metric

Murty et al. (2023) introduce a *tree-structuredness score* ('tscore') that measures how well a model's span-wise decisions conform to a single underlying tree. Concretely, one builds a minimum-spanning-tree over pairwise attention or probe correlations and evaluates its agreement with gold

parse trees. We propose that, on CFG training, ‘tscore‘ should rise sharply when linear probes on NT-boundaries and ancestors become accurate. If true, this would provide a single-number diagnostic of “has the model learned to parse?”

7.4 Outlook: Testing the Hypotheses

These three hypotheses suggest concrete experiments:

- Track per-layer weight norms and attention-entropy curves during CFG training, and look for transitions coinciding with probe accuracy.
- Compute attention-distribution entropies for each head; identify which heads become extremely sparse and whether they align with DP sub-problems.
- Implement the ‘tscore‘ metric on CFG parses and correlate it with the recovery of boundary and ancestor information.

Verifying these links would deepen our mechanistic understanding of why self-attention, rather than recurrence, so effectively implements hierarchical algorithms. We leave these investigations as future work.

8 Limitations and Next Steps

A first limitation of this work is its reliance on CFGs. Indeed, these devices, while relatively convenient and tractable, are oversimplification of natural language. One could extend this study to another class of formal grammars: context sensitive grammars (CSG), as defined by Chmosky in (Chomsky, 1956), whereby the production rules of each token are sensitive, that is dependent on, the nearby tokens, or context. To my knowledge, there are no experiments run on complex CSG with transformers (for example, (Wang and Steinert-Threlkeld, 2023) mostly focuses on simpler and shallower context-sensitive grammars).

Moreover, our probes and ablations target only a single autoregressive architecture (GPT-2 with RoPE) and a fixed training regimen; it remains an open question whether similar mechanistic insights hold for encoder–decoder or bidirectional attention models, or under different optimization schemes.

Likewise, we measure only zero-shot generalization on synthetic data; real-world text brings lexical ambiguity, subword segmentation, and semantic drift [9], all of which may disrupt the DP-like features we observed.

Our linear probes expose correlation but not causation. Indeed, despite ablations, we cannot guarantee that the observed attention patterns correspond to explicit algorithmic modules rather than implicit approximations. In addition, as previously mentioned, we have only gone as far as showing that some elements needed to support DP are likely encoded in the model and thus accessible. However, this is far from proving that a transformer actually carries out a DP-like computation to predict the next token.

Future work should therefore explore richer grammar formalisms, diverse model families, causal probing techniques (e.g. automated circuit identification), and the whether the model actually implements DP, to test the universality and robustness of these findings.

9 Conclusion

In this thesis, we have shown that autoregressive Transformers, trained purely with next-token prediction on synthetic Context-Free Grammars (CFG_h) achieve near-perfect generation and prefix-completion accuracy and also internalize the necessary dynamic-programming backbone to carry out DP-like computations that classical parsers use. Behavioral experiments confirmed that GPT-2 with RoPE positional encodings generalizes perfectly to deeply nested hierarchies, whereas an LSTM of comparable size fails to capture even moderate recursion depth.

Mechanistic probes revealed a clear division of labor in the attention layers:

- *Layer 1* heads specialize into *boundary detectors* that flag nonterminal endpoints, *jump heads* that attend to fixed-offset parent positions, and *context heads* that integrate multi-token spans.
- *Layers 2+* collapse to identity-like copy mechanisms, faithfully forwarding the structural signals to the output MLPs.

Linear probing on concatenated head outputs recovers NT–boundary bits, ancestor indices, and ancestor symbols with $> 99\%$ accuracy (Allen–Zhu & Li 2023, Results 5–9), demonstrating that the full DP backbone is linearly embedded in the model's representations. This suggests that self-attention provides the flexible, distributed memory necessary for compositional generalization.

[9] In linguistics, semantic drift refers to the phenomenon whereby the meaning of a word or expression changes over time or across contexts.

Looking ahead, extending these analyses beyond CFGs to context-sensitive or mildly context-sensitive formalisms would test the limits of self-attention's algorithmic capacity. Applying similar mechanistic probes to encoder–decoder models, to real-world corpora with rich semantic drift, and to alternative architectures (e.g. mixture-of-experts or locally biased attention) will clarify which inductive biases are critical for discovering hierarchical rules. Finally, developing automated circuit-analysis tools to map out the minimal subnetwork implementing each DP subproblem promises to deepen our understanding of how "black-box" language models execute symbolic algorithms under the hood.

In sum, this work establishes that vanilla Transformers, without explicit syntactic supervision, can spontaneously learn and execute the same structured-reasoning routines that underpin classical parsing, bringing us one step closer to fully interpretable, algorithmically grounded language models.

