Assessing The Syntactical Abilities Of Transformer Architectures

Bachelor's Thesis of

Michael Khal

Artificial Intelligence for Language Technologies (AI4LT) Lab
Institut for Anthropomatics and Robotics (IAR)
KIT Department of Informatics

Reviewer:	Prof. Niehues
Second reviewer:	Prof. Waibel
Advisor:	Prof. Niehues

15. August 2023 – 15. November 2023

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

PLACE, DATE

. . . . . .

(Michael Khal)

Abstract

This work investigates the syntactical abilities of transformer networks, focusing on their performance in tasks designed to assess their understanding of tree structures, as encountered for context-free grammars. The hypothesis posits that while transformers excel in many natural language processing tasks, their syntactical capabilities may be limited. Instead of relying on true generative principles, these models may employ what we term as "statistical heuristics" —- mechanisms that prioritize statistical patterns and heuristics rather than a deep understanding of syntactical structures. These statistical heuristics may involve a reliance on lower-level syntactical features to construct approximations for higher-level structures.

Two tasks, **BRACKET** and **MASK**, were designed to evaluate the ability to learn syntactical structures and dependencies of various transformer models, as well as one long short-term memory serving as a baseline. In **BRACKET**, models generate parse trees for sequences based on an unknown grammar, revealing that incentives are necessary to drive the learning of syntactical structures. The results suggest that transformers tend to prioritize lower-level syntactical features, using them as building blocks for approximating higher-level structures. Task design and tokenization strategies significantly influence the models' learning behaviors in this context. In **MASK**, models predict the class of a masked word within a sequence, highlighting the models' tendency to compute over linear word order rather than syntactical distance. The study further explores the models' dependency on data distributions, revealing that stochastically learned heuristics break for out-of-distribution data.

We discuss the implications of these observations, acknowledging the inherent limitations of our study. In addition, we propose promising directions for future research to further illuminate the intricate ways in which neural networks navigate and approximate the complex realm of syntax in human language.

Zusammenfassung

In dieser Arbeit werden die syntaktischen Fähigkeiten von Transformernetzen untersucht; mit dem Schwerpunkt auf Baum-Strukturen, wie sie bei kontextfreie Grammatiken begegnet werden. Unsere Hypothese besagt, dass Transformer zwar in vielen Aufgaben der Verarbeitung natürlicher Sprache herausragen, ihre syntaktischen Fähigkeiten jedoch begrenzt sein könnten. Anstatt sich auf echte generative Prinzipien zu stützen, verwenden diese Modelle möglicherweise das, was wir als *statistische Heuristiken* bezeichnen - Mechanismen, die statistischen Mustern und Heuristiken Vorrang vor einem tiefen Verständnis der syntaktischen Strukturen geben. Über Heuristiken über Distributionen einfacherer syntaktischer Strukturen können Netze hiermit höhere, möglicherweise unerreichbare Stukturen approximieren, insoweit sie für ihre jeweilige Aufgabe von Nöten sind.

Zwei Experimentreihen, **BRACKET** und **MASK**, wurden entwickelt, um die Fähigkeit verschiedener Transformer und einem long short-term memory (LSTM) zu bewerten, syntaktische Strukturen und Abhängigkeiten zu lernen. In **BRACKET** generieren die Modelle Parse-Bäume für Sequenzen, die auf einer unbekannten formalen Grammatik basieren. Es stellt sich heraus, dass Anreize notwendig sind, um das Erlernen von syntaktischen Strukturen zu steuern. Die Ergebnisse deuten darauf hin, dass Transformatoren dazu neigen, syntaktische Merkmale auf niedrigerer Ebene zu priorisieren und sie als Bausteine für die Annäherung an Strukturen auf höherer Ebene zu verwenden. Das Aufgabenentwurf und die Tokenisierungsstrategien beeinflussen das Lernverhalten der Modelle in diesem Zusammenhang nicht vernachläsigbar. Bei **MASK** sagen die Modelle die Klasse eines maskierten Wortes innerhalb einer Sequenz voraus, was die Tendenz der Modelle verdeutlicht, eher lineare Wortreihenfolge als syntaktische Distanz zu benutzen. Die Studie untersucht außerdem die Abhängigkeit der Modelle von der Datenverteilung und zeigt, dass stochastisch gelernte Heuristiken bei Daten außerhalb der Verteilung versagen.

Wir erörtern die Implikationen dieser Beobachtungen, wobei wir die inhärenten Grenzen unserer Studie anerkennen. Darüber hinaus schlagen wir vielversprechende Richtungen für künftige Forschungen vor, um die komplizierten Wege, auf denen neuronale Netze den komplexen Bereich der Syntax in der menschlichen Sprache navigieren und approximieren, weiter zu beleuchten.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Why is syntax important for understanding?

Language can be viewed from three different perspectives: that of semantics, i.e. meaning, that of pragmatics, i.e. how language is used, and that of grammar, i.e. how and in what form language is expressed. Grammar may be broadly divided into syntax, morphology and phonology. Phonology deals with the structure of the sounds of language, morphology deals with the structure of components of words and syntax deals with the structure of sentences. All of those components may interact. In a broader sense, we may include morphology into syntax. For the task of sequence modeling we need limited access to sound or pragmatics, but are especially interested in the question of how to form *proper*, i.e. syntactically correct, and *meaningful*, i.e. semantically enriched, sentences.

While semantics captures the content of language, syntax captures its form. The semantical representation of a sentence expresses a structure of objects or notions, whereas its syntax captures the structure of their respective signifiers, i.e. words or sub-words. Here it might become apparent that the distinction between semantics and syntax cannot be strictly uphold. Syntactical information largely influences how to semantically interpret a sentence, whereas contextual information obtained from previous sentence meaning may influence the way a sentence is parsed.

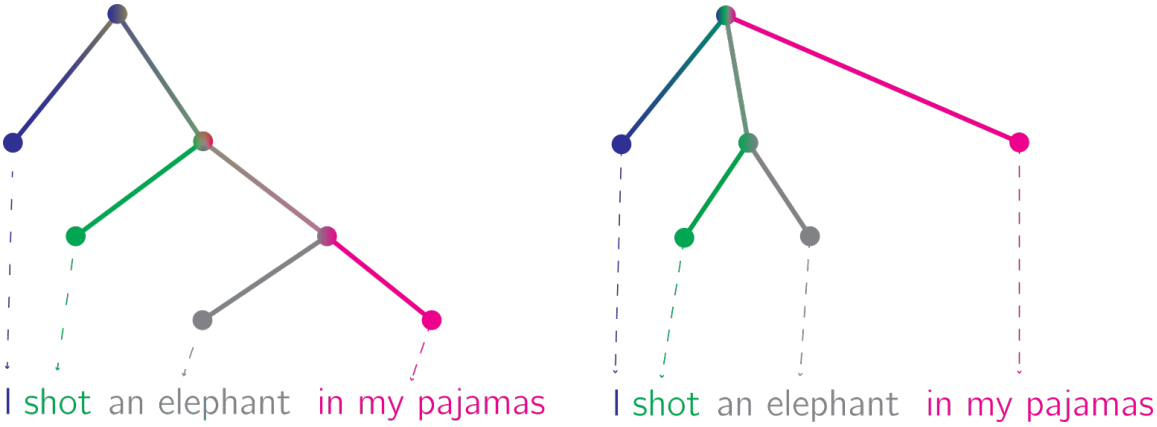


Figure 1.1: Two pseudo parse trees for “I shot an elephant in my pajamas.”

Take, for example, the sentence: “I shot an elephant in my pajamas.”. The sentence may be interpreted in two ways. The narrator wears pajamas or the elephant does. Although the sequence remains the same, the underlying syntactical representations give a clear

idea of what the sentence intended to convey.

Thus, generating a meaningful sentence, linguist Noam Chomsky argues [12], implies generating the underlying syntactical structure. This is achieved through a sequence of generative operations, that may partially be influenced by "semantical" features. On the surface, this representation will be transformed into a sequence of words. Understanding a sentence will therefore involve running this process in the opposite way, i.e. parsing a sequence into a higher level syntactical representation that imposes semantical meaning.

1.2 What does this have to do with transformers?

Transformer networks, like all artificial language models, are man-made tools for performing language-related semantic transduction tasks, whether they may be text summarizing, translating, or answering questions. All these tasks require the models to have semantic knowledge. To obtain this knowledge, as argued previously, a model must have to learn syntax. As we will see, this behavior can also be observed emerging somewhat in "self-taught" artificial models, where the models seem forced to parse syntactical features first to obtain semantic information in higher layers.

In the example above, humans parsing that sentence would vaguely form the depicted trees, that impose meaning. If we want language models to model natural language successfully, they should approach such sentences in a similar way. In fact, to make sure that a model correctly interpreted a sentence, there is a point to be made that it should parse them in the same way as humans do. For this, two things are needed: models need to have the computational capability to learn human language and must be intrigued to do so during training. Syntactical capacity in regard to computational systems is greatly formalized in a way that we can form hierarchies: the Chomsky hierarchy tells us what kind of languages can be processed by what kind of system. Human language is thought to be located between type-1 and type-2. Transformers seem to already struggle learning languages of type-3. This might be suprising, considering their overwhelming success in NLP tasks. We call this the *theory-practice gap*.

1.3 How might transformers process natural language?

Our work contributes to making sense of the theory-practice gap. From what we have said, we hypothesize that transformers might employ the limited capacities they have to find what we call *statistical heuristics* for the syntactical distributions they are supplied with.

It's crucial to recognize that the notion of *statistical heuristics* should not merely imply that models learn simple Markov chains over individual tokens. Models may, in fact, integrate different, somewhat nontrivial lower-level syntactical features to approximate higher-level generative principles that would else be unattainable. This means, that they

may find vast amounts of correlations over syntactical features of differing complexities.

This approach would not impair performance much for conventional language tasks, but may indicate a substantial deficit regarding learned syntactical representations.

1.4 Our Research questions

From this hypothesis, we may form the following more specific research questions.

Q0 Does a given transformer model learn the structure with which a language was generated?

Q1 Are there inductive biases that force the model to learn complex solutions involving generalizable high-level features instead of "lazy" statistical ones?

Q2 What aspects of real-world implementations have an effect on the motivation to learn generalizable, high-level structures?

Q3 How sensible are those statistical heuristics to training and evaluation data distributions?

We design two tasks that deal with aspects of our research questions: **BRACKET** and **MASK**. **BRACKET** will be our main task, covering most of our work. It will try to analyze how far transformers learn the real structure with which a language was generated (**Q0, Q1**) and what factors can influence its tendency to do so (**Q2**). The task will also try to show how strong the inductive bias of the model leans towards finding statistical heuristics, instead of structure. **MASK** resembles common pretraining tasks, increasing the transferability of our results to field observation. With it we will try to confirm notions that the model employs *statistical heuristics* that may depend greatly on data distributions seen during training (**Q3**), which would be consistent with our hypothesis.

For this, we will first clarify preliminary knowledge in chapter 2 and elaborate on the state of related research in chapter 3. Here, research revolving our aforementioned theory-practice gap will be further illuminated.

In chapter 4 our hypothesis will be discussed and an overview and the reasoning behind our conducted experiments will be provided, aspects of which will be more precisely defined in chapter 5. Our results and analysis thereof will be presented in chapter 6. Finally, we will conduct a broader reflection of our work and discuss limitations in chapter 7, culminating in our conclusion in chapter 8.

2 Concepts

2.1 Formal Grammars

2.1.1 Formalism

Formally, a **language** L is a potentially infinite set of sequences $w_i = x_0^i x_1^i \dots$ consisting of elements, called **letters**, that are part of the languages **alphabet** $\Sigma = \{x_1, x_2, \dots\}$. In most cases, the alphabet is finite. A **grammar** G provides a set of rules to generate a language $L(G)$. This is achieved by defining a start symbol S and production rules $P = \{r_0, r_1, \dots\}$, which are pairs of sequences $r_i = (\alpha_i, \beta_i)$ of **variables** $V : \alpha_i, \beta_i \in V$. The rules are most commonly pictured as $\alpha_i \rightarrow \beta_i$. The grammars **derivation operation** ($\overset{G}{\Rightarrow}: V* \rightarrow V*$) takes a sequence $a = a_0 a_1 \dots a_n$ and produces a new sequence $b = a_0 \dots a_{t-1} \beta_i a_{t+k+1} \dots a_n$ by replacing a subsequence $\alpha_i = a_t a_{t+1} \dots a_{a_t+k}$ by β_i if $(\alpha_i, \beta_i) \in P$. To define which of these produced sequences is part of the generated language and not a mere intermediary computation step, we split V into two complementary sets: **nonterminals** N and **terminals** T. Terminals are precisely the alphabet of the produced language and nonterminals are arbitrary syntactical symbols used in the intermediary production process that do not occur in the produced words, like the starting symbol.

As there may be multiple possible applicable rules for a sequence, we can define ($\Rightarrow: [V*] \rightarrow [V*]$) more broadly as taking a set of sequences and producing the set of all possible derivations. Furthermore, if $\overset{k}{\Rightarrow}$ means applying this operation k times and $\overset{*}{\Rightarrow}$ means applying it arbitrary number of times, we may define the produced language $L(G)$ as follows: $L(G) = \{w | S \overset{*}{\Rightarrow} w \wedge \forall_{w_i \in w} w_i \in T\}$.

To be more precise, such grammars are called **phrase structure grammars** or **constituency grammars**.

The following sections will explain some fundamental concepts about the algebraic theory of formal languages laid out in 1959 by Chomsky and Schüzenberger [11].

2.1.2 Chomsky-Schützenberger Hierarchy

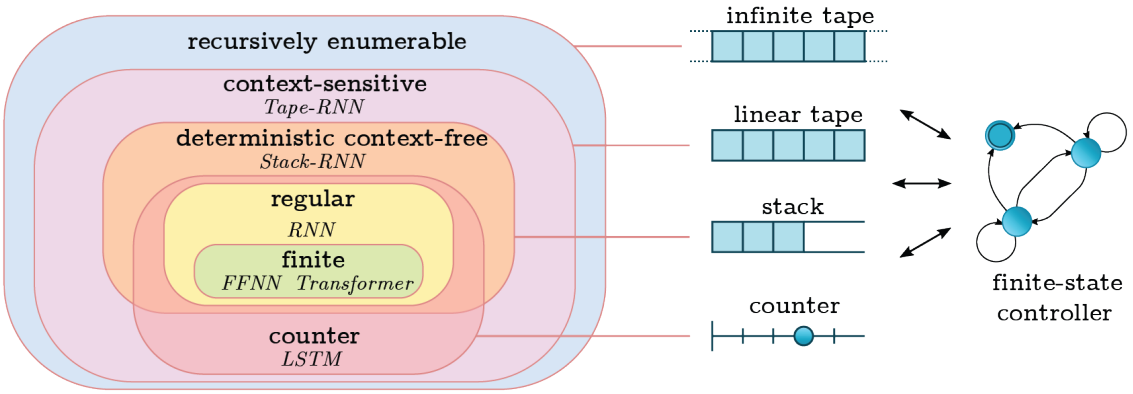


Figure 2.1: Chomsky hierachy with corresponding automata. Note, that Transformers were placed in the regular class by Deletang et al. (2022).
Source: [15]

2.1.2.1 General

By restricting the types of possible rules, we obtain a hierarchy in terms of expressiveness of grammars with the corresponding difficulty of computability. This can be further illustrated by constructing automata with certain computational capacities that accept the according language. The automaton will hereby iteratively receive a word as its input and will have to decide, by stopping in a predefined set of states, called the accepting states, whether or not the word belongs to the desired language. If grammar and automaton produce the same set of words, i.e. the language, they are equivalent in terms of expressiveness and computability.

For illustrative reasons, we will be using $A, B \in N; a, b \in T; \alpha, \beta \in V*; \gamma \in V+; \epsilon$ as the empty word and $S \in N$ as the starting symbol.

2.1.2.2 Regular Grammars

A **regular** grammar or a Type-3 grammar generates a regular language. Its production rules are constrained in the following way:

- The left-hand side (LHS) may only contain a singular nonterminal
- The right-hand side (RHS) may either be the **empty word** ϵ or a nonterminal followed by a terminal
- The form could be described as: $A \rightarrow aB \mid \epsilon$

More precisely, this kind of regular grammar is called right-regular. When swapping the terminal and nonterminal on the RHS we obtain a left-regular grammar. Both are equivalent in terms of expressiveness and computability.

Intuition One can imagine the production process of a regular grammar as successively adding the next letter to the word based on the current state, i.e. the previously underived nonterminal, and producing one new state, i.e. the new nonterminal or terminal. Letters at a certain position may therefore, through the currently underived nonterminal, only have a one-way dependency to letters to their left (or to their right, in case of left-regular languages). In other words, the next letter may, therefore, only be dependent on the current state.

Computational equivalence It might therefore be quite intuitive why regular languages are precisely the languages that can be accepted by finite-state automata.

Closure properties Regular languages can, in addition to the previous rule-based approach above, be described through their closure properties: Let the class of *basic languages* contain the single letter language $\{a\}$ for $a \in \Sigma$, the empty set empty set $\emptyset$ and the set of the empty word $\{\epsilon\}$. This class of basic languages is part of the regular languages. Any other regular language can only be obtained by its closure over union $\cup$, concatenation $\circ$ and the **Kleene star** $*$, where $A*$ denotes the set of arbitrary many repetitions of A, including none.

Regular languages are additionally closed under intersection $\cap$, set minus $\setminus$, homomorphisms, and complements $\overline{\square}$, among others.

2.1.2.3 Context-free Grammars

A **context-free** grammar or Type-2 grammar generates a context-free language. Here, the production rules are constrained in the following way:

- Again, the LHS may only contain a singular nonterminal
- The LHS may contain a non-empty, arbitrary combination of nonterminals and terminals
- The form could be described as: $A \longrightarrow \gamma$

Moreover, the rule $S \longrightarrow \epsilon$ may be added if S does not occur on a RHS of any other rule. This allows the empty word to be part of a context-free language.

Intuition Here, the production process is rather similar to the generation of a tree with nonterminals as its non-leafs and terminals as its leafs. A production step can be equated to taking a certain unfinished tree and expanding all of the leaf nodes that are nonterminals until all leafs are terminals. Each tree node would then resemble a certain area of the final word, the deeper the more granular. Thus two or more letters or areas may depend upon each other only via their tree nodes' common ancestors. Here, it might be evident why context-free languages cannot produce words in which two pairs of letters have cross-dependencies, i.e. variables within a region enclosed by two dependent nodes (or

variables) may only depend upon other variables within this area, not outside of which. This will be eluded clearer in the section on mildly context-sensitive grammars.

Computational equivalence context-free grammars are the languages that can be accepted by a finite-state automaton employing an additional stack, i.e. a pushdown automaton. The automaton may push and pop symbols of a predefined stack-alphabet onto the stack and consider the uppermost element for its state transition.

Closure properties Context-free grammars are closed under $\cup$, $\circ$, $*$, intersection with regular languages, and homomorphism, among others.

2.1.2.4 Context-sensitive Grammars

A **context-sensitive** grammar or Type-1 grammar generates a context-sensitive language. Here, the production rules are constrained in the following way:

- The LHS contains a nonterminal, that is enclosed by two arbitrary sequences of nonterminals and terminals which can be seen as its context
- The RHS contains an arbitrary non-empty sequence of variables enclosed by the context
- The general form can be described as: $\alpha A \beta \rightarrow \alpha \gamma \beta$

Intuition It is more complex to imagine the production process of context-sensitive grammars. As the name tells, rules here may depend on the variables that are near a given nonterminal.

Computational equivalence Context-sensitive grammars generate precisely the languages that can be accepted by a nondeterministic Turing machine, whose tape is linearly bounded, i.e., a linearly bounded automaton.

2.1.2.5 Unrestricted Grammars

Unrestricted Grammars or Type-0 grammars generate the class of *recursively enumerable* languages. They do not have any restriction on the kind of production rules, except that the LHS has to contain at least one nonterminal.

Computational equivalence This class covers all known formal languages. They correspond to all the languages that a Turing machine accepts. More precisely, the Turing machine has to halt on and accept strings belonging to that language, but does not necessarily have to halt on strings not part of that language.

2.1.3 Further concepts

2.1.3.1 Star-free regular languages

If we restrict regular languages to not use the Kleene star during construction, we can define a subset of regular languages: **star-free** regular languages. They are, among others, closed under boolean operations and concatenation. The restriction, however, does not prevent star-free languages to express repetitions:

Example: $a*$ is star-free

$$a* = \overline{\overline{\emptyset} \circ (\Sigma \setminus a) \circ \overline{\emptyset}}$$

Rather, star-free grammars intuitively generate the kind of language where a language, when repetitions are involved, can be accepted without counting parts of the repetition. As such, they are equivalent to *counter-free languages* as described by McNaughton and Seymour (1971) [38]. Note that this is not the same as the complement to counter languages, which we will define further below. In natural language, we rarely count the number of syntactical objects to verify that a sentence is syntactical. The biggest part of regular expressions in natural language is thus star-free.

Definition: Dot-depth

Suppose now that we define a closure operator $\mathfrak{B}$, which closes a class of languages under boolean operators, and an operator $\mathfrak{M}$, that closes a class of languages under concatenation. Let $\mathfrak{E}$ denote the class of *basic languages* as defined in subsubsection 2.1.2.2. We may now alternately apply our operators to $\mathfrak{E}$ to obtain star-free regular languages. The number of times we apply both of these operators is called the **dot-depth** of our language class. We may then form a hierarchy of star-free languages: the **dot-depth hierarchy**[43] [13]. For dot-depth n class $\mathcal{B}_n$ can thus be obtained inductively:

$$\mathcal{B}_0 = \mathfrak{E}, \forall_{n \in \mathbb{N}} : \mathcal{B}_n = \mathfrak{B}\mathfrak{M}\mathcal{B}_{n-1}$$

.

Example: Dot-depth

Following the above construction for $a*_{„}$ we find that we have to first apply boolean operators to a, Σ and $\emptyset$, then concatenate them and finally use boolean inversion $\overline{\square}$. We end up with 1.5 operations, which belongs to dot-depth 2.

2.1.3.2 Counter languages

Counter languages are languages that can be accepted by a deterministic finite automaton employing one or more counters. As such, they are a subset of regular languages. During state-transition the automaton may increase the values of the counters by an arbitrary

$m \in \mathbb{Z}$ or set them to zero according to the current state. The state transition, in addition to the current state and the input symbol, may depend on *zero checks* per counter. For each counter the automaton may hereby know whether it is zero or not. A word will then be accepted if the state is in an accepting state F with a predefined configuration of zero-checked counter states.

For a single counter, counter languages are a sub-set of context-free languages, where the counter may simply be simulated by the stack. For more than one counter, it can be shown that there is an nontrivial construction that is equivalent to a Turing machine.

2.1.3.3 Mildly context-sensitive Grammars

Mildly context-sensitive grammars [28] are conceptually situated between context-free and context-sensitive grammars. They do not occur in the original Chomsky hierarchy but were explored a little bit later as they seem to be a better approximation of the capacity of natural language than the other ones. Through the findings of certain attributes in some natural languages that exceed the abilities of context-free grammars, e.g. cross references in Swiss German [47], one could argue that the natural language may be stronger than context-free languages. On the other side, context-sensitivity seems to be a too loose definition to capture natural language.

The class of mildly context-sensitive languages does not have a unified grammar formalism like the other grammars. The class is instead defined by the following attributes:

- **Larger than context-free**
  The grammars generate (at least) all context-free languages.
- **Cross-serial dependencies**
  The grammars are able to generate a limited amount of cross-serial dependencies. As discussed previously, this is an attribute that cannot be obtained from context-free languages. This is equivalent for being able to solve the COPY task, i.e. being able to generate $\{w^n | w \in T*\}$ for a $n \geq 2$.
- **Constant growth**
  The lengths of the generated words grow linearly.
- **Polynomial parsing**
  The languages' membership problem must be solvable in deterministic polynomial time. This restriction ensures demands of efficient parsing by humans.

There exists a variety of formalisms that cover areas of the mildly context-sensitive grammar class, but up to this date none cover all possible languages. For instances, a way to achieve cross-serial dependencies for context-free languages is to add an operation that could alter a derived tree by expanding an inner node by adjoining a new tree at the respective node. This so called tree-adjoining grammar (TAG) formalism was the first to be introduced in the mildly context-sensitive grammar class. Now there have been found more various formalisms weakly equivalent to TAG [56] and even formalisms, that further

generalize similar grammar formalisms within this class (see Linear context-free rewriting systems (LCFRS) [30] and multiple context-free grammars (MCFG) [46]). However, even the more generalized form of the formalisms found here seems to not cover certain mildly context-sensitive languages. [29]. Note furthermore that "weakly equivalent" means that two grammar formalisms generate the equivalent sets of languages, not necessarily equivalent sets of syntactical representation. This is indeed not the same, as semantical information of a sentence relies vastly on the underlying syntactical representation.

