Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Learnability of probabilistic context-free languages using small transformer models

Bachelor's thesis in the course Informatics

Author:

David Rath

Matriculation Number: 7234000

Faculty of Mathematics, Informatics and Natural Sciences
Department of Informatics

First Reviewer & Supervisor: Dr. Michaela Regneri
Second Reviewer: Prof. Dr. Sören Laue

Hamburg, 23.10.2024

Abstract

This thesis investigates the relationship between memorization, generalization, and creativity in GPT-2 language models trained on artificial languages generated by probabilistic context-free grammars (PCFGs). Two PCFGs of differing complexities were used to generate synthetic corpora for training GPT-2 models with varying architectural complexities and data sizes.

By evaluating the models' ability to infill missing suffixes in sequences from the training data, we observed that both overfitting and underfitting can produce outputs that superficially resemble creative generation. Models often generated novel sequences not present in the training data; however, we argue that these sequences lacked genuine creativity, as they resulted from simple recombinations of memorized patterns. Merely verifying the absence of generated sequences in the training data is insufficient for assessing creativity.

We highlight the necessity of additional metrics to accurately evaluate creativity in language models. Analyzing the diversity and distribution of generated sequences—such as through suffix entropy and n-gram distributions—can be a start. Neither excessively complex models with vast amounts of data nor overly simplistic models with limited data optimally foster creativity. Achieving a balance in model complexity and training data is essential for models to learn underlying grammatical structures without defaulting to memorization or oversimplification. This study advances our understanding of language model behavior in regards to memorization and generalization and provides a foundation for developing more nuanced evaluation methods for memorization, generalization and creativity in AI.

Contents

1 Introduction

1.1 Motivation

Artificial intelligence (AI) has transformed modern technology, and driven innovation across many industries such as healthcare, finance, and transportation. Specifically two sub fields of AI have proven themselves capable of a wide variety of tasks: machine learning (ML) and deep learning (DL). Machine learning, a subset of artificial intelligence, involves the development of algorithms that enable computers to learn from and make decisions based on data. Deep learning, a further subset of ML, relates to the usage of neural networks to model complex patterns in data. Recently, Generative AI (Gen AI), a subset of deep learning, has gained significant attention for its ability to generate novel content such as text, images, and music (see Figure 1).

Unlike "traditional" programs that rely on hard-coded rules, machine learning algorithms learn from data by identifying complex patterns which enables them to handle tasks that are difficult to program explicitly. This process, referred to as training or fitting, involves adjusting the models internal parameters so that they more accurately represent the patterns found in the training data. A well-trained model should then be able to generalize from its data to make predictions about new, previously unseen data.

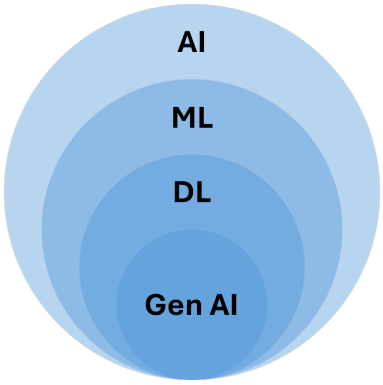

Figure 1: AI hierarchy

The remarkable effectiveness of machine learning and deep learning models in modeling complex data has led to their widespread use across various fields. In healthcare, AI assists in disease diagnosis by analyzing medical images and achieves dermatologist-level accuracy in skin cancer classification [1]. In fraud detection, machine learning algorithms enhance security by identifying anomalies indicative of fraudulent transactions [2]. Additionally, AI recommendation algorithms are able to provide personalized content suggestions on platforms like YouTube and Amazon [3, 4].

Deep learning, a further subset of machine learning, utilizes neural networks to model complex patterns in data. Neural networks are powerful computational models capable of approximating any continuous function, as established by the universal approximation

theorem [5]. This ability allows deep learning models to excel in tasks involving large-scale, complex data, such as image recognition, natural language processing, and autonomous driving.

Although these technologies demonstrate significant advancements, they are not without downsides. In their current state, generative AI models are prone to *hallucinations*, a phenomenon where models generate plausible, but factually incorrect or nonsensical outputs. This issue is especially relevant in deep learning due to its complex and often opaque model structure. Additionally, the performance and behaviour of AI systems can suffer from *biases*, should their training data be skewed. As an example, Buolamwini et al. reported that facial recognition systems from major companies had significantly higher error rates for darker-skinned and female faces, as their training data was mostly comprised of images of lighter-skinned and male faces [6].

Natural language processing (NLP) is a field that bridges computer science and linguistics, enabling computers to understand, interpret, and generate human language. Applications of NLP include machine translation, virtual assistants like Siri and Alexa, sentiment analysis in social media, and automated summarization of documents. This field has grown significantly with the advancements in AI, particularly through the development of models that are able to understand context, nuance, and the subtleties of language. As a result, we have seen substantial improvements in machine translation, sophisticated chatbots, sentiment analysis, and speech recognition technologies.

Large language models (LLMs) have become a cornerstone of NLP due to their ability to generate human-like text. After being trained on extensive corpora of text data, these models are able to produce coherent and contextually relevant responses across a wide range of topics. For example, OpenAI's GPT-3 model was trained on a dataset containing approximately 570 gigabytes of filtered text, which was comprised of 499 billion tokens sourced from books, articles, and web texts [7]. LLMs are able to generalize from their training data, which enables them to generate novel and relevant content that extends beyond the specific examples they have seen. LLMs have been instrumental in advancing research in applications such as conversational agents, automated content creation, and language translation. Despite their capabilities, LLMs are often considered "black boxes" because their internal decision-making processes are not fully understood. This lack of interpretability poses challenges in understanding how they generate outputs, particularly when they produce hallucinations. These

challenges are studied within the field of explainable AI (XAI) and AI interpretability (AIX), which aim to make AI systems more transparent and understandable.

In our work, we aim to study the relationship between memorization, generalization, and creativity in language models, with a focus on how these aspects manifest in models trained on artificial languages generated by probabilistic context-free grammars (PCFGs). By controlling the parameters of these grammars, we can analyze under which conditions models memorize specific patterns or generalize to produce novel outputs.

1.2 Experiment setup

Given the challenges associated with memorization and generalization in LLMs, we aim to explore how models transition between these modes and under what conditions creativity emerges. We design an experiment with the following structure: start with two PCFGs of low, but slightly differing complexity levels. Using these grammars, we generate large corpora of random sequences, which are subsequently used to train GPT-2 [8] models with architectures of varying complexities. We selected the GPT-2 architecture for its widespread adoption in research to make our findings comparable across various studies. Our experiment condition is chosen in a way as to specifically encourage overfitting, as research has shown that overfitting increases memorization [9]. Additionally, both a locally and a globally optimal trigram model are evaluated to serve as baseline comparisons.

After the training process is complete, models are evaluated by infilling the last characters of sequences from the training data. As sequences are drawn from the training data, models can produce valid continuations by either generating suffixes verbatim from their training data, or generate new sequences not before seen. This evaluation method allows models to either rely either on memorization or generalization to complete the sequences.

Our findings include:

- All GPT-2 models were able to learn the languages and consistently produce valid sequences.
- Both underfitted and overfitted models tended to generate sequences that, while valid according to the grammar, lacked diversity and originality. Overfitted models memorized frequent patterns and produced repetitive outputs, whereas underfitted models

defaulted to simple, common structures due to their limited capacity. This similarity in outputs makes it challenging to distinguish genuine creativity from mere memorization.

- Relying solely on whether a generated sequence is absent from the training data is insufficient to assess creativity. Models can produce novel sequences that are simply recombinations of memorized n-grams or frequent patterns. We argue that to obtain a clearer picture of a model's creativity, it is important to also analyze the diversity of generated sequences using metrics like suffix entropy or n-gram distribution.

- Our results indicate that neither overly complex models with excessive data (which may overfit) nor overly simple models with limited data (which may underfit) are optimal for enabling genuine creativity. An appropriate balance is necessary to allow models to learn the underlying structures without merely memorizing or defaulting to simple patterns.

The remainder of this thesis is structured as follows. In Section 2, we provide background on formal languages, context-free grammars, and language models. Section 3 details our experimental setup, including the construction of grammars, model training evaluation procedures. In Section 4, we present the results of our experiments, followed by a discussion in Section 5. Finally, Section 6 summarizes our findings and suggests directions for future research.

2 Background

This chapter presents the theoretical background on which the background of our research is based. We will start with an introduction of formal languages and context-free grammars, followed by an overview of the methods of natural language processing and language modeling we utilize.

2.1 Formal Languages and Grammars

2.1.1 Formal Languages

Formal languages are mathematical abstractions used to describe sets of sequences of symbols of an alphabet. These languages allow for the definition of rules, known as grammars, which determine whether a given sequence of symbols (called a word) is valid within the language.

The atomic components of formal languages are single arbitrary *symbols*. A finite, nonempty set of symbols is called an *alphabet*, and will here be denoted by the letter Σ. *Words* can be constructed by concatenating a finite number of symbols of some alphabet. For a given alphabet Σ, we denote the set of all words that can be constructed from it as Σ^* (where $*$ denotes the Kleene-star operation) [10].

As an example, let $\Sigma_{abc} = \{a, b, \ldots, z\}$ be an alphabet whose symbols are the lowercase english alphabetical characters. The word *dog* is an element of Σ^*_{abc}, whereas 123 is not.

Any subset $L \subseteq \Sigma^*$ is called a *language*. Languages are called empty, if they contain no words, finite if they contains a finite number of words or infinite, if they contain an infinite amount of words.

We denote that a word w is member of a language L as $w \in L$. Deciding whether a word is a member of some particular language by comparing it to a list of known words is impractical for large languages and impossible for infinite languages. Instead, *formal languages* allow us to define a set of rules or procedures, to decide whether or not a word is contained in the language. For a word w to belong to a language L, it must both be made up of only symbols of Ls alphabet Σ and be well-formed. Well-formed means, that w conforms to the rules of the language.

As an example, let L be the language of even-length strings over our previously defined alphabet Σ_{abc}. The word "hello" would not be a member of L as it has an odd length, but

"hi" would be a member. Similarly, "1234" would not be a member of the language as it is made up of symbols outside of Σ_{abc}.

2.1.2 The Chomsky Hierarchy

Formal languages can be categorized into different classes depending on their complexity as defined by the Chomsky hierarchy. The Chomsky hierarchy, originally introduced in the paper *Three Models for the Description of Language* [11], consists of four classes (See Figure 2):

1. **Regular languages** are the simplest class and can be recognized by finite automata. These are limited to patterns expressible by regular expressions.
2. **Context-free languages** are more expressive and can be generated by context-free grammars (CFGs). They can describe nested structures which makes them suitable for approximating the syntax of natural language.
3. **Context-sensitive languages** are more powerful than context-free languages and are defined by context-sensitive grammars. These languages can be recognized by linear bounded automata, a type of non-deterministic Turing machine with limited memory.
4. **Recursively enumerable languages** are the most general class of formal languages. They can be described by Turing machines and encompass all computable languages.

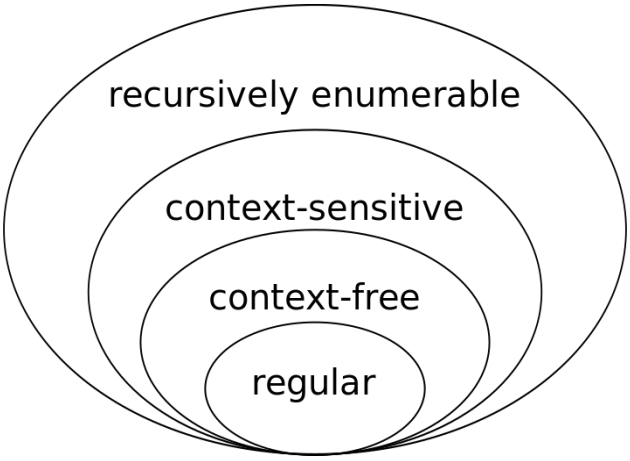

Figure 2: Set inclusions of languages as described by the Chomsky hierarchy

The system of rules governing a formal language is referred to as a *grammar*, and can be described using different formalisms, each modeling decision procedures in various ways. The principal methods for defining grammars include state or pushdown automata, production rules, Turing machines, or other algorithmic decision procedures. The choice of formalism has a significant impact, as they vary in terms of complexity and the classes of languages they can recognize. For instance, the language $L = \{a^n b^n | n > 0\}$ can be recognized by a pushdown automaton, but not by finite-state automata.

This work focuses specifically on context-free languages, and so further discussion of other formal language types falls outside the scope of this analysis.

