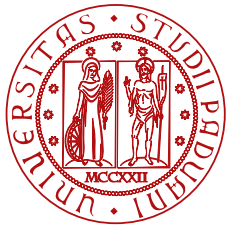


UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

MASTER THESIS IN COMPUTER ENGINEERING

Can Transformers Learn Synchronous Context-Free Translations? An Analysis of Pretraining on Automatically Generated Datasets

MASTER CANDIDATE

Nicola Calzone

Student ID 2087003

SUPERVISOR

Prof. Giorgio Satta

University of Padova

CO-SUPERVISOR

Assoc. Prof. Ondřej Bojar

Charles University

ACADEMIC YEAR
2024/2025

Abstract

This thesis presents an experimental investigation into a previously unexplored capability of Transformers: their ability to infer Synchronous Context-Free Grammars (SCFGs), i.e. to learn and generalize over particular grammars just from example pairs of strings. More specifically, two experiments were conducted. The first experiment explored Transformers' capacity to translate between synthetic languages corresponding to the source and target side of an SCFG grammar. The second experiment sought for a Transformer configuration which would be capable of SCFG parsing, i.e. identifying the ability to recognize licensed SCFG pairs of strings based on only positive and negative training examples. With a sufficiently large model, Transformers proved capable to learn this task to a high accuracy (up to 96.7%) even for very long inputs, longer than any training items. Experiments show limitations and variability that leave parts of the problem open to further research.

Sommario

Questa tesi presenta un'indagine sperimentale su una capacità precedentemente inesplorata dei Transformer: la loro abilità di inferire Synchronous Context-Free Grammars (SCFGs), ovvero di apprendere e generalizzare grammatiche specifiche a partire semplicemente da coppie di stringhe di esempio. Nello specifico, sono stati condotti due esperimenti: il primo esplora la capacità dei Transformer di tradurre tra linguaggi sintetici, che corrispondono rispettivamente al lato sorgente e al lato target di una grammatica SCFG. Il secondo esperimento consiste nella ricerca di una configurazione di Transformer in grado di effettuare il parsing di SCFGs, ovvero la capacità di riconoscere coppie di stringhe licenziate da una SCFG basandosi solo su esempi di addestramento positivi e negativi. Con un modello sufficientemente grande, i Transformer hanno dimostrato di poter apprendere questo compito con un'elevata accuratezza (fino al 96,7%), anche per input molto lunghi, più estesi di qualsiasi elemento durante l'addestramento. Gli esperimenti dimostrano variabilità e limitazioni che lasciano il problema aperto ad ulteriori ricerche.

Contents

List of Figures

List of Tables

List of Acronyms

AI Artificial Intelligence

BLEU Bilingual Evaluation Understudy

BPE Byte Pair Encoding

CALR Cosine Annealing Learning Rate

CE Cross-Entropy

CFG Context Free Grammar

CYK Cocke-Younger-Kasami

CNF Chomsky Normal Form

DL Deep Learning

FN False Negative

FP False Positives

ML Machine Learning

MT Machine Translation

NLP Natural Language Processing

NMT Neural Machine Translation

NT nonterminal

RHS Right-Hand Side

SCFG Synchronous Context Free Grammar

SDTS Syntax-directed translation schemata

Seq2Seq Sequence-to-Sequence

SOV Subject-Object-Verb

SVO Subject-Verb-Object

TN True Negative

TP True Positives

VSO Verb-Subject-Object

1

Introduction

"In the beginning was the Word, and the Word was with God, and the Word was God." [1]

Over recent years, Natural Language Processing (NLP) [1] has achieved exceptional capabilities and this happened thanks to deep learning models, particularly Transformers [2, 3]. These models have achieved human-like quality in machine translation [4], and their impact extends to other domains such as image processing [5] and speech processing [6]. The separation of Transformer encoder and decoder architectures has led to the development of large language models [7], which have brought the whole field into unprecedented popularity while also revealing limitations such as hallucination and biased outputs. Despite these advancements, uncertainty persists regarding how Transformers learn, with ongoing debates about whether their performance stems from genuine generalization or merely memorizing vast datasets [8, 9, 10, 11]. This thesis bridges two domains: deep learning, with its reliance on Transformers, and the pre-neural machine translation era, which utilized synchronous context-free grammars (SCFGs). These special grammars historically enabled the generation of sentence pairs in two languages using formal rules.

Initially, this work aimed to extend Tom Kocmi's doctoral thesis [12], which focused on enhancing Transformer understanding of low-resource languages. However, it evolved to investigate the generalization capabilities of Transformers using synthetic languages generated by SCFGs. These synthetic languages exhibit two key charac-

[1] The Bible, John 1:1

teristics: they can resemble natural languages, and they are recursive. This research emphasizes the recursive nature of synthetic languages, without really mimicking human languages, exploring how well Transformers can generalize over deeply recursive structures using significantly smaller datasets than those typically required for large language models powering modern chatbots and agents. Specifically, this study examines the Transformer's ability to learn a SCFG from varying sizes of example sets and apply it to two tasks: **parsing** (Chapter 3) and **translation** (Chapter 4). Through experiments, we seek evidence that Transformers can successfully generalize from observed examples rather than merely memorizing positive versus negative cases.
This approach is novel and largely unexplored, positioning this thesis as the first known attempt to analyze the generalization capabilities of Transformers through the lens of synchronous context-free grammars.

1.1 Research Objectives

Transformers perform well on tasks with large, unstructured datasets, but their ability to handle structured, rulebased tasks, such as translations based on synchronous context-free grammars (SCFGs), is not explored yet. In the following chapters, we will examine how SCFGs provide a formal framework for modeling translation as a bilingual parsing problem in rule-based machine translation systems. SCFGs are well-suited for translation tasks because they capture hierarchical and syntactic relationships between source and target languages, which naturally include hierarchies, compositionality, and recursion — key features of human language. However, two challenges arise:

- SCFGs are limited in capturing the complexity of some natural languages, particularly those with non-projectivity (e.g., Czech), potentially causing Transformers to overfit to synthetic patterns due to this simplification;
- deep recursion in SCFGs may make it difficult for Transformers to distinguish valid language patterns from invalid ones.

Many real-world translation scenarios, such as code translation, formal language processing, or bilingual alignments in domains like legal or technical texts involve structured patterns. Understanding whether Transformers can generalize to these structured tasks could broaden their applications. This thesis builds on the work of Tom Kocmi [12], whose doctoral research showed that Transformers achieve significantly better performance in machine translation for low-resource languages after transfer learning from

high-resource language pairs. If Transformers can learn from synthetic SCFG data, this could enable knowledge transfer to other text types and languages, enhancing translation systems in structured and low-resource settings.

1.2 Thesis Organization

This thesis is structured as follows:

- **Chapter 2** presents the algorithms employed in this study, along with an explanation of relevant theoretical concepts, including formal grammars and the deep learning models used, to facilitate understanding of subsequent chapters.
- **Chapter 3**, referred to as the "Acceptor Experiment" describes the parsing (i.e. determining whether a string is licensed by the grammar) experiment. It describes the experimental setup, dataset preparation, model architecture, then the results and their limitations.
- **Chapter 4** details the machine translation experiment. It follows a similar structure as the previous chapter, covering experimental setup, dataset preparation, model architecture, results, and limitations.
- **Chapter 5** concludes the thesis with a summary of the contributions of this work, discussing potential paths for future exploration, and addressing all the limitations encountered.

2

Background and Related Work

"If as one people speaking the same language they have begun to do this, then nothing they plan to do will be impossible for them." [1]

This chapter expands on the concepts introduced earlier, providing a deeper explanation of key topics essential for understanding the project. These fundamentals are necessary to explain all the artifacts and algorithms developed to make the system work. Specifically, the project required several components, including: a random grammar generator, a sentence-pair generator based on SCFG, a parser for binarised SCFGs and finally a neural *Acceptor* and a *Translator* both built using Transformer technology.