Computational equivalence Automaton equivalences depend on the formalism employed. It has been shown, for instances, that tree-adjoining grammars generate the languages that can be accepted by an embedded pushdown automaton, i.e. an automaton with a stack of stacks.

2.1.3.4 Dyck-Languages

Definition: Brackets of N types

Brackets of **N types** are symbols of the unification of a pair of arbitrary, but equally sized finite alphabets $T \cup \bar{T}$, where $|T| = |\bar{T}| = N$. Furthermore, this implies a defined bijective mapping $\psi : T \to \bar{T}$. For an arbitrary, but fixed enumeration of T we will denote T_i as $\mathbf{BR}_i$ and call it the **opening bracket of type** i and $\psi(T_i)$ as $\mathbf{BL}_i$ and call it its respective **closing bracket of type** i. For our following thoughts the choice of $T, \bar{T}$ and ψ will not be of interest.

Definition: Correctly bracketed and hierarchically bracketed

Let $s = s_0 s_1 \dots s_m$ be a sequence of brackets of N types, $s_{0:k} = s_0 \dots s_k, k \leq m$ denote a sub-sequence thereof and let $\mathbb{B}_t : \Sigma* \to \mathbb{N}$ be a function that returns the number of opened ($\mathbb{B}_t^O$) and closed ($\mathbb{B}_t^C$) brackets of type t in a given sequence. Further, let $\mathbb{B}^\Delta = \mathbb{B}^O - \mathbb{B}^C$. We may then define:

$$s \text{ is } \textbf{correctly bracketed} \Leftrightarrow \forall_{t \in N} \forall_{k \leq m} \mathbb{B}_t^\Delta(s_{0:k}) \geq 0 \text{ and } \mathbb{B}_t^\Delta(s) = 0$$

$$s \text{ is } \textbf{hierachically bracketed} \Leftrightarrow \forall_{s_i \in s} \exists_{s_j \in s} s_{min(i,j):max(i,j)} \text{is correctly bracketed}$$

Note:

$$s \text{ is hierarchically bracketed} \Rightarrow s \text{ is correctly bracketed}$$

Named after the mathematician Walther von Dyck, Dyck-languages represent the class of context-free languages containing all **hierarchically bracketed** bracket expressions of N types. We will denote the Dyck language consisting of N types of brackets as $\mathcal{D}_N$. Dyck-languages cannot be regular. For an automaton sequentially parsing a Dyck string to accept the next symbol, it has to track not only the count of open brackets (*correct*) , but also the order in which they have been encountered (*hierachical*). For arbitrary strings this is only possible with infinitely many states. In contrast, a pushdown automaton can

track both of these things by pushing the open bracket symbols on the stack and popping them off when the respective closing bracket has been encountered.

Dyck-languages are a sub-sets of **Shuffle Dyck** languages. **Shuffle-N Dyck** will hereby denote a language consisting of N brackets that is **correctly bracketed**, but not necessarily hierarchical. It will therefore hold that $L(\mathcal{D}_k) \subseteq L(\text{Shuffle-N Dyck})$. By our previous analysis, this language is also not regular, but can instead be accepted by a counter automaton employing one counter per bracket type.

2.1.3.5 Chomsky-Schützenberger representation theorem

There is a deep connection between context-free languages and Dyck languages. For a small intuition, we should take a closer look on how pushdown automatons accept context-free languages. Consider a context-free grammar G. We want to accept and only accept all $s \in \mathcal{L}(G)$. s will precisely be part of G, when it can be generated out of the starting symbol S. As previously described, the generation of s from S will hereby look like a tree, with the nonterminals being the inner nodes and the terminals or words being the leafs. As we will later see, such trees are representable as bracket expressions, where each kind of node, i.e. each variable, will be assigned a bracket type. String of context-free gramars are therefore hidden bracket expressions. The Chomsky-Schützenberger representation theorem validates our intuition, stating the following:

Definition: Chomsky-Schützenberger representation theorem

A language L over the alphabet Σ is context-free iff there exists a matched alphabet alphabet $T \cup \bar{T}$, a regular language R over $T \cup \bar{T}$, and a homomorphism $h : (T \cup \bar{T})^* \to \Sigma^*$ such that

$$L = h(\mathcal{D}_{|T|} \cap R)$$

Context-free languages can thus be represented by a homomorphism over a Dyck-N language interesected with a regular one. Most proofs thereof are constructed in a way, that N will be precisely the number of terminals and nonterminals of our our grammar, e.g. [3]. As we will see, a lot of research is thus revolved around the question whether certain systems can model $\mathcal{D}_N$. If so, they should also be strong enough to model any other context-free language. More precisely, already showing that a system can model $\mathcal{D}_2$ might be sufficient. The intuition behind this is that for any N we can construct a hierarchical bracket language $D_2 = \mathcal{D}_2 \cap R_2$ for some regular language R_2 that can be transformed to $\mathcal{D}_N$ via a morphism $h_2 : (T_2 \cup \bar{T_2})^* \to (T_N \cup \bar{T_N})^*$. The language will simply binary encode each bracket type using its two bracket types, making sure that the encodings are suffix-free, and then decode the encodings h_2. When viewing the corresponding pushdown automata instead, this is equivalent to binary encoding the stack alphabet in the same way, employing additional states to decode and encode such symbols. This is not possible for only a single bracket type ($\mathcal{D}_1$) as then the encoding would necessarily be non suffix-free and the homomorphism impossible.

2.2 Language Generation Models

2.2.1 Formalism

A *sequence-to-sequence* (Seq2Seq) or *transduction* task **TASK** consists of pairs $\mathbf{TASK} = \{(S^b, T^b) \mid 1 \leq b \leq B\}$ of sequences $S^b = \{s_i^b | 1 \leq i \leq n, s_i^b \in \Sigma_S\}$ and $T^b = \{t_i^b | 1 \leq i \leq m, t_i^b \in \Sigma_T\}$, where for an arbitrary pair $(S, T) = (S^b, T^b) \in \mathbf{TASK}$ a computational model $\mathcal{M}$, equipped with S has to predict $\hat{T}$, so that T and $\hat{T}$ are as near as possible according to a predefined loss metric.

Common tasks for natural language include translation into another language or text summarizing. For this, the model has to learn to extract semantical information, and thus necessarily syntactical information. In order to produce an according output sequence the extracted semantical information needs to be transformed into a new form, according to the respective task, after which it has to be poured into its final syntactical form. Therefore, we can observe two phases of a Seq2Seq process: information retrieval and output generation. In natural language processing for this kind of task the standard approach is to employ an encoder-decoder architecture. The encoder $\mathcal{E} : \Sigma_S^n \to \Theta$ will hereby cover the retrieval of information by converting the input sequence to a hidden state $\theta \in \Theta$. The decoder will use this state to generate the output sequence. As squeezing all the information from the sequence into a single hidden state often ends up being a major bottleneck regarding information flow, most encoders instead produce a hidden state $\dot{\theta}_i$ for each s_i of our sequence: $\mathcal{E} : \Sigma_S^n \to \dot{\Theta}^n$, which den can be flexibly aggregated together $\theta = \text{aggr}(\dot{\theta}_0 \dots \dot{\theta}_s)$.

For this, **auto-regressive** generation is the go-to approach. A decoder model $\mathcal{D}_\theta$, equipped with the inner state $\theta = \mathcal{E}(S)$, will hereby iteratively produce a probability distribution $\hat{T}_i$ over Σ_T, from which one can sample $\hat{t}_i \sim \hat{T}_i$. For this, the model will have access to its previously sampled predictions $\hat{t}_0 \sim \hat{T}_0 \dots \hat{t}_{i-1} \sim \hat{T}_{i-1}$. Thus, we can view our model as a probability function conditioned on previously predicted samples $\hat{T}_j = \mathcal{D}_\theta(\Sigma_T | \hat{t}_0 \dots \hat{t}_{j-1})$. Some models instead only use a *window* of previous predictions or solely rely on their inner state. For window size k, we can denote this as $\hat{T}_j = \mathcal{D}_\theta(\Sigma_T | \hat{t}_{j-k-1} \dots \hat{t}_{j-1})$.

In the course of the iterative generation process, the decoder will update its inner state, making it time dependent: $\hat{T}_j = \mathcal{D}_{\theta_|}(\Sigma_T | \hat{t}_0 \dots \hat{t}_{j-1})$. Furthermore, to accelerate and stabilize the training process, the decoder will often be provided with the left context of the ground truth: $\hat{T}_j^{training} = \mathcal{D}_\theta(\Sigma_T | t_0 \dots t_{j-1})$. This is called **teacher forcing**.

In the following, we will alter our current definition of formal languages to consider sentences instead of words. For this, we will define a language L as a set of sequences, called **sentences**: $L = \{s_i = x_0^i x_1^i \dots | 1 \leq i\}$ consisting of **words** or sub-words instead of letters. This doesn't change the formalism, but is just a redefinition of terms.

2.2.2 Tokenization and embedding

When employing language models, we need to find a way of transforming sequences of text into mathematical objects. For this, we will first have to cut our sequence into granular parts that carry semantical or syntactical meaning. These will then be the elements of what our model will perceive of a sequence. The elements that are obtained from this process are called tokens and thus the process is called tokenization. Our

sequence $S = \{s_i | 1 \leq i \leq n, s_i \in \Sigma_S\}$ will therefore be transformed into another sequence $\dot{S} = \{\dot{s}_i | 1 \leq i \leq \dot{n}, s_i \in \Sigma_{tok}\}$ using a reversible morphism $tok : \Sigma_S^+ \rightarrow \Sigma_{tok}^+$. While theoretically possible, it is uncommon for a tokenizer to join successive words or sub-words thereof into one token. Instead, words will rather be split into granular sub-words. Therefore, we can define $tok : \Sigma_S \rightarrow \dot{\Sigma}_S^+$. The most simple tokenization strategy is **word-wise** tokenization, where tokenization is just the identity function ($tok = \infty$).

Having obtained the final model vocabulary $\Sigma_{tok} = \{\sigma_0 \dots \sigma_N\}$, we finally need to convert the tokens into tensors, so that our model can compute them. This is achieved by representing each token σ_i as a basis vector of the N dimensional vector space $\mathbb{R}^N$. More precisely, if we call this transformation h we can represent it as

$$h(\dot{s}_i) = (x_0 \dots x_N)^T, \text{ where } x_i = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

This is called **one-hot encoding**.

As tokenization vocabularies tend to get quite large, so does this vector space. We thus need to reduce the dimensionality of our vectors. This is achieved by an **embedding** operation $\text{embed} : \mathbb{R}^N \rightarrow \mathbb{R}^H$, which in most cases is just a simple fully collected linear layer that transforms vectors of dimensionality N to dimensionality H, which is the hidden size. By choosing this layer to be part of our model, it may choose which kind of transformation suits its desires best.

2.2.3 Recurrent neural networks

A recurrent neural network is designed to process a sequence $S = \{s_i | 1 \leq i \leq n, s_i \in \Sigma_S\}$ iteratively. Starting with an initial hidden state h_0, for a given time step t it will take the next element s_t of the sequence and its current hidden state h_t to produce a new hidden state h_{t+1}. The output sequence will then be obtained by transforming the hidden states through an easy transformation. When training such a network using *backpropagation through time* (bptt), the gradients for each time step will be multiplied according to the derivative chain run. For long sequences and activation functions, that may produce small gradients. This leads to exponentially small values, which cause numerical errors. As a result, long-range dependencies may get lost. This problem is called the **vanishing gradient problem**.

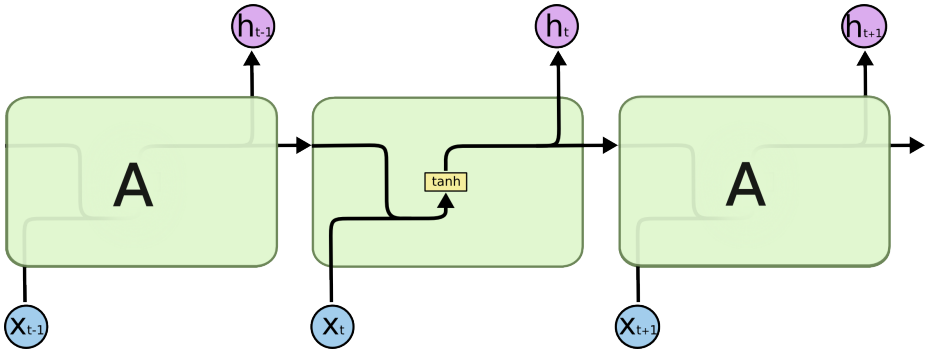


Figure 2.2: A basic RNN cell employing *tanh* activation.
Source: [39]

2.2.4 Long short-term memory

Long Short Term Memories (LSTMs) were introduced in 1997 by Hochreiter and Schmidhuber [27] and had been state-of-the-art in NLP, before transformers took over the field. A LSTM is a recurrent network consisting of one or more LSTM cells. These cells are designed to give the model explicit control over the information flow within a time step, weakening the problem of vanishing gradients. In addition to the hidden state, the so-called **cell state** c, which contains long-distance information, will be implicitly passed from cell to cell. Using the current sequence element x_t and the previous hidden state h_{t-1} the model can alter the cell state c_{t-1}. It may forget aspects of it, using the so-called **forget gate**, update it, using the **input gate**, and decide how to assemble the next hidden state h_t, using the **output gate**. Thus, LSTMs have a window size of $k = 1$.

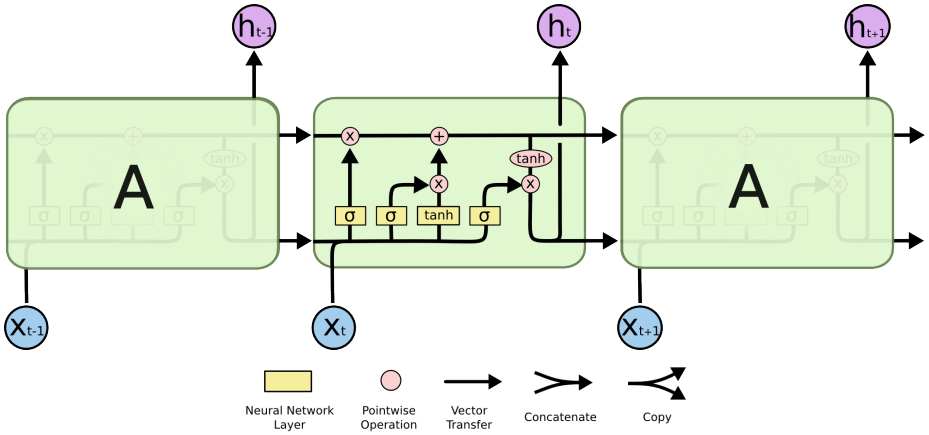


Figure 2.3: A basic LSTM cell. The upper horizontal arrow symbolizes the information flow of the cell state. To alter states, the three gates employ the sigmoid activation function in combination with element wise multiplication.
Source: [39]

2.2.5 Transformers

Transformers were introduced by Vaswani et al. in 2017 in their paper "Attention Is All You Need" [55]. There is an encoder and a decoder part to the model, but LLMs can also only employ encoder layers, such as BERT [16], decoder layers, such as GPT-3 ([10]) or use both, such as BART [34]. The core of transformers consists of **self-attention**, which allows it to compute powerful word embeddings, i.e. hidden states, by enabling token positions to attend to each other. In the following, we will discuss the architecture a little more in detail.

2.2.5.1 Positional Encoding

After tokenization, we end up with a sequence $\dot{S} = \{\dot{s}_i | 1 \leq i \leq n, \dot{s}_i \in \Sigma_{tok}\}$ of tokens. Using aforementioned operations we obtain n vectors of size $\mathbb{R}^H$, $E = \{\dot{e}_i \mid 1 \leq i \leq n, e = \text{embed}(h(\dot{s}_i))\}$. As we will see, transformers are permutation-invariant, making them highly parallelisable and thus efficient to train. However, language vastly relies on word order. Positional information thus needs to be encoded into our embeddings, so our model can infer syntactical dependencies. This can be achieved explicitly by adding either fixed or learnable, position-dependent vectors to our embedding; or it can be achieved implicitly in the decoder of our model (**positional masking**). The original paper proposes **sinusoid** positional encoding, which is an explicit and fixed positional encoding approach. Sinusoid functions with different frequencies will be computed over the dimensions of our embedding space. Based on the position of $\dot{e}_i$, these functions will be evaluated at the position i. By doing this, any neighboring positional embedding vector is obtainable through a linear transformation. However, LLMs usually employ learnable positional encodings, as such are more adaptable to data distributions.

2.2.5.2 Attention mechanism

We first need to understand how the core part works: the attention mechanism. In most cases, all the information will be provided by only one set of embeddings. If so, this process is called **self-attention** as opposed to **cross-attention**, which will be explained later.

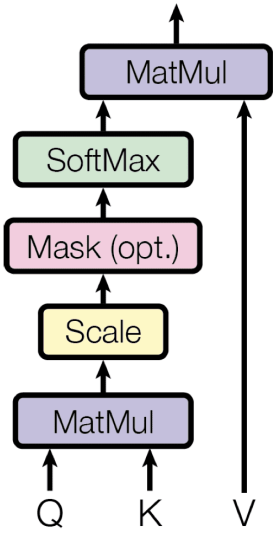


Figure 2.4: The self attention mechanism. The scaling factor is $d_k^{-\frac{1}{2}}$, d_k being the dimension of our keys and queries.

Source: [55]

As in the name, the attention block takes in n embeddings and transforms them by letting them attend to each other. Each embedding will hereby have the possibility of viewing all the other embeddings and decide which parts of itself to update. This enables finding dependencies between each other, as is often the case in language. For this, there are three weight matrices: the query matrix W_Q, the key matrix W_K and the value matrix W_V. Each embedding e_i, using these matrices, will generate its query q_i, key k_i and value v_i. Intuitively, q_i expresses what our word is searching for, k_i a description of what it has to offer, and v_i the content it has to offer. Therefore we will compare our query to every key, including our own, to obtain a weighting over all the embeddings and use this weighting in combination with the values to update our embedding. Mathematically, this is done by stacking all keys, values, and queries to the matrices Q, K and V and applying according operations. The comparison between queries and keys will be performed using **dot product multiplication**, which is equivalent to multiplying the matrices Q and K . The product will then be down-scaled to mitigate the problem of vanishing gradients. After that, to obtain a weighting factor between 0 and 1 , our product will be normalized through the $softmax$ function. The weighting factors will then be multiplied by our value matrix V to obtain the final result.

2.2.5.3 Multi-head attention

In reality, these operations are not performed along the entire embedding dimension. Instead, we define a number of **heads**, each possessing their own query, key, and value matrices. Each head will be supplied with a portion of the embedding dimension, for which it will then compute the above attention mechanism. The results will then be combined again to obtain a vector of the initial embedding dimension. This construction allows

the transformer to learn different subspace features more easily, as each head will focus on another aspect of the embeddings. Note, however, that the split between heads is only logical. The splitting is mathematically equivalent to partitioning our matrices into independent vector spaces by masking off all cross-dependencies of them. Thus, we end up with block matrices for Q, K and V, with equal block sizes = $\frac{\text{hidden dim}}{\text{number of heads}}$.

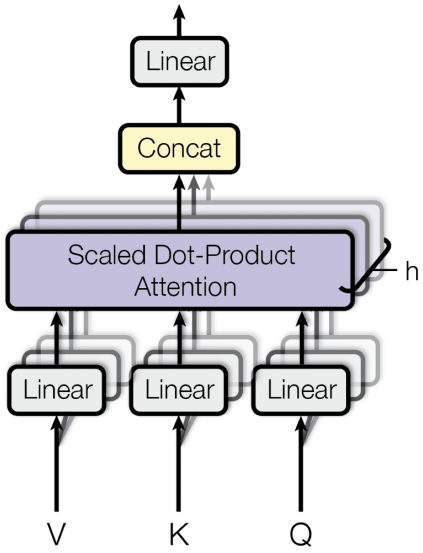


Figure 2.5: Multi-head attention.
Source: [55]

2.2.5.4 Cross-attention

In addition to self-attention, our model will employ cross-attention in the decoder. This serves the purpose of incorporating the input embeddings provided by the encoder into the decoding process. Following our former formalism, this can be described as using the yielded hidden states $\dot{\theta}_0 \dots \dot{\theta}_s$ to update the current decoder state θ_t using the attention mechanism. For this, the cross-attention block will be provided by queries of the output embeddings Q^{out} and the keys and values of the input embeddings K^{in} and V^{in}.

2.2.5.5 Assembling it all together

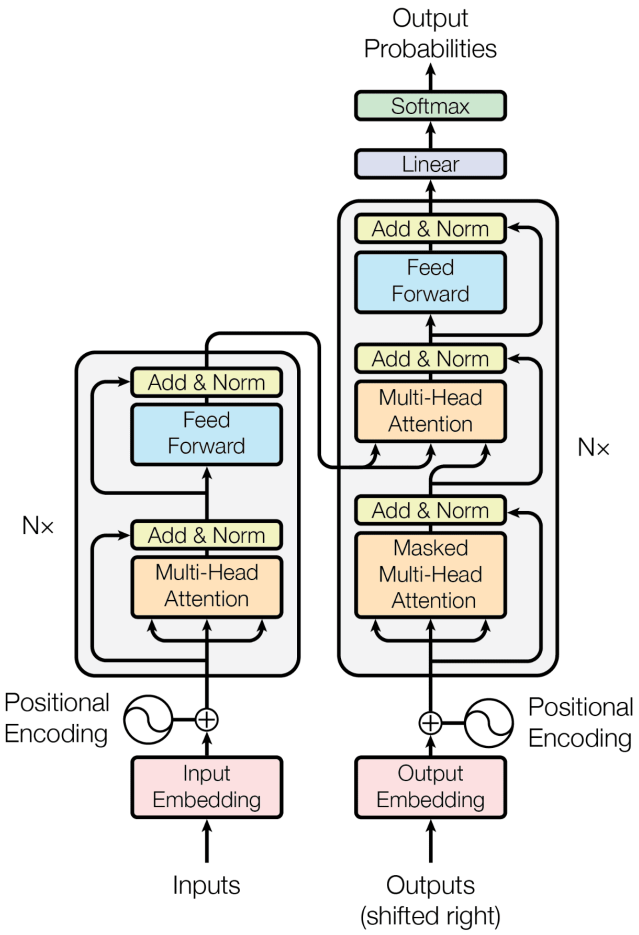


Figure 2.6: The Vanilla transformer introduced be Vaswani et al. (2017). The left and right blocks are part of the encoder and decoder, respectively.
Source: [55]

We are now equipped with the basic tools to assemble our transformer. The architecture can be seen in Figure 2.6. After embedding our words, we obtain our hidden vectors. These will then be processed through a number of encoder layers. Each of these layers consists of two sublayers; first a self-attention sublayer and then a feedforward sublayer. Each of these sub-layers will have a residual connection around it [25] followed by layer normalization [4]. The Encoder is build in a way that the model will iteratively per-lay first find dependencies in the embeddings and the have some "thinking time" in the feed forward sub-layer.

During *training* with **teacher forcing**, the decoder will also have a number of output embeddings at hand. These will similarly first pass an attention sublayer. We do not want our model to peek to embeddings it has not produced yet, so the attention matrices have to be masked in a way, so that embeddings can only attend to previous positions. The block is therefore called *masked* multi-head attention. After that we enter our cross-attention block,

again with a residual connection and layer normalization. Finally, we pass the embeddings through a feedforward sublayer to obtain our layer-wise result. For the prediction, the yielded embeddings need to be transformed back to a distribution in $\mathbb{R}^N$. For this, the dimension of our embeddings is first expanded to N, using a linear layer, after which we obtain our probabilities by *softmaxing* them.

During *inference* the decoder will not have the output embedding at hand. Instead the decoder is fed the output sequence in an auto-regressive way. For time step $i \leq n$ it will be supplied by $\hat{t}_{0:i}$ and predict $\hat{t}_{i+1}$. The window size is thus $k = n$.

3 Related work

3.1 Field research

3.1.1 Transformers success in NLP

Since introduction by Vaswani et al. [55], transformers have been out-performing comparable architectures in most NLP tasks: The original transformer scores top results in translation and constituency parsing with far less training cost. For example. Lakew et al. (2018) show how transformer-based models outperform recurrent networks reliably in neural machine translation [32].

Transformers employed as large language models (LLMs), like BERT [16], T5 [44] or GPT-3 [10], perform extraordinary results on established language benchmarks like GLUE or SuperGLUE, i.e. multitask general language understanding benchmarks [57, 58], so that as of October 2023 nine of the top ten scoring models on SuperGlue employ transformers with the only exception being the human base line. All of this success would be quite impossible without building up dedicated syntactical knowledge. Naturally one would ask, what features of natural language they seem to understand so well and where difficulties may lie. A number of researchers have dedicated their work to investigating the syntactical abilities of transformers employed as LLMs.