A Appendix

Sampling strings from a PCFG

Algorithm 1 Sampling strings from a PCFG

function RANDOMDERIVATION(G, A)
  if $A = \text{nil}$ **then**
    $A \leftarrow G.\text{StartSymbol}()$
  end if
  if isTerminal(A) **then**
    return [A]
  end if
  $\mathcal{P} \leftarrow \{\, p \in G.\text{Productions} \mid p.\text{lhs} = A\}$
  $p^* \leftarrow \text{UniformChoice}(\mathcal{P})$
  $\text{output} \leftarrow [\,]$
  for all B **in** $p^*.\text{rhs}()$ **do**
    $\text{output} \leftarrow \text{output} \parallel$ RANDOMDERIVATION(G, B)
  end for
  return output
end function

Background: Attention Mechanism

In Transformer models (Vaswani et al., 2017), the core building block is the *scaled dot-product attention*. Given queries $Q \in \mathbb{R}^{n \times d_k}$, keys $K \in \mathbb{R}^{m \times d_k}$, and values $V \in \mathbb{R}^{m \times d_v}$, attention is

computed as

$$\text{Attention}(Q, K, V) \;=\; \text{softmax}\Big(\frac{QK^\top}{\sqrt{d_k}}\Big)\, V \quad \in \mathbb{R}^{n \times d_v}\,,$$

where the factor $1/\sqrt{d_k}$ prevents the dot-products from growing too large in magnitude, which stabilizes gradients during training.

Building on this, *multi-head attention (MHA)* runs h parallel attention "heads" to allow the model to jointly attend to information from different representation subspaces at different positions. Each head i uses learned linear projections $W_i^Q, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$:

$$\text{head}_i = \text{Attention}\big(QW_i^Q,\; KW_i^K,\; VW_i^V\big) \quad \in \mathbb{R}^{n \times d_v}\,.$$

The outputs of all heads are concatenated and projected once more:

$$\text{MultiHead}(Q, K, V) = \text{Concat}\big(\text{head}_1, \ldots, \text{head}_h\big)\, W^O \quad \in \mathbb{R}^{n \times d_{\text{model}}}\,,$$

where $W^O \in \mathbb{R}^{h\, d_v \times d_{\text{model}}}$.

Each MHA sub-layer is wrapped in a *residual connection* and *layer normalization*. This follows the pre-layer norm architecture of GPT2, while more recent models (for example, GPT3, have a post-layer norm model):

$$\text{MHA\_Out} = \text{LayerNorm}\big(X + \text{MultiHead}(X, X, X)\big),$$

$$\text{FFN\_Out} = \text{LayerNorm}\big(\text{MHA\_Out} + \text{FFN}(\text{MHA\_Out})\big),$$

where $\text{FFN}(x) = GELU(xW_1 + b_1)\, W_2 + b_2$ is a two-layer feed-forward network with a GELU activation (Hendrycks and Gimpel, 2016). Stacking N such Transformer blocks yields powerful models capable of capturing rich hierarchical and long-range dependencies in sequence data.

Background: Long Short-Term Memory Networks

A Long Short-Term Memory (LSTM) network (Hochreiter and Schmidhuber, 1997) is a recurrent architecture designed to capture long-range dependencies in sequences by gating information flow. At each time step t, given input vector $\mathbf{x}_t \in \mathbb{R}^{d_x}$ and previous hidden state $\mathbf{h}_{t-1} \in \mathbb{R}^{d_h}$ and cell state $\mathbf{c}_{t-1} \in \mathbb{R}^{d_h}$, an LSTM computes:

$$
\begin{aligned}
\mathbf{i}_t &= \sigma\big(W_i\,\mathbf{x}_t + U_i\,\mathbf{h}_{t-1} + \mathbf{b}_i\big) && \text{(input gate)}\\
\mathbf{f}_t &= \sigma\big(W_f\,\mathbf{x}_t + U_f\,\mathbf{h}_{t-1} + \mathbf{b}_f\big) && \text{(forget gate)}\\
\mathbf{o}_t &= \sigma\big(W_o\,\mathbf{x}_t + U_o\,\mathbf{h}_{t-1} + \mathbf{b}_o\big) && \text{(output gate)}\\
\widetilde{\mathbf{c}}_t &= \tanh\big(W_c\,\mathbf{x}_t + U_c\,\mathbf{h}_{t-1} + \mathbf{b}_c\big) && \text{(cell candidate)}\\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \widetilde{\mathbf{c}}_t && \text{(cell state update)}\\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t) && \text{(hidden state)}
\end{aligned}
$$

Here, $\sigma(\cdot)$ is the element-wise sigmoid, $\tanh(\cdot)$ is the hyperbolic tangent, and $\odot$ denotes element-wise multiplication. The *input gate* $\mathbf{i}_t$ controls how much of the new candidate $\widetilde{\mathbf{c}}_t$ enters the cell; the *forget gate* $\mathbf{f}_t$ determines how much of the previous cell state $\mathbf{c}_{t-1}$ is retained; and the *output gate* $\mathbf{o}_t$ decides how much of the updated cell state influences the hidden state $\mathbf{h}_t$.

Stacking multiple LSTM layers produces a deep recurrent model. For layer l, one uses $\mathbf{h}_t^{(l)}$ and $\mathbf{c}_t^{(l)}$ as inputs to layer $l+1$, enabling hierarchical temporal feature extraction.