2.1.3 Probabilistic Context-free Grammars

We continue with an explanation of context-free languages. Context-free languages are languages, which can be generated by a context-free grammar (CFG). Formally, a context-free grammar is a 4-tuple $G = (V, \Sigma, R, S)$, where:

- V is a set of nonterminal characters
- Σ is a set of terminal characters (analogous to the previously defined alphabet)
- R is a set of production rules, relations between nonterminal and terminal characters of the form $V \times (V \cup \Sigma)^*$ (usually written as $V \to (V \cup \Sigma)^*$)
- S is the start symbol ($S \in V$)

To more easily differentiate between nonterminal and terminal characters, nonterminal characters are usually written as capitalized letters, and terminal characters as lowercase letters.

Production rules are defined as pairs of the form (α, β) with $\alpha \in V, \beta \in (V \cup \Sigma)^*$ and specify, how nonterminal characters can be replaced by other nonterminal or terminal characters. Here, α is called the *left hand side variable*, and β the *right hand side variable*. To apply a rule $r \in R$ to a word $w \in (V \cup \Sigma)^*$, replace an occurence of the left hand side variable with the right hand side variable(s). Let $u, v \in (V \cup \Sigma)^*$. If there exists a rule $r \in R$, that when applied to u yields v, we write $u \Rightarrow v$. If v can be produced by a chain of applications of rules to u, we write $u \overset{*}{\Rightarrow} v$. As an example, let $w = AbAb$ and $r = (A, ba)$

(or in standard notation: $A \to ba$). Applying r to w yields the word *babbab*. This can also be written as:

$$AbbAb \Rightarrow babbbab$$

A word w is a member of a context-free language L, if $S \overset{*}{\Rightarrow} w$, meaning it can be produced by repeatedly applying production rules to the start symbol S. Alternatively we will say, that w is *valid* in L.

Probabilistic context-free grammars (PCFGs) are an extension of context-free grammars that allow us to assign probabilities to specific production rules. Formally, a PCFG is defined by the 4-tuple $G = (V, \Sigma, R, S, P)$. The difference to CFGs is the element P, which describes a corresponding set of probabilities on the production rules R, with the constraint that the sum of probabilities of a single left-hand side nonterminal must add up to 1. A rule (α, β) with probability p is written here as $\alpha \to \beta$ $[p]$. These probabilities enable a new sampling strategy for generating strings from the grammar: when expanding a nonterminal symbol during the derivation process, we select which production rule to apply by sampling according to the assigned probabilities of the rules [12].

Context-Free Grammars have been used in modeling the syntactic structures of natural languages [11]. While natural languages are not strictly context-free, CFGs and their probabilistic extensions (PCFGs) provide an approximation for many linguistic constructs. By incorporating probabilities, PCFGs can model the variability and ambiguity inherent in natural language.

The following is example of the concepts relating to PCFGs that are important in this work. Let $G_{MIU} = (V, \Sigma, R, S, P)$ be a PCFG with the following parameters (note that we do not explicitly mention P, but instead denote the probabilities directly at the rules of R):

- $V = \{M, I, U\}$
- $\Sigma = \{m, i, u\}$
- $R = \left\{ \begin{array}{ll} M \to m & [0.5] \\ M \to MIU & [0.5] \\ I \to i & [0.8] \\ I \to UI & [0.2] \\ U \to u & [0.7] \\ U \to UU & [0.3] \end{array} \right\}$

- $S = M$

We can now produce words that belong to L_{MIU}, the language of G_{MIU}, by starting with the nonterminal M, and iteratively expanding a random nonterminal by sampling and applying one of its production rules until the word consists only of terminal characters from Σ:

$$M \Rightarrow \underline{m}$$
$$M \Rightarrow \underline{MIU} \Rightarrow MI\underline{u} \Rightarrow \underline{m}Iu \Rightarrow m\underline{i}u$$
$$M \Rightarrow \underline{MIU} \Rightarrow \underline{m}IU \Rightarrow m\underline{UI}U \Rightarrow m\underline{u}IU \Rightarrow mu\underline{i}U \Rightarrow mui\underline{UU} \Rightarrow muiu\underline{U} \Rightarrow muiu\underline{u}$$

The final derivation sequence can also be expressed as $M \overset{*}{\Rightarrow} muiuu$ or $muiuu \in L_{MIU}$.

2.2 Machine learning

Machine learning is a subdiscipline of artificial intelligence that focuses on developing statistical models that allow machines to learn from data and make predictions without being explicitly programmed for their tasks. Today, machine learning is already used in a wide range of tasks, including generating media like text or images, offering optimal recommendations of products to users, or self driving cars.

In this work, we focus on *supervised learning*, a category of machine learning where models are trained on labeled datasets. The training data consists of entries of the form $(x_1, ..., x_n, y)$, where $x = (x_1, ..., x_n)$ is called the *feature vector*, and y the *label*. The goal is to have the model learn a function $f(x_1, ..., x_n) = \hat{y}$ that approximates the relationsship between the input and the labels (assuming that such a relation exists). As an example, x_1 could represent the price, x_2 the size and x_3 the color of a car and y represent whether or not the user likes the specific car. A properly trained model should be able to predict whether the user likes a car, based only on these three values.

During training, machine learning models learn by minimizing a *loss function*, such as cross-entropy loss for classification tasks or mean squared error for regression tasks. The loss function quantifies the difference between the predicted output $\hat{y}$ and the expected label y, and serves as an objective measure to guide the optimization of the model's parameters through algorithms like gradient descent.

2.2.1 Neural Networks

Neural networks are one of the most commonly used architectures in machine learning due to their ability to approximate any function, as established by the universal approximation theorem [5].

The architecture of a neural network typically comprises three main components: the input layer, the hidden layers, and the output layer. The input layer receives the initial feature vector x, which are then propagated and transformed through the hidden layers and finally given to the output layer, which yields the prediction $\hat{y}$. Each hidden layer consists of neurons interconnected with those in adjacent layers. Neurons receive weighted inputs from the preceding layer, perform computations on those inputs, and transmit them to subsequent layers.

Let N be a neuron in hidden layer k which is connected to the outputs $x_1, ..., x_n$ of neurons from the preceeding layer $k-1$. Also, let σ be an unary function (called *activation function*) and $w_1, ..., w_n, b \in \mathbb{R}$ (called *parameters*). The computation performed by neuron N involves summing the weighted inputs from the preceding layer, applying the activation function, and adding a bias term b:

$$\sigma(x_1 \cdot w_1 + x_2 \cdot w_2 + ... + x_n \cdot w_n + b)$$

The resulting value is then transmitted to neurons in the subsequent layer $k+1$. The parameters m_i and b are initialized with random values and optimized during the training process in a way that minimizes the models loss function. During training, the model generates predictions for data points in the training data and subsequently adjusts its parameters in such a way that aligns its predictions $\hat{y}$ closer to the correct label y. This process is implemented by *gradient descent* and *backpropagation*, which compute the gradients of the loss function with respect to the model parameters. After computing the gradients, parameters are updated according to the following formula:

$$\text{New Parameter} = \text{Old Parameter} - \text{Learning Rate} \cdot \text{Gradient}$$

The learning rate is a hyperparameter of the model architecture that controls the magnitude of parameter updates during the training process. Essentially, it determines the step size taken along the gradient direction to minimize the loss function. A higher learning rate

leads to larger parameter updates, which can accelerate convergence but may risk overshooting the optimal solution. A lower learning rate results in smaller updates, which leads to a more stable but slower convergence process.

2.2.2 Embeddings

An embedding is a mapping from discrete data (such as words, tokens, images, etc.) to continuous vector representations $v \in \mathbb{R}^n$, called *embeddings*, which represents the semantic information of the token in an abstract format usable by machine learning models [13, 14]. The vector space of all possible embeddings $\mathbb{R}^n$ is commonly refered to as the *latent space*.

Tokens are able to be embedded with regards to different contextual information. In the context of natural language, embedding is done with respect to the context in which a specific word or token occurs, so that words that occur in similar contexts should be transformed into similar vector representations. As an example, the word *"King"* should be mapped to a similar vector as "Queen".

There are various ways to measure the similarity between two embeddings $e = (e_1, ..., e_n)$ and $f = (f_1, ..., f_n) \in \mathbb{R}^n$, such as cosine similarity, Euclidean distance, and the dot product. Cosine similarity, for instance, measures the cosine of the angle between two vectors and provides a normalized similarity score that accounts for differences in vector magnitudes. However, in the context of this work, we focus on the dot product as it is fundamental to the attention mechanism.

Exactly how similar two embeddings are can be measured by calculating their dot product. This computation sums the pairwise products of corresponding elements in the embeddings, which effectively measures their alignment. A large dot product indicates a high alignment between embeddings, which suggests some kind of similarity in the semantic meanings they represent. The dot product is calculated as follows:

$$e \cdot f = \sum_{i=0}^{n} e_i \cdot f_i = (e_1 f_1 + e_2 f_2 + ... + e_n f_n)$$

As an example, let e_1 be the context embedding of "King", e_2 of "Queen" and e_3 of "Broccoli". Then we expect the dot product $e_1 \cdot e_2$ to be comparatively large due to the semantic similarity between "king" and "queen", while $e_1 \cdot e_3$ or $e_2 \cdot e_3$ should be relatively small.

2.3 NLP and Language Models

Natural language processing (NLP) is a field of artificial intelligence which aims to enable computers to understand and generate human language in a meaningful way. A natural language is any language commonly spoken by humans, as for instance english or german. NLP methods are applied in a broad range of tasks, including but not limited to speech recognition and understanding, dialogue systems, lexical analysis, parsing, question answering, sentiment analysis, and more.

This field is inherently challenging due to the ambiguity and variability of natural languages. For example, words can have multiple meanings depending on context (polysemy), and grammatical structures can vary widely between languages. Recently, the study of language models has become a significant area within NLP.

2.3.1 Language Models

A *language model* is a statistical model that, given an input sequence of tokens, generates a probability distribution over all possible subsequent tokens. A *token* denotes the smallest unit of text that the model processes, as defined by the model's specific tokenizer. For a given model, the tokenizer determines exactly how text is broken down into tokens, which remain consistent throughout the model's operation. Tokens can be individual characters, whole words, or more commonly, subword units that capture meaningful segments of words, depending on the model's design. When using PCFGs in this work, we use their terminal symbols as tokens because they are the smallest elements of their language.

Because language models predict the likelihood of subsequent tokens of a token sequence, they can also be used to generate new token sequences by iteratively sampling tokens and appending them to the input sequence. This is called the *decoding strategy*, and a common choice is greedy decoding, which always selects the next best token with the highest predicted probability.

2.3.2 N-gram Models

N-grams are a concept of statistical language modeling and describe fixed-length sequences of n tokens, usually extracted from a corpus of text. N-grams are used in various NLP tasks, including language modeling, text generation, and information retrieval.

N-grams can be extracted for texts by counting the occurences of fixed-length sequences with a sliding window of size n. Often, sequences are padded with special tokens which allows n-grams to model the beginnings and endings of sequences as well.

N-gram models are language models that build on n-grams and generate the probability distribution of tokens given the last $n-1$ tokens of the input sequence. The higher the n, the more context the model can take in and the more complex it is [15]. As with other language models, different decoding strategies exist for n-gram models as well, with the most common one being greedy decoding. In this work, we will implement two n-gram models: one using greedy decoding and another employing an optimal decoding strategy as a baseline for comparison.

Trigram	**Frequency**
##a	1
#aa	1
aaa	3
aab	1
abb	1
bbb	2
bb#	1
b##	1

Table 1: Character-level trigrams of “aaaaabbbb”

Table 1 shows an example for the character-level trigrams extracted from the word “aaaaabbbb” which has been padded with the token “#” [15]. A greedy trigram model (or 3-gram model) that was trained on the trigram frequencies seen in Table 1 and asked to predict the continuation of “aa” would pick the token “a” instead of “b”, as the trigram “aaa” is more common than “‘aab”.

2.3.3 Transformers

Transformers are a type of language model based on deep learning architectures that generate a probability distribution of tokens based on all tokens in their input sequence up to a maximum context window size. Introduced by Vaswani et al. in 2017 [16], transformers have become a significant area of research in the field of deep learning, particularly in natural language processing. By attending to all tokens in the input sequence simultaneously (within the context window), transformers can capture long-range dependencies and contextual relationships more effectively than previous models. This chapter will discuss the basics of their architecture, functionality, and how transformers are utilized in this work.

A central component of the transformer architecture is the *self-attention mechanism*, which allows the model to weigh the importance of different parts of the input sequence and compute new representations of each token based on weighted combinations of all tokens

in the sequence. By attending to all tokens simultaneously within its context window, the transformer can capture relationships between tokens regardless of their positions.