3.1.2 Field research regarding syntactical abilities

This part of research focuses on finding syntactical knowledge in LLMs trained on natural **field** data as opposed to constructed **laboratory** data.

Quantitatively, syntactical capacities are measured by testing the comperative performance of LLMs on various superficial syntactical tasks, like part of speech (POS) tagging or named entity recognition (NER), finding those models to perform extraordinary well.

For example Goldberg (2019) tests BERTs' ability on subject-verb agreement using zero-shot examples and finds that it outperforms regular LSTMs. Although this may be due to the larger training sets and bidirectionality, he argues that "real syntactical generalization is taking place" [22].

Our work is driven by the question of whether the abilities of transformers only suffice to complete traditional natural language processing tasks, or if they can create syntactical structures of the sophistication of those of humans. By a quantitative approach, this knowledge is difficult to attain. As previously argued, sequences may not be a good indicator

of what the output of the human language generating capacity might be, so it might also not be the best predictor of a language models capacity. Two identical sentences may be produced through arbitrary syntactical constructs, including nonsensical ones. A model that reaches high perplexity scores and reliably predicts masked-out words may therefore not necessarily have learned any complex syntactical structures. As we will see in our experiments (see subsection 6.3.2), the comparative results measured by Goldberg (2019) do not necessarily express whether a model understands the syntactical principles that dictate which subject agrees with which verb.

Performance of LLMs measured by black-box language modeling tasks is therefore in some way decoupled from syntax understanding. Instead, empirical research on the syntactical abilities of artificial models should be complemented by qualitative analysis that aims to reveal knowledge about the generated syntactical features.

Qualitatively, field research is often conducted by performing *probing tasks*. Hereby it is assumed that if syntactical knowledge of a certain degree is encoded into model weights, a classifier equipped with those weights should be able to perform better on other syntactical tasks. Varying probing tasks and comparing performance between different models will yield statements about syntactical abilities.

A broad study by Jawahar et al. (2019) hints, among others, that BERT encodes tree-like structures in the upper layers, while lower levels focus on surface-level phrasal syntax [21]. In a more elaborate way, Hewitt and Manning (2019) used probing tasks to explore syntactical information in LLM word representations. They find that through linear transformations, one can extract syntactical metrics like dependency tree distance or depth and obtain the original dependency tree. Examining the size of syntactical information encoded in vector space, they find it to be rather small, where probes of dimensionality of around 64 to 128 stop increasing performance. [26].

As this finding is quite similar to what we want to assess, we shall elaborate a little more on the limitations of it. For one, as noted by the authors, their construction is quite strict and is "designed not to test for some notion of syntactic knowledge broadly construed". Instead, their work tries to extract the kind of knowledge a LLM already contains. Dependency grammars, especially those without labeled nodes, as used here, generally construct fewer (or no) explicit grammatical nodes. Subphrases will therefore only be reconstructable by grouping child nodes. It would be interesting to know whether more sophisticated grammar formalisms with explicit abstract syntactical objects (e.g. tree nodes) could also be obtainable by this formalism. Contrary to this grammar, we will, as described below, construct phrase grammars with explicit, syntactical nodes. This will instead enable us to classify the computational power of our model using common theoretical concepts.

The underlying assumption of optimism in regard to this kind of research is that syntactical structure will naturally emerge out of the data. According to this, by successively combining trivial inductions, these may approach human level sophistication. If this pre-

vails, then surface level performance, e.g. perplexity and BLEU scores, should be predictors for the emerging of human level syntactical abilities. This is due to the fact that, following the assumption, data by its nature drives models to learn the right rules. A parallel can be drawn here to the same notion of human language acquisition that *empiricist* linguists uphold, which is highly challenged in the field of linguistics. On the contrary, *nativists* emphasize an innate mechanism or inductive bias that guides the language acquisition process. For this, the poverty of the stimulus arguments (POS arguments) state that all children learn in a short period of time a identical set of rules out of nearly infinitely many while only being exposed to little, erroneous and incomplete data (compare to e.g. [33], [24] and [35] for such findings). This knowledge could therefore not have been attained by general induction principles and the data alone. Humans must therefore have explicit or implicit *a priori* knowledge, hence the name nativisim. A big part of syntactical information about the structure of human languages will therfore not be found in that data. If so, the architectural biases of model will be of great importance. A perfect model would then have to entail the same inductive biases or a priory principles as humans do. In our work, we will thus particularly emphasize findings about inductive biases.

Explicit research thereof is rather hidden. One such example is a study conducted by Liu et al. (2019), where they test the generalizability of syntactical and semantical representations within LLMs using a variety of probing tasks. They find that for one, LSTMs seem to encode general syntactical knowledge in the lower levels, getting more task specific in the higher ones. For transformer models, this hierarchy cannot be upheld, with the most generalizable encodings in the middle layers. Furthermore, pretrained models seem to perform worse than models trained on related tasks, implying that pretraining seems to incentivize the model to learn pretraining specific task representations. They validated this with an empirical finding showing how layers that perform worse on probing tasks are the ones that perform better on conventional language modeling tasks, indicating a trade-off between generalizability and task specificity. Generalizable or fine-grained linguistic features seem accordingly only to be learned insofar they relate to pretraining objectives, resulting in rather course-grained syntactical knowledge [37]. On the contrary, models with strong syntactical biases should be rather driven to learn generalizable features, irrespective of, or rather complementary to, task-specific behavior.

This insight can be key in understanding similar findings like that of Tenney et al. (2019) and similar work, who find that BERT encodes surface-level syntactical knowledge, like POS tags, in the lower layers to obtain high-level semantic information, like co-reference, in higher layers. This is consistent with our construction in chapter 1 and the general linguistic parsing approach, thus titling their work "BERT rediscovers the classical NLP pipeline" [54]. Liu et al. argue that this behavior, rather than indicating that LLMs extract generalizable features in lower level to gradually build higher level representations, may be due to the nature of the pre-training tasks, making semantical knowledge rather useful in higher layers. The sophistication to which those syntactical features are built up may therefore not exceed what is task-relevant, most probably resulting in rather surface-level syntactical features that stem from task incentives rather than from inductive biases.

3.2 Laboratory research

3.2.1 Theory

Laboratory research focuses on the theoretical abilities of transformer networks and aims to find the limits of computational capacity when trained on specific syntactical tasks. For this, researchers usually examine the ability or disability of a given transformer architecture to learn recognizing classes of formal languages with certain computational power, e.g. levels of the Chomsky hierarchy. Chosen grammars will therefore serve the purpose of encapsulating attributes of computational importance, e.g. counting, periodicity, and hierarchy, rather than covering syntactical phenomena of natural language. Those are instead believed to be composed of the base features or emerge out of them. This can be achieved by using mathematical proofs (*theory*) or by examining their performance on certain language formalisms (*empirics*).

In the best case, transformers are just Turing-complete. There are several papers that claim to prove Turing completeness for transformer networks. For instances, Peréz et al. (2019, 2021), assuming infinite precision, as not uncommon, supply a finite depth transformer construction that may simulate a given Turing-machine computation [42] [41]. Hereby the token positions take on the role of the computation steps, which a Turing machine would go through. However, to make their construction work, the width of the network will be bounded by the number of steps of the computation. Similarly, Bhattamishra et al. (2020a) tried to prove Turing-completeness for the vanilla transformer. Their construction has similarities to previous constructions in so far as the width of the Transformer will once again be bounded by metrics of the specific computation instance; in this case, the memory usage [7]. Both of these constructions do not prove Turing completeness in the proper sense, as one could even construct finite state automata through those restrictions. We refer to [2] for a clearer presentation of this issue. In contrast, RNNs are Turing complete in the proper sense, following famous findings by Siegelmann and Sonntag (1992, 1994) [51][50] and Siegelmann (1999) [49] for Elman-RNNs (1990) [18]. Under more practical assumptions, as we will see, Weiss et al. (2018) find LSTMs and Elmain-RNNs to only be capable of modeling counter automata, with weaker simple RNNs only capable of recognizing regular languages [61].

Consistent with our assessment, Hahn (2020) proved theoretically that soft-attention-based sequence models, contrary to LSTMs, even with infinite precision are not able to achieve perfect cross-entropy scores for periodic regular languages or hierarchical structure, namely Dyck-2, which are two core aspects of natural language. For hard attention, this restriction is even stronger, stating impossibility and thus theoretically condemning such models to sub-regularity. He proved this by providing a procedure that, by carefully choosing input embeddings that force the network to focus on parts and this to ignore other parts of the sequence, constructs inputs that a given model mislabels by nature of the attention-based aggregation. This is possible for sufficiently sensitive languages, like parity. His findings are asymptotic, i.e. for fixed-size inputs one can, in fact, still construct a transformer that could correctly label all input data. As the aforementioned procedure

might require the construction of large inputs, he admits that further research on the practicality of this discovery should be carried out [23].

Taken together, these discoveries tell us that transformer capabilities are strongly asymptotic. That means, given a suitably bounded computation, transformers are in theory capable of computing it, although they break down for inputs whose computation exceeds said bounds.

Ackermann and Cybenko (2020) summarized research regarding networks' computational capacity in a more realistic manner by assuming linearly bounded computation time with regard to input size and finite precision. They denote this by *IBFP* (input-bounded finite-precision). They state that, asymptotically, while RNNs seem to correlate to regular languages

$$\mathcal{L}(IBFP - GRU) = \mathcal{L}(IBFP - SRNN) = \boldsymbol{REGULAR}$$

and LSTMs seem to be enclosed by the set of simplified k-counter languages (**SKCL**) and the general set of counter languages (**CL**) as shown by Weiss et al. (2018) [61]

$$\boldsymbol{SKCL} \subseteq \mathcal{L}(IBFP - LSTM) \subseteq \boldsymbol{CL}$$

transformers do not even contain all regular languages:

$$\boldsymbol{REGULAR} \not\subset \mathcal{L}(IBFP - TRANSFORMER)$$

Note, however, that they assume transformers without positional encoding, which mitigates its abilities to learn position-dependent tasks. They admit that this insight may not hold for positional encodings that do not repeat on sufficiently large inputs. Furthermore, they discuss how access to memory might overcome computational limits as contemporary networks are forced to drag syntactical information implicitly from layer to layer. They expand on this thought by arguing how memory-augmented neural networks (MANN) show potential in climbing the Chomsky hierarchy, as we will see later. [1]

3.2.2 Empirics

Theoretical research provides us with an upper bound for what transformers are in theory capable of and for what they, in practice, may be incapable of. While the theoretical upper bound (bounded "Turing-completeness") seemed promising, each sequence model trained by stochastic gradient descent (SGD) comes with inductive biases, i.e. preferences of directions to choose when iterating through parameter space. Although this might help in finding an acceptable solution, it might also inhibit finding the optimal one. Therefore, it is crucial to explore the de facto capabilities of transformers.

3.2.2.1 Regular languages

Bhattamishra et al. (2020) showed that transformers, even with a single layer, are theoretically able to learn some counter-languages that do not require reset operations, like

Shuffle-Dyck. They generalize their proof to a simplified version of **stateless counter-languages** *RCL*. This is a specific, fairly restricted set that does not include all regular languages and only some nonregular, somewhat permutations-invariant counter languages.

Aside from this, they also show that transformers seem to perform well only on a weak subset of regular languages, namely star-free languages with dot depth 1. This behavior is greatly influenced by the chosen positional encoding scheme, where implicit positional encoding via masking, as employed in above construction, may generalize well on Dyck-1, but fails to do so for some star-free languages, where instead it may profit from an explicit positional encoding. This oddly does even apply to permutation-invariant tasks, e.g. checking an input sequence for parity. For non-star-free languages, transformers fail even for simple ones like $(aa)^*$. [6]

Deletang et al. (2022) further complemented research by providing a broad empirical study of neural network architecture performance on transduction tasks ranging over the complexity classes regular (*R*), context-free (*CF*), and context-sensitive (*CS*), as well as covering attributes such as permutation-invariance and counting necessity. In addition to previous research, they also employ memory-augmented models. The authors find that LSTMs accurately and reliably learn regular tasks and even a counting-reliant context-sensitive task. Transformers seem to show nontrivial learning only on permutation-invariant tasks, but not even all of them, irrespective of positional encoding. For the residual ones, including three out of four regular tasks, transformers seem to not generalize well. The bottleneck again seems to be the positional encoding, since positional encodings for embeddings of unseen indices may fall out of distribution. They validated this by visualizing first-layer activations by sequence lengths, indicating that large sequences diverge in resemblance compared to those for smaller ones. On the other hand, as expected, RNNs seem to generally solve regular tasks, Stack-RNNs context-free tasks, and Tape-RNNs tasks up to the level of context sensitivity. Some irregularities occurred that can be attributed to subtleties of task design and training, but the Chomsky hierarchy proved more or less accurate for RNNs [15].

3.2.2.2 Dyck languages

We may briefly summarize that for regular languages, transformers seem to perform well only on permutation-invariant and star-free regular languages. Furthermore, they are capable of learning Shuffle-Dyck by iteratively counting occurrences of symbols from left to right, as long as such counting is somewhat permutation invariant. This finding can be transferred to $\mathcal{D}_1$ = Shuffle-1 Dyck. Here, one has to note that while accepting $\mathcal{D}_1$ by the given construction, the model does not per-se "match" brackets, but only counts number of open and closed brackets in a similar way as our definition of **correctly bracketed** expressions in subsubsection 2.1.3.4. For $k = 1$ *correctly bracketed* and *hierarchically bracketed* coincide. We thus still need to find out, whether transformers can also model hierarchies, i.e. $\mathcal{D}_{>1}$. Various researches find this not to be the case. For example, experiments conducted by Suzgun et al. (2019) show that RNNs enhanced with differentiable stacks (Stack-RNN) are well suited to learn $\mathcal{D}_n$: $n \in \{1, 2, 3, 6\}$, contrary to conventional

LSTMs and transformers. [53]

We find papers trying to mitigate this deficit. For example, when adding an arbitrary starting symbol T to the beginning of sequences, Ebrahami et al. (2020) found that the models' ability to learn $\mathcal{D}_2$ drastically increased, being on par with LSTMs. The starting symbol enabled the transformer to approximate the simulation of an automaton by attending to it at the end of a clause, i.e. a correctly bracketed sub-string, or at the end of the sequence. This resembles the popping operation of a stack-based automaton, which is also the operation that the stateless counter automaton mentioned above lacks. [17]. However, both models remain insufficient for learning $\mathcal{D}_2$ (compare with [52] for LSTMs). A more promising approach has been provided by Yao et al. (2021). Here, the performance for $\mathcal{D}_{N,k}$ languages was evaluated, where D denotes the maximum nesting depth of that language. In addition, they employ a "single fixed scalar monotonic positional encoding", namely pos/n. They theoretically prove that for soft attention:

"$\forall N, k \in \mathbb{N}+$, there exists a 2-layer soft-attention network that can generate $\mathcal{D}_{N,k}$"

This assumes an input size of *n O(log n)* precision and *O(log k)* memory size per layer. Through these assumptions and the nesting depth bound, they circumvent Hahns [23] theorem. In their construction, the first layer would calculate depths for each bracket and the second would match brackets according to their depth score. These insights could be experimentally validated, showing that transformers are well suited to learn $\mathcal{D}_{N,k}$ languages, even for $D \in [2, 128]$ and $k \in [3, 15]$. They constantly outperform LSTMs that have much more trouble for increasing nesting depths. Furthermore, their construction predicts why the second-layer attentions of two-layer transformers produce virtual hard attentions; a pattern often observed in larger language models [64].

However, in our branch of research, this finding is not as positive as one might think. To realize this, it should first be noted that the restriction of a bounded hierarchy condemns $\mathcal{D}_{N,k}$ to regularity. To sketch a proof, we would first define:

$$\begin{aligned} \text{extend}_j(A) &= (\mathbf{BL}_j \circ A* \circ \mathbf{BR}_j) \cup A \\ \text{extend}_{(x_1 \ldots x_N)}(A) &= \text{extend}_{x_N}(\ldots \text{extend}_{x_1}(A) \ldots) \end{aligned}$$

It follows:

$$\begin{aligned} \mathcal{D}_{N,0} &= \{ \epsilon \} \\ \mathcal{D}_{N,k+1} &= \Big(\bigcup_{(x_1 \ldots x_n) \in \mathcal{P}(N,N)} \text{extend}_{(x_1 \ldots x_N)}(\mathcal{D}_{N,k}) \Big) * \end{aligned}$$

$$\begin{aligned} \text{E.g. } \mathcal{D}_{2,1} &= (\text{extend}_{(1,2)}(\mathcal{D}_{N,0}) * \cup \text{extend}_{(2,1)}(\mathcal{D}_{N,0}) *) * \\ &= ((\mathbf{BL}_2 \circ (\mathbf{BL}_1 \circ \epsilon \circ \mathbf{BR}_1) * \circ \mathbf{BR}_2) * \cup (\mathbf{BL}_1 \circ (\mathbf{BL}_2 \circ \epsilon \circ \mathbf{BR}_2) * \circ \mathbf{BR}_1) *) * \end{aligned}$$

Here, $\mathcal{P}(a, b)$ denotes all the possible configurations for a out of b entries. In the case of $a = b$, there will be $a!$ possibilities. Our proof is possible because regular grammars are,

among others, completed in terms of $\cup$, $*$ and $\circ$. Furthermore, this procedure is finite, that is, for all N and k this procedure tells us how to define $\mathcal{D}_{N,k}$ in a finite way with the given operators. It takes advantage of the fact that any tree structure restricted by depth can be reduced to tree structures of a lower depth, which can be enumerated in the end without "vertical recursion".

Practically, this means, with respect to hierarchies, that the supposed recursivity, a believed core feature of natural language, will be approximated by limited enumeration, given sufficient capacities. Furthermore, the produced underlying syntactical objects (tree-like for CFGs, sequence-like for regular grammars) are qualitatively different: Given a string s of nesting depth $\leq k$ two parsers P_{CFG}, P_R equipped with $\mathcal{D}_N$ and $\mathcal{D}_{N,k}$, respectively, while both accepting s, produce rather different syntactical structures. In fact, it cannot be said of P_R to model tree structures, and therefore recursive hierarchies, at all, just as a parser memorizing 80.000 examples of $\mathcal{D}_{N,k}$ produces no syntactical objects of interest whatsoever. Consequently, when an LLM produces a sequence like s, because it was theoretically shown that it cannot learn $\mathcal{D}_N$, but it can very well do so for $\mathcal{D}_{N,k}$, the underlying generation and its abstract syntactical representation may also not be (recursively) hierarchical. This is consistent with the findings of Ebrahimi et al. (2020) and Yao et al. (2021).

Although initially not sounding like one, the given restriction is thus both theoretically and practically a hard restriction of our grammar, making the *Chomsky-Schützenberger theorem* not applicable. Therefore, the finding that transformers can learn $\mathcal{D}_{N,k}$ may be a rather negative than positive finding with respect to their ability to model sup-regular grammars.

3.2.3 Transformers and the Chomsky Hierarchy

All these results proved it hard, compared to RNNs or LSTMS, to place transformers in a designated drawer of the Chomsky hierarchy, as their computational system does not fit the automata-based theories thereof. It could be argued that this is due to the parallelism in which a transformer computes the tokens compared to the sequential folding operations RNNs and classical automata conduct. In contrast, Liu et al. (2023) show that transformers of *o(T)* layer size, T being input size, are not only able to asymptotically simulate automata, but reliably find "shortcuts" for such computations, vastly reducing the amount of "computation steps", i.e. layers, to $L \ll T$. [36]. We will therefore assume that transformers are theoretically capable of simulating automata equivalent to levels of the Chomsky hierarchy. Through our prior analysis we remain inconclusive about its de-facto capacity. Depending on architectural specifics, we find that transformers cover some areas of regular languages and some specific kinds of counter languages, but they seem to be very likely *sub-context-free* and quite possibly even *sub-regular*.

Another approach would be to accept that transformers do not conform to the automata-based models. To establish a new computational model for transformer models, Weiss et al. (2021) constructed a programming language, **restricted access sequence processing** (RASP), for transformer networks. RASP consists of element-wise operations and selection

and aggregation operations, which resemble the ones done by a transformers' feedforward and attention components, respectively. They show how solving a given task using RASP helps predict model parameters and may be a good indicator for how the model approaches the task. By doing so, they provide an upper bound for what kind of computations are feasible and an intuition for the complexity of certain computations from a transformer's point of view. Certain tasks easily learnable for automata-like models, like LSTMs, might therefore be highly nontrivial for transformers and vice versa [62]. However, this model seems to be too specific to compare its computational capability with conventional models, and not much research has been done so far. Furthermore, formal language theory is built around conventional automata-based models. We will thus remain in the conventional frame work until *RASP* has been somewhat integrated into it.

3.3 The theory-practice gap

In summary, vanilla transformers employing common position encoding schemes seem incapable of learning regular non-star-free languages and are strongly restricted to learning hierarchical context-free languages like $\mathcal{D}_{>1}$. As natural language is expected to be situated above the context-free language class, quite possibly within the mildly context-sensitive class, this seems surprising [29]. After all, following the Chomsky-Schützenberger theorem [11], each Context-Free language may be represented by a homomorphism of the intersection of a $\mathcal{D}_n$ and a regular language. If a transformer is incable of modeling $\mathcal{D}_2$, then it would follow that neither can it model context-free, let alone mildly context-sensitive. How can we then explain its overwhelming empirical success regarding natural language data? We refer to this problem as **the theory-practice gap**.

One way to resolve this might be to argue, as some researchers have done, that contemporary linguistic models for human language might not be restrictive enough. Jäger and Rogers (2012) argue that the overwhelming majority of regular utterances are star-free, where a vanilla transformer already suffices. Similarly, Karlsson (2007) tried to show empirically that the maximum centre embedding depth of written sentences is around three [31]. This is why Yao et al. (2022) imply that their construction for restrictively nested languages may already be employed by large transformers trained on natural language, as natural data already meets the restriction of nesting depth [64]. If human language indeed is bounded, just as the grammars that transformers seem to learn, then the syntactical representations of such models might indeed be not far away from human representations, which allows one to ascribe *real understanding* to such models.

However, one has to note that natural sentences following a generally assumed generative grammar are hierarchical in construction per se, irrespective of occurrences of embedded clauses. Assuming boundedness of hierarchy would furthermore render natural language regular, contradicting most of linguistic research and assumptions. More importantly, one has to be generally cautious when inferring statements about the human language *capability* from empirical findings in externalized sequences, as discussed in chapter 1. Statistical findings of that kind may instead fall into the domain of pragmatics

and not necessarily syntax. To form arguments about the capability, one had to argue rather biolinguistically. For example, in the case of boundedness of hierarchy, one would have to find neurocomputational reasons to disprove the general assumption of (hierarchical) recursion, an assumed core aspect of human cognitive computation (compare to [14], [19] and especially [9]). We refer to Watumull et al. (2014) for a clearer disentanglement of the relevance of such empirical findings in regard to statements about the human capability for recursion [60].

4 Assessing the syntactical abilities of transformer networks

4.1 Hypothesis

We have seen that transformers perform extraordinarily well on NLP tasks, but posses quite limited syntactical capabilities. We are thus left explaining how an assumed weak model, i.e. transformers, can process a complex computation, i.e., natural language. We have found various reasonings that tried to either lower assumption regarding the complexity of the computation or increase assumptions about the capability of the model. In chapter 3 we tried to sketch how attempts of lowering assumptions about the complexity of natural language are highly controversial. Furthermore, theoretical and empirical research seems to imply that transformers are rather unsuitable for learning sup-regular, let alone natural language.

We instead hypothesize that large transformer networks, although possibly suited well enough to model a quantitatively large portion of externalized human language, do have rather weak computational power that restricts them from finding general generative principles of languages.
They may instead employ their limited capacities, like modeling star-free languages and restricted hierarchies, to find approximations over distributions of syntax. We call this behaviour "finding *statistical heuristics*".

The most straightforward *statistical heuristic* on could think of involves leveraging data distributions to create a Markov chain based on raw input tokens. This approach might be already effective enough for predicting a substantial number of tokens that rely on preceding ones – such as predicting that the word "am" is likely to follow the word "I". A more refined version of this heuristic entails initially predicting part-of-speech tags using simple Markov chains and then constructing a higher-level Markov chain over these features. Subsequently, these features can be utilized to model various types of sentence sub-phrases and higher-level features.

Example: Solving subject-verb agreements with statistical heuristics

Consider how models might learn subject-verb agreement when explicitly trained on this task. Here, for a given subject or verb, the model needs to identify the predictor word, which is entangled via a grammatical notion, e.g. grammatical number. The human approach would be to simply parse the sentence into the syntactical representation it was generated with, as we do implicitly all the time, and find the respective, syntactically near word. For a model where this process is unavailable or too complex, a simpler approach would be to predict POS-tags of all words and find distributions. The simplest one, picking the nearest verb or noun in the sequence, would already suffice for a vast majority of sentences. On top of this, the model might build more sophisticated features that identifies edge cases, like embedded sentences. Later, our second task **MASK** will confirm this intuition.