2.1 Overview of Natural Language Processing

Human language exhibits several fundamental properties: it is intrinsically *ambiguous*, with linguistic units carrying different meanings depending on context; it is *compositional*, where the meaning of a larger unit, such as a sentence, is a function of the meanings of its components and their combination; it is *recursive*, allowing units to be repeatedly combined through grammatical rules to generate complex recursive structures; additionally, it has hidden structures, where local changes, such as altering a single word (e.g., "The trophy doesn't fit into the suitcase because it is too {small, large}") can have global effects on a sentence's interpretation, revealing underlying dependencies.

[1] Genesis 11:5–6, "The Tower of Babel"

Understading, generating and translating natural language are all fields that go under natural language processing (NLP). Often synonymous with computational linguistics, NLP is a field of artificial intelligence (AI) that enables machines to understand and generate natural language [13]. NLP today works for a large range of applications, from chatbots (e.g. ChatGPT) and virtual assistants (e.g. Alexa, Siri, Google Assistant) [14, 15] to machine translation [4], sentiment analysis and fake news detection [16]. These technologies are widely used in more industry sectors and daily transform the way we interact with machines.

2.1.1 MACHINE TRANSLATION

Machine translation, one of the key areas of NLP, developed through different stages [1]. Its conceptual foundations emerged in the early 20th century and can be defined as follows:

Definition 1 ***Machine Translation (MT):*** *automatic translation of text or speech from one language to another without human involvement.*

This branch of NLP saw its first real progress in the 1950s. In 1954, the "Georgetown-IBM" system became the first successful machine translation system for English-Russian [17]. Between this period and the 1990s, progress remained limited; scientists used symbolic models with rules created by linguists (*Rule-Based MT*), these models required thousands of rules and struggled to handle the real-world complexity of natural languages. In the 1990s, statistical models emerged (*Statistical MT*), using large datasets — with expert annotations in machine translation case — to achieve good results. Since the 2010s, neural models have successfully used machine learning to perform well in tasks like translation and text generation (*Neural MT*). In 2017, the invention of the Transformer revolutionized this field, enabling NLP models to achieve remarkable results previously unimaginable. In this thesis work we will use the rules of Rule-Based MT and the Transformer models, which are the current state-of-the-art solution in the field.

In MT we define *Source* language the original language of the text to be translated, while the *Target* language is the one into which the text is translated. The words in the target language may not match the source language in terms of number or order. For example, consider the translation of the following constructed English sentence into Japanese:

(1) (a) English: He wrote a letter to a friend

(b) Japanese:	tomodachi	friend
	ni	to
	tegami-o	letter
	kaita	wrote

The word order differs significantly between the two languages. In English, the verb "wrote" appears in the middle of the sentence, whereas in Japanese, the verb "kaita" is placed at the end. Languages differ in word order for simple declarative clauses [1]. English, German, French, and Mandarin follow an *SVO* (Subject-Verb-Object) structure, with the verb between subject and object. Hindi and Japanese are *SOV*, with the verb at the clause's end. Irish and Arabic are *VSO*. Moreover, it is possible that one language may have a lexical gap: which means that no word or periphrasis of the target language can express the exact meaning of a word in the source language.

2.2 The Transformer Model

The standard architecture for machine translation (MT) is the encoder-decoder Transformer, a particular kind of a sequence-to-sequence (Seq2Seq) model [1]. As mentioned in the introduction, Transformers [2, 3] are neural network architectures that revolutionized artificial intelligence (AI) by using self-attention mechanisms to process sequential data. Self-attention allows the model to focus on all parts of the input at once by assigning weights to their importance, enabling parallel processing instead of sequential computation, which makes training and inference much faster. This innovation has allowed machine translation to reach human-level quality [4] for some languages and has similarly transformed fields like image [5] and speech [6] processing. The separation of encoder and decoder architectures led to the development of large language models (LLMs) [7]. The popularity of these models also revealed limitations, such as hallucination or biased outputs [18]; these limitations highlight the need to understand Transformers' formal capabilities, as we will explore in this thesis. The formal capabilities of Transformers are still being actively researched, as their theoretical limits and how training can achieve their full potential [8, 9, 10, 11].

Despite their power, Transformers exhibit a paradoxical behavior in their computational capabilities, which this thesis aims to explore. Two key papers present conflicting findings [19]: one proves that Transformers are Turing-complete [20], while another shows that Transformers, when processing a string of 0s and 1s, cannot determine whether the number of 1s is odd or even [8, 21]. This thesis investigates this

issue using formal languages, described later in Section 2.5. Specifically, we study the Transformer's ability to learn a synchronous context-free grammar (SCFG), detailed in Section 2.5, with different sizes of example sets applied to two tasks: *translation* and *parsing*. For the *parsing* experiment, presented in Chapter 3, we use an Encoder-Only Transformer architecture. For the *translation* experiment, described in Chapter 4, we employ a Sequence-to-Sequence Transformer architecture.

Sequence-to-Sequence (Seq2Seq) The Seq2Seq Transformer, as detailed in [2, 3], was created for machine translation tasks and is hereby used for the Machine Translation experiment conducted in Chapter 4. It consists of an encoder and a decoder, both composed of stacked layers of multi-head self-attention, cross-attention (in the decoder), and feed-forward networks. The encoder processes the input sequence, transforming it into a set of contextualized representations by attending to all tokens simultaneously, capturing long-range dependencies effectively. In traditional encoder-decoder models, the encoder's final hidden state acts as a fixed context vector. The encoder processes the source language input tokens $X = x_1, \ldots, x_n$ and produces an intermediate representation $H_{\text{enc}} = h_1, \ldots, h_n$ through a stack of encoder blocks, creating a bottleneck that struggles to represent all input information, especially for long sequences. The attention mechanism introduced in [22] addresses this by dynamically computing a context vector c_i for each decoding step i, derived as a weighted sum of all encoder hidden states $\{h_1^e, \ldots, h_n^e\}$, as defined by Equation (2.1), where α_{ij} are attention weights computed via a softmax over similarity scores computed as dot products between the decoder's state and encoder hidden states. These weights focus on parts of the input relevant to the current output token, enabling the decoder to access all encoder representations via cross-attention. The decoder takes this vector into its hidden state computation, as given by Equation (2.2).

$$c_i = \sum_{j=1}^{n} \alpha_{ij} h_j^e, \tag{2.1}$$

$$h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i), \tag{2.2}$$

where $\hat{y}_{i-1}$ is the previous output token, h_{i-1}^d is the previous decoder hidden state, and g represents the decoder's feed-forward network combining these inputs [2]. This architecture is well-suited for translation tasks, as it maps a source sequence to a target sequence, facilitating the learning of SCFGs by aligning syntactic structures across languages. In this work, we use the Seq2Seq Transformer as it is implemented in the

`fairseq` framework [23], presented in Section 2.4.1.

Most MT tasks treat sentences as independent units, aiming to translate a source language sentence into a target language sentence. MT systems employ supervised machine learning, training on a *parallel corpus* or *bitext* (ours are described in Section 4.2), which pairs source language sentences with their target language translations. The goal is to maximize the probability of the target token sequence $y_1, \ldots, y_m$ given the source token sequence $x_1, \ldots, x_n$:

$$P(y_1, \ldots, y_m \mid x_1, \ldots, x_n) \tag{2.3}$$

Encoder-Only In contrast, the Encoder-Only Transformer used for the Acceptor experiment detailed in Chapter 3, does not use the decoder component and solely consists of the encoder one. This architecture — used for important models as BERT [24] — processes the input sequence to produce contextualized representations, which are then used directly for tasks that do not require generating a new sequence, as in our case for *parsing* where we use this architecture for the *classification* task. In this thesis, the Encoder-Only Transformer is employed for the parsing experiment in Chapter 3 to classify the input sequence as licensed by the synthetical language. For this project no causal mask is applied and each self-attention layer attends to all token positions simultaneously, making the model bidirectional. This BERT-like architecture ensures that each token's representation depends on tokens both to its left and to its right.