First, each token t_i is embedded with its base context information and then enriched with its positional encoding, which yields the vector $E_i \in \mathbb{R}^d$, where d is the dimensionality of the embedding space. Next, the *query*, *key*, and *value* embeddings are computed for each token by projecting the embeddings E_i using learned linear transformations:

$$Q = EW^Q, \quad K = EW^K, \quad V = EW^V$$

where W^Q, W^K, and W^V are the trainable projection matrices, and Q, K, and V are matrices containing the query, key, and value vectors for all tokens in the sequence. These projection matrices are optimized during the training process to most accurately predict the next tokens in their training sequences. W^Q and W^K are of size $d \times d_k$ and W^V of size $d \times d_v$.

The key and query embeddings Q and K, which hold the base context information as well as the positional encoding, can then be used to calculate how much one token t_i should relate to another token t_j, by calculating the dot product $Q_i \cdot K_j$. Next, the result is divided by $sqrt(d_k)$ to prevent the dot products from growing too large in magnitude when d_k is large, which can prevent small gradients during backpropagation. We define the attention score a_ij as:

$$a_{ij} = \frac{Q_i K_j^\top}{\sqrt{d_k}}$$

As the dot product does not inherently produce probabilities but instead unbounded logits, a softmax function is applied so that the sum of the influence weights across all tokens for a given token sums to one and can be used as a probability distribution [17]. The softmax function is a function that converts a vector of real numbers into a probability distribution. As an example, let $X = x_1, ..., x_n$ be a vector of logits. Then the softmax of $x_i \in X$ can be calculated as follows:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{n} e^{x_k}}$$

The attention weight $f(i, j)$ by which a token t_i should be influenced by another token t_j, can be calculated with the following function:

$$f(i,j) = \frac{\exp(a_{ij})}{\Sigma_{k=1}^{n} \exp(a_{ik})}$$

If $f(i,j)$ is large, we know that token j has a strong relation to token i. It should be noted that this formula is not symmetric and does not show any relation from i to j (for which we would have to calculate $f(j,i)$).

In practice, the vector of a single embedding is adjusted by all other embeddings in the sequence at the same time. The following formula shows how an embedding E_i is transformed by the tokens in the sequence:

$$E_i^{\text{new}} = E_i + \sum_{j=1}^{n} f(i,j) V_j$$

As an example, let the S be the sentence *"The blue dog and red cat are playing"*. In this example, let the words $w_1, ..., w_8$ of S also be its tokens $t_1, ..., t_8$. Transforming each w_i into its embedding E_i captures only the inherent information of each word, but none of the important context between them.

To determine how much the token *dog* (t_3) should be influenced by the token *blue* (t_2), we compute the attention weight $f(3,2)$:

$$f(3,2) = \frac{\exp(a_{32})}{\sum_{k=1}^{8} \exp(a_{3k})}$$

After calculating the attention weights for all pairs of tokens, we can compute the new embedding for *dog* as follows:

$$E_3^{\text{new}} = E_3 + \sum_{j=1}^{8} f(3,j) V_j$$

Following this procedure and iteratively adjusting every embedding by every other embedding, should thus encode a deeper semantic context in each embedding.

One iteration of adjusting embeddings after the previously mentioned formula, is called *scaled dot-product attention* and encapsulated in a single layer. *Multi-Head Attention* describes the process of running multiple independent scaled dot-product attentions in parallel, called *heads*. Each head possesses its own projection matrices W^Q, W^K and W^V, which allows heads to specialize on different parts of the latent space or the input sequence. Figure

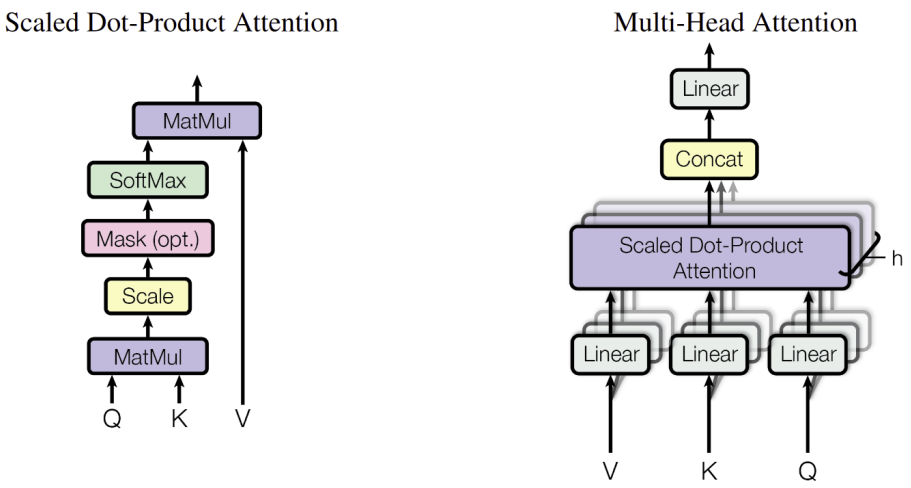

Figure 3: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel. The Figure is taken from "Attention Is All You Need" [16]

3 shows the structure of the scaled dot-product attention and multi-head attention layers.

Ater computing the multi-head attention, the outputs are passed through a fully connected feed-forward network (FFN):

$$FFN(E_i) = max(0, E_i W_1 + b_1)W_2 + b_2$$

where W_1, W_2 are weight matrices, and b_1, b_2 are bias vectors. This allows the model to capture non-linear relationships and further transform the representations. The procedure of calculating the multi-head attention and passing the outputs through the FFN is repeated N times (with N typically being 6 or 12).

The full transformer architecture consists of two main components: an *encoder* and a *decoder*, both composed of multiple layers that utilize the attention mechanism. Figure 4 shows a simplified layer structure of the transfomer model, with the inputs on the bottom and the output probabilities on the top. In the original Transformer model introduced by Vaswani et al. [16], the encoder processes the input sequence to generate a set of continuous representations that capture rich contextual information. The decoder then generates the output sequence by attending to both the encoder's output and the previously generated tokens, which makes it applicable for sequence-to-sequence tasks such as machine translation,

where the input and output sequences are different.

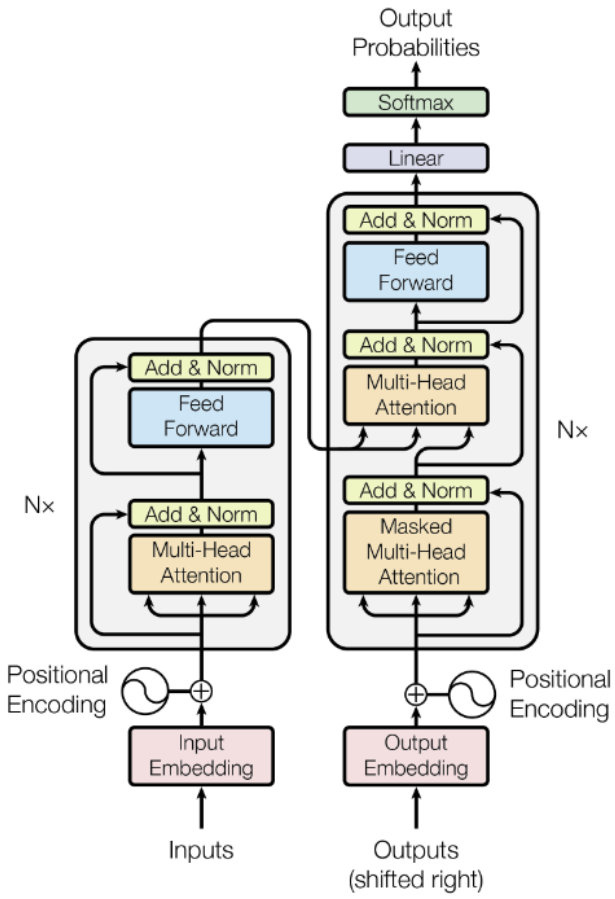

Figure 4: The Transformer-model architecture. The Figure is taken from "Attention Is All You Need" [16]

The encoder receives the input token sequence, for instance, a sentence in English. Each token is transformed into its base context embedding, and a positional encoding is added to provide the model with sequence order information, since the self-attention mechanism is inherently order-invariant [16]. Afterwards, the multi head attention is computed, which yields a sequence of input embeddings $E = [E_1, E_2, \ldots, E_n]$ (where $E_i \in \mathbb{R}^d$). Additionally, the architecture uses residual connections and layer normalization after each attention and FFN sub-layer. Residual connections facilitate gradient flow and mitigate issues like the vanishing gradient problem, while layer normalization maintains consistent data distributions across layers. Together, these techniques help accelerating the training speed and improving

its stability.

The output of the encoder is a sequence of embeddings $E = (E_1, E_2, ..., E_n)$, which now contain information not only about the individual tokens, but also about their relationssships to each other within the sequence.

The decoder in the transformer architecture is designed to generate an output sequence (e.g., a translated sentence in natural language) by attending to both the previously generated tokens and the output of the encoder. While its structure is similar to the encoder, the decoder possesses two differences:

Firstly, the decoder employs *masked* multi-head self-attention. In addition to the regular self-attention we discussed, this layer masks future tokens during training so that each token in the output sequence can only attend to earlier tokens. This prevents the model from "looking ahead" during training, and forces the model to base its generation on only the previous tokens.

Secondly, the decoder incorporates an encoder-decoder attention layer. This layer allows the decoder to integrate contextual information from the entire input sequence while generating each output token.

2.3.4 GPT Models

Generative Pre-trained Transformer (GPT) models are a family of transformer language models based on the transformer architecture, developed by OpenAI [18]. GPT models implement the transformer decoder to perform language modeling tasks, and already achieve impressive results.

GPT-2 in particular is an improved version of the original GPT model with a larger architecture of 1.5B parameters [8]. talk about the results of it and why we use it

In our experiments, we train GPT-2 models from scratch on sequences generated by probabilistic context-free grammars (PCFGs). By adjusting the model complexity and the amount of training data, we investigate how these factors influence the model's ability to learn grammatical structures, memorize training data, and generate novel sequences.

2.3.5 Memorization and Generalization in LLMs

As large language models (LLMs) generate text, their behavior can broadly be divided into two modes: *memorization* and *generalization*. In memorization, models repeat sequences

that were directly contained in their training data. In generalization, models produce new sequences based on learned patterns. While generalization allows models to create novel sequences, it also carries the risk of producing incorrect information, often referred to as *hallucinations*. Therefore, sequences produced through generalization can be further categorized into factual, desired outputs (referred to here as creative sequences) and unfactual, unwanted hallucinations (see Figure 5).

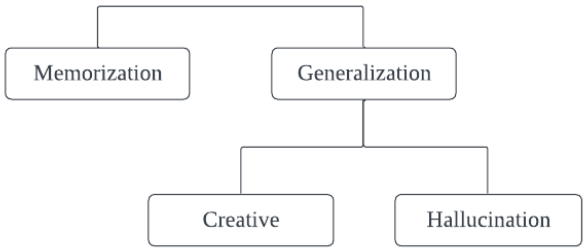

Figure 5: The hierarchy between text produced by memorization and generalization, and hallucinations

Memorization poses significant privacy risks. If the training data includes sensitive personal information or copyrighted material, the model may inadvertently leak this data. Recent research has shown that membership inference or training data extraction attacks can be effective against LLMs, enabling attackers to retrieve specific information about a model's training data by strategically querying it [19, 20]. For example, Carlini et al. demonstrated that attackers could extract verbatim pieces of training data from GPT-2, including personally identifiable information such as names, phone numbers, email addresses, IRC conversations, code snippets, and even 128-bit UUIDs [9]. Although model extraction and parameter extraction attacks have received limited research attention, these findings highlight the potential for significant privacy breaches [19].

On the other hand, generalization is not directly constrained by the training data and can produce a diverse variety of novel sequences. Nevertheless, generalization carries the inherent risk of generating unwanted hallucinations, controlling which remains a complex challenge [21]. Since LLMs generate content based on learned statistical patterns rather than explicit knowledge representations, they may lack the ability to verify the factual accuracy of their outputs. Techniques such as incorporating factual verification mechanisms, improving training data quality, or using reinforcement learning from human feedback have

been explored to address this issue [22].

2.4 Related Works

The themes of memorization, generalization, and creativity in LLMs have been extensively explored in recent literature. Understanding how these phenomena interact is crucial for developing models that can produce innovative and reliable outputs while minimizing risks such as data leakage and hallucinations.

Similar to our experiment, Carlini et al. analyzed the memorization capabilities of LLMs by training GPT-2 models on natural language and prompting them with prefixes from the training data [23]. Their work focuses on data extraction attacks and investigates under what conditions models generate sequences that were directly contained in their training data. They found that memorization increases proportionally with model complexity, sequence repetition in the training data, and the size of the context.

While most experiments relating to memorization are conducted on natural language, some studies have explored the learning of formal languages by neural networks. Allen-Zhu and Li trained GPT-2 models on complex context-free grammars (CFGs) and analyzed the internal parameters to build a model of how transformers approximate and learn the grammar's rules [24]. They found that the inner workings correlate with dynamic programming algorithms, which allow the models to learn even complex CFGs. However, their work does not directly test memorization effects, as the training data is not queryable, and they assume that generating something identical to the training sequence has a low probability and hence does not occur.