The process of attaining higher-level features is only possible as far as such features are attainable by the models' capabilities. But even when so, the model might not employ such: identifying suitable, complex features may demand more computational resources than forming heuristics based on statistical occurrences of less complex features. With stochastic gradient descent, models might therefore be unmotivated to learn more intricate structures, particularly when an increase in strategy complexity fails to substantially reduce loss or even increase it. This effect might be magnified by the given data distribution, if the biggest part of it can already be approximated with lower level features.

In a sense, the term *heuristic* already implies that any strategy that does not involve the real generative principle is some kind of *statistical heuristic*. However, we would rather like to emphasize the differing complexity levels at which such heuristics can be formed. The higher level the learned features are, the more we can attribute the strategy generalizablity and thus *real understanding* as far as that term would be applicable.

We posit that the extent to which more complex structures are learned is a result of the interplay between the models' theoretically limited capabilities and their inductive bias. The complexity of these features is, therefore, bounded by the models' capacity, and their utilization depends on the inductive bias and the specific incentives of the given task. For our first task, **BRACKET**, we thus especially looked out for parameters of models and task designs that incentivize the model to learn syntax.

In the following we want to illustrate the design of our experiment and motivate the reasoning behind choices. For a more detailed description of these aspects we refer to chapter 5.

4.2 General approach

Our research follows a primarily theoretical approach, drawing parallels to the theoretical *laboratory research* discussed in section 3.2. In essence, our general methodology aligns with this approach: we prepared models for evaluation, including baseline models, and selected appropriate grammars to generate large datasets. With these datasets and models in place, we designed syntactical tasks to evaluate the models' performance on the selected grammars. Further evaluation, employing various metrics, allowed us to analyze how well the models learned specific aspects of the tasks, such as syntactical structures.

The study encompasses a comprehensive comparative analysis, considering various models, grammars, and settings. The chosen tasks enable us to make informed statements about our research focus—the models' ability to learn structure. The grammars employed (denoted as **1D**, **2D**, **SF**, **NSF**, and **LARGE**) are crafted not only to cover computational classes but also to address a range of syntactical phenomena. Our model's design, including the positional encoding strategy, and evaluation settings, such as sequence lengths, draw inspiration from "real-world" examples to ensure comparability with fieldwork.

Our work is **exploratory** in essence and thus subject to limited expressivity in areas where more detailed analyses would go beyond the scope. As such, it spans a broad field of research areas for which more fine-grained future work would be possible. Instead, this work attempts to pave the way for such research both by testing our assumptions through wide-ranging comparisons and by presenting methodological and analytical tools in the process. The limitations of our methods and our general approach will be discussed both in our analysis of our results in chapter 6 and in our general discussion in chapter 7.

The following sections provide a rough overview of aspects and reasoning behind our experiments. Each aspect will be illuminated in greater detail in chapter 5.

4.3 Grammars

4.3.1 General idea

As natural language does not have a holistic formalization of syntactical structures of sentences, we must restrict ourselves to formal grammars whose syntactical representations are well studied. For this, four rule-based grammars have been designed for the purpose of fitting both formal computational classes, i.e. regular and context-free grammars, and covering natural syntactical phenomena, i.e. (linguistic) agreement and recursion.

All grammars will produce natural-like sentences using words in the English or German vocabulary and permissible sentence structures with the exception of the large grammar, which by design may also produce odd sentences. By doing so, LLMs pre-trained on large sets of natural language may be more compatible with the training set. To ensure this, we will use the repectively versions of our pretrained models that were trained on English or

German data, respectively. Furthermore, by doing so we mitigate the risk of using idealized forms of syntax that may influence performance on field data. One of such idealizations is the tokenization strategy, that might split words into different tokens in real world setups. We will explain this aspect in further detail in subsubsection 4.4.1.2.

4.3.2 Dyck languages

1D and **2D** are two context-free grammars that test grammatical agreement. They are constructed so that they resemble Dyck-1 and Dyck-2, respectively. In fact, they can be obtained by morphisms from these two. Instead of matching parentheses, we employ logically matched words: subjects and verbs that share a common grammatical number (singular, plural).

Example: Constructed context-free grammars

- $L(\mathbf{1D}) \ni$ *die Mutter , deren Tochter schwimmt , singt* $\Leftrightarrow$ **(())**
- $L(\mathbf{2D}) \ni$ *die Katja , deren Hündin tanzt und deren Teller fliegen , spielt* $\leftrightarrow$ **(()[])**

Additionally to the previously mentioned advantages of natural-like sentences, this construction enables us to avoid another idealization of laboratory work. To explain this, it is necessary to shortly recall the general theoretical motivation of evaluating on $\mathcal{D}_{N>1}$. As discussed in chapter 2, the Chomsky-Schützenberger theorem lets us divide every context-free language L into a regular language R and a Dyck-N language $\mathcal{D}_N$, such that we can find a homomorphism h for which to which applies:

$$L = h(R \cap \mathcal{D}_N)$$

We also showed how $\mathcal{D}_2$ is sufficient to mock $\mathcal{D}_N$. Thus, to conclude whether a given model can learn context-free languages, it seems sufficient to investigate whether it can learn $\mathcal{D}_2$, as reasonably undertaken by related laboratory work. A concrete instance of a context-free language L_0, however, will be vastly reliant on R_0. Intuitively, in the above construction the Dyck language will build the hierarchical skeleton of L, but it will be the task of R to choose which parts of this skeleton concretely define L, e.g., which branches of all the possible $\mathcal{D}_N$ trees will be cut off. $R \cap \mathcal{D}_N$ will thus symbolize a concrete distribution of tree structures over $\mathcal{D}_N$. The more restricted and nuanced this distribution is, the easier will it be for a model to approximate distributions of such trees. If evaluating models on $\mathcal{D}_N \cap \Sigma* = \mathcal{D}_N$ only, the results undermine the fact that real world performance may vastly rely on modeling R . This is especially the case, when assuming, as we do, that transformers are only capable of approximating $\mathcal{D}_N$, but are very good in approximating distributions over low-level syntactical features. Employing “normal” context-free languages will therefore enable the model to exploit distributions more easily, which can then be contrasted with how much structure was instead learned.

4.3.3 Star-Free Regular Languages

SF and **NSF** are two regular grammars that test large-scale recursion. The languages are equivalent to $a * b$ and $(a * b)*$. Both of these are star-free, i.e. the kind of regular type that is commonly encountered in natural language. **NSF** is a bit more complex than **SF**. We employ these grammars mainly as baselines for comparisons.

Example: Constructed regular grammars

- *Sunny denkt, dass Toni denkt, dass Sunny sagt, dass alles ok ist.* $\in L(\textbf{SF})$
- *Sunny sagt, dass es morgen schneit und Toni denkt, dass Sunny sagt, dass alles ok* $\in L(\textbf{NSF})$

4.3.4 Large language

Furthermore, a fifth large language **LARGE** has been used with a large number of production rules and variables that serve the purpose of quantitatively approximating natural language. For constructing training sets, it proved nearly impossible to construct a meaningful distribution of our grammar that does not break computational boundaries, i.e. memory. The **LARGE** set will therefore not include all terminals, let alone all nonterminals. Evaluation on this set shows how important this fact seemed to be to task performance.

Example: Larger grammar

- *is there a flight from memphis to los angeles.*
- *please show me the flights from chicago to detroit that arrive at six p.m. next tuesday.*
- *i need a flight from philadelphia to westchester county.*

4.3.5 Further remarks

We expect general performance on those grammars to follow computational theory, where regular languages will be less complex than context-free languages. From our constructed grammars, **2D** is the most complex. A more detailed definition of our grammars and examples thereof will be provided in section 5.1.

To generate the training data, it is hard to follow implicit approaches, as is commonly done for regular grammars in theoretical research. Instead, we follow a rule-based approach and provide a generator written in Java for this purpose. Employing rule-based grammars, instead of implicit ones, yields the benefit of being more flexible in terms of sentence structure and available grammar formalisms, but comes with difficulties in design and generation. The design and difficulties of such generations are discussed in section 5.3.

4.4 Tasks

4.4.1 BRACKET

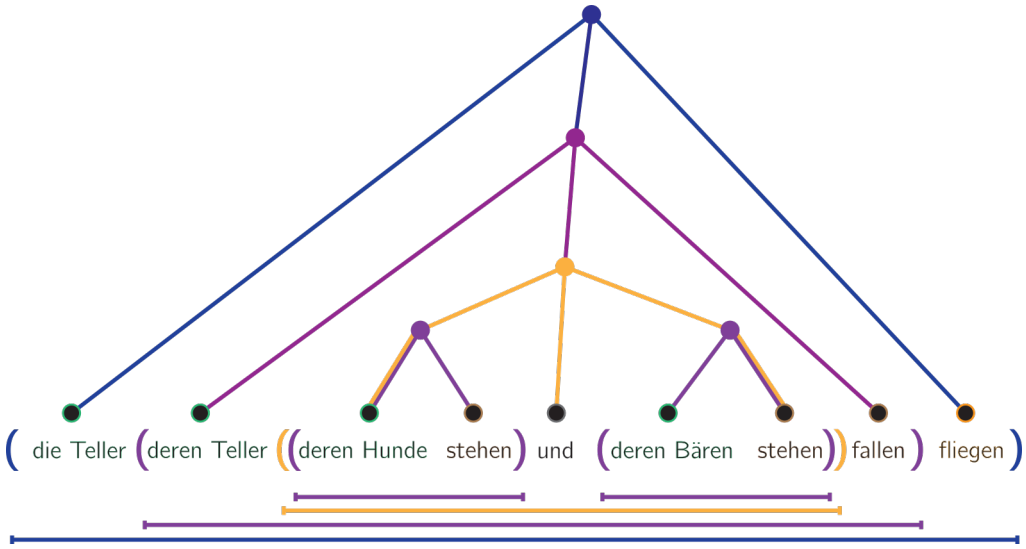


Figure 4.1: Depiction of **BRACKET**. Tree node colors symbolize different variables.

4.4.1.1 General design

Common pretraining tasks (e.g., word shuffling, masking, etc.) do not explicitly give an incentive to learn syntactical structures. In fact, it is hard to think of such a task design when dealing with natural language, as we are not sure how such structures would look like. For most NLP tasks, those representations don't matter much as long as the sequentialized output of those models can properly be interpreted by humans. This design makes it hard to judge what kind of structure the model has learned and also hard to measure its ability to do so in relation to a variety of factors. Our task will therefore need to explicitly provide syntactical structure.

We choose **BRACKET** for this, which is the task of producing a parse tree for a given sequence of an unknown grammar G. This is done by sequentializing the parse tree in a way in which every pair of brackets represents one subtree. The respective sequence elements, i.e. words, will be injected in between the respective brackets. **BRACKET** mocks the parsing process that a model should otherwise implicitly conduct if it is equipped with such capabilities. As such, it is equivalent to being able to accept $\mathcal{D}_N$, where N is equivalent to the number of nonterminals of the respective grammar. By explicitly forcing our model to perform this task, we will be able to evaluate which parts of this task it succeeds on and which parts it fails on. For a more detailed description and a more elaborate reasoning of this task and according metrics we refer to subsection 5.2.1.

Furthermore, from inaccuracies regarding the brackets, which resemble the parse tree nodes, conclusions about the syntactical abilities can be drawn. We will compare the

general accuracy to predict the right tokens with the accuracy restricted to bracket tokens (*bracket accuracy*) to see how much of the task performance can be attributed to learning syntax. This effect will be compared with various variables mentioned in the following. As there is no explicit prioritization of aspects of the predictions, by doing so, we may judge which task variables incentivize models to learn real syntax as opposed to statistical heuristics. Further analysis hereof and results can be found in subsection 6.2.4.

4.4.1.2 Further task variables

We are interested in how theoretical limitations play out in non-idealized real-world scenarios that might bare subtleties that theoretical work does not capture. One of these subtleties is the tokenization strategy, which may yield more or less tokens per word. LLMs vastly rely on byte-pair encoding (**bpe**), wheras theoretical research is often conducted using word-wise tokenization, because words of formal grammars are already the smallest information-baring units. We hypothesize that, at least for our task, but quite possibly also in field work, this may have an influence on how much structure is learned. Our idea behind this is that having more tokens that have no syntactical relevance may lead to deincentivizing the model to learn structure, due to uniformity of weighting during loss computation. The results of this analysis can be found in subsection 6.2.6.

We further hypothesize that certain predictor tokens may make it easier for the model to learn where a subphrase starts or ends. For our context-free grammars, one of such candidates would be commas, which in German are always placed at the start and end of an embedded sentence. We evaluated the grammar $\mathbf{2D}_{\mathrm{CL}}$, which is the same as **2D** but without commas, and compared the performance with **2D**. The results can be viewed in subsection 6.2.8.

4.4.2 MASK

If our model employs statistical heuristics, they must arise purely from the input data, whereas syntactical structures, even of the limited complexity of such models, may also arise from architectural biases. With **MASK**, we try to sketch how input distributions may influence its ability to form statistical heuristics. The task consists of predicting the class, i.e. nonterminal, of a masked word within a sequence. A common property of natural language (and, in fact, formal language) is that word dependencies are a reflection of their syntactical distance rather than their sequentialized distance, which is called "linear" distance. The masked word greatly depends on syntactically near words, which are not necessarily linearly near. However, these two metrics coincide greatly over large sets of generated sentences when sampling sentences in an equally distributed way. The input distribution of our training data may therefore incline the model to learn local statistical heuristics instead of "real structure", deliberately tolerating errors on inputs, where those two distances do not coincide. We thus evaluate our model on sequences where this correlation breaks to confirm our hypothesis. To measure this, we first define a metric, **span width**, which captures the notion of divergence of linear distance and syntactical distance. Applied to a masked word, it tells us the linear subphrase length over words that are syntactically near. A clear definition of the metric can be found in

subsection 5.2.2. We conducted our analysis using both our regular evaluation set that shares the same residual distribution with the training set and a newly designed set of a different distribution to see whether other distribution-dependent factors may have been exploited. We further evaluate how stable assumed stochastic heuristics are by analyzing how distributions of words of little task relevance have an influence on breaking them. For a more detailed description of both task and the span width metric we refer to subsection 5.2.2. Furthermore, the results and a more fine-grained analysis can be found in subsection 6.3.2.

4.5 Models

We will compare performance of three model architectures: a bidirectional lstm (**LSTM**), serving as a baseline, a Vanilla transformer ($\textbf{SIMPLE}_{\text{NLayers}}$) with varying layer sizes $N \in \{1, 3, 6\}$, to test the influence of layer sizes, and a pre-trained model ($\textbf{BART}_{\text{BASE}}$), to evaluate how structures learned during pretraining may transfer to our task.

The results of influence of layer size can be found in subsection 6.2.5. The role of pre-training will be elaborated on in subsection 6.2.7.

We equipped the LSTM with bidirectionality to stay comparable to the bidirectional nature of transformers. Furthermore, as LSTMs often suffer from a bottleneck effect when aggregating hidden states between the encoder and decoder, we employed Bahdanau attention [5] as the aggregator function, allowing the decoder to have more flexible access encodings provided by the encoder.

The transformer models were standard implementations provided by *hugging face* [63]. We will explain model architectures and reasoning of choices in greater detail in *chapter* 5.

4.6 Evaluation lengths

Theoretical research can be said to assume automata-like sequence processing and thus tests the generalization ability of the model by evaluating on "out-of-domain" data, i.e. sequences that exceed lengths during training. Sequence lengths has been found to be a known modulator for mistakes. The greater of length than seen during training sequence get, the more likely it is for the model to misclassify. These "out-of-domain" sequences are employed to verify the models generalization capacity. This assumes that the model learns folding operations that successively divide and conquer large sequences, i.e. nontrivial recursive solutions. As experiments show (e.g. [15], [6], [59]) models do not generalize well on longer sequences. Here it is argued that the positional encoding scheme plays a crucial role. Token positions that are not seen during training may lead to embeddings that the model does not recognize at all, as already argued above. While certain positional encoding strategies seem to restrict this deficit for certain tasks, we are interested in bridging the theory-practice gap and will therefore employ standard (sinusoidal or learned) positional encoding as employed by BART/BERT or the Vanilla transformer. Moreover, LLMs do not necessarily need to generalize syntactical knowledge to out-of-domain sizes if trained on sufficiently sized inputs. After all, practical lengths of sentences as found in large

text corpora will rarely exceed common LLM input sizes. For comparison: assuming a common rule of thumb of 0.75 words per token [40], BERT, employing up to 512 token positions, could be trained on sequences of up to 384 words. In fact, BART is used to receiving two sentences as input for some pre-training tasks. Furthermore, assuming the longest distribution of sentence lengths stated by Sichel (1974) ($\mu = 26.77 \sigma = 18.36$) [48], who performed empirical analysis on large corpora of English and Greek prose texts, the probability of sentences exceeding 100 words is already much lower than 0. 1 %. Although the ability to generalize to larger inputs satisfies the conditions of theoretical automata-based models, it might not play a major role in LLM performance.

We thus evaluate our model on partly on in- and out-of-domain data, attaching not much importance to generalizing well on sequences that are out-of-domain. You can find an evaluation of the effect of sequence length in subsection 6.2.1.

5 Experimental Setup

5.1 Grammars

5.1.1 Star Free Regular

Rules: SF

- $S \longrightarrow P$ *sagt, dass* S.
- $S \longrightarrow A$

Here P can be derived to a set of persons or personal pronouns (e.g. *Sarah, Timo, he, she, someone, etc.*) and A can be derived to any statement. In general, words may also be replaced by synonyms to obtain more training data.

Example: Sentences for SF

- Sunny sagt, dass Toni denkt, dass alles ok ist.
- Toni sagt, dass Sunny denkt, dass Tone denkt, dass alles ok ist.

By a homomorphism, $a * b$ can be obtained from our grammar, from which it is easy to prove that it is star-free and regular. This follows from the concatenation of b and the known star-free language $a*$, or:

$$a * b = \underbrace{\overline{\overline{\emptyset} \circ (\Sigma \setminus a) \circ \overline{\emptyset}}}_{a*} \circ b$$

The dot depth of this grammar is thus exactly 2.

5.1.2 N Star-Free Regular

Rules: NSF

- $T \longrightarrow S$
- $T \longrightarrow S$ *und* T
- ... Rules from **SF**

This grammar is an addition to the upper one. It retains all the upper rules, but defines a new start symbol T through which it can repeat multiple expressions of **SF** successively.

Example: Sentences for NSF

- Leo sagt, dass Thommy meint, dass es morgen schneit und Marie will, dass alles ok ist.
- Leo sagt, dass Marie will, dass es morgen schneit und Leo will, dass alles ok ist und Toni will, dass es morgen schneit.

This language, by the above construction, can be said to be equivalent to the language $(a * b)*$, which is also star-free, but more complex as it contains two nested points of recursion. The proof is roughly:

$$(a * b)* = b \cup \{a, b\} * \circ b$$

For which:

$$\{a, b\}* = \overline{\overline{\emptyset} \circ (\Sigma \setminus (a \cup b)) \circ \overline{\emptyset}}$$

5.1.3 Dyck-N experiments

As mentioned before, Dyck-N denotes the class of languages of correctly nested bracket expressions consisting of N different bracket types, where crosswise parentheses must not occur (e.g. *"([)]"*)

For our Dyck-N experiments, subject-verb congruence is used in German, specifically in numerus, that is, in number. In the sentence "Die Bärin schwimmt." (the bear swims), "Die Bärin" (the bear) is congruent with "schwimmt" (swims), while the word "schwimmen" (swim) would instead be congruent with a subject in the plural. Moreover, since there is a verbal phrase for each nominal phrase, the singular and the plural correspond to different types of parentheses. To achieve nesting, possessive subordinate clauses are used, which have the property that a new subject-verb pair can be injected into the same sentence between the nominal and verbal phrases.

According to this reasoning, the sentence "*Die Fische, deren Wasser stinkt, schwimmen.*" is of the form "[()]" and the sentence "*Der Fisch, dessen Flossen, deren Farbe glänzt, schimmern, schwimmt.*" is of the form "([()])".

In addition to nesting, parallel bracket expressions must be allowed. This is accomplished with the linking word "und"(and), which makes the expression "*Der Fisch, dessen Wasser stinkt und dessen Flossen, dessen Farbe scheint, schimmern, schwimmt und ein Vogel, dessen Flügel funktionieren, fliegt.*" (*The fish, whose water stinks and whose fins, whose color shines, shimmers, swims and a bird, whose wings work, flies.*) equivalent to "[()[()]([])".

5.1.4 1D

Rules: 1D

- $S \rightarrow S\ und\ S.$
- $S \rightarrow DN_s\ V_s$
- $S \rightarrow DN_s, P, V_s$
- $T \rightarrow N_s\ V_s$
- $T \rightarrow N_s, P, V_s$
- $P \rightarrow deren\ T$
- $P \rightarrow P\ und\ P$

For 1-Dyck, the singular form is always used (see assumed "s" on derivation rules). The top main clause (S) requires a determiner in the case of a singular noun; this is not used in the possessive subordinate clauses T. Therefore, there is some duplication in the derivational rules of S and T, since for the main clauses DN_s ("determiner-noun") must be used instead of N_s. Furthermore, there is duplication in the concatenations of main clauses and subordinate clauses, since in possessive clauses the possessive pronoun must precede each noun. Consequently, there are two rules for "parallelise" derivations; one for "outside" and one for "inside".

Example: Sentences for 1D

- die Tochter, deren Mutter tanzt, lacht und eine Mutter, deren Tochter lacht, tanzt. ↔ (())(())
- diese Bärin, deren Tochter, deren Katja, deren Mutter spielt, lacht, geht, lebt. ↔ ((((()))))

5.1.5 2D

The language 2-Dyck results from what has been discussed so far by adding corresponding plural rules. To do this, for each rule containing an imputed **s** in **1D**, an identical one is added, replacing all occurrences with **pl** instead. Furthermore, only nouns of the feminine are used for simplification, since for these, the possessive pronoun "*deren*" in the singular agrees with the plural. For the sake of clarity, the rules are not listed here.

Example: Sentences for 2D

- die Mutter, deren Tochter schwimmt, singt und die Bären, deren Teller fallen, liegen.. ↔ (())[[]]
- die Flaschen, deren Mutter lebt, fliegen und die Katja, deren Hunde stehen, singt. ↔ [()]([])

5.1.6 LARGE

Furthermore, a large grammar has been employed. For this *atis.cfg* was used from the Python Natural Language Toolkit (NLTK) [8], which contains over 4.5k production rules. Our data-set produced sentences with 448 different Non-Terminals and 611 words. As we've discussed, it is no easy task producing a data set that captures a relevant distribution of such large grammars. Furthermore, as grammars like this are rather used to parse existing sentences, than generating new ones, this grammar by design produces odd sentences that humans cannot make sense of.

Example: Sentences for LARGE

- i need a flight from charlotte to las vegas that makes a stop in saint louis .
- show me flights from chicago to kansas city leaving around seven p.m. thursday .
- can you tell me about the flights from saint petersburg to toronto again .
- you both had this sixth less and they select less francisco they .

5.2 Tasks

5.2.1 BRACKET

5.2.1.1 Formalism

Given a tree $T = (V, E \subset V \times V)$, a set of c colors $C = \{ i \mid 1 \leq i \leq c \}$ and a coloring function color : $V \to C$, we may sequentialize its structure into a bracket expression sequence $S = s_1 \circ \ldots \circ s_n$ of a sequence alphabet A, $\forall_{1 \leq j \leq n} : s_i \in A$. For this we will first define a bracket pair for each color $\forall_{i \in C}$: $\mathbf{BL}_i \in A$ and $\mathbf{BR}_i \in A$ and use the helper function children : $V \to [V]$. Furthermore, we will provide a leaf sequentialization function serializeLeaf : $V \to$ *sequence of A*. We will use the following recursive depth-first algorithm:

Algorithm 1 SequentialiseTreeNode(Node, color, serializeLeaf)

```
N ← Node
i ← color(N)
if children(N) = ∅ then
    return serializeLeaf(N)
end if
Seq ← BL_i
for Node child in children(N) do
    Seq ← Seq ∘ SequentaliseTreeNode(child, color, serializeLeaf)
end for
Seq ← Seq ∘ BR_i
return Seq
```

If we pass our root to our algorithm, we get our desired sequence.

Given a formal grammar $\mathbf{G} = (S, NT, T, P)$, a sentence $s = s_1 \ldots s_n \in \mathcal{L}(\mathbf{G})$ and a parse tree $T = (V, E = V \times V)$ for s, we may use the above construction to obtain a sequentialized parse tree. The node color would hereby be their respective terminal or nonterminal. As non-terminals can yield different, equivalent words (e.g. $N_{singluar}$ may yield *Herbert*, *the table* or *she*) and we do nott wish to add a new color for each one of these, we construct a dictionary oracle $\mathcal{D}$: $NT \to$ *sequence of A* that returns a random element out of a predefined set of sequences. It follows serializeLeaf = $\mathcal{D}$.