2.3 Training

Both models, discussed further in Chapter 3 and Chapter 4, are trained using the Cross-Entropy (CE) loss function [25] and the Adam optimizer [26]. The CE loss measures the difference between the predicted and true probability distributions. The Adam optimizer is a popular method for Transformers, adjusting the learning rate efficiently using first-order gradients and low-memory estimates, which works well for large, noisy, or changing datasets with minimal tuning. Additionally, for the experiment presented in Chapter 4, a Cosine Annealing scheduler is used to adjust the learning rate dynamically during training. This scheduler changes the learning rate over time (e.g., per epoch) to help the model converge better, avoid large steps, and fine-tune effectively.

2.3.1 Cross-Entropy Loss

The Cross-Entropy Loss is defined as

$$L_{\text{CE}} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{2.4}$$

where L_{CE} represents the Cross-Entropy Loss, C denotes the number of classes, y_i is the true probability for class i, and $\hat{y}_i$ is the predicted probability for class i.

2.3.2 Adam Optimizer

The Adam optimizer [26], an adaptive method combining momentum and RMSProp, updates model parameters using first and second-order moments of the gradients. It is particularly effective for training Transformer-based models on large datasets due to its adaptive learning rates and minimal hyperparameter tuning requirements.

The parameter updates are given by

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \tag{2.5}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \tag{2.6}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}, \tag{2.7}$$

$$\theta_{t+1} = \theta_t - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}, \tag{2.8}$$

where m_t and v_t are the exponentially decaying averages of the gradient g_t and its squared value, respectively; β_1 and β_2 are decay rates (typically 0.9 and 0.999); η is the initial learning rate; ε is a small constant for numerical stability (typically 10^{-8}); and θ_t represents the model parameters at step t. This formulation enables efficient convergence by scaling updates based on gradient magnitude, making it robust to noisy or sparse gradients common in deep learning tasks.

2.3.3 Cosine Annealing Learning Rate

The Cosine Annealing Learning Rate (CALR) scheduler [27], implemented with the PyTorch framework [28], is defined as

$$\eta_{t+1} = \eta_{\min} + (\eta_t - \eta_{\min}) \cdot \frac{1 + \cos\left(\frac{(T_{\text{cur}}+1)\pi}{T_{\max}}\right)}{1 + \cos\left(\frac{T_{\text{cur}}\pi}{T_{\max}}\right)} \tag{2.9}$$

where η_t represents the learning rate at step t, $\eta_{\min}$ is the minimum learning rate, T_{cur} denotes the number of epochs since the last restart, and $T_{\max}$ is the maximum number

of epochs in a cycle. This scheduler adjusts the learning rate dynamically during training by following a cosine function, which allows the learning rate to decrease from an initial value η_t to a minimum value $\eta_{\min}$ over the course of a training cycle $T_{\max}$. As T_{cur} increases, the cosine term smoothly reduces the learning rate, reaching its lowest point at the end of the cycle, promoting finer optimization as the model converges. The process can restart with a higher learning rate after each cycle, enabling the model to escape local minima and continue exploring the loss landscape, thus enhancing training stability and performance. It is very important to mention that the learning rate with CALR is dependent on the total number of epochs set for the training. For example, in a 5-epoch run, the learning rate drops to its minimum by the end of epoch 5. In a 20-epoch run, the learning rate decreases more slowly, so it is higher at epoch 5 compared to the 5-epoch run. We will se this better in Chapter 3.

2.4 INFERENCE

Greedy Decoding In sequence generation, decoding algorithms select the most likely output sequence [1]. The simplest approach, greedy decoding, chooses the highest-probability word at each time step t, computed as:

$$\hat{w}_t = \arg\max_{w \in V} P(w|w_{<t}), \tag{2.10}$$

where V is the vocabulary. However, this strategy may lead to suboptimal sequences, as a high-probability choice at step t might prove bad at step $t+1$.

Beam Search To address this, beam search explores multiple possible sequences by maintaining a set of k hypotheses, where k is the beam width [1, 29, 30]. Unlike greedy decoding, beam search models decoding as a search through a tree, where branches represent token generation actions and nodes represent partial sequences (prefixes). In NMT, beam search approximates the maximum conditional probability $P(y|x)$ for a translation y given input x [31]. At each decoding step, beam search:

(1) Computes a softmax over the vocabulary for each of the k current hypotheses.

(2) Extends each hypothesis by generating all possible next tokens, yielding $k \times |V|$ candidates.

(3) Scores each candidate using the product of the current token's probability $P(y_i|x, y_{<i})$ and the probability of its preceding path.

(4) Prunes the candidates to retain only the top-k hypotheses, ensuring a fixed memory footprint.

Beam search generates candidate sequences by exploring multiple paths until all retained paths produce an end-of-sequence (EOS) token, at which point the highest-scoring path is selected as the final sequence. This approach balances local and global optimality, improving upon the "short-sightedness" of greedy decoding [1]. In the rare case of an empty stack, where no paths remain due to pruning or invalid sequences, the algorithm may terminate early, typically returning no valid sequence or a fallback option, such as the highest-scoring partial sequence.

2.4.1 FAIRSEQ AND PYTORCH FRAMEWORKS

In this thesis, we utilize two deep learning frameworks, PyTorch and Fairseq, to implement, train and perform inference with Transformer models for different tasks involving synthetic languages, as detailed in Chapters 3 and 4. **PyTorch** is used in Chapter 3 for the acceptor task, which involves determining whether a pair of strings from two synthetic languages represents a valid translation. While Fairseq is well-suited for sequence-to-sequence tasks, it lacks native support for Encoder-Only Transformer models, requiring customization. To address this, we implemented a custom Encoder-Only Transformer from scratch in PyTorch. The specific architectures employed are detailed in Section 3.3. **Fairseq** is employed in Chapter 4 for the machine translation task, where the model translates between pairs of synthetic languages. Fairseq provides a flexible interface for constructing Transformer models and adjusting their hyperparameters directly via command-line arguments, facilitating the execution of multiple experiments. Detailed descriptions of the specific architectures used are provided in Section 4.3, with further details on inference procedures discussed in Section 4.3.3.

2.4.2 METRICS FOR MACHINE TRANSLATION

Several metrics have been developed to evaluate MT systems, with *chrF* [32], *BLEU* [33] and *METEOR* [34] being among the most important ones. This thesis adopts the BLEU metric, as discussed in Chapter 4, due to its automatic computation within `fairseq`, the training framework employed.

BLEU BLEU is an automatic evaluation metric that measures how similar a machine-generated translation is to one or more human reference translations. It was introduced

by Papineni et al. [33] and remains one of the most widely used metrics in machine translation evaluation. Unlike human evaluation, which is time-consuming and expensive, BLEU provides a fast and automatic way to assess translation quality. The BLEU metric operates on several fundamental principles. The first important component of BLEU metric is the computation of *n-gram precision*. BLEU examines 1-grams (individual words), 2-grams (word pairs), 3-grams (three consecutive words), and 4-grams (four consecutive words). The second most important component is *modified precision*, which prevents cheating the system. Without this modification, a translation could achieve high scores by simply repeating high-frequency words from the reference. The third component is the *brevity penalty*, which addresses the tendency of precision-based metrics to favor shorter translations. Since precision naturally increases when the denominator (total n-grams in candidate) decreases, very short translations could achieve artificially high scores. The brevity penalty specifically penalizes translations that are shorter than the reference translations.

2.5 Formal Languages

Formal languages provide a mathematical framework for defining and analyzing the structure of languages, both natural and artificial (or synthetic). The Chomsky hierarchy, introduced by Chomsky [35], classifies languages into four types: regular, context-free, context-sensitive, and recursively enumerable. This classification is based on their generative grammars and corresponding automata. This hierarchy serves as a foundation for studying the expressive power of Transformers in learning synchronous context-free grammars, as explored in this thesis.

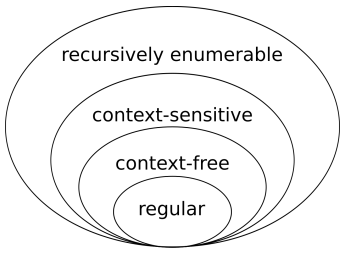


Figure 2.1: The Chomsky Hierarchy

2.5.1 SYNCHRONOUS CONTEXT FREE GRAMMARS

Synchronous Context-Free Grammars (SCFGs) are rooted in the theory of compilers, where they are called syntax-directed translation schemata (SDTS) where they are used to model compilers for programming languages [36, 37].