To be able to generate novel and relevant sequences, models need to be creative. The notion of creativity in LLMs has been measured with experiments similar to those that measure creativity in humans, such as divergent association or alternative use tasks. Given these tasks, Hubert et al. found that LLMs, specifically GPT-4, had no problems with these tasks and were even significantly more creative to their human counterparts [25]. Chen and Ding obtained similar results and found that temperature scaling allows models to generate more creative outputs, but with the tradeoff of reduced stability [26].

To investigate the relationship between memorization and generalization, Hu et al. conducted a study on syntactic generalization in neural language models [27]. They investigated how these models handle systematic variations in natural language and found that increas-

ing the complexity of the model architecture significantly improves the ability to generalize syntactic structures, more so than simply enlarging the dataset size. This suggests that a more complex model can capture intricate patterns and rules within a language, but this increased capacity also raises the potential for memorization of training data, which can lead to overfitting and reduce the model's ability to generate truly novel outputs.

Our approach differs from prior work by focusing on small languages and small datasets to deliberately induce overfitting, and examining under what conditions models switch from memorization to generalization. This contrasts with previous studies that often use large models and datasets, making it difficult to assess memorization effects directly. By adjusting the complexity of both the models and the languages, we can investigate how these factors influence the balance between memorization and creativity, and contribute to a deeper understanding of the underlying mechanisms in LLMs.

3 Experiment Setup

This section will serve as an overview of the experimental setup of this work. We start by constructing two PCFG grammars of different complexity (Chapter 3.1), from which we generate two corpus of sampled sequences (Chapter 3.2). Afterwards, these corpus ared used as training data for GPT-2 models, one with the default amount of parameters and one with reduced amount, as well as trigram models to be used for baseline comparison (Chapter 3.3). Figure 6 shows the data pipeline of this experiment.

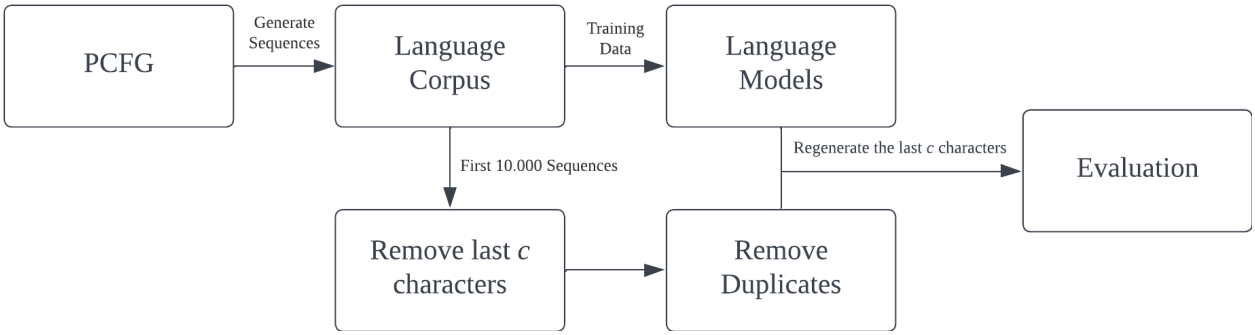

Figure 6: The Data Pipeline of the experiment

3.1 Constructing the Grammars

This subsection will explain and characterize the two PCFGs we use to generate our training data. the PCFGs and the languages they generate are chosen in such a way as to model the characteristics of natural language, and as such our generated corpora possess a long tail distribution of sequences and allow duplicates. Table 2 shows the number of terminals, nonterminals and production rules of each grammar.

Parameter	Simple Grammar	Complex Grammar
Terminals	5	7
Nonterminals	6	11
Production Rules	11	20

Table 2: Statistics of the grammars

3.1.1 Simple Grammar

The simple PCFG is designed to maintain low complexity while remaining sufficiently complex to generate a non-regular language. The complexity arises from the loop between the production rules of nonterminals A and E, which are both able to generate the other. This loop also allows generating sequences of infinite length, which makes the generated language an infinite one. Due to the fact that A can both be transformed into CE as well as EC, sequences are able to expand in both the left and right direction, with expansion to the left being more likely due to the higher probability assigned to that direction. The terminals B, C and D can also be expanded into a diverse set of terminal characters b, c, d and e. Table 3 shows the 10 most common generated words of the simple grammars corpus and their occurences, Figure 7 the production rules of the simple grammar.

Sequence	**Count**
acdccdc	34,023
cdccdc	21,559
acdccdb	14,776
abdccdc	14,701
acdcbdc	14,543
acdbcdc	14,496
bdccdc	9,248
cdcbdc	9,209
cdccdb	9,185
cdbcdc	9,072

Table 3: The 10 most common words of the simple grammar corpus

Simple Grammar:

- S → A [1.0]
- A → 'a' [0.4]
- A → C E [0.1]
- A → E C [0.5]
- B → C D [1.0]
- C → 'b' [0.3]
- C → 'c' [0.7]
- D → 'd' [0.8]
- D → 'e' [0.2]
- E → A B [0.8]
- E → C D [0.2]

Figure 7: The production rules of the simple grammar

3.1.2 Complex Grammar

The complex grammar contains more terminals, nonterminals and production rules than the simple grammar. It contains additional connections between nonterminals which create loops that increase the complexity of the generated language. For example, the nonterminal A now has multiple production paths originating from B, E, and H, which adds to the complexity of potential sequence structures. The loops ensure that the language generated by the complex grammar is infinite. A comparison between the generated corpora also shows how the increased complexity of the grammar is translated into a more diverse set of generated sequences. Table 4 shows the 10 most common generated words of the complex grammars corpus and their occurences, Figure 8 the production rules of the simple grammar.

Sequence	**Count**
aiehh	121,967
iaiiehh	31,113
aifech	21,928
faiehc	21,879
faiehh	19,864
aifehh	19,472
ehaiehh	11,750
fffec	10,980
fffeh	9,724
iiaiiiehh	8,047

Table 4: The 10 most common words of the complex grammar corpus

Complex Grammar:

- S → A [1.0]
- A → B G [0.8]
- A → F J [0.2]
- B → 'e' [0.6]
- B → D B [0.2]
- B → I A [0.2]
- C → 'i' [0.8]
- C → A E [0.2]
- D → 'e' [0.2]
- D → 'f' [0.8]
- E → B A [0.2]
- E → H C [0.8]
- F → D B [1.0]
- G → 'h' [1.0]
- H → 'a' [0.4]
- H → A H [0.2]
- H → C I [0.4]
- I → H C [1.0]
- J → 'c' [0.9]
- J → 'g' [0.1]

Figure 8: The production rules of the complex grammar

3.2 Generating the Datasets

As training data for our language models, we generate datasets of 1,000,000 sequences from each grammar. Sequences are randomly sampled as described in chapter 2.1.3.

Problems arised during generation and evaluation both for very short, and very long sequences. Allowing short sequences lead to the training data being very uniform, being made up of mostly a small number of sequences. As an example, for the simple grammar 40% of all generated sequences were just the letter 'b'. To allow more complexity and variety, we implemented a minimum sequence length requirement of five characters.

Additionally, checking whether a long sequence belongs to the language of a PCFG is computationally expensive, so we limited the sequence length to 50 characters to avoid intractability.

The difference in complexity between the two grammars becomes visible when comparing the corpora they generated. One way to measure this complexity is through *entropy*, which quantifies the unpredictability or randomness of the sequences in the corpus. High entropy indicates a greater diversity of sequences, while low entropy suggests more uniform or predictable patterns. Following Bentz et al. [28], we calculated the entropy $H(C)$ of a corpus C as follows:

$$H(C) = -\sum_{t=1}^{S} p(w_t) \log_2 p(w_t)$$

Here, S is the number of unique sequences and $p(w_i)$ the relative frequency of a single sequence $w_i \in C$.

For the simple grammar, the distribution of sequences tended to be more uniform, which resulted in a lower entropy. The complex grammar exhibited a higher degree of variability in the distribution of generated sequences, which was reflected in a higher entropy. Table 5 shows key properties of both corpora.

Metric	Simple Grammar	Complex Grammar
Total Tokens	9,385,058	13,953,443
Unique Sequences	120,211 (12.02%)	326,764 (32.68%)
Avg. Sequence length	9.39	13.95
Sequence level Entropy	11.40	12.36

Table 5: Properties of the generated corpora

A statistical evaluation of the datasets shows that the word frequency distributions in both corpora adhere to Zipf's law. Zipf's law describes a phenomenon in certain distributions where the rank of a word is inverse to its frequency (i.e. the most frequent word occurs twice as often as the second most frequent, three times as often as the third, and so on). This pattern is commonly observed in natural languages and reflects a long-tail distribution, where a few words are very frequent while many others occur rarely. Formally, Zipf's law can be expressed as $f(n) \propto 1/n^s$, where $f(n)$ is the frequency of the n-th most common word, and s controls the steepness of the distribution [29]. In natural language texts, s is typically close to 1, which indicates a heavy-tailed distribution.

By using linear regression and filtering out words with fewer than 10 occurrences, the derived optimal values for s are 0.74 for the simple grammar corpus and 1.02 for the complex grammar corpus (rounded to two decimal places). Figures 9 and 10 plot the distributions of word frequencies against their ranks.

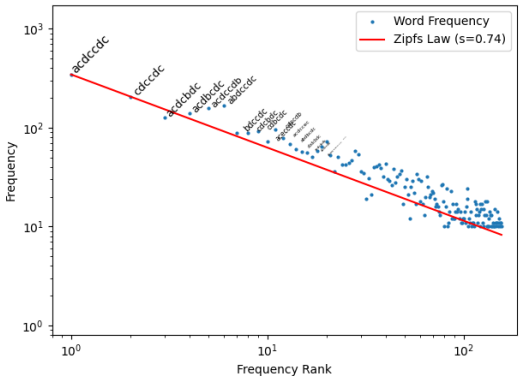

Figure 9: Word frequency distribution on the simple grammar corpus ($s = 0.74$)

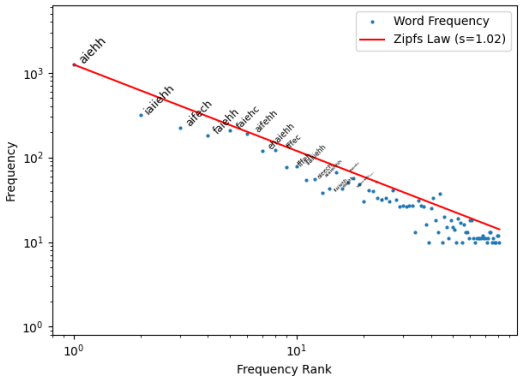

Figure 10: Word frequency distribution on the complex grammar corpus ($s = 1.02$)

The alignment of the distributions with Zipf's law suggests that the constructed corpora mirror statistical properties of natural language. Specifically, the complex grammar corpus closely mirrors natural language distributions with s near 1.0, which we argue reinforces the applicability of using these datasets as models for natural language in our experiments.

3.3 System Overview

This section provides an overview of the system, the tokenization process and the models used. We begin with a description of our custom tokenizer, followed by an outline of the different models we trained and evaluated.

3.3.1 Tokenization

A custom tokenizer was created for the purposes of this project. The vocabulary of the tokenizer consists of a grammars terminals, as well as the special control tokens `<BOS>` (used to mark the beginning of a sequence), `<EOS>` (used to mark the end of a sequence) and `<PAD>` (used to pad sequences in a batch to equal length).

A sequence w in a batch B is encoded as follows:

1. Prepend a `<BOS>` token to the start of the sequence
2. Append an `<EOS>` token to the end of the sequence
3. Let w_{max} be the longest sequence in batch B. Pad the sequence with as many `<PAD>` tokens, so that it has exactly $|w_{max}| + 2$ tokens.

Rule three also guarantees, that all sequences in B have the same length before being used for training. As an example, let $w =$ baabb and $|w_{max}| = 7$. Then w has the following form when encoded:

<BOS>baabb<EOS><PAD><PAD>

3.3.2 Model Overview

1. **GPT2-DEF (FULL)**: This model is configured with the default settings of the GPT-2 architecture and trained on the entirety of the training data (`n_embd=768, n_layer=12, n_head=12`).
2. **GPT2-DEF (HALF)**: This model is configured with the default settings of the GPT-2 architecture, but trained on only half of the training data to examine the impact of reduced training data (`n_embd=768, n_layer=12, n_head=12`).

3. **GPT2-MIN (FULL)**: To investigate the effects of model simplification, we trained a GPT-2 instance with reduced complexity on the complete training dataset (`n_embd=192, n_layer=3, n_head=3`).