The task will be, although without initial knowledge of our grammar, to return the output of *SequentialiseTreeNode* for a given sentence s.

5.2.1.2 Task metrics: Accuracy scores

To recapitulate on transduction tasks, as described in section 2.2: for a given pair $(S, T) \in$ **BRACKET** our model will predict $\hat{T}$, so that T and $\hat{T}$ are as near as possible according to our loss function. S and T will hereby be sequences of our tokens. Using the tokenization strategies described in subsection 5.5.2 we end up with two types of tokens. Tokens that are part of the sequentialized parse-tree (*bracket tokens*) and tokens that make up the words of

the input sequence (*non-bracket tokens*). We will use per-token accuracy on bracket-tokens to measure how much structure our model has learned as opposed to general per-token accuracy. Unlike similar theoretical research, that is interested in whether a model can fully produce a given formal language class, our aim is to measure how much of structure our model learns during regular performance. However, it's hard to compare a fully generated, quite-possibly uninterpretable tree with our ground-truth parse-tree. We will thus rather measure which nodes of the tree were predicted faulty, assuming everything else to be correct. This means, providing the ground-truth left context, we will measure the predicted probability distribution $\hat{T}_j = \mathcal{D}_\theta(\Sigma_t | t_0 \dots t_{j-1})$ and greedily retrieve the predicted token $\hat{t}_j = \text{argmax}(\hat{T}_j)$. We can then average the number of correctly predicted tokens in a sequence or in a batch of sequences to obtain the **general per-token accuracy** $\text{Acc}^{\mathcal{M}}(G)$ for our model $\mathcal{M}$ and our grammar G. If we instead average over the bracket tokens only we end up with the **bracket accuracy** $\text{Acc}^{\mathcal{M}}_{BR}(G)$. Comparing these two metrics tells us whether our model over or under-proportionally predicted bracket tokens, i.e. structure nodes, incorrectly.

5.2.1.3 Task metrics: Nesting height

Algorithm 2 NestingHeight(Node)

```
1: if children(N) = ∅ then
2:     return 0
3: end if
4: M ← 0
5: for Node child in children(N) do
6:     M ← max(M, NestingHeight(child))
7: end for
8: return M + 1
```

As not all bracket tokens have the same kind of complexity, we will also evaluate this in regard to how far up in the parse tree the node of the bracket is. We will thus categorize the bracket tokens according to their **nesting height**. This is the depth of the subtree spanned by the node according to the bracket token, which is equivalent to the longest downward distance from the said node to any leaf node. Algorithm 2 provides pseudo-code for computing the nesting height of a given node. Note that leafs, i.e., non-bracket tokens, will have a nesting height of 0. We hypothesize that modulation of performance through nesting heights correlates with incentivization and capabilities to model hierarchies in a non-trivial way, i.e. without resorting to statistical heuristics. The following section may provide a deeper understanding why we judged this metric to be of use.

5.2.1.4 Analytical assumptions

We can divide **BRACKET** into two subtasks: parsing the input sequence into a tree structure (**PARSE**) and copying the sequence entries to their respective place (**COPY**). **PARSE** is the structural component of our task and mimics the desired and correct parsing

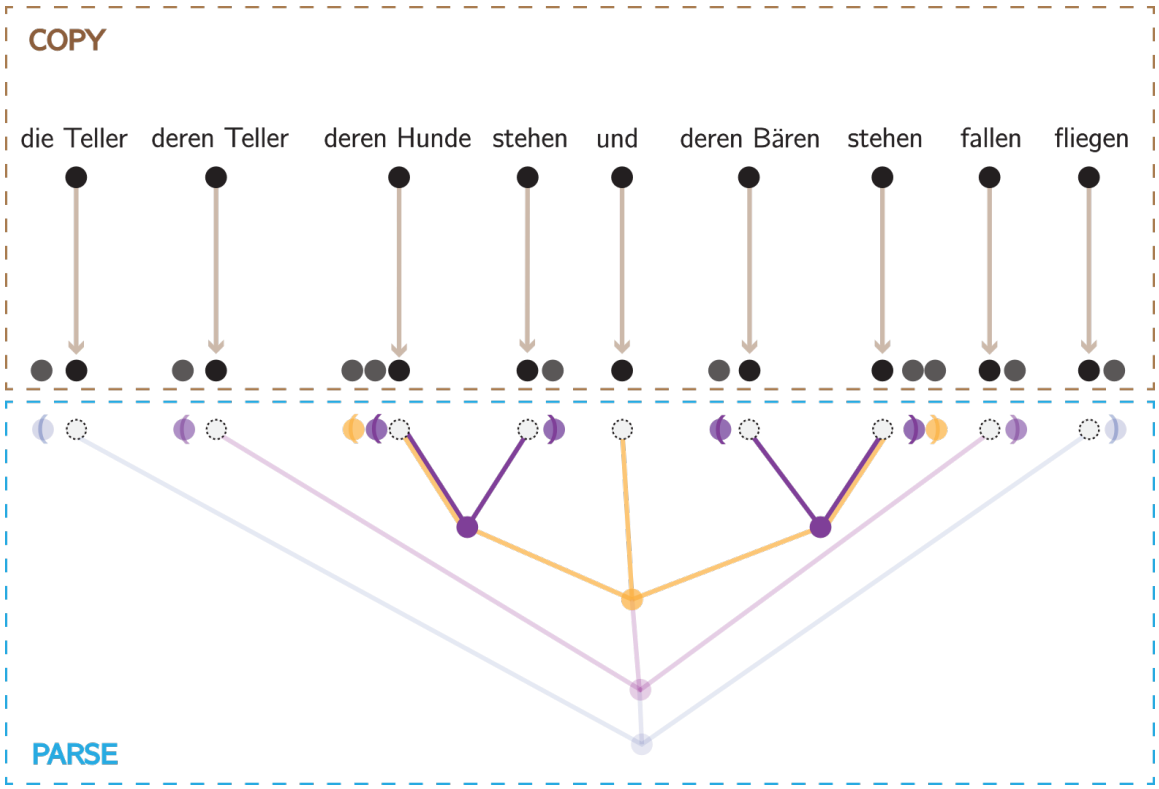


Figure 5.1: Depiction of analytical division of **BRACKET** into **PARSE** and **COPY**. The dots int the middle represent the output token positions for both tasks, respectively.

of context-free grammars. Mastering parse is therefore equivalent to learning $\mathcal{D}_N$, where N is the number of Non-Terminals of our grammar. Note that this is irrespective of the employed grammar. Therefore, even for our regular grammars **PARSE** and by extension **BRACKET** will be of the computational level of pushdown automata.

We hypothesize that **PARSE** becomes more complex and less likely to be approximated by simple statistical means the higher the level of the parse tree. This is due to two interacting reasons.

	1D	2D	SF	NSF	LARGE	**Average**
$Perc(BR_{\geq 0})$	100%	100%	100%	100%	100%	100 %
$Perc(BR_{\geq 1})$	53.6%	53.9%	53.5%	54.5%	57.9%	54.4 %
$Perc(BR_{\geq 2})$	14.1%	14.9%	13.6%	17.1%	30.1%	16.9%
$Perc(BR_{\geq 3})$	8.8%	9.3%	13.1%	13.9%	10.4%	10.2%
$Perc(BR_{\geq 4})$	5.0%	5.5%	12.7%	10.1%	3.1%	7.1%
$Perc(BR_{\geq 5})$	1.5%	3.2%	12.3%	8.3%	0.5%	5.6%

Table 5.1: Average percentage of bracket tokens of total tokens per training set for tokenization strategy $\mathcal{T}^{words}$. $BR_{\geq N}$ means that only brackets have been counted whose nesting height, that is, the depth of the appropriate subtree, is greater than or equal to N. Non-brackets have nesting heights of 0. The total percentage of bracket tokens is equal to $BR_{\geq 1}$.

The first reason is decreasing incentives. Our model might not find motivation to learn **PARSE** for higher nesting heights, since the reward, that is, decrease in loss, comparatively decreases the higher the node in the tree. This is due to the fact that higher-level tree nodes, i.e. nonterminals, cover an increasingly broader area of the input sequence, making their respective bracket tokens rarer. As seen in table 5.1, the percentage of brackets decreases drastically with increasing nesting height. We judge tokens of nesting height one as being trivial, as they only symbolize the words non-terminal and thus do not need much effort to infer. On average, only 16.9% of all tokens are nontrivial type brackets. In fact, only 31% of all bracket tokens are nontrivial. Note, that these numbers are even lower for tokenization with byte-pair encoding, as here words would be split into separate tokens yielding comparably more non-bracket tokens. This will be discussed in more detail later. Depending on grammar and a proper definition of triviality, the percentage may be even lower. For example, for **SF** and for most subphrases of **NSF**, the vast majority of opening brackets, while having high nesting heights, can be inferred by trivial statistical means.

The second reason is the increasing complexity. For a model using a recursively hierarchical strategy, even if bounded by the layer count, brackets of higher nesting heights would not necessarily be more complex to identify. However, a model that cannot resort to such mechanisms will contrarily have more hardship the higher the nesting heights. This is due to the fact that higher tree nodes logically depend on lower-level tree nodes when reconstructing the parse tree bottom-up. With no recursively-hierarchical mechanism

available, uncertainties of stochastic approximation would exponentially multiply.

Thus, **PARSE** might become un-rewarding quite fast, especially for narrowly branching and deep trees. On the other hand, **COPY** only relies on a rough idea of the general tree structure, i.e., how many higher-level bracket tokens separate two adjacent words. For some words, this might even be constantly 0, if they share a subphrase within the tree. Moreover, non-bracket tokens or lower-level bracket tokens that can be inferred by local statistical rules will occur more frequently than (higher level) bracket tokens. This makes learning **COPY** more rewarding.

Furthermore, note that the separation between **PARSE** and **COPY** is analytical in essence. This means that it is not unthinkable (or it is rather most likely the case) that certain strategies learned by the model would be useful for both tasks and, moreover, that advances in one task are strictly dependent on advances in the other. Nevertheless, this separation holds explanatory power in the sense that we and was therefore chosen to be used to illustrate reasoning of choices taken by the model.

We will try to affirm this analysis by finding correlations of errors and higher nesting heights in our experiment. The results can be seen in subsection 6.2.4.

5.2.2 MASK

5.2.2.1 Formalism

Given a formal grammar $\mathbf{G} = (S, NT, T, P)$, a sentence $s = s_1 \dots s_n \in \mathcal{L}(\mathbf{G})$, a mask index m, the mask token *[MASK]* and the masked sequence $\bar{s} = (\bar{s}_1 \dots \bar{s}_n)$

$$\bar{s}_i = \begin{cases} s_i & i \neq m \\ [MASK] & i = m \end{cases}$$

the task will be to predict the non-terminal of s_m, i.e. $\mathcal{D}^{-1}(s_m) \in NT$. For our constructed context-free grammars (**1D**, **2D**) we restrict masking to nouns and verbs. This will be especially of importance for our analysis, described in the following.

5.2.2.2 Task metric: span width

We will analyze performance according to what we said in chapter 4 using a syntactical distance metric: **span width**. In the following, we will define it and describe the experimental setup in greater detail.

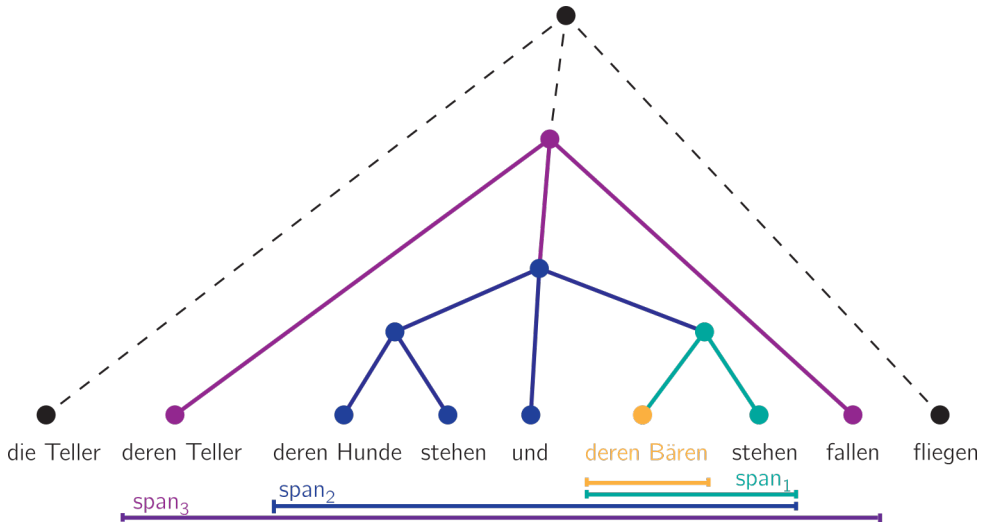


Figure 5.2: Span widths in simplified **2D** parse tree for an example sentence. For the yellow leaf node span widths of rank 1, 2 and 3 are shown. Note that span width counts variables and not tokens, resulting in: $span_1 = 2$, $span_2 = 5$ and $span_3 = 7$.

Definition: span width

Let **G** be a context-free grammar, and $s \in \mathbf{G}$ be a sentence of this grammar. Further, assume that there is exactly one parse tree $\mathbf{T_G(s)} = (V_s, E_s)$ for that word, where V_s defines all the nodes of that parse tree and $E_s \subset V_s \times V_s$ defines all the edges. Naturally, each word of that sentence is a leaf of that tree: $\forall w \in s w \in V_s$. This tree may be fully described by defining the following $parent^T : V_s \rightarrow V_s$ function:

$$parent^T(e) = \begin{cases} \text{p} & \text{if } \exists!_{p \in V_s}\ (p, e) \in E_s \\ \text{e} & \text{if } \nexists!_{p \in V_s}\ (p, e) \in E_s \end{cases}$$

Further define the set of Rank $r \in \mathbb{N}^+$ grand-parents by the following:

$$grandparents_r^T(e) = \begin{cases} \{e\} & \text{if } r = 0 \\ \{parent^T(g) | g \in grandparents_{r-1}^T(e)\} \cup grandparents_{r-1}^T(e) & \text{if } r > 0 \end{cases}$$

Then we can define the span width of Rank r of a word $w \in s$ like so:

$$span_r^T(w) = |\{v | v \in s \text{ and } grandparents_r^T(w) \cap grandparents_r^T(v) \neq \emptyset\}|$$

So $span_r^T(w)$ denotes the number of words that share a common grandparent with w up to rank r. For context-free grammars those words form a sub-string of s. This is due the tree-structure of their parse trees.

5.2.2.3 Experimental setup

We will analyze our task using a combination of our span width metric and the **2D** grammar. By restricting masked words to the verbs and nouns (which resemble the brackets Dyck-2 grammar), we ensure that there will always be a predictor word. The syntactical distance between those two words will always be the same, while their linear distance may differ. We will use this to evaluate the data dependency and stability of assumed stochastic heuristics as described in chapter 4.

More precisely, in the **2D** setup, there will be an entangled subject-predicate sharing a common grammatical number (t), whose one part will be masked. For clarity, we will assume that the verb will be masked and the subject will be visible, but the same will apply vice versa. In our case, the only word that reliably predicts t will be our predictor noun. The task will now be to pick the right predictor of multiple other possible ones, i.e. attractors, similarly to subject-verb congruency tests as conducted by Goldberg (2019) [22]. He already showed that performance in the presence of so called"attractors" decreases, but we will further examine this empirically according to our hypothesis. For this, we measure the performance with respect to different span widths. For **2D**, $span_1$ will be especially important, as this will be precisely the length of the subphrase that specifies the grammatical number of the subject-predicate pair, i.e. the distance between the masked word and the predictor. Other $spans_{\geq 2}$ should have negligible influence on task performance as they express broader contexts that do not add any informational value

to predictions. This can be assumed for idealized, uniformly distributed sentences. If we still find correlations here, this may hint that predictions are highly dependent on data distributions learned during training.

5.3 Data Generation

Data generation has the goal of generating a finite number of samples from which an infinite principle is reconstructable. Therefore, it should aim to produce a *representative* distribution of data samples. What representable means remains unclear. The most straightforward approach would have been to use a **probabilisitic context-free grammar** (PCFG). Here, each rule for a given Non-Terminal will be assigned a probability. By iteratively sampling possible derivations, one would then acquire a set of sentences. When using uneven probabilities, this approach yields unbalanced trees. This might incentivize the model to learn statistical heuristics. However, using equally distributed probabilities may lead to gigantic trees that are hard to compute and unsuitable for our models. Instead, we employed a hybrid approach. Generation is split into two parts: *tree building* and *derivation*.

Given a grammar G and a starting symbol S during *tree building* the tree of all possible derivations would be lazily computed. For this, our tree would have two alternating kinds of nodes: **choice** nodes and **product** nodes.

Example: Computing the possibility tree

Rules: $G_{example}$

- $S \rightarrow TUT$
- $T \rightarrow SU \mid SS \mid UUS$
- $U \rightarrow fizz \mid buzz$

Suppose we want to compute the possibility tree over TUT. This would be equivalent to all possibilities for T concatenated with all posibilities of U and concatenated with all possibilities for T again. We may define $\pi(X)$ as the set of all possible derivations of X and denote our above observation as :

$$\begin{aligned} \pi(TUT) =& \pi(T \circ U \circ T) \\ =& \pi(T) \times \pi(U) \times \pi(T) \end{aligned} \tag{5.1}$$

To compute the possibilities of T we get:

$$\begin{aligned} \pi(T) =& \pi(SI|SS|UUS) \\ =& \pi(SI) \cup \pi(SS) \cup \pi(UUS) \end{aligned} \tag{5.2}$$

We refer to computations like Equation 5.1 as product nodes and to ones like Equation 5.2 as choice nodes. Note, that a product node is constructed by multiplying some choice nodes, which then again are constructed by multiplying some product nodes etc. We may recursively compute this tree to some degree, stopping in the case we hit nonterminals. To retrieve all possible phrases out of a possibility tree we would iteratively collect subphrases from each node, concatenating samples for product nodes and picking one sample for choice nodes. This was implemented through recursive iterators, upon whom we shall not further expand here. To make sure the size of our tree does not explode above our wished data set size, we would only expand leaf nodes after measuring the number of possibilities. This can be computed by the recursive function defined in algorithm 3.

Algorithm 3 CalculatePossibilities(Node)

$N \leftarrow$ Node
$childs \leftarrow$ children(N)
if $childs = \emptyset$ **then**
    return 1
end if
if N is of type **product node then**
    return $\Pi_{i=0}^{n} CalculatePossibilities(childs[i])$
end if
if N is of type **choice node then**
    return $\Sigma_{i=0}^{n} CalculatePossibilities(childs[i])$
end if

Given a sample size T, the leaf nodes of the lazy tree would then be successively expanded in a breadth-first manner until the number of possibilities would just exceed T. We will thus end up with $P \geq T$ leaf nodes, i.e. possible phrases, that may still include non-terminals. Given $N \geq T$ leaf nodes, in the *derivation* phase those would then be forcibly derived until the end, i.e., until all variables become terminals. This will be done in a non-deterministic way, giving production rules that may lead to a faster termination and were used less frequently a higher chance of being applied.

Each of these fully derived sentences would then be processed by a task-specific generator, yielding a number of input-label pairs per sentence.

- For **BRACKET**, this means bracketing the sentence according to its derivation tree, as discussed above.

  Input: the unaltered sequence.
  Label: the parsed tree

- For **MASK**, this means masking out a single word.

  Input: the masked sequence.
  Label: the non-terminal representing the class of syntactically equivalent words

This pipeline was generated via a Java generator project spanning over 5.8k lines of code specifically developed for this purpose. The generator code can be openly retrieved from Github [45] . The exact data sets used in this work can be obtained by using the configuration supplied with it, including the random seed.

5.4 Models

5.4.1 Vanilla Sequence-To-Sequence Transformer

For comparison, the Vanilla transformer was used, that is, the original transformer introduced by Vaswani et al. [55]. Prior to passing the input through the Transformer, the tokens will first be passed through an embedding layer, yielding vectors of the dimension of the hidden size on which then the proposed sinusoidal positional encoding will be conducted. The labels during training will then be passed through the network, where the decoder is forced to predict next-token probabilities in a left-to-right manner by supplying a triangular matrix mask to the cross-attention block. This means that for a given token, only previous predictions may be used to make a choice. After passing the input through, the predicted token probabilities will then be retrieved by reverting the embedding through a simple fully connected linear layer, yielding vectors of the size of the vocabulary. The models performance with 1, 3 and 6 respective encoder and decoder layers was evaluated. A hidden size of 512 was considered sufficient, as larger sizes did not contribute to improving model performance. The models will be referenced by $\textbf{SIMPLE}_{\textbf{1L}}$, $\textbf{SIMPLE}_{\textbf{3L}}$, and $\textbf{SIMPLE}_{\textbf{6L}}$.

5.4.2 Vanilla Classification Transformer

Similarly, a Vanilla classifier transformer was used for the masking and classification task. After the token prepossessing mentioned above, the input would be passed through a stacked encoder-only layer on the top of which the BART simple classification head was placed. For masking experiment, only one configuration was used: 6 encoder and decoder layers and a hidden size of 512.

5.4.3 Bidirectional Long Short-Term Memory

A bidirectional long short-term memory was used as a baseline. Once embedded, the sequence will be successively encoded by one LSTM layer from left to right and then from right to left, producing vectors of size $2 * \textit{hidden size}$ per token. The seq2seq model will then employ a decoder that aggregates those embeddings using Bahdanau attention [5]. Similarly, the classification head will instead use this attention mechanism to yield a probability distribution on the output classes. This mechanism enables the model to focus on different parts of the input sequence for each step of the output sequence, providing more flexibility to capture dependencies in variable-length sequences. This mitigates the bottleneck effect of encoder state aggregation from which most LSTMs suffer, making it more suitable as a baseline. The model will be referenced by **LSTM**.

5.4.4 BERT

For the masking task, BERT (Bidirectional Encoder Representations of Transformers) was used [16]. BERT is, as the name says, a bidirectional encoder-only Transformer. Other than the vanilla transformer, models build on this architecture are able to access token positions in both directions during encoding, allowing it to capture intricate language context. It was trained on large text corpora in an unsupervised way by masking and then predicting text snippets. The pre-trained model should then be fine-tuned on the downstream task. As previously described, BERT outperformed most other LLMs when it was introduced. The two base variants of BERT are $\textbf{BERT}_{\text{BASE}}$ and $\textbf{BERT}_{\text{LARGE}}$. For this work $\textbf{BERT}_{\text{BASE}}$ was judged to suffice, having a hidden size of 768 and 12 layers. The model weights were loaded into *huggingfaces BertForSequenceClassification*, which is just BERT with a simple linear layer, normalization and dropout as the classification head. The model will be referenced by $\textbf{BERT}_{\text{BASE}}$.

5.4.5 BART

For the sequence-to-sequence task, *meta's* BART was used [34]. BART is built upon the BERT bidirectional encoder and the GPT auto-regressive decoder. BART is trained similarly to BERT by corrupting and predicting large sets of natural language. Due to its encoder-decoder architecture, its vanilla architecture can be used for sequence-to-sequence tasks, like the one in this work, more easily. More specifically, the *BartForConditionalGenration* model, initialised by the pretrained weights of $\textbf{BART}_{\text{BASE}}$ (6 encoder and 6 decoder layers, 768 hidden size) from *huggingface* [63] was employed. [34]. The model will be referenced by $\textbf{BART}_{\text{BASE}}$.

5.5 Model Configurations

5.5.1 Number of layers

As already mentioned, for the seq2seq task, the effect of the number of layers will be compared for $\textbf{SIMPLE}_1$, $\textbf{SIMPLE}_3$, and $\textbf{SIMPLE}_6$.

5.5.2 Tokenization strategy

Byte-Pair Encoding (bpe) is a popular tokenization strategy first described by P. Gage in 1994 [20]. Starting with characters, it iteratively merges tokens according to their occurrences within a given corpora. Almost all LLMs prefer bpe over other tokenization strategies, as they might be too task-specific or unreliant. This leads to rather large vocabularies with over 50.000 tokens (GPT-3). For comparison: a native English speaker knows a range of 20.000 to 30.000 vocabulary words.

Although bpe seems a good choice for natural text, subword tokenization, and especially more granular ones like bpe, do not necessarily contribute anything to formal language tasks. Formal language tasks have a predefined vocabulary and do not rely on subword information. However, the language model may exploit subtleties in data sets or grammar-specific information structures that might be hard for humans to detect. Furthermore, similar to the findings of Ebrahami et al. (2020) (see chapter 3), the attention mechanisms may use spare tokens as additional computation space.