SCFGs generalize Context-Free Grammars (CFGs) to generate two strings simultaneously and they can be used to generate one sentence as the direct translation of the other. SCFGs consists of three main parts: a CFG for the source language, a CFG for the target language, and a pairing relation that links the rules and nonterminals of the two grammars. Each pair of rules, called a *synchronous production*, connects *nonterminals* through *matching indices* (denoted in curly brackets as in Section 2.5.2). When a synchronous production is applied, it rewrites both nonterminals (with same matching index) together, and the pairing relation is updated to maintain alignment. Formally, let G be an SCFG and w a string in the source language:

- The **translation relation** $T(G)$ is the set of all possible $[u, v]$ pairs that the grammar G can produce, defined as in Equation (2.11).
- The **image** of w, written $T(w, G)$ is the set of all targets v that are translations of w, defined as in Equation (2.12).

$$T(G) = \{[u, v] \mid [S^{(1)}, S^{(1)}] \Rightarrow_G^* [u, v]\} \tag{2.11}$$

$$T(w, G) = \{v \mid [w, v] \in T(G)\} \tag{2.12}$$

An example of SCFG is given by Chiang [38] to present an English-Japanese synchronous grammar:

$$S \rightarrow \text{NP}\{1\}\ \text{VP}\{2\}//\ \text{NP}\{1\}\ \text{VP}\{2\} \tag{2.13}$$

$$\text{VP} \rightarrow \text{V}\{1\}\ \text{NP}\{2\}//\ \text{NP}\{2\}\ \text{V}\{1\} \tag{2.14}$$

$$\text{NP} \rightarrow \text{i}//\ \text{watashi} \tag{2.15}$$

$$\text{NP} \rightarrow \text{the box}//\ \text{wa hako} \tag{2.16}$$

$$\text{V} \rightarrow \text{open}//\ \text{akemasu} \tag{2.17}$$

The SCFG rules use the following notation:

- **Nonterminals**: Represented by uppercase alphabet literals. In the previous example, symbols such as NP, VP, and V represent nonterminal categories, which

are placeholders for syntactic structures in the grammar.
In this thesis we use uppercase alphabet letters such as S (default start symbol), $A, B, C, \ldots, Z$.

- **Terminals**: In the example above, English words (e.g., `i`, `the box`, `open`) and Japanese words (e.g., `watashi`, `wa hako`, `akemasu`) represent terminal symbols, the basic units of the source and target languages.
  In this thesis we use lowercase alphabet letters such as $a, b, c, \ldots, z$.

- **Left-Hand Side (LHS)**: A single nonterminal (e.g., S) preceding the $\rightarrow$ operator.

- **Production rule operator ($\rightarrow$)**: The $\rightarrow$ operator, as in Context-Free Grammars (CFGs), connects a single nonterminal on the left-hand side (LHS) to its production on the right-hand side (RHS).

- **Matching Indices**: Matching indices (e.g., $\{1\}$, $\{2\}$) are used in SCFGs to indicate the correspondence between nonterminals in the source and target component of a same production. These indices ensure that paired nonterminals (e.g., NP$\{1\}$ and VP$\{2\}$) maintain their alignment across the synchronous production, allowing for reordering or structural differences (e.g., NP$\{1\}$ VP$\{2\}$//VP$\{2\}$ NP$\{1\}$). This mechanism enables SCFGs to model translations where the order of constituents may vary between languages while preserving syntactic relationships.

- **Synchronous production separator**: The // symbol separates the synchronous production into two components: the *source* production (left) and the *target* production (right), defining the alignment between the source and target languages.

- **Right-Hand Side (RHS)** //: Defines a synchronous production, consisting of a *Source* production and a *Target* production, separated by "//". Each component of the synchronous production is either:

  - **Terminal Production**: Features a single terminal on the *Source* side and a corresponding single terminal on the *Target* side, e.g., $A \rightarrow a//e$.
  - **Recursive Production**: Includes two nonterminals on the *Source* side and the same two nonterminals on the *Target* side, with possible order inversion indicated by index annotations (e.g., $S \rightarrow A\{1\}B\{2\}//B\{2\}A\{1\}$), where indices $\{1\}$ and $\{2\}$ ensure alignment between corresponding nonterminals across the *Source* and *Target*.

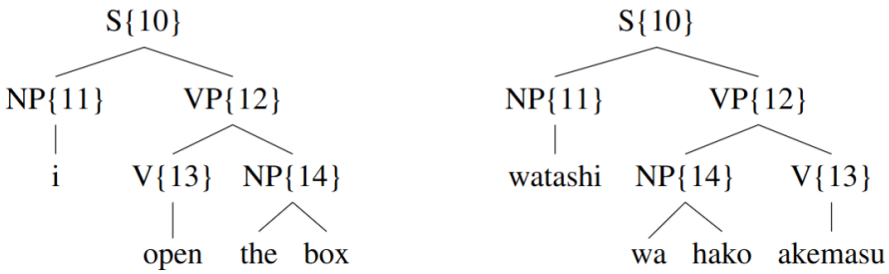


Figure 2.2: Synchronous parse trees for the English sentence “i open the box” and its Japanese translation “watashi wa hako akemasu”.

An SCFG produces pairs of strings or parse trees as shown in Figure 2.2, representing the source sentence and its translation. This makes SCFGs suitable for the translation experiment in Chapter 4, where a Seq2Seq Transformer learns to align source and target sequences, and for the parsing experiment in Chapter 3, where syntactic structures are analyzed.

2.5.2 DEFINED SCFGS

For the purposes of our experiment we defined three grammars: G, G_6 and G_{14}. The first grammar G is the main grammar used in both experiments, while the other two grammars G_6 and G_{14} are used only in the parsing experiment presented in Chapter 3 as negative examples. All three grammars are Synchronous Context-Free Grammars (SCFGs) that generate pairs of strings in a source language and a target language. Each terminal in the source language maps to only one terminal in the target language at each production, preserving equal token counts of length n in both produced sentences. The grammars are provided as Python strings and parsed using the `fromstring` function, following the methodology employed by the NLTK library [39]. These strings adhere to a well-defined structure, as outlined in Section 2.5.1.

(1) G: (manually crafted)

```
S -> A{1} B{2} // B{2} A{1}
A -> A{1} B{2} // A{1} B{2}
A -> C{1} F{2} // C{1} F{2}
B -> B{1} F{2} // F{2} B{1}
B -> D{1} A{2} // D{1} A{2}
C -> C{1} D{2} // D{2} C{1}
C -> F{1} B{2} // B{2} F{1}
```

```
D -> F{1} A{2} // F{1} A{2}
D -> D{1} C{2} // D{1} C{2}
F -> D{1} B{2} // B{2} D{1}
F -> F{1} C{2} // C{2} F{1}
A -> a // e
A -> b // f
A -> a // g
B -> b // f
B -> a // e
C -> c // g
C -> d // f
D -> d // h
D -> c // g
F -> c // g
F -> a // h
```

(2) G_6 (also called 'anti-grammar', manually crafted):

```
S -> A{1} B{2} // B{2} A{1}
A -> F{1} E{2} // F{1} E{2}
A -> C{1} F{2} // C{1} F{2}
B -> A{1} F{2} // F{2} A{1}
B -> D{1} A{2} // D{1} A{2}
C -> D{1} D{2} // D{2} D{1}
C -> B{1} B{2} // B{2} B{1}
D -> F{1} A{2} // F{1} A{2}
D -> S{1} C{2} // S{1} C{2}
F -> D{1} B{2} // B{2} D{1}
F -> F{1} C{2} // C{2} F{1}
E -> D{1} C{2} // D{1} C{2}
E -> F{1} A{2} // A{2} F{1}
A -> d // e
A -> c // f
A -> b // g
B -> a // f
B -> c // e
```

```
C -> d // g
C -> d // h
D -> c // h
D -> c // g
F -> a // g
F -> b // e
E -> b // h
```