4. **GPT2-MIN (HALF)**: This model combines reduced model complexity and reduction of training data size (`n_embd=192, n_layer=3, n_head=3`).

5. **TRI-OPT**: We implemented our own version of an optimal trigram model. This model chooses the continuation token sequence of the overall highest probability and can additionally recognize the end of sequences.

6. **TRI-GRE**: A simple trigram model that given a bigram, greedily predicts the next token by choosing the one with the highest relative frequency.

Each architecture was trained on both the simple grammar and complex grammar corpora for three epochs, resulting in a total of 10 trained and evaluated models. We specifically combined simple grammars with a high amount of training data as to artificially induce overfitting.

For the GPT-2 models, we used GPT2LMHeadModels from the huggingface transformer library. Each model was trained for 3 epochs with a batch size of 8. We set context window size for all models to the *length of the longest word in the corpus + 2*, which results in 51 tokens for the models trained on the simple grammars corpus, and 50 tokens for the complex grammars corpus. We add two, as each sequence is padded with at least two control tokens.

3.3.3 Trigram Models

To have a comparable baseline to our GPT-2 models, we implement two trigram models - one that selects subsequent tokens greedily, and one that selects them optimally. As previously mentioned, n-gram language models generate a probability distribution of subsequent tokens based on the previous $n-1$ tokens as context. Let Σ be a vocabulary and $f(x, y, z)$ with $x, y, z \in \Sigma$ be the relative frequency of the trigram xyz. Given a bigram xy, a trigram model predicts the next token in the sequence $z \in \Sigma$ based on its implementation.

Greedy Trigram Model

The greedy trigram model makes greedy choices at each step. Given a bigram xy, the model returns the token z that maximizes the function $f(x, y, z)$.

While this greedy approach is computationally efficient and often produces reasonable results, it may not always yield the most coherent or contextually appropriate sequences. By focusing only on the immediate next token, its possible that the model might miss out on better overall sequence structures that could be achieved by considering longer-term dependencies.

For that reason we present an alternative, non-greedy trigram model. This model extends the standard approach by evaluating the entire chain of trigrams to find the sequence with the highest overall probability.

Optimal Trigram Model

To provide an additional baseline comparison for the evaluation, we implemented a non-greedy trigram model. Unlike the greedy approach, our model continues sequences by choosing the fixed length chain of trigrams with the highest overall probability. Formally, let $w = w_1 w_2 \dots w_n$ be a sequence in a language L with alphabet Σ. To generate a continuation of m tokens $k_1, \dots, k_m \in \Sigma$ we iterate through all possible continuations and choose the one which maximizes the following function:

$$P(w_1, \dots, w_n, k_1, \dots, k_m) = f(w_{n-1}, w_n, k_1) \cdot f(w_n, k_1, k_2) \cdot \dots$$
$$\cdot f(k_{m-2}, k_{m-1}, k_m) \cdot f(k_{m-1}, k_m, \#) \cdot f(k_m, \#, \#)$$

Additionally, we add the frequencies of the trigrams $f(k_{m-1}, k_m, \#)$ and $f(k_m, \#, \#)$ to the product, which allows the model to recognize and adapt to the end of sequences. This implementation allows the model to generate the overall most likely continuation, rather than making locally optimal choices at each step.

It should be noted, that this method is computationally heavy and intractable for larger values of n. The computational complexity required to make a prediction follows $\mathcal{O}(n^{|\Sigma|})$, which scales exponentially with the size of the vocabulary $|\Sigma|$.

Models	Simple Language		Complex Language	
	cdccdccdbcdc	**bbbdbcecdccd**	**fiaiffechaiiehc**	**iehaiiaifechh**
GPT2-DEF (FULL)	cdccdcc<u>dccdc</u>	bbbdbce<u>cdccd</u>	fiaiffechai<u>iehc</u>	iehaiiai<u>ehhha</u>
GPT2-DEF (HALF)	cdccdcc<u>dccdc</u>	bbbdbce<u>cdccd</u>	fiaiffechai<u>iehc</u>	iehaiiai<u>iiieh</u>
GPT2-MIN (FULL)	cdccdcc<u>dccdc</u>	bbbdbce<u>cdccd</u>	fiaiffechai<u>iehh</u>	iehaiiai<u>iiieh</u>
GPT2-MIN (HALF)	cdccdcc<u>dccdc</u>	bbbdbce<u>cdccd</u>	fiaiffechai<u>iehh</u>	iehaiiai<u>iiehh</u>
TRI-OPT	cdccdcc<u>dccdc</u>	bbbdbce<u>cdccd</u>	fiaiffechai<u>iiieh</u>	iehaiiai<u>iiiieh</u>
TRI-GRE	cdccdcc<u>dccdc</u>	bbbdbce<u>ccdcc</u>	fiaiffechai<u>iiiii</u>	iehaiiai<u>iiiii</u>

Exact Matches

Novel Valid Sequences

Invalid Sequences

Table 6: Selected text examples of each model for $c = 5$. Underlined are the suffixes regenerated by the model.

4 Evaluation and Results

This chapter presents the results of our evaluation. We start with an explanation of our methodology and collected metrics, followed by a general interpretation of the behavior of the models with regard to the complexity of the corpus they were trained on. Afterwards, we analyze and discuss the results.

Our experiment design is modeled after the k-extractability approach defined by Carlini et al. In their work, the authors denote the suffix s of a sequence as *k-extractable*, if there exists a (length-k) prefix p, such that the concatenation $[p\|s]$ is contained in the training data for a model f, and f produces s when prompted with p using greedy decoding [23]. Their methodology specifically fixes the length of the prefix p to k characters and allows the length of the suffix s to be variable.

In our approach, we modify the experimental setup by restricting the suffix s to a fixed length c. This change addresses a practical issue observed in preliminary tests, where models tended to generate suffixes of infinite length when left unrestricted.

Our experiment tests each models performance in regenerating a fixed number of removed

characters from the end of sequences taken from their training data. Each evaluation cycle tests a model on $10,000$ unique sequences for a fixed parameter $c \in \mathbb{N}$ ranging from 1 to 10, which controls how many characters are removed from the end of a sequence and regenerated by the model. Given a sequence $w = w_1 w_2 \dots w_n$, we refer to the first $n - c$ characters ($w_1 w_2 \dots w_{n-c}$) as the *prefix* and the last c characters ($w_{n-c+1} w_{n-c+2} \dots w_n$) as the *suffix* of the sequence. Additonally, we call the concatenation of a prefix and a suffix generated by a model a *generated sequence* (even though the prefix remains unchanged).

As an example, let $w =$ `"hello world"` and $c = 3$. Then `"hello wo"` would be the prefix and `"rld"` the suffix of w. In the experiment, models would be tasked to generate a continuation of $c = 3$ characters, given the input sequence `"hello wo"`.

Removing trailing characters from unique sequences can result in identical sequences (i.e. removing two characters from `"aaaaa"` and `"aaabb"` both result in prefix `"aaa"`, a duplicate). Because the behaviour of models is deterministic, this would lead to duplicate results. For that reason, we skip duplicate prefixes. While this limits the total number of evaluated sequences, especially for higher values of c, we still believe that the findings remain robust and representative.

By limiting suffixes to a fixed length c, we ensure that each prefix has at least one corresponding suffix present in the training data (specifically the original suffix). This allows the model to regenerate the original sequence through memorization. Conversely, if the model generalizes beyond memorization, it is able to produce valid suffixes that were not explicitly present in the training data.

We classify a generated sequence w into one of three categories (where C is the corpus generated by grammar G with starting nonterminal S). See Table 6 for selected examples of generated sequences and their classifications. The categories are defined as follows:

- **Exact Matches:** The generated sequence w is identical to a sequence in the training data ($w \in C$).
- **Novel Valid Sequences:** The generated sequence w is not in the training data ($w \notin C$) but is contained in the grammars language ($S \overset{*}{\Rightarrow} x$).
- **Invalid Sequences:** The generated sequence w is not in the training data ($w \notin C$) and is not contained in the grammars language ($S \overset{*}{\not\Rightarrow} w$).

We are specifically interested in the relationship between exact matches and memorization, as well as the relationship between novel valid sequences and generalization and creativity. We hypothesize that a high rate of exact matches is indicative of memorization, while the generation of novel valid sequences indicates the model's ability to generalize beyond its training data.

While measuring the accuracy of the models is not directly feasible, we instead offer two proxy metrics: **validity**, which measures how many generated sequences are able to be produced by the underlying PCFG (see Figure 11, Add. Table 7) and **perplexity**, which measures a models "confidence" in its generation (see Figure 16, Add. Table 12). Additionally, we measure the entropy of the suffixes as an indicator of the variance of a models generated sequences (see Figure 12, Add. Table 8). In the following sections, we will examine the values of these metrics collected during the evaluation and discuss them.

4.1 Validity

The first metric discussed here is the *validity* of the generated sequences (see Figure 11, Add. Table 7). Validity is indicative of a model's effectiveness in producing linguistically and syntactically coherent outputs, and is also important for evaluating generalization, as generalization involves producing sequences that are both valid and novel. Note that sequences classified as exact matches will always be classified as valid.

As a baseline we consider the probability that a randomly generated sequence would be valid. This probability is proportional by the number of terminals of the PCFG $|\Sigma|$ and the length of the suffix c. Specifically, for any given prefix, the minimal probability that a sequence adheres to the PCFG is $\frac{1}{\Sigma^c}$, where Σ^c represents all possible suffixes that could follow the prefix.

Models trained on the simple grammar are able to generate valid suffixes for almost any given prefix. While these values are mostly homogeneous, it is worth noting that the GPT2-MIN model trained on half of the data shows a marginally higher performance compared to the model trained on the full data. However, given that the difference is negligible (a difference of at most 0.04%), it is likely a result of statistical noise rather than a meaningful trend.

The results of the complex grammar display a more varied distribution and we find that here, the validity of the generated sequences is proportional to the training data size and

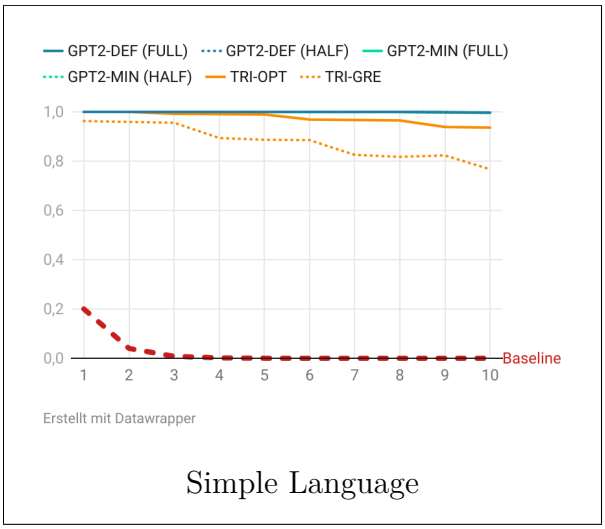

Figure 11: The percentage of generated valid sequences

model capacity. While the trigram models perform comparably to the GPT-2 models on the simple grammar, their performance declines when confronted with the complex grammar. As trigram models have a context size of 2 tokens, they can accurately predict suffixes up to a suffix length of 2, but incur a sharp drop in validity when tasked to generate longer suffixes. This suggests that the trigram models are underfitting and have problems to capture the more complex patterns of the grammar, likely due to their limited model capacity.

Compared to the trigram models, the GPT-2 models trained on the complex language demonstrate a noticeable improvement in generating valid sequences. While the performance declined for larger values of c, the GPT-2 models consistently outperformed the trigram models, particularly for longer sequences that required a deeper understanding of the underlying structure.

4.2 Suffix Entropy

The analysis of suffix entropy provides insights into the variability of the suffixes generated by the models compared to the training data. We calculate the suffix entropy of a corpus or model by averaging the entropy $H(s)$ across all suffixes s.

A model that captures the complexity of the language should generate suffixes with an entropy distribution similar to that of the training data, as this would indicate that the

model is producing a wide range of valid suffixes rather than over-relying on a limited set of patterns. Therefore, we expect the suffix entropy of the generated sequences to be correlated with that of the training data (see Figure 12, Add. Table 8).