In this work, performance of wordwise ($\mathcal{T}^{words}$) tokenization (i.e. each word of the vocabulary will be a separate token) will be compared to byte-pair encoding ($\mathcal{T}^{bpe}$).

Note furthermore that, regardless of tokenization, bracket tokens used for the tree parsing task will be tokenized as single tokens. Each respective bracket (opening or closing) will be tokenized separately. This means that for a grammar containing $|N|$ nonterminals and $|T|$ terminals, there will be $2 * (|N| + |T|)$ bracket tokens. $\forall t \in N \cup T$ we will denote $\mathbf{BL}_i$ as the opening token and $\mathbf{BR}_i$ as the respective closing bracket token.

5.5.3 Positional Encoding

Transformers are per-se permutation invariant, making them highly parallalizable and thus efficient to train. On the other hand, this is a disadvantage for position-reliant tasks, such as language modeling. Thus, some kind of positional information needs to be encoded into the token embeddings.

We will employ standard positional encodings, meaning that BART and BERT will keep their learned positional encoding scheme, and the Vanilla transformers will use the proposed sinusoidal scheme. While other positional encodings, as shown in related work, may improve model performance for specific tasks, this work focuses on how transformers employed as LLMs learn syntax *in the wild*. Performance with different positional encoding strategies, especially the promising $\mathbf{p}/\mathbf{n}$ scheme [64], may leave room for future work.

5.6 Training

All models were implemented using *PyTorch*. Predefined models were loaded from *huggingface*. Depending on the task and the model, the following training features were chosen:

Dataset entries	80.000 - 120.000
Learning rate	$\{5e-7, 5e-6, 5e-5\}$
Learning rate scheduler	{*None*, **NoamOptim** [55]}
Gradient clipping norm	{*None*, 0.5, 1.0}
Token positions	{256, 512}
Batch size	{4, 8}
Epochs	1-5
Optimizer	**Adamw**
Loss function	**Cross-Entropy**

The training was carried out on a GeForce GTX 1080Ti. Early stopping was used for models that converged faster, like the pre-trained models, but generally, considering the size of the data sets, 1-2 epochs would mostly suffice. We tracked validation loss and used L2 regularization to mitigate the risk of overfitting. Automatic mixed-precission (AMP) and dynamic batching with rather small batch sizes was used to optimize memory usage, as our bracket task needed 512 token positions and CUDA memory was running low. AMP usage, if not carefully conducted, sometimes leads to numerical inaccuracies; due to this and the nature of task and model design, especially for our Vanilla models, gradient clipping was occasionally employed to mitigate exploding gradients. Gradients would hereby be clipped to a threshold value, usally around 0.5. The learning rates would thus be kept relatively low to allow finding fine adjustments.

6 Results and analysis

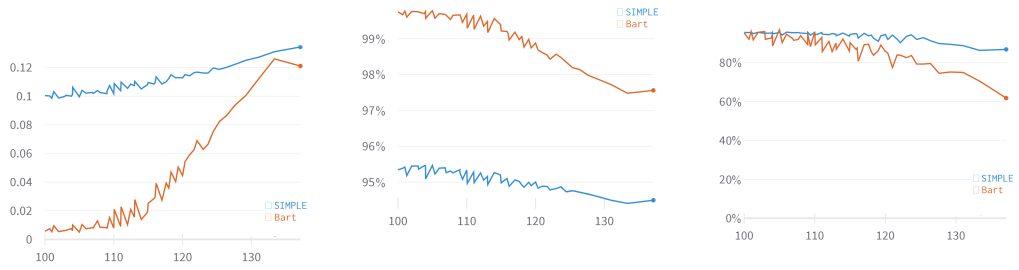
6.1 Notation

For **BRACKET** we measure bracket accuracies and accuracies for 5 models, 5 grammars and 2 tokenization strategies.

To provide a clearer notion of which values we are currently talking about, we will be using $(\mathcal{M}\ \text{G}\ \text{TOK})$ to denote the configuration consisting of model $\mathcal{M}$, grammar **G** and tokenization strategy **TOK**. If any dimension is missing, we refer to the averaged value over that dimension. For example, $\mathbf{Acc}_{BR}(\ (\text{SIMPLE}_{3L}\ \mathcal{T}^{words})\ )$ denotes the bracket accuracy of $\mathbf{SIMPLE}_3$ for word-wise tokenization averaged over all grammars. For only one dimension, we will not use parentheses. For a given grammar and model, we may also use $\mathbf{Acc}^{\mathcal{M}}(G)$ to denote the accuracy (or bracket accuracy) averaged over both tokenization strategies.

6.2 Tree Bracketing

6.2.1 Sequence length



a) Loss per input length

b) General accuracy per input length

c) Bracket accuracy per input length

Figure 6.1: Performance in respect to sequence length for $\mathbf{BART}_{\text{BASE}}$ and $\mathbf{SIMPLE}_{6L}$ on the grammar **2D**.

This section should affirm what we have said so far about modulation by sequence length. As already stated, we do not necessarily expect our model to generalize well on out-of-domain data, and thus do not put much emphasis on testing such sequences. Keeping this in mind, we evaluate our model on data that is both partially in-domain and out-of-domain. This is done by sampling 30% of the longest string sequences of our generated data sets, which are not necessarily the longest tokenized sequences. For

evaluation, we sort our inputs by length to inspect model performance in relation to input size. The results in Figure 6.1 show that $\textbf{SIMPLE}_{\textbf{6L}}$ and $\textbf{BART}_{\textbf{BASE}}$ generalize well on this in-domain training set, although observing a downward trends for the accuracies and an upward trend for the loss.

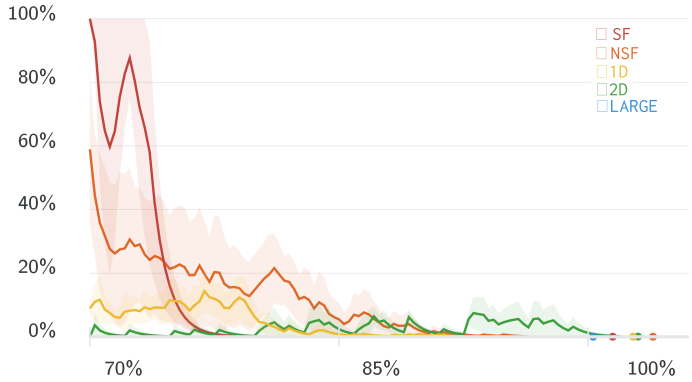


Figure 6.2: Full hit accuracy per length. As grammars differ in yielded lengths, the x-axis denotes the resepctive part of evaluaton set sorted by length. For example, 70% refers to sequences whose lengths are larger than 70% and smaller than 30% of the training corpus for their respective grammar.

When measuring the full hit accuracy, i.e., how many sequences were predicted completely correctly, we observe performance dropping drastically when entering lengths that were not seen during training. This can be seen in Figure 6.2. This is consistent with related research and our hypothesis about the limited ability of such models to generalize.

Although not a goal of our research, it would still be interesting to test our models on even larger inputs to see whether per-token accuracy breaks down completely. We leave this for future work.

6.2.2 Grammar comparison

	1D	2D	SF	NSF	LARGE
Acc	89.6	88.4	**93.4**	88.8	84.4
$\textbf{Acc}_{\textbf{BR}}$	77.0%	77.0	**85.3**	78.5	60.3

Table 6.1: Per-token average general (**Acc**) and bracket ($\textbf{Acc}_{\textbf{BR}}$) accuracy averaged over all models and tokenization strategies.

For the five grammars **1D, 2D, SF, NSF** and **LARGE** we average scores over all models and tokenization strategies. The results can be seen in 6.1. As expected, **SF** performed better than **NSF**. **1D** and **2D**, however, performed nearly identical with **1D** only performing slightly better. Furthermore, the constructed context-free grammars ($\textbf{Acc}_{BR}(\textbf{CFG}) = 76.97\%$) performed worse than the regular ones ($\textbf{Acc}_{BR}(\textbf{REG}) = 81.85\%$).

All the constructed grammars performed overall better than **LARGE**, where bracket accuracy was especially low. Best performance can be seen on our simple star-free regular language **SF** matching expectations.

We are especially surprised to find the bracket accuracies for our context-free grammars to be nearly identical, as they should rather correlate with tree complexity than general token accuracy. We hypothesize that the constructed context-free grammars' complexities might have been neglectable in contrast to general task complexity. In contrast, the difference of these two compared to **LARGE** is substantial, especially apparent regarding the bracket accuracy. Bracket accuracies, however, were generally subject to more fluctuation, as they are not per se necessary for low loss values, as we will discuss below.

6.2.3 General model performance

	$\mathbf{BART_{BASE}}$	$\mathbf{SIMPLE_{1L}}$	$\mathbf{SIMPLE_{3L}}$	$\mathbf{SIMPLE_{6L}}$	**LSTM**
1D	99.2%	65.6%	92.2%	92.4%	99.7%
2D	98.5%	59.8%	91.0%	93.3%	99.6%
SF	99.1%	72.0%	98.5%	98.6%	99.5%
NSF	99.2%	67.1%	88.9%	88.9%	100.0%
LARGE	97.2%	59.6%	78.3%	87.2%	99.6%
Average	98.6%	64.8%	89.8%	91.9%	99.6%

Table 6.2: Average accuracy scores per model and grammar ($\mathbf{Acc}^{\mathcal{M}}(\mathbf{G})$)

The pretrained BART model outperformed the vanilla model even though both have the same number of layers. The effects of pre-training will be discussed in subsection 6.2.7. As seen in the averages scores in Table 6.2, the following hierarchy of model performance appears to hold more or less:

$$\mathbf{LSTM} \geq \mathbf{BART_{BASE}} > \mathbf{SIMPLE_{6L}} > \mathbf{SIMPLE_{3L}} > \mathbf{SIMPLE_{1L}}$$

Matching expectations, **LSTM** performed slightly better than the best transformer model, regardless of pretraining. This observation is consistent with related theoretical research, like [15], who find the syntactical capabilities of LSTMs in theoretical laboratory work to exceed those of transformer networks.

Furthermore, the layer count turns out to be a modulator of model performance for the vanilla transformer models.

$$\forall_{G \in \{1D,2D,SF,NSF,LARGE\}} \mathbf{Acc}^{\mathbf{SIMPLE_{1L}}}(G) \leq \mathbf{Acc}^{\mathbf{SIMPLE_{3L}}}(G) \leq \mathbf{Acc}^{\mathbf{SIMPLE_{6}}}(G)$$

underpinning this further. We will illuminate this aspect especially in subsection 6.2.1.

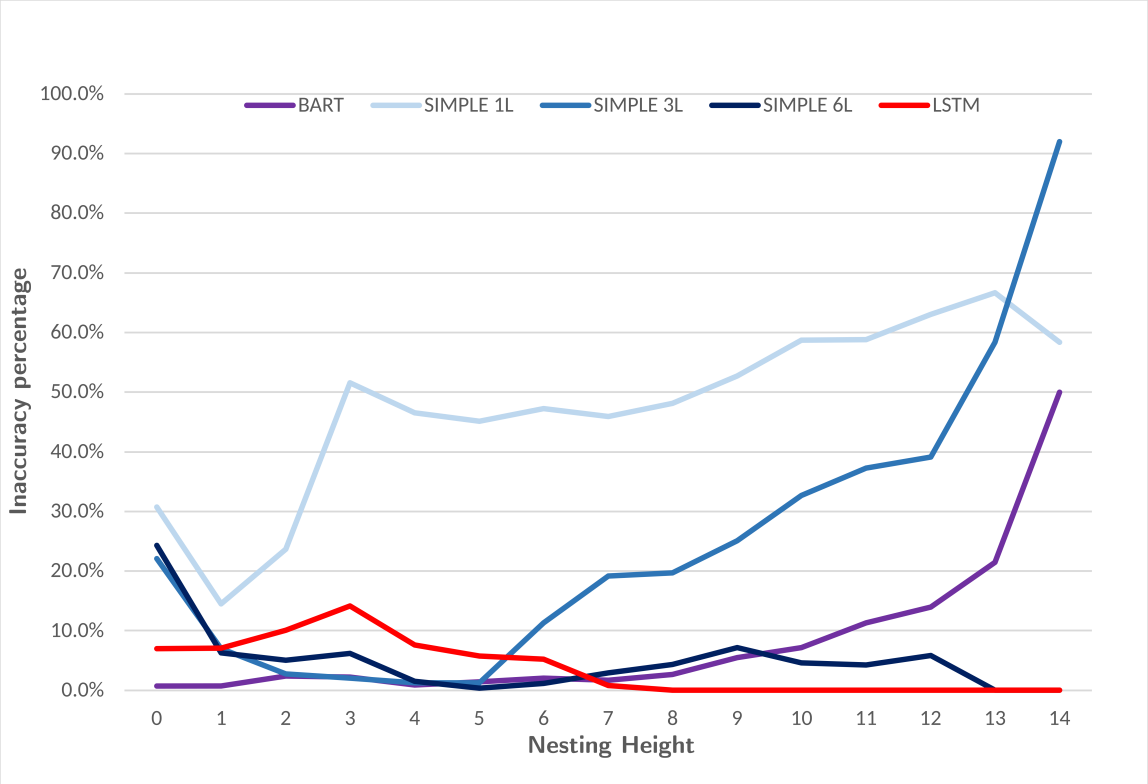


Figure 6.3: Percentage of erroneously predicted tokens in respect to their nesting height for $_{(\text{2D } \mathcal{T}^{words})}$.

6.2.4 Performance on structure

6.2.4.1 Performance per nesting height

The data in Table 5.1 already gave a small indication to what we discussed in chapter 5. Grammars whose bracket heights occur more evenly distributed, like **SF** and **NSF**, were judged less complex and were performed better on, as seen previously in subsection 6.2.2. To further verify this hypothesis, 10.000 random sequences from our evaluation set were sampled. The nesting heights of the erroneous tokens were compared, which yielded the results in Figure 6.3. We observe that our transformer models, except for $\textbf{SIMPLE}_{\textbf{6L}}$, were generally more likely to make errors with increasing nesting heights. For these models, the effect seems to start taking place after a certain threshold. For $\textbf{SIMPLE}_{\textbf{1L}}$, we already experience this taking place at nesting height 1 with a jump after nesting height 2. When equipped with two more layers, the vanilla transformer first begins omitting this effect after nesting height 5. The bracket error percentage for our pre-trained model only starts to gradually rise after nesting height 7. We hypothesize that these thresholds might be correlated with the depth limit N for learning $\mathcal{D}_{k,N}$ as discussed in chapter 3. Following this, for nesting heights higher than that bound, our model would need to or be incentivized to resort to statistical methods, resulting in the observed effect. We observe these bounds to correlate with model capacities, i.e, number of parameters. To state this

with greater certainty, a more detailed analysis regarding those depth bounds would be appropriate, which we will leave for future work.

In contrast to this, **SIMPLE$_{6L}$** and **LSTM** appeared to have no problem learning even higher nesting heights. This might mean that either a non-trivial hierarchical strategy was learned or that the formerly assumed bound was sufficient for this data set. As the same architecture employing less layers still suffered under said effect, we assume the latter for the transformer, although future experiments would need to confirm or deny this assumption.

Predicted	Non-Bracket		Bracket	
True	Non-Bracket	Bracket	Non-Bracket	Bracket
BART$_{BASE}$	14.4%	3.0%	4.7%	77.9%
SIMPLE$_{1L}$	69.8%	0.00%	1.0%	29.2%
SIMPLE$_{3L}$	47.9%	0.5%	0.3%	51.3%
SIMPLE$_{6L}$	43.3%	6.6%	0.1%	55.0%
LSTM	60.5%	17.9%	16.4%	5.2%
Average	47.18%	5.6%	4.5%	43.72%

Table 6.3: Match percentages of token kinds (non-bracket, bracket) of model mistakes.

We further analyze the kind of errors that our models made in respect to whether they at least expected the kind of token (non-bracket or bracket token). The data can be found in Table 6.3. We find that on average, 90.9% of predictions predicted the right kind of token. Analyzing our data further, we find, although still for our limited sample, nearly half (*43.72%*) of made mistakes by models could have been mitigated if brackets had no annotated type. This finding is strongest for our pretrained model **BART$_{BASE}$**. This would reduce our problem to be loosely equivalent to $\mathcal{D}_1$, which transformers have been found to learn non-trivially (see chapter 3). Furthermore, to solve **COPY** robustly, the type of bracket tokens does not play much role. **COPY** only relies on predicting where the next non-bracket token should be placed, which means that only the bare skeleton of our tree would be necessary.

This may indicate that the model might learn a more restricted structural version than is necessary by this task. Here, a more elaborate analysis would be quite informative, which might allow one to clarify whether this behavior indeed occurs as hypothesized, and, if so, whether it is task-specific or whether transformers generally employ Dyck-1-like parsing. Surely, repeating this experiment without annotated brackets would yield promising data for this.

6.2.4.2 Comparing token accuracies

Because the models were trained using standard cross-entropy loss, the accuracy scores correlate greatly with loss: Higher accuracies generally mean lower losses. Loss computation does not discriminate between ordinary words or bracket tokens: the more predictions that match the labels, the lower the loss. This is not necessarily true for the precision of

		1D	2D	SF	NSF	LARGE
Non brackets	$\mathcal{T}^{words}$	18	24	9	15	611
	$\mathcal{T}^{bpe}$	38	47	21	41	695
Brackets		$2 * 9$	$2 * 11$	$2 * 5$	$2 * 7$	$2 * 298$

Table 6.5: De-facto number of unique tokens, i.e. output vocabulary, per data set and tokenization strategy. Note that $\textbf{BART}_{\textbf{BASE}}$ has 30.522 possible tokens, of which only a portion was used in our data sets.

the brackets. Strategies that increase overall accuracy may be learned at the expense of learning structure. In the following, we will take a closer look on the bracket accuracies regarding this.

As we have seen, in order to achieve as many correctly labeled tokens as possible, the model may learn **COPY** and for low-level bracket types **PARSE**. We may further measure this when the loss decreases or, more precisely, when the overall accuracy (**Acc**) increases, whereas mastering **PARSE** would result in an increase in bracket accuracy ($\textbf{Acc}_{\textbf{BR}}$).

	$\textbf{BART}_{\textbf{BASE}}$	$\textbf{SIMPLE}_{\textbf{1L}}$	$\textbf{SIMPLE}_{\textbf{3L}}$	$\textbf{SIMPLE}_{\textbf{6L}}$	**LSTM**	**Average**
Acc	98.6%	64.8%	89.8%	91.9%	**99.6**%	88.9%
$\textbf{Acc}_{\textbf{BR}}$	90.3%	21.3%	86.0%	84.0%	**96.5**%	75.6%

Table 6.4: Model-wise average accuracy and bracket accuracy scores

Given the general loss computation and the above, one would expect the bracket accuracy to be generally lower than the general accuracy. This can also be found in the data, seen in 6.4: average general accuracy (**88.9%**) is higher than average bracket accuracy (**75,6%**), which does also hold for nearly all per-grammar and per-model average accuracies.

There are some notable exceptions when using $\mathcal{T}^{words}$. More precisely, these exceptions occur only for the models $\textbf{SIMPLE}_{\textbf{3L}}$ and $\textbf{SIMPLE}_{\textbf{6L}}$. Here, bracket accuracies for nearly all grammars exceed general accuracies: e.g. $\textbf{Acc}^{3L}(NSF) = 84,53\% > \textbf{Acc}^{3L}_{BR}(NSF) = 94,37\%$. Only **LARGE** is exempt from this. In Table 6.6 this effect can be viewed averaged over all grammars. $\textbf{BART}_{\textbf{BASE}}$, **LSTM** and $\textbf{SIMPLE}_{\textbf{1L}}$, and no model employing $\mathcal{T}^{bpe}$ shows this effect.

We hypothesize that this may be due to the fact that $\mathcal{T}^{words}$ has fewer non-bracket tokens than $\mathcal{T}^{bpe}$, as seen in Table 6.5. As we discussed above, the comparative advantage of learning **COPY** instead of **PARSE** can be modulated by increasing the percentage of trivial or non-bracket tokens. $\mathcal{T}^{bpe}$ splits words that would otherwise become separate tokens, thus increasing the comparative amount of non-brackets, while, by our tokenizer construction (see subsection 5.5.2), the number of bracket tokens stays the same. More non-bracket tokens mean a comparatively greater reward for learning **COPY**. Therefore, learning

structure will not be as appealing as with $\mathcal{T}^{words}$. On the other hand, **LARGE** already consists of a large number of different tokens and yields the most complex grammatical structures. Thus, word-wise tokenization may not substantially improve performance.

$\textbf{SIMPLE}_{\textbf{3L}}$	$\mathcal{T}^{bpe}$	$\mathcal{T}^{words}$
Acc	**92.35%**	87.19%
$\textbf{Acc}_{BR}$	81.69%	**90.25%**

$\textbf{SIMPLE}_{\textbf{6L}}$	$\mathcal{T}^{bpe}$	$\mathcal{T}^{words}$
Acc	**94.42%**	89.71%
$\textbf{Acc}_{BR}$	78.11%	**89.84%**

Table 6.6: Scores averaged over all grammars for the vanilla with employing three (left) and six (right) layers.

Why this behavior cannot be observed with the other models may be because they perform either too well or too bad. $\textbf{SIMPLE}_{\textbf{1L}}$ might simply not have the capacity to approximate **PARSE**, as already seen in Figure 6.3. This would rely on approximating recursive hierarchies and, as already discussed, this would most likely require at least two layers. On the other hand, $\textbf{BART}_{\textbf{BASE}}$ and **LSTM** reach quite similar bracket accuracies as the vanilla networks, but higher general accuracies. Their general architectural advantage might push the model to finding more general solutions to the problem instead of greedily focusing on one of the two subtasks.

6.2.5 Layer size

We previously found that layer size enhances the models' performance, measured in the overall accuracy scores seen in Table 6.2. Surprisingly, $\textbf{SIMPLE}_{\textbf{3L}}$ performed extraordinarily well. Despite the fact that it only had half the layers of $\textbf{SIMPLE}_{\textbf{6L}}$, it had a comparable performance to it and even exceeded the accuracy of $\textbf{SIMPLE}_{\textbf{6L}}$ regarding bracket accuracy (see Table 6.4). This could be explained by one of the constructions of Yao et al. (2021) [64], who showed how a minimum of two layers are needed for a soft attention transformer to learn or approximate hierarchy, a core part of context-free languages. To verify this, further experiments with two layers as well as experiments that compare the attention patterns found by the ones described by the researchers should be conducted. This would go beyond the scope of this work.

As for bracket accuracy performance, we have to restrict our previous observation:

$$\textbf{LSTM} \geq \textbf{BART}_{\textbf{BASE}} > \textbf{SIMPLE}_{\textbf{NL}}$$

Regarding the modulation by layer size, while this holds when comparing $\textbf{SIMPLE}_{\textbf{1L}}$ with $\textbf{SIMPLE}_{\textbf{3L}}$ and $\textbf{SIMPLE}_{\textbf{6L}}$, we find the average bracket accuracy for the three layer network to be higher than the six layer network. Furthermore, a more fine-grained analysis of our data also finds this to be the case for nearly each grammar individually. Combining this with our finding in subsubsection 6.2.4.1, we hypothesize that $\textbf{SIMPLE}_{\textbf{3L}}$ might have rather focused on finding strategies for lower-level features, while $\textbf{SIMPLE}_{\textbf{6L}}$ focused on a rather holistic strategy. Because lower lever brackets occur vastly more often than higher-level brackets, as already discussed in subsection 5.2.1, this strategy might have lead to a higher bracket accuracy, although learning less complex structure. Rather than

emitting inductive biases for the opposite behaviour, we thus observe models to rather follow task incentives, which is consistent with the analysis by Liu et al mentioned in chapter 3.

6.2.6 Tokenization strategy

	$\mathbf{BART_{BASE}}$		$\mathbf{SIMPLE_{1L}}$		$\mathbf{SIMPLE_{3L}}$		$\mathbf{SIMPLE_{6L}}$		**LSTM**	
	$\mathcal{T}^{bpe}$	$\mathcal{T}^{words}$	$\mathcal{T}^{bpe}$	$\mathcal{T}^{words}$	$\mathcal{T}^{bpe}$	$\mathcal{T}^{words}$	$\mathcal{T}^{bpe}$	$\mathcal{T}^{words}$	$\mathcal{T}^{bpe}$	$\mathcal{T}^{words}$
Acc	**99.1**%	98.1%	**73.6**%	56.0%	**92.4**%	87.2%	**94.4**%	89.7%	99.6%	**99.7**%
$\mathbf{Acc}_{BR}$	**90.5**%	90.1%	**33.7**%	8.8%	81.7%	**90.3**%	78.1%	**89.8**%	95.3%	**97.6**%

Table 6.7: Scores averaged over all grammars

The general accuracy performance was slightly better with $\mathcal{T}^{bpe}$, increasing **Acc** on average by 10.9%. This can be attributed to the fact that even when mislabeling a given nonbracket word at a certain position, subparts of that word that are not captured by $\mathcal{T}^{words}$, may randomly still be correctly predicted.