(3) G_{14}: (generated at random as described in Section 2.6)

```
S -> N{1} Y{2} // N{1} Y{2}
Y -> A{1} R{2} // R{2} A{1}
N -> D{1} N{2} // N{2} D{1}
R -> S{1} A{2} // S{1} A{2}
D -> N{1} R{2} // N{1} R{2}
N -> D{1} R{2} // R{2} D{1}
D -> R{1} N{2} // N{2} R{1}
R -> Y{1} D{2} // Y{1} D{2}
N -> A{1} N{2} // N{2} A{1}
A -> Y{1} N{2} // Y{1} N{2}
N -> R{1} A{2} // R{1} A{2}
A -> R{1} R{2} // R{1} R{2}
Y -> R{1} Y{2} // Y{2} R{1}
D -> A{1} R{2} // R{2} A{1}
Y -> k // m
R -> k // s
A -> d // m
Y -> d // t
D -> l // p
D -> i // s
A -> i // v
Y -> k // s
A -> h // u
Y -> l // v
```

Manually Crafted Grammars with Shared and Modified Productions The manually crafted grammars G and G_6 share identical terminals and some identical productions, while certain productions in G_6 are modified by swapping nonterminals relative to G. This design choice creates an "anti-grammar" in G_6, which generates negative examples — sentences not licensed by G — for the dataset G+G_6 described in Section 3.2.1.

Overlap Between G and G_6 There is a small overlap in the languages generated by G and G_6, as some sentences are valid in both grammars. However, the modifications in G_6 ensure that the majority of the sentences generated by G_6 are not valid in G, and the fraction of sentences that belongs to both grammars are filtered out during dataset construction thanks to the deterministic Parser that we explain later in Section 2.7.

2.6 Implemented Algorithms for SCFGs

For our experiment we needed to implement three algorithms related to SCFGs: (1) an algorithm for generating random SCFGs, (2) an algorithm for generating random sentences from a given SCFG, and (3) an algorithm for generating random sentence pairs as negative samples. The implementations of these algorithms are described in the following paragraphs.

Algorithm for Random Grammar Generation This algorithm generates random Synchronous Context-Free Grammars (SCFGs) for the experiments described in Chapters 3 and 4. It takes as input: the number of nonterminals (`nt`), terminals (`t`), productions (`p`), and the start symbol (default is "S"). The algorithm creates `nt` nonterminals, named `nt1` to `nt_nt`, shared equally between the *Source* and *Target* grammars, ensuring a common set of nonterminals. Additionally, it generates `t` terminals, which are randomly assigned to either the *Source* or *Target* grammar; while in principle terminals are allowed to be shared between both grammars, this assignment ensures a distinct distribution for the experiments. The algorithm then produces `p` synchronous productions, each consisting of either two nonterminals or one terminal. When a production includes two nonterminals, they are paired with matching indices, permitting inverted positions between the *Source* and *Target* grammars. When a production contains one terminal, it is randomly selected from the respective *Source* or *Target* terminal sets. Finally, the algorithm verifies that all nonterminals and productions are reachable from the start symbol S. If any are unreachable, the grammar is discarded, and a new one is generated.

The advantage of having randomly generated grammars is to test the robustness of the models against different grammar structures, making it automatic to generate multiple grammars with different characteristics: more productions (which means more recursion), more terminals, etc.

Algorithm for Sentence Generation from Grammar The algorithm for generating random sentences from grammars extends an existing Context-Free Grammar (CFG) implementation in the NLTK library [39]. For our experiment the grammars provided to this algorithm are G, G_6, and G_{14}; they are given in input as strings and parsed into synchronous production objects, which store parallel *Source* and *Target* rules with positional indices. The generation process is controlled by an initial depth parameter, set to a maximum of 100 in this implementation, limiting the recursion depth and influencing the maximum achievable sentence length. The algorithm iteratively selects nonterminal productions using a base probability factor (p_{factor}) that adjusts dynamically: as the recursion depth decreases, the adjusted probability (given by: $p_{\text{adjusted}} = p_{\text{factor}} + (1.0 - p_{\text{factor}}) \cdot (1 - (\text{depth}/\text{initial\_depth}))$) increases, favoring terminal productions over expandable ones. This depth-dependent mechanism ensures termination and results in a class imbalance in the generated dataset, where shorter sentences (e.g., lengths 14 to 50) are produced in thousands of samples, while longer sentences are generated in fewer hundreds (as clearly visible in Figure 3.1). Finally, the terminal nodes are linearized into *Source-Target* string pairs $[u, v]$.

Algorithm for Random Sentence Generation for Negative Samples The algorithm generates synthetic negative sentence pairs for our SCFG experiments of Chapter 3 and Chapter 4 to train models to distinguish valid grammatical structures from random noise. Specifically, it generates the "*rand*" and "$rand_{14}$" components of the respective datasets G+rand and G_{14}+rand$_{14}$, presented in Section 3.2.1, by producing random noise proven to be outside the language defined by the respective grammars. It takes an integer for the number of pairs and a list of valid (source, target) sentence pairs to mimic their length distribution. It defines source and target vocabularies from the input pairs. For each pair, it samples a `sentence_length` from the valid sentences' distribution (computed as word counts). It then creates source and target sentences by randomly selecting words from their respective vocabularies up to `sentence_length` length. The output is a list of (source_sentence, target_sentence) tuples. Finally, all generated pairs are filtered by the SCFG Parser (described in the following Section 2.7) to remove any unintentionally valid pairs.

2.7 SCFG PARSING

SCFGs parsing extends the recognition and parsing problems of CFGs. The recognition problem determines whether an SCFG can generate a pair of strings $[w, w']$. The parsing problem builds a compact parse forest containing all tree pairs that the SCFG produces for $[w, w']$ [40]. Using dynamic programming, recognition and parsing are closely linked, as recognition steps can be stored to create the parse forest through memoization. Recognizing a pair $[w, w']$ takes polynomial time, depending on the grammar's rule structure and reordering patterns [40]. To reduce complexity, practical machine translation systems often use binary rules, with no more than two nonterminals on the right-hand side (RHS), as they are effective and easier to process [41], though this comes at the cost of reduced expressivity. This thesis uses binary rules, as defined in Section 2.5.2, for the experiments in Chapter 3 and Chapter 4.

Algorithm 1 Sync-CYK Parsing Algorithm

for all i, j, i', j' **do**
  $c[i, j, i', j'] = \emptyset$
for all axioms $[A, i, j, i', j']$ **do**
  add $[A, i, j, i', j']$ to $c[i, j, i', j']$
for $m = 1$ **to** $n + n$ **do**
  for all i, k, i', k' s.t. $k - i + k' - i' = m$, $0 \leq i \leq k \leq n$, $0 \leq i' \leq k' \leq n$ **do**
    for nonterminal symbols $A \in V$ **do**
      if $[A, i, k, i', k']$ can be proven from items in c **then**
        add $[A, i, k, i', k']$ to $c[i, k, i', k']$
if $[S, 0, n, 0, n] \in c[0, n, 0, n]$ **then**
  accept $\langle w, w' \rangle$

Implementation The SCFG Parser, also known as the Bi-Text Parser [38], takes three inputs: a synchronous grammar in Chomsky Normal Form (CNF)—ensured by the pre-binarized grammars seen in Section 2.5.2—, a source sentence $w = w_1 \cdots w_n$, and a target sentence $w' = w'_1 \cdots w'_n$. Multiple valid translations may exist for a single source sentence, and the parser accepts any valid pair $[w, w']$. It outputs **True** if the pair $[w, w']$ is derivable from the grammar, or **False** if it is not. To do this, the parser follows the Cocke-Younger-Kasami (CYK) algorithm adapted to SCFG parsing as described by Chiang [38] in Algorithm 1. The parser first checks if all characters in w and w' are valid terminals in the grammar. It then loops over spans of increasing size in both strings, trying to derive nonterminals for each span pair. The parsing table is filled

bottom-up, starting with smaller spans and using their results to compute larger spans. To avoid recomputation, the algorithm checks if a nonterminal is already in the table. The implementation was tested thoroughly with `unittest` framework, verifying both positive and negative cases across various input dimensions to ensure correctness.