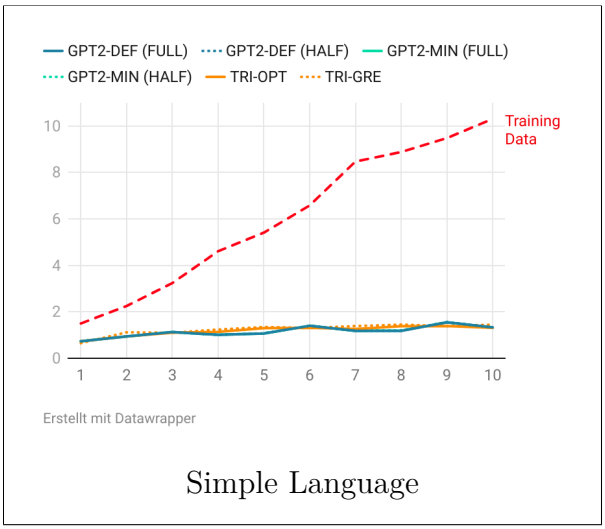

Figure 12: The entropy of the generated suffixes

Analysis shows, that only the GPT-2 model trained on the complex language effectively modeled the variability of their training data. These models display a suffix entropy that increases proportional to the suffix length, while the values of the other models are distributed between 0 and 2. These models do not fully match the entropy levels of their training data. However, even these models do not fully match the entropy levels of the training data. This suggests that the models may be reaching an upper bound in their ability to replicate the diversity of the language, likely due to overfitting.

In contrast, the models trained on the simple language are not capturing the full complexity of the language, but instead generating a regular subset (in the formal linguistic sense) of suffixes. As these suffixes are more homogenous compared to the ones generated by the GPT-2 models, the suffix entropy is comparatively lower.

4.3 Memorization

This section evaluates the proportion of generated sequences with exact matches in the training data of each model (see Figure 13, Add. Table 9). A high rate of exact matches

is expected to provide evidence of memorization, as it suggests the model is reproducing sequences it has encountered during training.

As with validity, the proportion of exact matches is affected by the suffix length and the number of terminals. As the number of possible suffixes increases exponentially with c, shorter suffixes have a higher likelihood of matching sequences from the training data. The amount of exact matches is additionally affected by the size of the training data. Models with smaller training data size have an increased chance of generating sequences not contained in their training data. However, this potential increase may be offset by the comparative weaker performance of models trained on datasets of smaller size.

This behavior aligns with the bias-variance tradeoff observed in machine learning models. Models trained on smaller datasets tend to generalize more (lower variance), which results in fewer exact matches. Conversely, models trained on larger datasets are more likely to memorize training data (lower bias), which leads to an increase in exact matches.

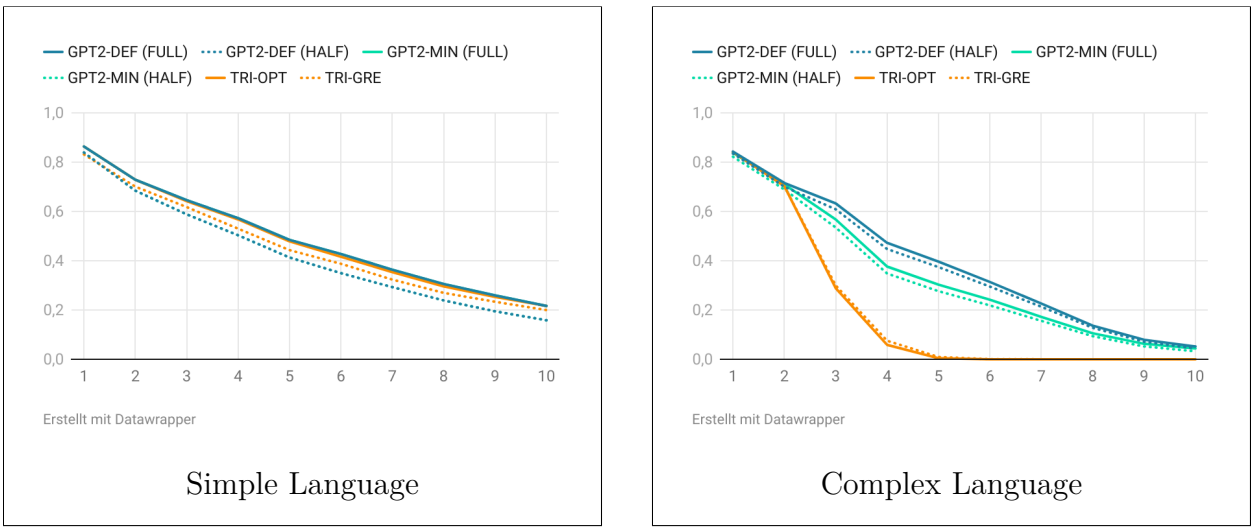

Figure 13: Percentage of generated sequences with exact matches in the training data

On average, models that were trained on the simple language possess a higher exact match rate compared to those trained on the complex language. For a suffix length of up to three, exact match rates are similar (excluding the trigram models of the complex language), however for longer sequence lengths the rates decline more steeply for models trained on the complex language. Additionally, the exact match rate is affected by both the model complexity and the training data size for the complex language, but only by the

training data size for the simple language.

The GPT-2 models' ability to produce valid sequences in the simple language suggests that the differences in exact match rates are primarily due to variations in training data size, rather than model behavior. A smaller dataset reduces the probability of generating sequences that are already present in the training data, thereby decreasing the exact match rate. For suffix lengths up to 4, the majority of generated valid sequences have exact matches, while for suffix lengths greater than 5, most are novel valid sequences.

In the context of the complex language, both model complexity and training data size have an impact on the rate of exact matches. Here, the majority of generated sequences have exact matches for suffix lengths up to 7, after which most are novel valid sequences.

4.4 Generalization

This section will discuss the proportion of generated novel valid sequences and interpret the results. We classify a sequence as novel and valid if it is not contained in the training data of its model and if it adheres to some set of rules, which here means that the sequence is contained in the language of its PCFG (See Figure 14, Add. Table 10). A higher proportion of novel valid sequences is expected to indicate the model's ability to generalize, as these sequences demonstrate the model's capacity to generate valid outputs beyond those explicitly present in the training data.

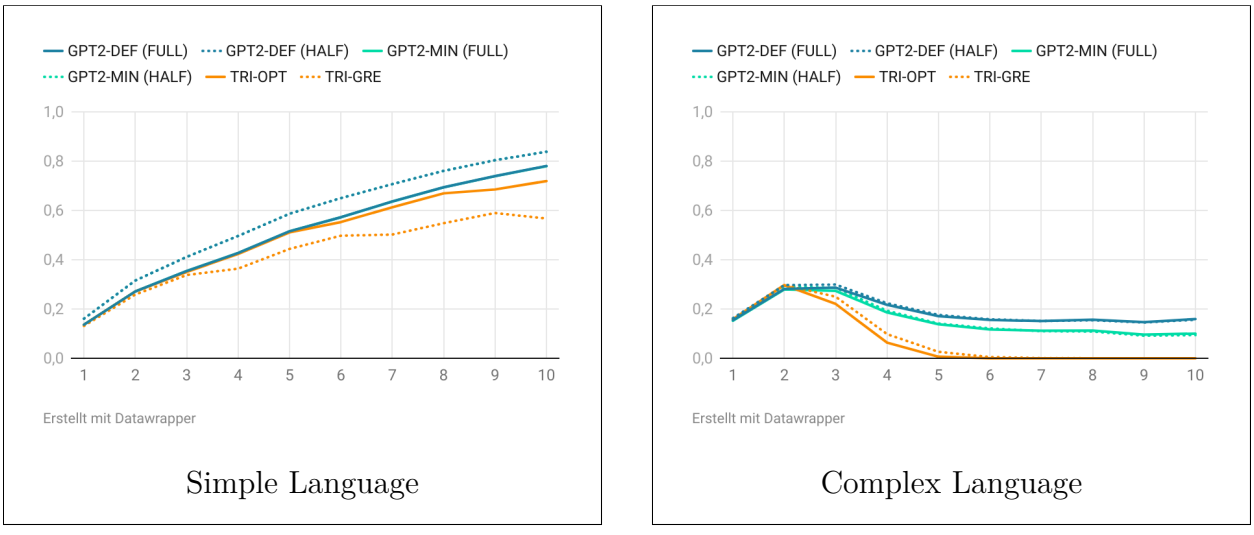

Figure 14: Percentage of generated novel valid sequences

As mentioned previously, models trained on less data have a larger chance of generating sequences not contained in their training data. As such, models with less training data and higher validity scores will display a higher proportion of novel valid sequences.

Trigram models consistently generated a lower proportion of novel valid sequences compared to GPT-2 models, largely due to their reduced ability to produce valid sequences in general. We can conclude that models with smaller context sizes and lower capacity tend to generate fewer novel valid sequences compared to models with larger capacity and more complex architectures.

Models trained on the simple language generate a higher amount of novel valid sequences compared to those trained on the complex language. While for smaller suffixes, models have a good chance of generating sequences already contained in the trainined data, long suffixes lead to higher proportions of novel valid sequences.

The models trained on the complex grammars corpus show an overall lower probability of generating novel valid sequences, which is in part caused by an overall lower probability of generating valid sequences in the first place. While the creativity scores of the simple language seem to increase, the scores of the complex language plateau at $c = 2$ and decline afterwards. This phenomenon is likely to stem from overall lowered validity scores as well. To migitate this bias, we present another evaluation of novel valid sequences in which we restrict the analysis to a population of only valid sequences (excluding invalid sequences). The results of this evaluation are shown in Figure 15 (see also Add. Table 11).

Noticable is that now the proportions of novel valid sequences between the models are more similar and proportional to the suffix length. Again, scores seem to be influenced more by training data size than by model complexity. As expected, models trained on smaller datasets display a slightly higher proportion of novel valid sequences. However, beyond training data size, we did not observe any discernible patterns in the relationship between model architecture and the rate of novel valid sequences.

The large fluctuations in values for the trigram models trained on the complex grammar are a consequence of the limited number of valid sequences they generate. Because there are so few valid sequences, even minor changes in the number of novel valid sequences cause significant fluctuations in their percentage among the total valid sequences, leading to the observed large jumps in the graph.

Our results indicate that novel valid sequences are not the exception, but the norm, even

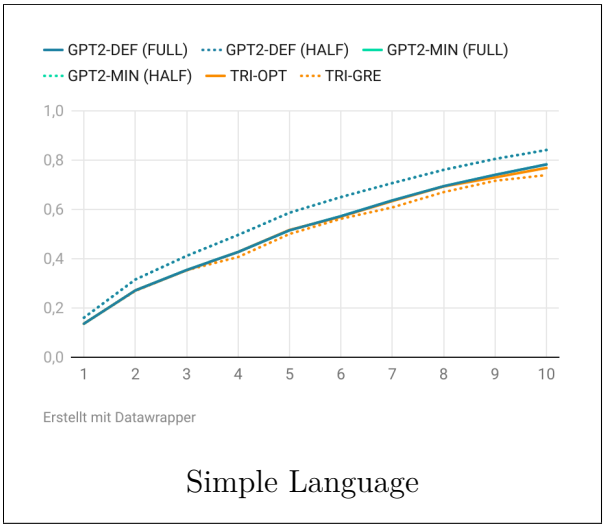

Figure 15: Percentage of generated novel valid sequences, ignoring invalid sequences

for low complexity architectures like the trigram models. As such, valid sequences appear to be consistently produced as a result of the model's generalization capabilities, especially when tasked with generating longer suffixes.

4.5 Perplexity

The final metric discussed is the *perplexity*, which measures a model's "uncertainty" in predicting the next token. Lower perplexity indicates that the model is more confident and assigns higher probabilities to likely next tokens, while higher perplexity suggests more uncertainty, with probabilities spread more evenly across possible tokens (See Figure 16, Add. Table 12). While lower perplexity scores are associated with better language modeling in general, perplexity can also be influenced by other factors like the model's temperature.

Perplexity is mathematically defined as the exponential of the entropy of the distribution of the next-token predictions. For a sequence of tokens $X = (x_1, \ldots, x_n)$ and a model's probability distribution P over the next possible tokens, we calculate the perplexity as follows:

$$Perplexity = e^{H(P)} = e^{-\sum_{x \in X} p(x) log(p(x))}$$

Here, $H(P)$ represents the entropy of the probability distribution of the next-token predictions and $p(x)$ the probability assigned to each token x. The overall perplexity of a model is determined by averaging the perplexity values of each generated sequence.

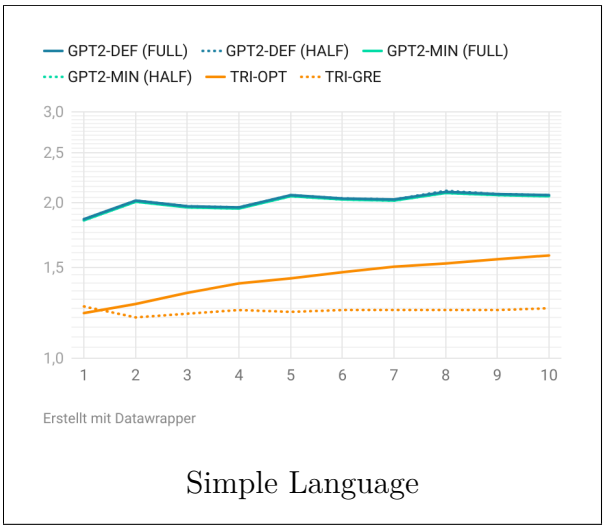

Figure 16: The average perplexity of each model over all sequences

Again, the perplexity values for the GPT-2 models trained on the simple language are very similar across configurations, which suggests that model capacity and training data size have little effect on the perplexity in this case. This consistency indicates that the simple language is relatively easy for the models to recreate, and even smaller models or less training data yield confident predictions.