For $\mathbf{SIMPLE_{1L}}$ this improvement was more striking, increasing performance by a total of 17,5 percantage points or by 31,26% comparatively. Here, the restricted capacities of having only one layer may have been leveraged by having more token positions at hand. Furthermore, tokens that do not carry any meaning relevant to the task, e.g. when a token always precedes a certain other predictor token, may be used for additional computations similarly to how the starting token in the experiments conducted by Ebrahimi et al. (2020) [17] was used to indicate the end of a sub-sequence. Generally when employing a statistical solution to data that by nature of it's generation shows a certain task independent information structure the model profits from having more data points, i.e. tokens, as this increases the probability of finding such.

This might also explain why, when examining bracket accuracies this finding becomes even more apparent. When using byte-pair encoding, for $\mathbf{SIMPLE_{1L}}$ accuracy increased by 24,9 percentage points which more than quadrupled the bracket accuracy (increase of **282,5**%). For wordwise tokenization, **PARSE** occupies a larger share of the overall task, making identifying brackets more important. However, it might have been nearly impossible for the network to predict bracket tokens in a non-trivial way, due reasions discussed above. Contrary, for $\mathcal{T}^{bpe}$ spare tokens may have helped the model to find statistical heuristics for the bracket tokens, instead.

On the other side for $\mathbf{SIMPLE_{3L}}$ and $\mathbf{SIMPLE_{6L}}$, comparative performance of $\mathcal{T}^{words}$ increased bracketing performance by 9.5% and 13%, respectively. As already discussed, this may be due to the fact that while the models have the capacity to learn **PARSE** in a non-trivial way, they are only incentivized to do so for $\mathcal{T}^{words}$, making $\mathcal{T}^{bpe}$ rather disadvantageous for increasing bracket accuracy. If this arguments prevails the six layer, $\mathcal{T}^{words}$ configuration may be a sweet spot of having the capacities to learn non-trivial syntactical representations and being highly incentivized to do so. As common language modeling tasks, including the ones LLMs are trained with, give little to no inclination

to learn explicit syntactical structures (compared to **BRACKET**) this finding may hint that additionally to (or maybe due to) their weak theoretical capacities, large transformer networks rely on explicit motivation to learn more complex syntactical structures, rather than relying on low-level statistical heuristics.

6.2.7 Influence of pre-training

We may use this small subsection to summarize findings regarding one aspect of our experiment: in how far pretraining was beneficial. We have seen that general accuracies of our pretrained model exceeded that of our vanilla transformer in subsection 6.2.3. We observed the same thing, although not quite as remarkable, for bracket accuracies in subsubsection 6.2.4.2. When assessing the heights of the brackets in subsubsection 6.2.4.1 however, we find that $\mathbf{BART_{BASE}}$, in contrast to $\mathbf{SIMPLE_{6L}}$, starts performing worse for higher bracket heights. We hypothesize that pretraining, following assumptions by Liu et al., may incentivize models to learn rather course grained, shallow features, which then may be employed to find statistical heuristics for more complex features. We cannot state this with certainty as model architectures, e.g. the positional encoding strategies, differed between both models. We find this area to be highly promising for future research.

6.2.8 Commaless grammar

		$\mathbf{BART_{BASE}}$		$\mathbf{SIMPLE_{1L}}$		$\mathbf{SIMPLE_{3L}}$		$\mathbf{SIMPLE_{6L}}$		**LSTM**	
		$\mathcal{T}^{bpe}$	$\mathcal{T}^{words}$	$\mathcal{T}^{bpe}$	$\mathcal{T}^{words}$	$\mathcal{T}^{bpe}$	$\mathcal{T}^{words}$	$\mathcal{T}^{bpe}$	$\mathcal{T}^{words}$	$\mathcal{T}^{bpe}$	$\mathcal{T}^{words}$
2D	**Acc**	99.1%	97.9%	63.7%	55.9%	94.1%	87.9%	95.1%	91.4%	99.7%	99.5%
	$\mathbf{Acc}_{BR}$	87.8%	96.6%	15.3%	10.49%	79.0%	92.0%	94.2%	95.2%	98.7%	99.4%
$\mathbf{2D}_{CL}$	**Acc**	87.6%	96.1%	62.8%	74.7%	93.9%	89.6%	94.8%	89.9%	99.7%	99.7%
	$\mathbf{Acc}_{BR}$	80.0%	96.3%	14.3%	9.4%	82.7%	93.6%	91.5%	94.2%	98.7%	98.7%

Table 6.8: Scores on **2D** and $\mathbf{2D}_{CL}$

We conduct a small comparison study by training our model on $\mathbf{2D_{CL}}$, which is **2D** without commas. As discussed in chapter 4, our idea behind this is that the commas, being situated before and after each embedded phrase, may be a strong hint for the desired parse tree. Our results can be seen in Table 6.8.

We surprisingly find performance to be generally quite similar for all models: Average general accuracy for $\mathbf{2D_{CL}}$ is at *88.9%*, as opposed to *88.4%* for **2D**, and bracket accuracy is at *75.9%* as opposed to *77.0%*. We hypothesize that commas might not be as good of a predictor as classifying verbs and nouns. These, even for unknown grammatical number, also tell whether embedded sentences start or end at their respective position. Contrary, commas do not differentiate between these two types making them weaker predictors. Testing with a grammar, where the start and end allow the same words would be insightful here. Having said this, other effects of having less words at hand might have played more important roles.

6.3 Masking

6.3.1 General model-wise performance

	1D	2D	SF	NSF	LARGE
BERT$_{\text{BASE}}$	100%	99,9%	100%	100%	7,8%
LSTM	100%	98,9%	100%	100%	3,6%
SIMPLE$_{\text{6L}}$	99,9%	83,6%	71,6%	58,1%	3,6%

Table 6.9: Masking accuracies of different models and data sets averaged over $\mathcal{T}^{words}$ and $\mathcal{T}^{bpe}$.

The general performance on **MASK** as seen in Table 6.9 shows that performance on our constructed grammars for **BERT$_{\text{BASE}}$** and *LSTM* was nearly perfect. **LARGE** seemed quite hard to predict, which can be explained by considering the vast amount of non-terminals. **SIMPLE$_{\text{6L}}$** had more trouble. We observe, that it generally performed worse on **2D** than on **1D** and worse on **NSF** than on **SF**. Performance across context-free and regular grammars is hardly comparable as we restricted masking to verbs and nouns (as described in chapter 5) for the context-free grammars, making the task easier as less candidates are available. We still observe that grammar wise comparison mirrors grammar complexity clearer than for our previous task. For our context-free grammars this is amplified by the fact, that the very thing that makes **2D** more complex than **1D**, namely the additional bracket type, is precisely what we test our models for. Accuracy on other token positions will thus not blur the statistics, resulting in a clearer picture of the complexity of our grammars.

6.3.2 Span width

To inspect whether $span_N$ had an influence on model performance, the errors of **2D** were compared to the respective span width of the masked token. We measured the percentage of errors in the evaluation data and compared it with $span_N$ of that word for $n \in \{1, 2, 3\}$. The results for $span_1$ can be seen in the diagram 6.4. This was first measured for our regular evaluation data set, resulting in the diagrams to the left. Note that the drawn lines serve the purpose of illustrating the possible distributions. Our experiment is discrete in nature (either the model correctly classifies a token or it does not), and data set span widths are not equidistant, so assumptions over distributions are necessary. The yellow line assumes normal distribution per data point, and the blue line is a simple spline interpolation.

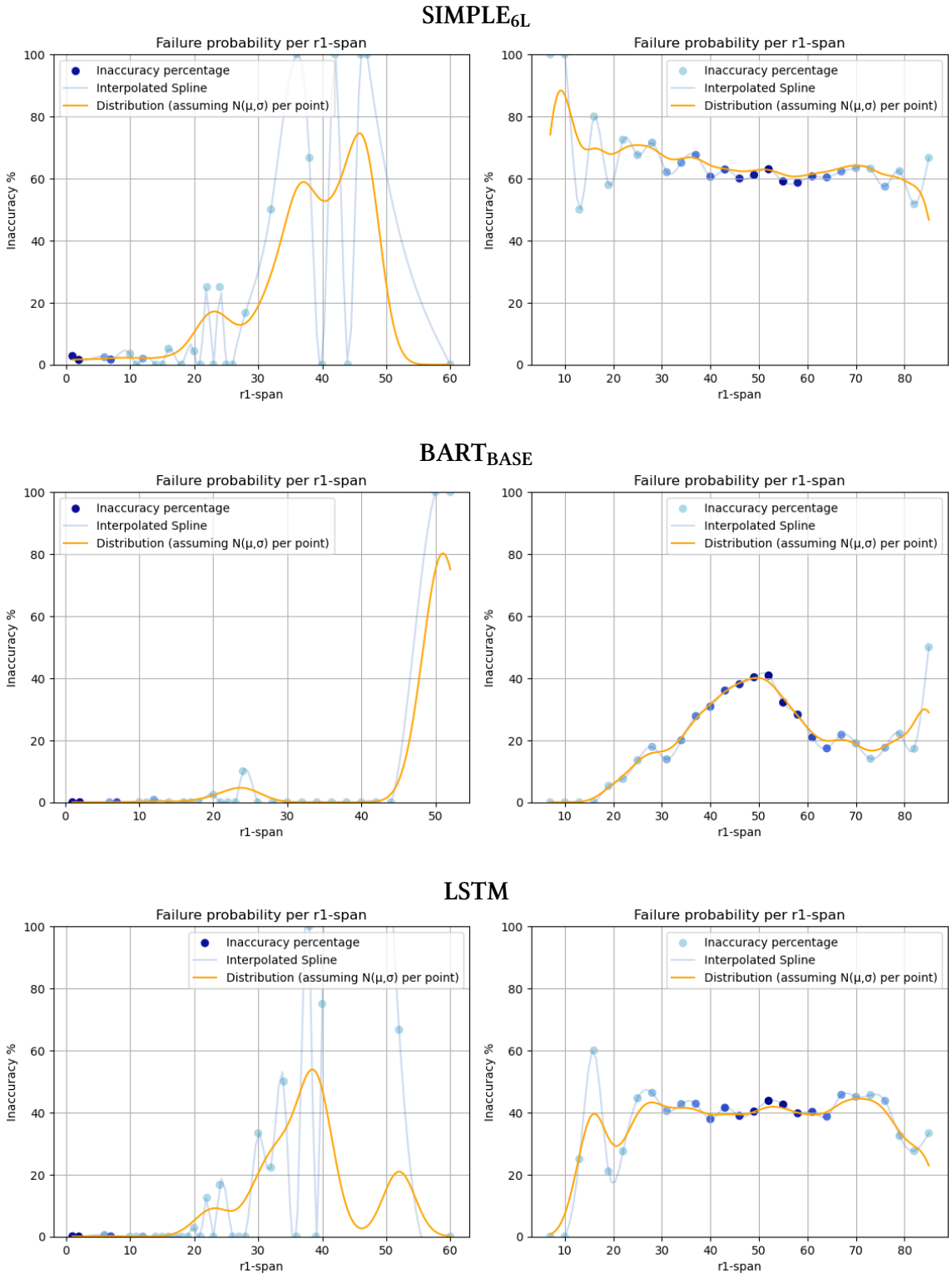


Figure 6.4: Inaccuracy percentages per $span_1$ for different models on **2D** (left) and **2D**$_{\text{SPAN}}$ (right). Distributions are not normalized. The strength of the data points indicate how often they have occurred.

To further verify this and mitigate the risk of influence of other variables, the model was tested on a new data set $\mathbf{2D_{SPAN}} \subset \mathcal{L}(\mathbf{2D})$ that was crafted just for this purpose. As the sequence length is a known modulator for the performance of the transformer, we ensured that all sentences in $\mathbf{2D_{SPAN}}$ are more or less the same length ($\Delta_{Lengths} \leq 8$). Additionally, $\mathbf{2D_{SPAN}}$ was designed to cover higher spanning widths that did not occur in the original training set. $\mathbf{2D_{SPAN}}$ could be loosely described in the following way:

Definition: $2D_{SPAN}$

Let N be the desired sequence length. To generate input-label pair, sample $n \in [0, N]$.

$$S \rightarrow NP_s \text{ , } \textbf{deren } pump(n) \text{ , } VP_s \textbf{ und } pump(N-n)$$

$$S \rightarrow NP_p \text{ , } \textbf{deren } pump(n) \text{ , } VP_p \textbf{ und } pump(N-n)$$

where $pump(n)$ returns a random sub-phrase of length n. This was constructed by using the linking word "und" in combination with arbitrary simple phrases.

As sequences within this data set, while part of the same grammar, were generated using another sampling and generation method, we will call those samples **out-of-distribution**, not to be confused with sequences that exceed span widths seen during training, that we will call **out-of-domain**. More precisely, all samples with $span_1 > 12$ will already be called out-of-domain as they lay within the last percentage of seen **2D** span widths of rank 1 (Regarding $span_1$ for **2D**: $\mu = 4.01, \sigma^2 = 14.54, min = 0, max = 47$).

The results for $\mathbf{2D_{SPAN}}$ can be viewed on the right-hand side of the figure 6.4. We can see that, while $\mathbf{SIMPLE_{6L}}$ generalizes well on in-domain and in-distribution samples, it fails to do so for out-of-domain or out-of-distribution samples, indicating both a tendency to employ statistical heuristics and a modulation of performance by span width. In particular, out-of-distribution samples that lie in-domain seem to confuse it, scoring accuracies partially far below the (nearly) zero-knowledge probability (50%), approaching it only for larger out-of-domain inputs. Lack of out-of-distribution generalization can also be seen for $\mathbf{BART_{BASE}}$. Here, performance for in-distribution but out-of-domain span widths is surprisingly well, solving the task reliably even for span widths up to **45**. However, this behavior cannot be observed for our out-of-distribution set, where, similarly to $\mathbf{SIMPLE_{6L}}$, the percentage of inaccuracy increases rapidly after span widths of around 15. Thus, $\mathbf{BART_{BASE}}$ seems to have found better heuristics for our original data set that are nevertheless statistical in essence, otherwise the same performance would have been observed for $\mathbf{2D_{SPAN}}$. Note here that this *distribution overfitting* is quite likely to have occurred, considering that $\mathbf{BERT_{BASE}}$ employs twice the number of layers as our vanilla model. For our **LSTM**, employing only one layer, we observe an increase in the percentage of inaccuracies for out-of-domain but in-distribution span widths, similar to $\mathbf{SIMPLE_{6L}}$. Contrary to our transformer models, for out-of-distribution data, the same behavior can be observed: only domain seems to modulate inaccuracies, indicating a rather distribution-invariant learning approach for out task. This is consistent with our prior hypothesis, stating that Transformer networks, contrary to more automata-like models,

may be rather vulnerable to settle for statistical heuristics.

To further validate this claim, we look at higher span widths. Following our aforementioned explanation, those should have a negligible influence on task performance. If we nevertheless observe a correlation, this could also be an indicator that exceptionally large sequences break statistical heuristics (which rely on broader contexts).

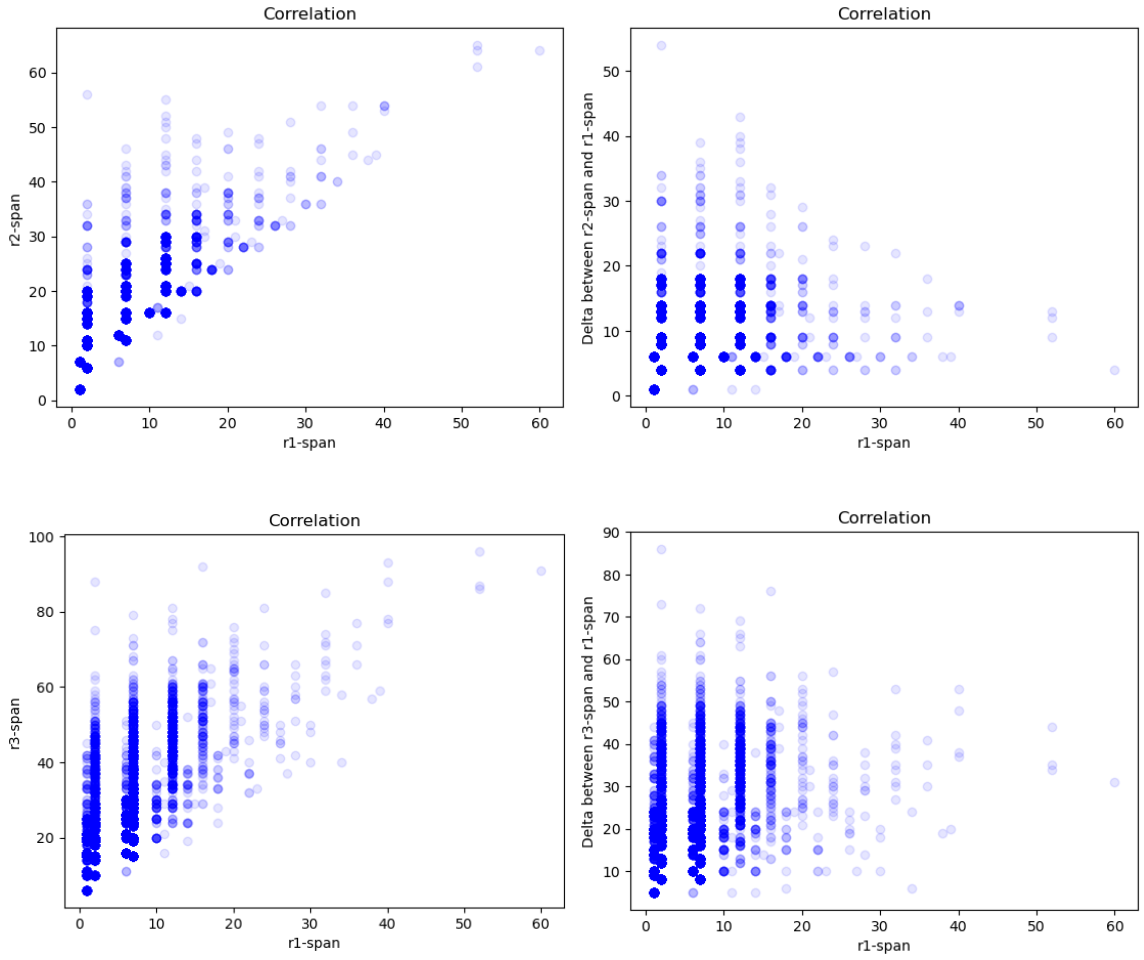


Figure 6.5: Correlation to $span_1$ of pure span widths (left) and delta span-widths (right).

However, observing $span_2$ and $span_3$ directly might not be a good idea because $span_1$ correlates with them:

$$\forall_{L \in L(CFG)} \forall_{w \in L} \forall_{N \in \mathbb{N}^+} : span_N(w) \leq span_{N+1}(w)$$

High $span_2$ and $span_3$ therefore may highly correlate with high $span_1$, as seen in Figure 6.5.

Instead, we examine the correlations of the delta of higher span widths to $span_1$ ($span_{\Delta_1^2}$ and $span_{\Delta_1^3}$) with inaccuracies. We do this only for in-distribution data, since $\mathbf{2D_{SPAN}}$, by construction, has neglectable span deltas. It would be interesting to verify this finding with another out-of-distribution data-set specifically crafted for this purpose.

The results can be seen in Figure 6.6. As expected, our **LSTM** seems to be indifferent to high delta spans. So does $\mathbf{BART_{BASE}}$, although we might trace this to its general in-distribution performance. We find a mild correlation of $span_{\Delta_1^2}$ and a stronger correlation of $span_{\Delta_1^3}$ to the performance of $\mathbf{SIMPLE_{6L}}$. This seems quite surprising, as theoretically we could construct arbitrary samples of any $span_{\Delta_1^3}$ with small $span_1$ that should be just as easy to solve as the bare sub-phrase of $span_1$ leaving us with no apparent causal dependency between $span_{\Delta_1^3}$ and $span_1$. Of course, such dependencies cannot be ruled out for this specific test set-up.

The size of the super-trees, in which the mask phrase is embedded, seems to modulate performance here. Assuming a statistical approach, this might not be surprising: if you have no reliable way of recognizing your sub-phrase (and thus your predictor), you may want to find statistical regularities that help you in finding its area. Similar to predicting the kind of leaf node within a simple subtree by statistical means, you can predict the kind of simple subtree embedded within your larger super tree. This might explain why larger contexts are considered. Larger supertrees generally occur in a higher variety than smaller ones. Masked words with smaller contexts (or slimmer sup-trees) are thus more suitable to be approximated statistically than larger ones. This is amplified by the fact that if we sample trees in a nearly equally distributed way, larger trees occur less often than smaller ones. This means that tokens near the masked token are more likely to be similarly structured the smaller the super tree is. This might explain why even for broader contexts, an increase in size may influence model performance. For a proper analysis on this part (and to proof or disproof our assumption for $\mathbf{BART_{BASE}}$ for out-of-distribution data), we would need to test out-of-distribution examples, too, as well as eliminate the factor of sequence length, to see if our assumptions uphold. Nevertheless, if this argument prevails, this may be a hint that transformers employing attention mechanisms, while, by doing so, theoretically overcoming limitations of restricted context accessibility (compare to vanishing gradient problem for RNNs), might be inclined to learn large statistical context dependencies on sufficiently distributed data sets even for tasks where an optimal solution would require learning efficient and restricted contexts, e.g. congruency in language. Here may be great potential for future work.

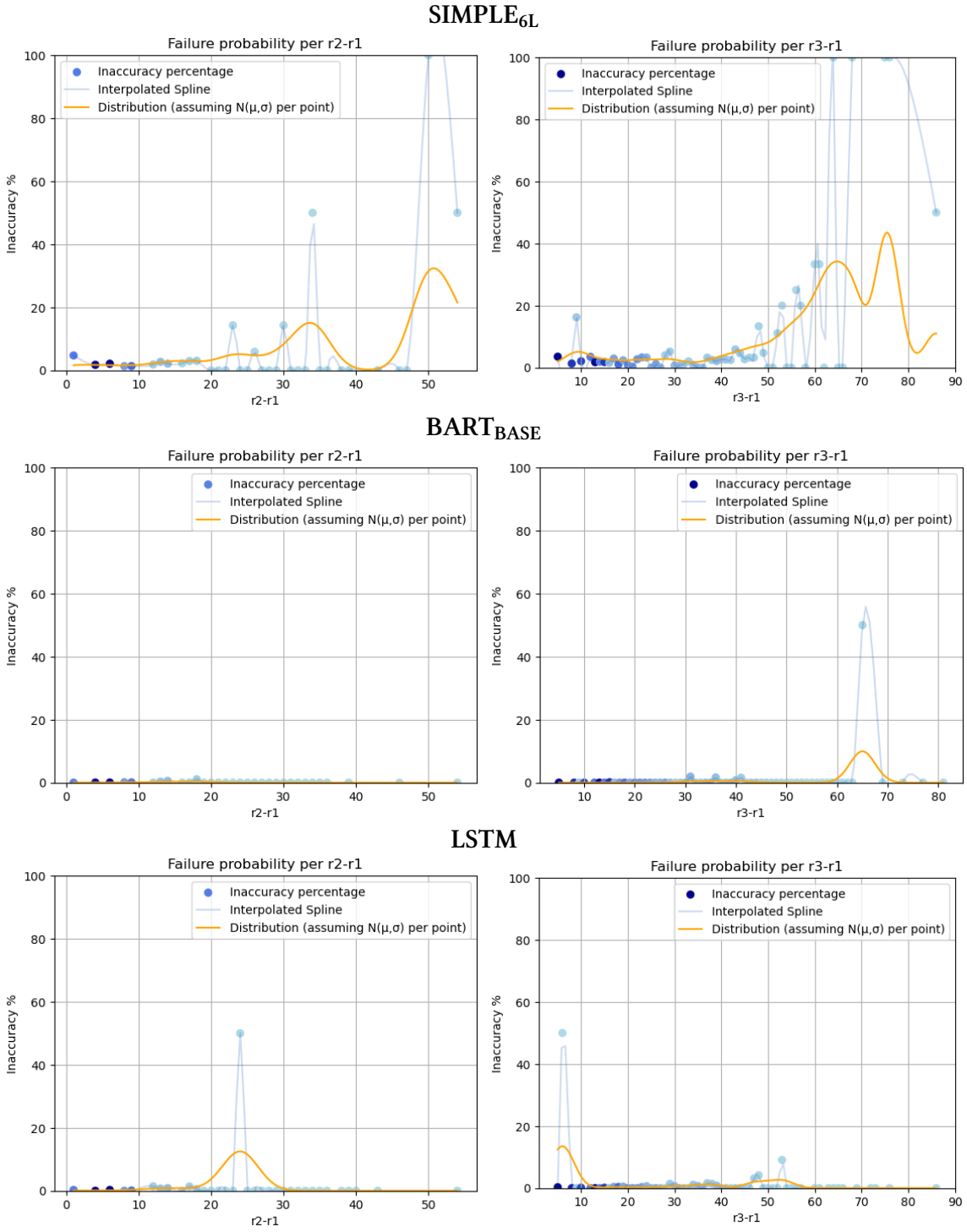


Figure 6.6: Inaccuracy percentages per $span_{\Delta_1^2}$ (left) and $span_{\Delta_1^3}$ (right) for different models on **2D**. Distributions are not normalized. The strength of the data points indicates how often they have occurred.