2.7.1 METRICS FOR TRANSFORMERS' PARSING OF SYNTHETIC SCFGS

To evaluate the performance of SCFGs for parsing, several metrics are used to assess the model's effectiveness. These metrics, namely accuracy, precision, recall, and F1 score, are derived from the confusion matrix, which categorizes predictions into true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) [42]. Each metric provides distinct insights into the model's performance, especially in the context of class imbalance cited in Section 2.6, where shorter sentences with lengths 14 to 50 have thousands of samples, while longer sentences have only hundreds. This imbalance makes precision, recall, and F1 score particularly informative for the Variance experiment using Bootstrap sampling which we will see in Section 3.6.3.

Accuracy Accuracy measures the proportion of correct predictions, both positive and negative, out of all predictions made by the model. It is defined as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2.18}$$

This metric is the most intuitive, answering the question of how often the parser is correct. It performs well for balanced datasets but can be misleading with imbalanced classes. In this thesis, where shorter sentences dominate, high accuracy may reflect performance on the majority class of shorter sentences rather than overall parsing quality. Accuracy treats all errors equally, which may not be suitable for applications where specific error types are more critical.

Precision Precision evaluates the accuracy of positive predictions, indicating how many predicted positive parses are actually correct. Its formula is:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.19}$$

Precision answers the question of how often the parser is correct when it predicts a positive parse. A high precision indicates a low false positive rate, which is crucial when incorrect positive predictions are costly, such as in SCFGs parsing.

Recall Recall, also known as sensitivity, measures the proportion of actual positive instances correctly identified by the model. It is defined as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.20}$$

Recall tells how many actual positive parses were correctly identified. A high recall indicates a low false negative rate. It can be high even if many false positives occur.

F1 Score The F1 score is the harmonic mean of Precision and Recall and provides a balanced evaluation metric, useful when there is an uneven class distribution.

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.21}$$

The F1 score balances the trade-off between precision and recall, making it suitable when both metrics are equally important. It ranges from 0 (worst) to 1 (best), with the harmonic mean penalizing extreme values more than the arithmetic mean. Given the class imbalance in this project, where longer sentences are much less represented than shorter ones, accuracy alone may be misleading and inflated by performance on the majority classes.

2.7.2 Variance and Bootstrap Resampling

Bootstrap resampling is a statistical method designed to estimate the variability and statistical significance of performance metrics by generating multiple virtual test sets from an existing dataset. In the context of evaluating Transformers at SCFGs parsing, this method is particularly valuable for addressing the variance inherent in performance metrics of Section 2.7.1, especially when dealing with the class imbalance described. The method, formalized by Koehn [43], involves repeatedly sampling with replacement from the test set to create new ones. Formally, given a test set of size n, bootstrap resampling creates B new virtual test sets, each of size n, by randomly drawing samples with replacement from the original dataset. In each resampled set, some samples appear multiple times, while others are omitted, simulating variability in the data. For each virtual test set, performance metrics are computed, yielding a distribution of values. A 95% confidence interval for all metrics of Section 2.7.1, is estimated by sorting these values and discarding the top and bottom 2.5%, leaving the middle 95% as the interval, which shows the likely range of the true value and its stability.

In this thesis, bootstrap resampling is applied to compare the performance of two Ac-

ceptor models from Chapter 3 with different parameters, specifically focusing on their ability to parse sentences of varying lengths. The method involves sampling approximately 59,000 unique indices from the test set with replacement over 10,000 iterations, with around 34,000 indices omitted in each iteration due to the sampling process. For each iteration, accuracy, precision, recall, and F1 score are calculated based on binary predictions derived from test set results. Two-tailed p-values are computed to assess the statistical significance of performance differences between the two models, determining whether one consistently outperforms the other across resampled test sets. Bootstrap resampling ensures that performance differences between two different models are not a result of random variations in the test data, enabling a more precise evaluation of SCFG parsing effectiveness across diverse sentence lengths.

3

Acceptor Experiment

"The limits of my language mean the limits of my world." [1]

As anticipated in Chapter 2, this experiment investigates whether a Transformer can classify a pair of input sentences as licensed by a SCFG [38] (for details see Section 2.5). The Acceptor Transformer, as used in this chapter (see Section 2.2), is an Encoder-Only architecture designed for binary classification. Given a pair of sentences in the source and target languages, it outputs a special token indicating acceptance (if the pair conforms to the target grammar), or rejection (otherwise). The experiment underwent extensive analyses, modifications, and iterations. It is, in fact, the most significant experiment in this thesis project, as it aims to recreate the Parser presented in Section 2.7 but using Transformer models, therefore creating a neural alternative to deterministic parsers of complexity $\Theta(n^6)$ for SCFGs. A working neural acceptor could potentially revolutionize parsing for CFGs and SCFGs, making the task not only extremely fast but also scalable, avoiding the computational bottlenecks of traditional parsing algorithms.

The primary objectives of this experiment are multifaceted. Successful classification of sentence pairs determining whether they belong to the SCFG's language or not, could indicate the Transformer's ability to generalize beyond mere memorization of training data. Such a capability would suggest an understanding of the grammar's complex structure, including recursion and potential ambiguity. To put it simply, neural translation alone, even assuming that it works very well, would not fully demonstrate whether the Transformer has learned the underlying grammar, as it might rely on some

[1] Ludwig Wittgenstein, "Tractatus Logico-Philosphicus", 1922

token-to-token mapped memorization (a plausible scenario, given the small grammars used and described in Section 2.5.2) and some recurrent permutation-patterns (since they are constrained and limited by the production rules). The idea that a Transformer can classify a sentence pair $(w, w') \in L$ or $(w, w') \notin L$, where L is any synchronous language produced by the given SCFG (Section 2.5.1) is more powerful: it could provide stronger evidence that the model truly recognizes the grammar's boundaries, especially if proving its robustness on unfamiliar, much longer pairs of strings (w, w'), with respect to the ones that the model has learned during training. For this reason, this experiment has been investigated more thoroughly than the translation one (Chapter 4), and the most important findings of this project stem from it.

3.1 Experimental Setup

This section details the experimental setup for the Acceptor Transformer, including the computational environment, model architectures, and key variations tested.

3.1.1 Environment and Infrastructure

The experiments were conducted on the MetaCentrum[2] clusters provided by Charles University of Prague, differing from the setup used in Section 4.1.

- **OS**: Debian Linux (kernel 6.1.0–37-amd64, 64-bit x86_64), with SMP and PREEMPT_DYNAMIC support.
- **CPU**: Dual-socket Intel Xeon E5–2630 v4 (2.2 GHz base, 3.1 GHz turbo), 10 cores (40 logical threads), 64-bit architecture.
- **GPU**: NVIDIA GeForce GTX 1080 Ti (11 GiB VRAM), CUDA 12.8, driver v570.133.20.
- **RAM**: 125 GiB total; 15 GiB swap space.

3.1.2 Software and Dependencies

The experiments relied on the following Python (v. 3.11.11) packages:

[2] `https://metavo.metacentrum.cz/`

- `pip` 22.0.2
- `lightning` 2.5.0.post0 (PyTorch Lightning for training orchestration)
- `torch` 2.5.1 (PyTorch core)
- `tensorboardX` 2.6.2.2 (training metrics visualization)

3.2 Dataset Preparation

Each Encoder-Only Transformer architecture was sequentially trained on three distinct datasets, each designed to evaluate different aspects of grammatical acceptance. All datasets derive from the grammars defined in Section 2.5.2 and share the unified vocabulary described later in Section 3.3.3.

Dataset Composition

To guarantee splits to have 50% positive examples and 50% negative examples, three dataset combinations were constructed:

(1) ***G*-based Datasets:**

- ***G*+rand**: Pairs from grammar G versus randomly generated strings.
- G+G_6: Pairs from grammar G versus those from its modified variant G_6.

(2) **G_{14}-based Dataset:**

- $G_{14} + \textbf{rand}_{14}$: Pairs from the synthetic grammar G_{14} versus random strings.