The models trained on the complex language display more variation in their perplexity outcomes. Here, both increased model complexity and larger training datasets contribute to reduced perplexity, which results in improved confidence in the predictions. Additionally, the plateau observed in the perplexity values beyond a certain suffix length suggests that the models reach a point where additional sequence length does not affect their performance.

5 Discussion

In this chapter, we will summarize our key findings and discuss their implications regarding the relationship between memorization, generalization, and creativity in language models. Our results indicate that all GPT-2 models were able to learn the languages and consistently produce valid sequences. For the simple grammar, both underfitted (trigram models) and overfitted models (GPT-2 models) generated sequences that lacked diversity and originality. Specifically, overfitted models memorized frequent patterns which lead to them producing repetitive outputs, while underfitted models relied on simple, common structures due to their limited capacity. This similarity in outputs makes it challenging to distinguish genuine creativity from mere memorization. As such, evaluating creativity only by checking if valid generated sequences are included in the training data is insufficient, and further metrics are required to effectively judge the creativity of outputs.

In our experiments with models trained on the simple grammar, we observed a low rate of exact matches despite evidence suggesting that the models are overfitting. Specifically, these models predominantly generate sequences using the trigrams 'cdc', 'ccd', or 'dcc', which results in homogeneous structures. Although these trigrams are valid according to the grammar, they are overrepresented in the generated outputs. In the original dataset, the trigram 'cdc' and its variations made up about 35% of the suffixes. However, in the sequences generated by all models, these trigrams accounted for at least 95% of the suffixes, with the greedy trigram model producing them exclusively. Instead of solving the problem of completing sequences within the non-regular language, the models rely on regular language patterns to produce valid sequences. These regular structures were rare in the training data and thus often classified as novel. Given these findings, we argue that this behavior does not constitute actual creativity, as the models are not generating novel sequences through genuine understanding or generalization of the language structure. Instead, they are repeatedly applying simple, memorized patterns, which indicates a reliance on memorization rather than creative generation.

As we increased the task complexity with the complex language, the differences between the models became more noticeable. The GPT-2 models showed a higher degree of diversity in their outputs, as indicated by increased suffix entropy. The parameters of the GPT-2 models also had an effect on their behaviour, as models with a more complex architecture or larger training data size achieve higher evaluation scores. In contrast, the trigram models

struggled with the complex language, and their performance declined significantly for longer suffixes. The results suggest that these trigram models were displaying underfitting, similar to the trigram models trained on the simple grammars language.

Our findings corroborate two observations previously reported by Carlini et al. [23]. First, *larger models tend to memorize more*; that is, models with more complex architectures and trained on larger datasets exhibited higher exact match rates in our experiment, which indicates a greater tendency for memorizing training data. Secondly, *repeated strings are memorized more*; in our experiments with the simple language, the trigram 'cdc' and its variations were particularly frequent in the training data and were consequently generated more often by the models.

Our findings indicate that evaluating creativity solely by checking if valid generated sequences are included in the training data is insufficient. Models suffering either from overfitting or underfitting are able to produce outputs that are technically new, yet lack true originality or meaningful variation, as seen with the simple language.

This suggests that additional factors are necessary in assessing the creativity of language models effectively. The diversity of generated sequences can be helpful in recognizing the behaviour of models in terms of their reliance on memorization versus their capacity for genuine creative generalization. By examining the variety and uniqueness of the outputs, we can better determine whether a model is producing novel content or merely reproducing learned patterns.

Additionally, it is more effective to measure memorization not on a macro level (i.e., whole sequences), but instead on smaller chunks of generated sequences. This approach can help to better distinguish between outputs resulting from memorization of frequent patterns and those generated through genuine creative processes.

6 Summary

In this work, we trained different language models on languages generated by probabilistic context-free grammars of different complexities and evaluated them based on their ability to fill in suffixes of variable length of sequences from the training data. Our primary goal was to explore the memorization and generalization effects in these models, particularly under conditions that induce overfitting. We trained GPT-2 models and trigram models on both simple and complex artificial languages to understand how model architecture and language complexity influence the emergence of creativity and the tendency to memorize.

We found that evaluating creativity solely on whether a full sequence is contained in the training data is insufficient. Models affected by overfitting or underfitting can produce outputs that seem novel but lack true originality or meaningful variation. Assessing creativity effectively requires considering additional factors, such as the diversity of generated sequences and analyzing shorter subsequences or n-grams, since full sequences are rarely repeated verbatim. By focusing on these finer-grained elements, we can better distinguish between true creative generalization and mere memorization of frequent patterns.

While our experiments provide initial insights, expanding the scope of experiments—especially by including natural language datasets and larger models—could offer greater insights into how models handle memorization and creativity in more realistic settings. Currently, it is not well understood how neural language models generalize over a grammar or the extent to which the grammar is reflected in the model's parameters. Gaining insights into these aspects could support the development of more efficient and reliable language models.

Limitations

While our experiment is a first step into exploring the mechanisms behind memorization and generalization in language models, it only covers a limited setup that requires further exploration. We offer further avenues for future research that could help understand the mechanisms behind memorization and generalization.

Experimenting with different model architectures could provide a better perspective on how various models handle memorization and creativity. Models such as Long Short-Term Memory networks (LSTMs), newer transformer variants, or state-space models might show different behaviors than the GPT-2 and trigram models used in this study. Investigating how these models perform with the same tasks could reveal architecture-specific tendencies towards overfitting, generalization and memorization.

Next, adjusting model parameters and training strategies may influence the emergence of creativity. For instance, experimenting with different decoding strategies—such as beam search, top-k sampling, or nucleus sampling—could affect the diversity and originality of the generated sequences. Additionally, further varying the amount of training data, employing regularization techniques, or using different learning rates might impact the models' ability to generalize without overfitting.

Our study explored only two grammars, which limits the breadth of our analysis. Investigating a wider range of grammars could provide additional insights into the memorization and generalization tendencies of models trained on PCFGs. Specifically, testing both ambiguous and non-ambiguous grammars might reveal how grammatical ambiguity affects model performance and their ability to generalize. Expanding the experiments to include languages from different levels of the Chomsky hierarchy (such as regular or recursively enumerable languages) would allow us to assess whether the observed behaviors are consistent across various grammatical complexities.

Understanding how models store memorized sequences within their parameters remains an open question. By examining the internal representations of these models, we may gain insights into the mechanisms behind memorization. Techniques such as probing individual neurons, attention heads, or layers could reveal where and how memorization occurs within the model's architecture.

An additional limitation of our study is that, by focusing exclusively on probabilistic context-free grammars (PCFGs), we are confined to examining memorization and general-

ization phenomena. The artificial languages generated by PCFGs lack semantic richness and context found in natural languages, which are necessary for assessing true creativity. It is also worth noting that there remains potential for future research exploration regarding the absence of minimum or maximum sequence length constraints. Consequently, natural language might be better suited for exploring the emergence of creativity in language models. Future research should consider expanding experiments to include natural language datasets and larger models to determine whether the behaviors we observed persist in a more complex and realistic setting.

References

[1] Andre Esteva et al. “Dermatologist-level classification of skin cancer with deep neural networks”. In: *Nature* 542.7639 (Feb. 2017). Erratum in Nature. 2017 Jun 28;546(7660):686. doi:10.1038/nature22985, pp. 115–118. DOI: 10.1038/nature21056.

[2] E.W.T. Ngai et al. “The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature”. In: *Decision Support Systems* 50.3 (2011), pp. 559–569. DOI: https://doi.org/10.1016/j.dss.2010.08.006. URL: https://www.sciencedirect.com/science/article/pii/S0167923610001302.

[3] Paul Covington, Jay Adams, and Emre Sargin. “Deep Neural Networks for YouTube Recommendations”. In: *Proceedings of the 10th ACM Conference on Recommender Systems*. New York, NY, USA: Association for Computing Machinery, 2016, pp. 191–198. DOI: 10.1145/2959100.2959190. URL: https://doi.org/10.1145/2959100.2959190.

[4] Brent Smith and Greg Linden. “Two Decades of Recommender Systems at Amazon.com”. In: *IEEE Internet Computing* 21.3 (2017), pp. 12–18. DOI: 10.1109/MIC.2017.72.

[5] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080.

[6] Joy Buolamwini and Timnit Gebru. “Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification”. In: *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*. Ed. by Sorelle A. Friedler and Christo Wilson. Vol. 81. Proceedings of Machine Learning Research. PMLR, 23–24 Feb 2018, pp. 77–91. URL: https://proceedings.mlr.press/v81/buolamwini18a.html.

[7] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL]. URL: https://arxiv.org/abs/2005.14165.

[8] Alec Radford et al. “Language Models are Unsupervised Multitask Learners”. In: 2019. URL: https://api.semanticscholar.org/CorpusID:160025533.

[9] Nicholas Carlini et al. *Extracting Training Data from Large Language Models*. 2021. arXiv: 2012.07805 [cs.CR]. URL: https://arxiv.org/abs/2012.07805.

[10] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. "Introduction to Automata Theory, Languages, and Computation". In: 2nd. Addison-Wesley, 2001, pp. 28–31.

[11] Noam Chomsky. "Three models for the description of language". In: *IRE Transactions on Information Theory* 2.3 (1956), pp. 113–124. DOI: 10.1109/TIT.1956.1056813.

[12] Christopher D. Manning and Hinrich Schütze. "Foundations of Statistical Natural Language Processing". In: Cambridge, MA: MIT Press, 1999, pp. 381–387.

[13] Yoshua Bengio et al. "A neural probabilistic language model". In: *Journal of Machine Learning Research* 3 (2003), pp. 1137–1155. ISSN: 1532-4435.

[14] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL]. URL: https://arxiv.org/abs/1301.3781.

[15] Li Deng and Yang Liu. "A Joint Introduction to Natural Language Processing and to Deep Learning". In: *Deep Learning in Natural Language Processing*. Singapore: Springer Singapore, 2018, pp. 1–22. DOI: 10.1007/978-981-10-5209-5_1. URL: https://doi.org/10.1007/978-981-10-5209-5_1.

[16] Ashish Vaswani et al. "Attention is All You Need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Curran Associates Inc., 2017, pp. 6000–6010.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016. ISBN: 9780262035613. URL: https://books.google.de/books?id=Np9SDQAAQBAJ.

[18] Alec Radford and Karthik Narasimhan. "Improving Language Understanding by Generative Pre-Training". In: 2018. URL: https://api.semanticscholar.org/CorpusID:49313245.

[19] Yifan Yao et al. "A survey on large language model (LLM) security and privacy: The Good, The Bad, and The Ugly". In: *High-Confidence Computing* 4.2 (2024), p. 100211. ISSN: 2667-2952. DOI: https://doi.org/10.1016/j.hcc.2024.100211. URL: https://www.sciencedirect.com/science/article/pii/S266729522400014X.

[20] Jean-Baptiste Truong et al. *Data-Free Model Extraction*. 2021. arXiv: 2011.14779 [cs.LG]. URL: https://arxiv.org/abs/2011.14779.

[21] Lei Huang et al. *A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions*. 2023. arXiv: 2311.05232 [cs.CL].

[22] Ziwei Ji et al. "Survey of Hallucination in Natural Language Generation". In: *ACM Computing Surveys* 55.12 (2023), pp. 1–38. ISSN: 1557-7341. DOI: 10.1145/3571730. URL: http://dx.doi.org/10.1145/3571730.

[23] Nicholas Carlini et al. *Quantifying Memorization Across Neural Language Models*. 2023. arXiv: 2202.07646 [cs.LG]. URL: https://arxiv.org/abs/2202.07646.

[24] Zeyuan Allen-Zhu and Yuanzhi Li. *Physics of Language Models: Part 1, Context-Free Grammar*. 2023. arXiv: 2305.13673 [cs.CL].

[25] Kent F. Hubert, Kim N. Awa, and Darya L. Zabelina. "The current state of artificial intelligence generative language models is more creative than humans on divergent thinking tasks". In: *Scientific Reports* 14.1 (2024), p. 3440. DOI: 10.1038/s41598-024-53303-w. URL: https://doi.org/10.1038/s41598-024-53303-w.

[26] Honghua Chen and Nai Ding. "Probing the "Creativity" of Large Language Models: Can models produce divergent semantic association?" In: *Proceedings of EMNLP*. 2023. URL: https://aclanthology.org/2023.findings-emnlp.858.