7 Discussion

7.1 Main insights

We tried to show what kind of research can be conducted under our hypothesis stated in chapter 4. For our **BRACKET** task, we found that models need strong incentives to learn structure, as opposed to statistical heuristics. In subsection 6.2.4 we find evidence that transformer models focus on lower level syntactical features over which approximations for higher level structure may be obtained. Rather than finding an inductive bias that purposely drives the model to build complex grammatical structures, we have reason to believe that task design rather needs to force it to do so – for example, by increasing comparative loss for structural components, as we observed for tokenization strategies in subsection 6.2.6. This is consistent with the analysis shared by Liu et al. (2019) [37], who emphasize pre-training-specific learning in LLMs. We find hints that models to approximate the task over less complex structural constructs, i.e. $\mathcal{D}_1$, for which related research has provided us with reasons to believe they are instead capable of. Furthermore, we observe how sequence level metrics, like our general accuracy metric, but even for our more refined bracket accuracy metric, do not necessarily justify judgement about learned syntactical abilities. Performance may very well come at the expense of learning less complex structures. Having found all of this on a task that provides explicit incentive to learn structural representations, we may only speculate that this behavior also prevails for common pretraining tasks.

For our second task, **MASK**, we find that, rather than containing inductive biases that drive models to learn syntax-driven metrics, as humans do, the models' induction principles compute over the linear order of the words. We further find that models seem to be vastly distribution-dependent, with stochastical heuristics breaking for out-of-distribution data. In the course of this, we show how not only overly long inputs, but even extremes of other seemingly irrelevant metrics seem to break approximations, when they correlate with data distributions. As similar in implementation, our findings could furthermore provide theoretical understanding of the factors leading to some deficits in agreement tasks.

7.2 Limitations of our work

7.2.1 General limitations of our work

As stated in chapter 4 this is a broad exploratory study with the aim of showing areas where finer-grained future work may provide further information regarding our hypothesis. We can therefore only validate findings regarding our specific test setup. We cannot certainly state that other model architectures using different training methods would have produced

different outcomes. Perhaps small changes to some of our models, like different positional encoding schemes or different model parameters, would have equipped our models with the syntactical inductive bias we were hoping to find. Neither can we be sure that other grammars or sample methods thereof may be more suitable for our models to develop such abilities.

7.2.2 Task specific limitations

Regarding **BRACKET**, some aspects of task design may have played a crucial role. Our metric choice of bracket accuracy could have been too restrictive or too loose. Correct guessing of brackets might have been more correlated with other factors than with the incentive of the model to "learn structure" and, as such, might have been too loose. On the other hand, a given set of sentences can be produced by arbitrarily many grammars. So, even when provided with the correct bracket types, our model might have learned different grammars with slightly different syntactical representations whose performance this task might not have captured. As such, our metrics might have been too restrictive. Furthermore, our task design, which is similar to parsing some variant of Dyck-N, might have been simply too complex. Perhaps, choosing brackets to have no types instead, rendering this task equivalent to Dyck-1, would already suffice to find different behaviors. Finally, the per-token approach, while more suitable for our purposes, has the downside of limited expressivity of evaluation data regarding the unguided generation of such sequences. Depending on the sampling method, the results of our models in real scenarios would differ. As our research had the ambition to analyze where difficulties may lie, instead of testing the realizability of our task, we still chose to use it. We must still admit that our high accuracy scores should not be taken as reasons to believe that our models had only minor difficulties of learning this task. In fact, even when provided with the entire left context of the label sequence, only a few percentage of overall produced sequences were completely correct.

For **MASK** we can state similar things about span width. We could not rule out the possibility of other correlations of other more important effects of our data set on span width performance. As such, a more granular metric, such as counting the syntactical distance per tree node, might have been more representative. As span widths were rather unevenly distributed and classifying tasks produce binary results, it proved hard to find continuous trends or correlations within discrete data points. As such, caution should be exercised when inferring statements on such results, especially for outliers in the data set. Therefore, the expressiveness of our results should be further validated.

7.2.3 Limitations of applicability to field work

As we have often noted, it is difficult to assert the applicability of theoretical research to models "in the wild". For one, we can only hypothesize that our tasks provide more or comparable incentives to the model to learn structure than the various pre-training tasks that are performed with LLMs. For another, our work focused on some formal grammars that, in the end, may not have as much significance for natural language. Despite the fact that we tried to mitigate the gap to field research by finding learning tendencies, we

cannot state for sure which behaviors prevail or emerge for large models, being trained on a vast amount of real data. Therefore, under our stated hypothesis, only findings in large, pre-trained language models can fully free us from doubt.

7.2.4 Discussion on the significance of our hypothesis

In our previous chapters, especially chapter 1, chapter 3 and chapter 4 we elaborated on the broader motivation of our hypothesis. Assuming that our hypothesis can be upheld, LLMs, as currently employed, may indeed be incapable of learning the true generative process of context-free languages and, by extension, of human language. We explained briefly how this deficit implies that the parsed syntactical – and thus semantical – meanings of sentences will therefore be unavailable for such models. Similarly, language generation will follow distributions of statistical rules of lower-level grammatical constructs with only limited knowledge of the meaning of the chosen syntax.

However, it remains unclear to what extent this disability becomes relevant for the proficiency in translation, summarizing, and other applications in most real-world scenarios. After all, having access to a vast amount of statistical heuristics, models seem well suited to produce in-distribution, sound sentences, judged by their overwhelming success. For sufficient bounds of language, research can show that the approximation abilities of such models are extraordinary. As an example, the finding of the ability to recognize bounded hierarchical constructs by Yao et al. (2021) can be named [64]. Regardless of the heated linguistic debate, in how far such findings are relevant to questions about the power of human language, externalized human language distributions seem to be well encaged by such bounds. One might thus justifiably question the importance of such measures of syntax for the design of LLMs. LLMs, one could argue, do not serve theoretical linguistic curiosity. They should rather be viewed as sophisticated, engineered tools to take over language related tasks. Just as an image generating network may have no "real" understanding of what characterizes a dog, both networks nonetheless may produce sufficient approximations for the purposes they are used for.

Following this view, one would be rather interested in where the limits of such approaches, i.e. statistical approximations over vast amount of data, lie. These limits would be rather qualitative in essence and thus not easily attainable. One possible approach would be to examine *field* findings in terms of nesting depth, to state with greater certainty whether transformers generally learn depth restricted languages and if so, whether k lies in dimensions that reveal this deficit in real world scenarios.

In conclusion, the significance of this work depends on the goals one seeks to achieve. For tasks such as text summarizing and sentiment analysis, the nuances of linguistic theory may matter less than the models' practical effectiveness. Nevertheless, a deeper exploration of the boundaries of these models' language understanding capabilities can provide valuable insights and guide the development of future language technologies.

7.3 Future work

In the process of conducting our analyzes in chapter 6, we came across many areas where further work would be necessary to produce more certain statements or complement our study. We may use this section to emphasize a few and name some more questions that have arisen during our work.

Regarding **BRACKET** we would be interested in further analyzing which parts of grammars are particularly complex for our model to form heuristics about. If we find such, we might have a better understanding of how far these findings apply to natural language. Furthermore, as discussed in subsection 6.2.4, we would be interested in conducting our experiments with untyped brackets, to see whether performance, especially for higher-level brackets, increases. If so, this could be a hint that such models tend to parse the structure of sentences, while still hierarchical, without creating explicit syntactical nodes. This could be complemented by repeating the study by Hewitt and Manning (2019), but instead by probing with typed trees [26]. Generally, more detailed studies on the inductive biases of learned syntactical structures and approximation strategies would be quite enlightening. Further work regarding other aspects of our methodology and model and data design aspects should also be carried out to verify our findings.

Regarding **MASK**, we find further experiments concerning the sensibility to data distributions quite promising. Above all, we look forward on narrowing down what features primarily constitute data distributions. Having figured out what kind of data distributions force the strongest generalizabilty out of our model, it will be easier to train models that learn more complex structures and thus solve tasks more reliably. We would also like to know if the observed effect can be mitigated through different tasks or architectures. We especially look forward for more research regarding models equipped with differentiable memory, as related work showed promising results.

Overall, the future work should aim to deepen our understanding of how neural networks learn or approximate syntax and shed light on the limits of these models in capturing the intricacies of human language. By addressing these aspects, future research can provide valuable insights into the capabilities and boundaries of language models, informing the development of more advanced and effective language technologies.

8 Conclusion

This research addressed the theory-practice gap in understanding how transformer networks learn and represent syntactical structures in languages. The central hypothesis posits that, while transformers can effectively model a substantial portion of human language, their computational limitations constrain their ability to discover overarching generative principles of language. Instead, these models may rely on statistical heuristics to approximate syntax, guided by their inductive biases and task-specific incentives.

Our study explores this hypothesis through two primary tasks: **BRACKET** and **MASK**. For **BRACKET**, it is revealed that models require strong incentives to learn structure over statistical heuristics. The findings suggest that models often focus on lower-level syntactical features, from which approximations of higher-level structures are derived. The research questions the existence of inductive biases that promote the development of complex grammatical structures, proposing that task design may need to compel models in this direction.

MASK and analyzation thereof delves into how models rely on linear word order rather than syntax-driven metrics, contrasting with human language comprehension. The research uncovers the model's dependence on data distributions, with statistical heuristics breaking when faced with out-of-distribution data. The study demonstrates how seemingly irrelevant metrics can influence model performance when correlated with data distributions.

While our research offers valuable insights, we acknowledge certain limitations. It primarily validates findings within the specific test setup, acknowledging the potential influence of alternative model architectures, training methods, and grammars. Task-specific limitations, such as metric choice and task design, are acknowledged, emphasizing the need for further validation and exploration. The study also highlights the challenges in applying theoretical research findings to real-world large language models, emphasizing the significance of future work on such models. We furthermore shortly elaborate on the the possible significance of our hypothesis, finding that while such behaviours may indeed prohibit transformers from "real" syntactical and thus semantical understanding, this might not impair its overwhelming success in its areas of application.

In conclusion, this research contributes to our understanding of how transformer models learn and represent syntactical structures in language, shedding light on the interplay of computational limitations, inductive biases, and task design. It paves the way for more fine-grained future work and complements the broader discourse on the syntactical capabilities of large language models.

Bibliography

[1] Joshua Ackerman and George Cybenko. *A Survey of Neural Networks and Formal Languages*. Tech. rep. ZSCC: 0000007 arXiv:2006.01338 [cs] type: article. arXiv, June 2020. DOI: 10.48550/arXiv.2006.01338. URL: http://arxiv.org/abs/2006.01338 (visited on 09/04/2023).

[2] Hessameddin Akhlaghpour. *Transformers Aren't Turing-complete, But a Good Disguise Is All You Need - Life Is Computation*. en-US. ZSCC: NoCitationData[s0]. Apr. 2023. URL: https://www.lifeiscomputation.com/transformers-are-not-turing-complete/ (visited on 11/06/2023).

[3] Jean-Michel Autebert, Jean Berstel, and Luc Boasson. "Context-free languages and pushdown automata". In: *Handbook of Formal Languages: Volume 1 Word, Language, Grammar* (1997), pp. 111–174.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. Tech. rep. ZSCC: 0009308 arXiv:1607.06450 [cs, stat] type: article. arXiv, July 2016. DOI: 10.48550/arXiv.1607.06450. URL: http://arxiv.org/abs/1607.06450 (visited on 11/05/2023).

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. Tech. rep. ZSCC: 0000005 arXiv:1409.0473 [cs, stat] type: article. arXiv, May 2016. DOI: 10.48550/arXiv.1409.0473. URL: http://arxiv.org/abs/1409.0473 (visited on 11/12/2023).

[6] Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. "On the Ability and Limitations of Transformers to Recognize Formal Languages". In: (Sept. 2020). arXiv: 2009.11264 [cs.CL].

[7] Satwik Bhattamishra, Arkil Patel, and Navin Goyal. *On the Computational Power of Transformers and its Implications in Sequence Modeling*. Tech. rep. ZSCC: 0000031 arXiv:2006.09286 [cs, stat] type: article. arXiv, Oct. 2020. DOI: 10.48550/arXiv.2006.09286. URL: http://arxiv.org/abs/2006.09286 (visited on 10/03/2023).

[8] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[9] Pauli Brattico. "Recursion Hypothesis Considered as a Research Program for Cognitive Science". en. In: *Minds and Machines* 20.2 (July 2010). ZSCC: 0000017, pp. 213–241. ISSN: 1572-8641. DOI: 10.1007/s11023-010-9189-8. URL: https://doi.org/10.1007/s11023-010-9189-8 (visited on 10/30/2023).

[10] Tom B. Brown et al. *Language Models are Few-Shot Learners*. Tech. rep. ZSCC: 0015354 arXiv:2005.14165 [cs] type: article. arXiv, July 2020. DOI: 10.48550/arXiv.2005.14165. URL: http://arxiv.org/abs/2005.14165 (visited on 10/16/2023).

[11] N. Chomsky and M.P. Schützenberger. "The Algebraic Theory of Context-Free Languages". In: *Computer Programming and Formal Systems*. Elsevier, 1959, pp. 118–161. DOI: 10.1016/s0049-237x(09)70104-1.

[12] Noam Chomsky. "Deep structure, surface structure and semantic interpretation". In: *1971* (1971), pp. 193–216.

[13] Rina S Cohen and Janusz A Brzozowski. "Dot-depth of star-free events". In: *Journal of Computer and System Sciences* 5.1 (1971), pp. 1–16.

[14] Michael C Corballis. "Recursion as the key to the human mind". In: *From Mating to Mentality*. Psychology Press, 2004, pp. 155–171.

[15] Grégoire Delétang et al. "Neural Networks and the Chomsky Hierarchy". In: (July 2022). arXiv: 2207.02098 [cs.LG].

[16] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: (Oct. 2018). arXiv: 1810.04805 [cs.CL].

[17] Javid Ebrahimi, Dhruv Gelda, and Wei Zhang. *How Can Self-Attention Networks Recognize Dyck-n Languages?* Tech. rep. ZSCC: 0000010 arXiv:2010.04303 [cs] type: article. arXiv, Oct. 2020. DOI: 10.48550/arXiv.2010.04303. URL: http://arxiv.org/abs/2010.04303 (visited on 02/09/2023).

[18] Jeffrey L Elman. "Finding structure in time". In: *Cognitive science* 14.2 (1990), pp. 179–211.

[19] Florian P Fischmeister et al. "Self-similarity and recursion as default modes in human cognition". In: *Cortex* 97 (2017), pp. 183–201.

[20] Philip Gage. "A new algorithm for data compression". In: *The C Users Journal archive* 12 (1994), pp. 23–38. URL: https://api.semanticscholar.org/CorpusID:59804030.

[21] Seddah Ganesh JawaharBenoît SagotDjamé. "What does BERT learn about the structure of language?" In: 2019.

[22] Yoav Goldberg. *Assessing BERT's Syntactic Abilities*. 2019. DOI: 10.48550/ARXIV.1901.05287.

[23] Michael Hahn. "Theoretical Limitations of Self-Attention in Neural Sequence Models". In: *Transactions of the Association for Computational Linguistics* 8 (Dec. 2020), pp. 156–171. DOI: 10.1162/tacl_a_00306.

[24] Morris Halle. *From memory to speech and back: Papers on phonetics and phonology 1954-2002*. Vol. 3. Walter de Gruyter, 2013.

[25] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[26] John Hewitt and Christopher D. Manning. "A Structural Probe for Finding Syntax in Word Representations". In: *Proceedings of the 2019 Conference of the North*. Association for Computational Linguistics, 2019. DOI: 10.18653/v1/n19-1419.

[27] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997). ZSCC: 0092980, pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[28] Aravind Krishna Joshi. “Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions?” In: (1985).

[29] Laura Kallemeyer. *Seminar Schwach kontextsensitive Grammatikformalismen*. 2011. URL: https://user.phil.hhu.de/~kallmeyer/GrammarFormalisms/.

[30] Laura Kallmeyer. “Linear Context-Free Rewriting Systems”. In: *Language and Linguistics Compass* 7.1 (2013), pp. 22–38.

[31] Fred Karlsson. “Constraints on multiple center-embedding of clauses”. en. In: *Journal of Linguistics* 43.2 (July 2007). ZSCC: 0000238, pp. 365–392. ISSN: 1469-7742, 0022-2267. DOI: 10.1017/S0022226707004616. URL: https://www.cambridge.org/core/journals/journal-of-linguistics/article/abs/constraints-on-multiple-centerembedding-of-clauses/9FB5CD5B24A8742C7CD18D84BC656CB5 (visited on 10/05/2023).

[32] Surafel M. Lakew, Mauro Cettolo, and Marcello Federico. *A Comparison of Transformer and Recurrent Neural Networks on Multilingual Neural Machine Translation*. Tech. rep. ZSCC: 0000130 arXiv:1806.06957 [cs] type: article. arXiv, June 2018. DOI: 10.48550/arXiv.1806.06957. URL: http://arxiv.org/abs/1806.06957 (visited on 10/16/2023).

[33] Howard Lasnik and Jeffrey Lidz. “The argument from the poverty of the stimulus”. In: (2016).

[34] Mike Lewis et al. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. Tech. rep. ZSCC: 0006732 arXiv:1910.13461 [cs, stat] type: article. arXiv, Oct. 2019. DOI: 10.48550/arXiv.1910.13461. URL: http://arxiv.org/abs/1910.13461 (visited on 10/10/2023).

[35] Jeffrey Lidz, Sandra Waxman, and Jennifer Freedman. “What infants know about syntax but couldn’t have learned: experimental evidence for syntactic structure at 18 months”. In: *Cognition* 89.3 (2003), pp. 295–303.

[36] Bingbin Liu et al. *Transformers Learn Shortcuts to Automata*. Tech. rep. ZSCC: NoCitationData[s0] arXiv:2210.10749 [cs, stat] type: article. arXiv, May 2023. DOI: 10.48550/arXiv.2210.10749. URL: http://arxiv.org/abs/2210.10749 (visited on 10/05/2023).

[37] Nelson F. Liu et al. *Linguistic Knowledge and Transferability of Contextual Representations*. Tech. rep. ZSCC: 0000660 arXiv:1903.08855 [cs] type: article. arXiv, Apr. 2019. DOI: 10.48550/arXiv.1903.08855. URL: http://arxiv.org/abs/1903.08855 (visited on 10/16/2023).

[38] Robert McNaughton and Seymour A Papert. *Counter-Free Automata (MIT research monograph no. 65)*. The MIT Press, 1971.

[39] Christopher Olah. *Understanding LSTM Networks*. Aug. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[40] OpenAI. *OpenAI Platform*. en. ZSCC: NoCitationData[s0]. URL: https://platform.openai.com/tokenizer (visited on 10/15/2023).

[41] Jorge Pérez, Pablo Barceló, and Javier Marinkovic. "Attention is Turing Complete". In: *J. Mach. Learn. Res.* 22.1 (Jan. 2021). ISSN: 1532-4435.

[42] Jorge Pérez, Javier Marinković, and Pablo Barceló. *On the Turing Completeness of Modern Neural Network Architectures*. Tech. rep. ZSCC: 0000109 arXiv:1901.03429 [cs, stat] type: article. arXiv, Jan. 2019. DOI: 10.48550/arXiv.1901.03429. URL: http://arxiv.org/abs/1901.03429 (visited on 10/03/2023).

[43] Jean-Éric Pin. "The dot-depth hierarchy, 45 years later". In: *THE ROLE OF THEORY IN COMPUTER SCIENCE: Essays Dedicated to Janusz Brzozowski* (2017), pp. 177–201.

[44] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. Tech. rep. ZSCC: NoCitationData[s0] arXiv:1910.10683 [cs, stat] type: article. arXiv, Sept. 2023. DOI: 10.48550/arXiv.1910.10683. URL: http://arxiv.org/abs/1910.10683 (visited on 10/16/2023).

[45] randomNumber101. *Phrase grammar sentence generator*. ZSCC: NoCitationData[s0] original-date: 2022-12-20T01:02:21Z. Dec. 2022. URL: https://github.com/randomNumber101/PhraseGrammarGenerator (visited on 11/12/2023).

[46] Hiroyuki Seki et al. "On multiple context-free grammars". In: *Theoretical Computer Science* 88.2 (1991), pp. 191–229.

[47] Stuart M Shieber. "Evidence against the context-freeness of natural language". In: *The Formal complexity of natural language*. Springer, 1985, pp. 320–334.

[48] H Sichel. "On a Distribution Representing Sentence-Length in Written Prose". In: *Journal of the Royal Statistical Society. Series A (General)* 137.1 (1974), p. 25. DOI: 10.2307/2345142.

[49] Hava T Siegelmann. "Computation beyond the Turing limit". In: *Science* 268.5210 (1995), pp. 545–548.

[50] Hava T Siegelmann and Eduardo D Sontag. "Analog computation via neural networks". In: *Theoretical Computer Science* 131.2 (1994), pp. 331–360.

[51] Hava T Siegelmann and Eduardo D Sontag. "On the computational power of neural nets". In: *Proceedings of the fifth annual workshop on Computational learning theory*. 1992, pp. 440–449.

[52] Mirac Suzgun et al. *LSTM Networks Can Perform Dynamic Counting*. Tech. rep. ZSCC: 0000053 arXiv:1906.03648 [cs] type: article. arXiv, June 2019. DOI: 10.48550/arXiv.1906.03648. URL: http://arxiv.org/abs/1906.03648 (visited on 10/20/2023).

[53] Mirac Suzgun et al. *Memory-Augmented Recurrent Neural Networks Can Learn Generalized Dyck Languages*. Tech. rep. ZSCC: 0000024 arXiv:1911.03329 [cs] type: article. arXiv, Nov. 2019. DOI: 10.48550/arXiv.1911.03329. URL: http://arxiv.org/abs/1911.03329 (visited on 10/04/2023).

[54] Ian Tenney, Dipanjan Das, and Ellie Pavlick. "BERT Rediscovers the Classical NLP Pipeline". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (2019) 4593-4601* (May 2019). arXiv: 1905.05950 [cs.CL].

[55] Ashish Vaswani et al. “Attention Is All You Need”. In: (June 2017). arXiv: 1706.03762 [cs.CL].

[56] Krishnamurti Vijay-Shanker and David J Weir. “The equivalence of four extensions of context-free grammars”. In: *Mathematical systems theory* 27.6 (1994), pp. 511–546.

[57] Alex Wang et al. *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*. Tech. rep. ZSCC: NoCitationData[s0] arXiv:1804.07461 [cs] type: article. arXiv, Feb. 2019. DOI: 10.48550/arXiv.1804.07461. URL: http://arxiv.org/abs/1804.07461 (visited on 10/16/2023).

[58] Alex Wang et al. *SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems*. Tech. rep. ZSCC: NoCitationData[s0] arXiv:1905.00537 [cs] type: article. arXiv, Feb. 2020. DOI: 10.48550/arXiv.1905.00537. URL: http://arxiv.org/abs/1905.00537 (visited on 10/16/2023).

[59] Yau-Shian Wang, Hung-Yi Lee, and Yun-Nung Chen. “Tree Transformer: Integrating Tree Structures into Self-Attention”. In: (Sept. 2019). arXiv: 1909.06639 [cs.CL].

[60] Jeffrey Watumull et al. “On recursion”. In: *Frontiers in Psychology* 4 (2014), p. 1017.

[61] Gail Weiss, Yoav Goldberg, and Eran Yahav. *On the Practical Computational Power of Finite Precision RNNs for Language Recognition*. Tech. rep. ZSCC: 0000286 arXiv:1805.04908 [cs, stat] type: article. arXiv, May 2018. DOI: 10.48550/arXiv.1805.04908. URL: http://arxiv.org/abs/1805.04908 (visited on 10/04/2023).

[62] Gail Weiss, Yoav Goldberg, and Eran Yahav. *Thinking Like Transformers*. Tech. rep. network. 2021.

[63] Thomas Wolf et al. *HuggingFace’s Transformers: State-of-the-art Natural Language Processing*. 2019. DOI: 10.48550/ARXIV.1910.03771.

[64] Shunyu Yao et al. “Self-Attention Networks Can Process Bounded Hierarchical Languages”. In: (May 2021). arXiv: 2105.11115 [cs.CL].