All datasets have negative examples ([n] is the gold label) preceding positive examples ([Y] is the gold label); train set is shuffled at training time while validation and test sets maintain original ordering. The objective of all architectures is to train on around 200,000 samples. To achieve this, 200,000 samples are initially generated for both the Positive and Negative categories, then the data is split as follows: training (56%) (around 224,000 examples), validation (22%) and test (22%). To learn more about our sentence generation, please refer to Section 2.6, while to learn more about the grammars G, G_6 and G_{14} that we used, please refer to Section 2.5.2.

3.2.1 $G + rand$ DATASET CONSTRUCTION

The *G*+rand dataset ("*output_200k_random*" in the graphs) was constructed as follows:

(1) **Positive Data:** Generated 200k sentence pairs using grammar *G* (lengths 2–100) via the "Random Sentence Generator" algorithm (see Section 2.6).

(2) **Negative Data:** Produced 200k strings using same *G*'s vocabulary and a "Random Sentence Generator for Negative Samples" algorithm (see Section 2.6), following the length distribution of the positive data.

(3) **Sync-CYK Parsing:** Verified acceptance/rejection using the deterministic parser from Section 2.7. Parsing happens up to length 20, as "false-positives" have an exponentially decreasing probability for longer strings and they are extremely rare already at length 10.

(4) **Data Split:** Divided into three subsets:

- Training (56%) and validation (22%) with lengths 2–14.
- Test (22%) with lengths 15–100 for challenging, unseen data.

The resulting dataset is detailed below in Table 3.1. It is possible to see that from the 224,000 expected pairs in training data, we actually have 214,000 pairs as around 10,000 of them were removed by the parsing stage.
In Table 3.2 it is possible to see how each sentence pair is annotated with "[Y]" (accept) and "[n]" (reject), special tokens following processing through the deterministic parsing algorithm. The token assignment reflects the grammatical validity determined by the parsing procedure.

Split	**Length Range**	**Accepted**	**Rejected**	**Total**
Training	2–14	∼107,000	∼107,000	∼214,000
Validation	2–14	∼40,000	∼40,000	∼80,000
Test	15–100	∼44,000	∼44,000	∼88,000
Total	2–100	192,225	192,225	384,450

Table 3.1: Dataset splits for the *G*+rand configuration.

```
<CLS> a b <SEP> h f <LABEL> [n]
<CLS> a a <SEP> g g <LABEL> [n]
...
<CLS> a b c a a b a d d c c c d d
    <SEP> e g g f h g f f e g g h e h <LABEL> [n]
<CLS> b b a a a a a d b d c c c b
    <SEP> g e e g e e g e g f h f g f <LABEL> [n]
<CLS> b a <SEP> e f <LABEL> [Y]
<CLS> a b <SEP> f e <LABEL> [Y]
...
<CLS> a c a d c a d a a b a b a a
    <SEP> e g e h e g e e f f e e h e <LABEL> [Y]
<CLS> a c b a a c b b a a b d b a
    <SEP> h f e e g f f f e e f g e e <LABEL> [Y]
```

Table 3.2: Training set structure for *G*+rand dataset. For easier orientation, we show the examples sorted by length and with negative cases first; when training, the order is random.

3.2.2 G+G_6 DATASET

The G+G_6 dataset (labeled "*output_200k_g6*" in the graphs) follows the same processing pipeline and partitioning scheme as the *G*+rand dataset, with a change in the second stage: the generation of negative samples. Instead of sampling the negative examples from random strings (Section 2.6), we sample them from outputs of our "negative grammar" G_6 (Section 2.5.2). This setup tests the model's ability to distinguish a highly plausible grammar with subtle deviations from valid constructions, rather than simply detecting random noise as in the previous dataset.

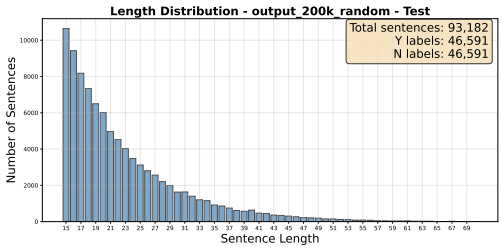
The G_6 grammar introduces deliberate changes to G's production rules, as detailed in Section 2.5.2. These changes include substituting some terminal symbols, reordering certain rules, and introducing controlled structural variations. However, the vocabulary remains the same, and sentences generated by G_6 are often very similar to those produced by G. The key difference lies in specific terminal swaps and rule permutations, which are critical for grammar recognition, as explained in Section 2.7. Even a single terminal swap or a different tree permutation can lead to the rejection of an input pair, despite its high similarity to sentences produced by the original grammar G. However, as noted in Section 2.5.2, G and G_6 similarity means that some sentences generated by G_6 might could actually belong to both grammars. Therefore these sentences are removed by the SCFG Parser described in Section 2.7 during the dataset construction.

3.2.3 $G_{14} + \text{RAND}_{14}$ DATASET

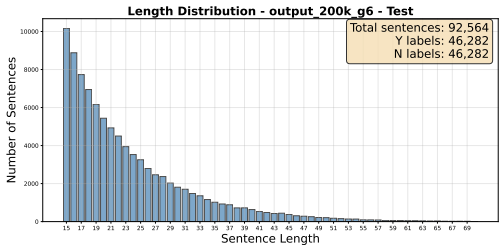
The $G_{14} + \text{rand}_{14}$ dataset (labeled "*output_g14_200000*" in the graphs) follows the same processing pipeline and partitioning scheme as the *G*+rand dataset, but replaces *G* with the more complex grammar G_{14} defined in Section 2.5.2. The negative samples are generated using the algorithm for random negative samples described in Section 2.6. The term "rand_{14}" indicates that these randomly generated negative pairs have the same sentence length distribution and vocabulary as original G_{14} generated sentence, consistent with the approach used for *G*+rand, as explained in Section 3.2.1.

3.2.4 SENTENCE LENGTH DISTRIBUTIONS FOR ALL DATASETS

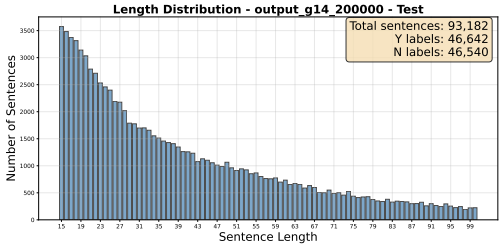
The sentence length distribution for the three datasets (*output_200k_random*, *output_200k_g6*, *output_g14_200k*) is shown below. Since the number of sentences with lengths between 70 and 100 is too low for the two *G*-related datasets, subsequent graphs will focus on sentence lengths ranging from 15 to 70.



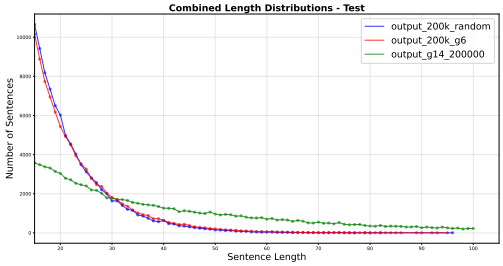
(a) Distribution of *G*+rand test set.



(b) Distribution of $G + G_6$ test set.



(c) Distribution of $G_{14} + \text{rand}_{14}$ test set.



(d) Combined distributions of all three datasets.

Figure 3.1: Test set distributions for *G*+rand, $G + G_6$, $G_{14} + \text{rand}_{14}$, and their combined view.

3.3 Model Configurations

In this work, we define **model architecture** as the structural design of a neural network, specifying its core components and their arrangement. For this experiment, an Encoder-Only Transformer architecture is employed, as detailed in Chapter 3, while a Seq2Seq architecture will be explored in subsequent Chapter 4. This design choice aligns with the binary classification nature of the acceptor task, where the model must evaluate sentence pairs rather than generate sequences.