[27] Jennifer Hu et al. "A Systematic Assessment of Syntactic Generalization in Neural Language Models". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020, pp. 1725–1744. DOI: 10.18653/v1/2020.acl-main.158. URL: https://aclanthology.org/2020.acl-main.158.

[28] Christian Bentz et al. "The Entropy of Words—Learnability and Expressivity across More than 1000 Languages". In: *Entropy* 19.6 (2017). ISSN: 1099-4300. DOI: 10.3390/e19060275. URL: https://www.mdpi.com/1099-4300/19/6/275.

[29] George K. Zipf. *Human Behaviour and the Principle of Least Effort*. Addison-Wesley, 1949.

Addendum

Evaluation Values

	Model	c									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Complex Language	Baseline	20.00%	4.00%	0.80%	0.16%	0.03%	0.01%	0.00%	0.00%	0.00%	0.00%
	GPT2-DEF (FULL)	99.78%	99.73%	91.93%	68.93%	56.74%	46.95%	37.76%	29.36%	22.60%	21.10%
	GPT2-DEF (HALF)	99.77%	99.72%	90.80%	67.09%	55.03%	45.35%	36.47%	28.22%	21.69%	20.03%
	GPT2-MIN (FULL)	99.03%	98.94%	84.09%	56.24%	44.11%	35.92%	28.39%	21.77%	15.91%	14.41%
	GPT2-MIN (HALF)	98.25%	98.16%	82.06%	54.11%	41.78%	33.98%	26.63%	20.22%	14.38%	12.65%
	TRI-OPT	100.00%	100.00%	50.86%	12.21%	1.07%	0.00%	0.00%	0.00%	0.00%	0.00%
	TRI-GRE	100.00%	100.00%	54.94%	17.33%	3.57%	0.56%	0.11%	0.02%	0.00%	0.00%
Simple Language	Baseline	14.29%	2.04%	0.29%	0.04%	0.01%	0.00%	0.00%	0.00%	0.00%	0.00%
	GPT2-DEF (FULL)	100.00%	100.00%	99.99%	99.98%	99.98%	99.98%	99.98%	99.97%	99.86%	99.71%
	GPT2-DEF (HALF)	100.00%	100.00%	99.99%	99.98%	99.98%	99.98%	99.98%	99.97%	99.86%	99.66%
	GPT2-MIN (FULL)	100.00%	100.00%	99.99%	99.98%	99.96%	99.96%	99.95%	99.94%	99.82%	99.62%
	GPT2-MIN (HALF)	100.00%	100.00%	99.99%	99.98%	99.98%	99.98%	99.98%	99.97%	99.86%	99.66%
	TRI-OPT	100.00%	100.00%	99.21%	99.11%	98.95%	96.90%	96.70%	96.54%	93.87%	93.59%
	TRI-GRE	96.27%	95.94%	95.58%	89.39%	88.69%	88.53%	82.56%	81.77%	82.32%	76.71%

Table 7: The percentage of generated valid sequences

	Model	c									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Complex Language	Training Data	0.86	1.91	2.76	3.72	4.75	5.98	6.96	7.89	8.71	9.46
	GPT2-DEF (FULL)	0.64	1.55	3.17	3.89	4.59	5.44	6.22	6.71	7.01	7.22
	GPT2-DEF (HALF)	0.60	1.51	3.20	3.92	4.61	5.46	6.24	6.76	7.08	7.34
	GPT2-MIN (FULL)	0.50	1.38	3.21	3.95	4.60	5.43	6.22	6.79	7.17	7.48
	GPT2-MIN (HALF)	0.52	1.36	3.24	3.96	4.60	5.42	6.20	6.80	7.23	7.57
	TRI-OPT	0.00	0.00	1.58	1.49	1.29	1.51	1.72	1.80	1.83	1.83
	TRI-GRE	0.00	0.00	1.61	1.38	1.14	1.35	1.50	1.54	1.58	1.59
Simple Language	Training Data	1.50	2.25	3.24	4.61	5.41	6.58	8.47	8.88	9.48	10.31
	GPT2-DEF (FULL)	0.73	0.94	1.13	1.01	1.07	1.40	1.18	1.18	1.55	1.33
	GPT2-DEF (HALF)	0.73	0.94	1.13	1.01	1.07	1.40	1.18	1.19	1.55	1.33
	GPT2-MIN (FULL)	0.73	0.94	1.13	1.01	1.07	1.40	1.18	1.19	1.55	1.32
	GPT2-MIN (HALF)	0.73	0.94	1.13	1.01	1.08	1.40	1.19	1.20	1.56	1.34
	TRI-OPT	0.73	0.94	1.10	1.14	1.30	1.31	1.26	1.38	1.39	1.31
	TRI-GRE	0.65	1.12	1.10	1.24	1.35	1.31	1.39	1.45	1.39	1.45

Table 8: Suffix Entropy

	Model	c									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Complex Language	GPT2-DEF (FULL)	84.30%	71.56%	63.27%	47.24%	39.62%	31.38%	22.63%	13.67%	7.93%	5.19%
	GPT2-DEF (HALF)	83.47%	70.02%	60.87%	44.75%	37.42%	29.51%	21.33%	12.78%	7.15%	4.39%
	GPT2-MIN (FULL)	83.79%	70.97%	56.74%	37.63%	30.29%	24.19%	17.22%	10.55%	6.29%	4.38%
	GPT2-MIN (HALF)	82.20%	68.92%	53.48%	34.82%	27.65%	21.87%	15.61%	9.38%	5.22%	3.24%
	TRI-OPT	84.13%	70.27%	28.83%	5.86%	0.43%	0.00%	0.00%	0.00%	0.00%	0.00%
	TRI-GRE	84.13%	70.27%	29.95%	7.53%	0.94%	0.05%	0.03%	0.00%	0.00%	0.00%
Simple Language	GPT2-DEF (FULL)	86.35%	72.88%	64.55%	57.27%	48.45%	42.73%	36.34%	30.56%	25.91%	21.66%
	GPT2-DEF (HALF)	83.91%	68.43%	58.82%	50.33%	41.32%	34.97%	29.29%	23.90%	19.47%	15.84%
	GPT2-MIN (FULL)	86.35%	72.88%	64.55%	57.27%	48.45%	42.73%	36.34%	30.56%	25.91%	21.66%
	GPT2-MIN (HALF)	83.91%	68.43%	58.82%	50.33%	41.32%	34.97%	29.29%	23.90%	19.47%	15.84%
	TRI-OPT	86.35%	72.88%	64.16%	56.77%	47.84%	41.62%	35.40%	29.60%	25.35%	21.66%
	TRI-GRE	83.13%	70.07%	61.78%	53.01%	44.31%	38.77%	32.35%	26.95%	23.35%	19.98%

Table 9: The percentage of exact matches

	Model	c									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Complex Language	GPT2-DEF (FULL)	15.47%	28.17%	28.66%	21.69%	17.12%	15.58%	15.13%	15.69%	14.67%	15.92%
	GPT2-DEF (HALF)	16.30%	29.70%	29.93%	22.34%	17.61%	15.84%	15.14%	15.43%	14.54%	15.64%
	GPT2-MIN (FULL)	15.24%	27.98%	27.35%	18.61%	13.82%	11.72%	11.17%	11.22%	9.63%	10.03%
	GPT2-MIN (HALF)	16.05%	29.24%	28.58%	19.29%	14.12%	12.12%	11.02%	10.85%	9.16%	9.42%
	TRI-OPT	15.87%	29.73%	22.04%	6.35%	0.64%	0.00%	0.00%	0.00%	0.00%	0.00%
	TRI-GRE	15.87%	29.73%	24.98%	9.80%	2.63%	0.50%	0.08%	0.02%	0.00%	0.00%
Simple Language	GPT2-DEF (FULL)	13.65%	27.12%	35.44%	42.71%	51.53%	57.25%	63.64%	69.41%	73.95%	78.05%
	GPT2-DEF (HALF)	16.09%	31.57%	41.16%	49.65%	58.66%	65.01%	70.68%	76.07%	80.39%	83.83%
	GPT2-MIN (FULL)	13.65%	27.12%	35.44%	42.71%	51.51%	57.23%	63.61%	69.38%	73.91%	77.96%
	GPT2-MIN (HALF)	16.09%	31.57%	41.16%	49.65%	58.66%	65.01%	70.68%	76.07%	80.39%	83.83%
	TRI-OPT	13.65%	27.12%	35.05%	42.34%	51.11%	55.28%	61.30%	66.94%	68.52%	71.93%
	TRI-GRE	13.14%	25.87%	33.80%	36.38%	44.38%	49.76%	50.21%	54.82%	58.96%	56.72%

Table 10: The percentage of novel valid sequences

	Model	c									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Complex Language	GPT2-DEF (FULL)	15.51%	28.25%	31.18%	31.46%	30.18%	33.17%	40.06%	53.44%	64.90%	75.42%
	GPT2-DEF (HALF)	16.34%	29.79%	32.96%	33.30%	32.00%	34.93%	41.52%	54.70%	67.05%	78.08%
	GPT2-MIN (FULL)	15.39%	28.27%	32.52%	33.10%	31.32%	32.64%	39.34%	51.53%	60.49%	69.61%
	GPT2-MIN (HALF)	16.34%	29.79%	34.83%	35.65%	33.81%	35.66%	41.40%	53.64%	63.72%	74.42%
	TRI-OPT	15.87%	29.73%	43.33%	51.98%	59.41%	N.A.	N.A.	N.A.	N.A.	N.A.
	TRI-GRE	15.87%	29.73%	45.48%	56.56%	73.59%	90.38%	70.00%	100.00%	N.A.	N.A.
Simple Language	GPT2-DEF (FULL)	13.65%	27.12%	35.45%	42.72%	51.54%	57.26%	63.65%	69.43%	74.05%	78.28%
	GPT2-DEF (HALF)	16.09%	31.57%	41.17%	49.66%	58.67%	65.03%	70.70%	76.09%	80.50%	84.11%
	GPT2-MIN (FULL)	13.65%	27.12%	35.45%	42.72%	51.53%	57.25%	63.64%	69.42%	74.04%	78.26%
	GPT2-MIN (HALF)	16.09%	31.57%	41.17%	49.66%	58.67%	65.03%	70.70%	76.09%	80.50%	84.11%
	TRI-OPT	13.65%	27.12%	35.33%	42.72%	51.66%	57.05%	63.39%	69.34%	73.00%	76.86%
	TRI-GRE	13.65%	26.96%	35.36%	40.70%	50.04%	56.20%	60.82%	67.04%	71.63%	73.95%

Table 11: The percentage of novel valid sequences, ignoring invalid sequences

	Model	c									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Complex Language	GPT2-DEF (FULL)	2.26	2.21	2.32	2.36	2.34	2.36	2.37	2.38	2.38	2.37
	GPT2-DEF (HALF)	2.30	2.24	2.35	2.39	2.37	2.39	2.40	2.41	2.41	2.41
	GPT2-MIN (FULL)	2.60	2.44	2.53	2.59	2.56	2.57	2.58	2.59	2.59	2.59
	GPT2-MIN (HALF)	2.74	2.53	2.61	2.66	2.64	2.65	2.66	2.67	2.68	2.68
	TRI-GRE	1.10	1.10	1.14	1.17	1.17	1.15	1.14	1.13	1.13	1.14
Simple Language	GPT2-DEF (FULL)	1.86	2.02	1.97	1.96	2.07	2.04	2.03	2.10	2.08	2.07
	GPT2-DEF (HALF)	1.86	2.02	1.97	1.96	2.07	2.04	2.03	2.11	2.08	2.07
	GPT2-MIN (FULL)	1.85	2.01	1.96	1.95	2.06	2.03	2.02	2.09	2.07	2.06
	GPT2-MIN (HALF)	1.85	2.01	1.96	1.95	2.06	2.03	2.02	2.09	2.07	2.06
	TRI-GRE	1.26	1.20	1.22	1.24	1.23	1.24	1.24	1.24	1.24	1.25

Table 12: The average perplexity of each model over all sequences

Code

We provide our code and sample data under `https://anonymous.4open.science/r/PCFGS_Generalisation-FB3B/`. The repository contains the following:

- `documents`: The training data, as outlined in chapter 3.2.
- `evaluation`: The evaluation results for each model.
- `generated`: The sequences produced by each model.
- `grammars`: The production rules of each grammar.
- `ngrams`: The n-gram frequencies of the corpora.
- `src`: The code used to build the grammars, generate the documents and train and evaluate the models.
- `schedules`: JSON dictionaries that specify training/evaluation parameters.

Use of Generative AI

Generative AI has been used in this work for grammar and syntax correction. No original content was produced by generative AI.

Ich bin damit einverstanden, dass meine Arbeit in den Bestand der Bibliothek eingestellt wird.

Ort, Datum

Unterschrift

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Bachelorstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe

Ort, Datum

[redacted]

Unterschrift