Models Hyperparameters

Table 5: Key Training and Model Hyper-parameters

Data	
Batch size	128
Sequence length	512
Model	
Transformer (Decoder-Only)	
Hidden size, d_{model}	768
Feed-forward size, d_{ff}	3072
# layers	12
# heads	12
Dropout	0.1
LSTM	
Embedding Size	768
Hidden size	768
Layers	16
Dropout	0.1
Optimizer & Scheduler	
Optimizer	AdamW (Loshchilov and Hutter, 2019)
Optimizer Paramas (β_1, β_2)	$(0.9, 0.98)$
Learning rate	3×10^{-4}
Weight decay	0.01
Scheduler	Cosine Annealing & Linear Warmup

References

Kabir Ahuja, Vidhisha Balachandran, Madhur Panwar, Tianxing He, Noah A. Smith, Navin Goyal, and Yulia Tsvetkov. Learning syntax without planting trees: Understanding hierarchical generalization in transformers. *Transactions of the Association for Computational Linguistics*, 13: 121–141, 2025. doi: 10.1162/tacl_a_00733. URL https://doi.org/10.1162/tacl_a_00733. Accessed: 2025-05-20.

Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 1, learning hierarchical language structures. *arXiv preprint arXiv:2305.13673*, 2023. Version 3 posted June 2, 2024.

Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):e0130140, 2015. doi: 10.1371/journal.pone.0130140. URL https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0130140.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020a.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are unsupervised multitask learners. Technical report, OpenAI, 2020b. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf. Accessed: 2025-05-17.

Francesco Cagnetta, Leonardo Petrini, Umberto M. Tomasini, Alessandro Favero, and Matthieu Wyart. How deep neural networks learn compositional data: The random hierarchy model. *arXiv preprint arXiv:2307.13673*, 2023.

Lawrence Cayton. Algorithms for manifold learning. Technical Report 12(1–17):1, University of California at San Diego, 2005.

Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956. URL https://chomsky.info/wp-content/uploads/195609-.pdf. Definition of formal grammars in §2 ("A grammar is a device . . . ").

Nai Ding, Lucia Melloni, Hang Zhang, Xing Tian, and David Poeppel. Cortical tracking of hierarchical linguistic structures in connected speech. *Nature Neuroscience*, 19(1):158–164, 2016. doi: 10.1038/nn.4186.

Yanzhuo Ding, Yang Liu, Huanbo Luan, and Maosong Sun. Visualizing and understanding neural machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1150–1159, Vancouver, Canada, 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1106. URL https://aclanthology.org/P17-1106/.

J. Earley. *An efficient context-free parsing algorithm*. PhD thesis, Carnegie Mellon University, 1970.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

John Hewitt and Christopher D. Manning. A structural probe for finding syntax in word representations. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4129–4138, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1419. URL https://aclanthology.org/N19-1419/.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

Hugging Face OpenAI Community. Hugging Face model card: openai-community/gpt2. https://huggingface.co/openai-community/gpt2, 2025. Accessed: 2025-05-17.

T. Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. In *Scientific report AFCRL-65-758*, 1965.

K. Lari and S. Young. The estimation of stochastic context-free grammars using the inside–outside algorithm. *Computer Speech & Language*, 4(1):35–56, 1990.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

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

William Merrill, Vijay Ramanujan, Yoav Goldberg, Robert Schwartz, and Noah A. Smith. Effects of parameter norm growth during transformer training: Inductive bias from gradient descent. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3445–3464. Association for Computational Linguistics, 2021.

Shikhar Murty, Pratyusha Sharma, Jacob Andreas, and Christopher D. Manning. Grokking of hierarchical structure in vanilla transformers. *arXiv preprint arXiv:2305.18741*, 2023.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, pages 8024–8035, 2019.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Short Papers*, pages 464–468, 2018. URL https://aclanthology.org/N18-2074/.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021. doi: 10.48550/arXiv.2104.09864. URL https://arxiv.org/abs/2104.09864.

M. Tomita. An efficient augmented-context-free parsing algorithm. In *Proceedings of the 23rd ACL*, pages 31–41, 1986.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008, 2017. URL https://arxiv.org/abs/1706.03762.

Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 5797–5808, 2019. URL https://aclanthology.org/P19-1580/.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small, 2022. URL https://arxiv.org/abs/2211.00593.

Nelson Elhage Wang, Neel Nanda, Catherine Olsson, Nicholas Joseph, Andy Chen, Tom Henighan, Jacob Buckman, and Dario Amodei. Attribution in the wild: The building blocks of interpretability. *Transformer Circuits Thread*, 2025. https://transformer-circuits.pub/2025/attribution-graphs/biology.html.

Shunjie Wang and Shane Steinert-Threlkeld. Evaluating transformer's ability to learn mildly context-sensitive languages. In *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 271–283, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.blackboxnlp-1.21. URL https://aclanthology.org/2023.blackboxnlp-1.21/.

Nick Whiteley, Annie Gray, and Patrick Rubin-Delanchy. Statistical exploration of the manifold hypothesis. *arXiv preprint arXiv:2208.11665*, 2022.

D. H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, $10(2) : 189 - -208, 1967$.