Multiple Encoder-Only Transformer configurations were evaluated, with parameters ranging from only *275k* up to *13M*, to assess scalability and performance trade-offs. We define **model configuration** as the specific set of hyperparameters applied to a given architecture, determining its operational characteristics. We conducted an extensive hyperparameter search by training separate models for each combination of the following configurations:

- **Attention heads**: 8, 16, 32
- **Layers**: 1, 2, 3, 5
- **Model dimension**: 32, 64, 128, 256, 384, 512
- **Training epochs**: 1, 3, 5, 8, 12, 20

While this set of combinations theoretically yields 432 possible paths, practical constraints limited testing to around 200 configurations. The largest model actually evaluated was 3 layers with 512 dimension and 32 heads (13M parameters) — some extreme combinations (particularly 5-layer models at higher dimensions) were omitted due to time constraints. This selective evaluation still provided comprehensive coverage of the design space, revealing trends in the accuracy-complexity tradeoff while focusing computational resources on the most promising configurations.

3.3.1 Training Methodology

The experimental protocol (see Section 2.3) employed consistent training parameters across all configurations:

- **Optimizer**: Adam (lr = 1e-3 = 0.001)
- **Scheduler**: Cosine Annealing

- **Loss**: Cross-Entropy (CE) Loss
- **Batch size**: 256
- **Training duration**: 1–20 epochs (depending on configuration)
- **Gradient clipping**: 1.0 (norm)

Each model configuration undergoes sequential training across three distinct datasets (detailed later in Section 3.2) to systematically evaluate performance consistency under varying grammatical specifics (see Section 2.5.2). This multi-dataset approach serves the dual purpose of validating model robustness while calibrating task difficulty. The protocol's value lies in its ability to distinguish between general grammar understanding/handling and dataset-specific memorization, where consistent performance across all three datasets *might be* evidence of robust pattern recognition capabilities.

By maintaining identical training parameters across all configurations while varying training data reflecting various grammatical settings, the experiment isolates the impact of model hyperparameters choices on the model's ability to learn fundamental syntactic principles rather than surface-level features.

3.3.2 Variability and Randomness

To assess the robustness of our findings, we conducted reproducibility tests by training multiple instances of selected configurations (particularly the best-performing ones) using distinct random seeds. This approach served two key purposes: to determine whether performance trends remained consistent across different initializations and to quantify how sensitive model outcomes were to random initialization effects. All the results are presented in Section 3.6.

3.3.3 Model's Vocabulary

The Encoder-Only Transformer employs a unified vocabulary spanning all three grammars (G, G_6, and G_{14}), enabling direct processing of any dataset combination without architectural modification. This design capitalizes on the inherent token overlaps between grammars (visible in Section 2.5.2). The vocabulary includes five special tokens:

- `<CLS>`: Initial token marking the start of each input sequence

- `<SEP>`: Delimiter separating source and target language sentences in the input
- `<LABEL>`: Prefix token indicating the start of the ground truth classification
- `[Y]`: Positive class label (indicating the string pair $\in L$)
- `[n]`: Negative class label (indicating the string pair $\notin L$)

This joint vocabulary approach was chosen to ensure consistent embedding spaces across languages while simplifying the training pipeline. Though currently hard-coded for these specific grammars, the system could be adapted to other formal languages by modifying the vocabulary layer while preserving the core classification mechanism.

3.4 Results and analysis

As mentioned in Section 3.3, approximately 200 configurations were executed across all three previously described datasets: G+rand, G+G_6, and $G_{14} + \text{rand}_{14}$. Our objective was to identify a starting point in the model's hyperparameter space from which the models could begin to learn and generalize across the three datasets, and possibly to locate a point where the models started to overfit or ceased learning, merely memorizing the data.

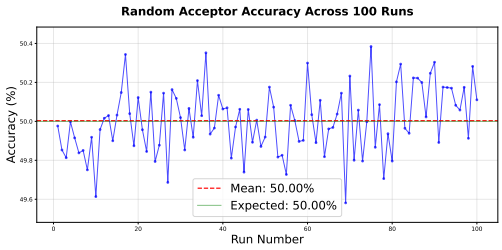
This task proved challenging and has not yielded linear or entirely interpretable results thus far. In general, smaller configurations demonstrated limited capability in accepting or rejecting strings effectively. Conversely, larger configurations consistently ranked among the top performances in our analysis. However, certain observations deviated from expectations: some larger configurations performed significantly worse than anticipated, and certain smaller configurations achieved performance comparable to larger ones. Other results remained difficult to interpret, particularly cases where models performed well on one dataset but poorly on others. We will examine all such results.

3.4.1 Random Guessing Baseline

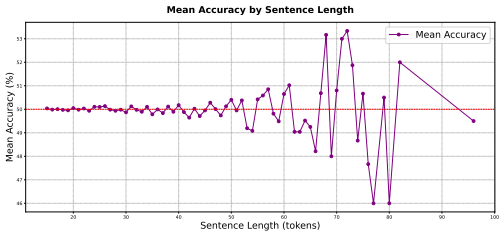
To contextualize our model's performance, we establish an elementary, deliberately simplistic baseline using random guessing, representing the most naive approach possible. We observe that a model failing to learn exhibits performance between 45% and 55% for two possible reasons: (1) it predicts all yes or all no, (2) it learns some patterns but fails to generalize and it essentially guesses randomly, consistently yielding near

50% accuracy.

In this binary classification task, a model randomly predicts whether a sentence pair belongs to language L or not ($\notin L$). Given a balanced dataset with a 50% chance of selecting the correct label per example, the probability of achieving 90% accuracy by chance is effectively zero. We validated this empirically by running a random acceptor 100 times on the test set, as shown in Figure 3.2. The plot confirms consistent convergence to the expected 50% accuracy, with observed accuracies tightly clustered between 49.6% and 50.4% (y-axis) across all runs (x-axis). Additionally, Figure 3.2b illustrates the mean accuracies of this naive random guesser across all sentence lengths in the test set, showing increased variance ($\pm \sim 1.5\%$) for very long sentences, but this happens only because the number of samples for very long sentences is much lower.



(a) Accuracies of the Random Guesser model across 100 independent runs.



(b) Random Guesser model mean accuracies across all lengths of test set.

Figure 3.2: Random Guesser Baseline.

3.5 TEST ACCURACIES

The performance results shown in Figures 3.3 to 3.6 need careful analysis due to our unique training approach. Instead of using checkpoints from one long training session, we ran multiple separate training sessions for each Transformer model (see Section 3.3), with training durations varying from 1 to 20 epochs. This is important because the learning rate at a specific epoch, like epoch 5, differs depending on the total number of epochs. This difference comes from our use of cosine annealing decay for the learning rate (see Section 2.3.3). With cosine annealing, the learning rate at a given epoch depends on the total number of epochs set for the training. For example, in a 5-epoch run, the learning rate drops to its minimum by the end of epoch 5. In a 20-epoch run, the learning rate decreases slower, so it is higher at epoch 5 compared to the 5-epoch run. This training method has two main goals: (1) it checks how consistent the models are across different training lengths; (2) it helps find the best number of epochs for each

model configuration. This approach was inspired by scaling law studies [44] to identify the best mix of model parameters (like attention heads, layer depth, and model size) and the ideal training duration.

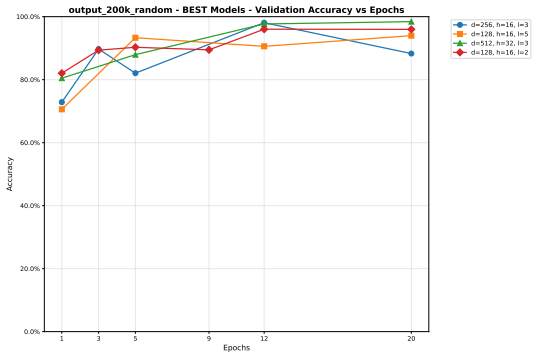
3.5.1 Performance Extremes in Model Evaluation

The terms "best" and "worst" models in this context describe relative performance boundaries within our experimental results, not absolute capabilities. A "best" model designation does not imply a complete solution to the Acceptor problem, just as a "worst" model classification does not indicate complete failure to learn the grammars defined in Section 2.5.2. These labels simply mark the upper and lower performance thresholds observed across our specific configuration space. Here, a "best model" refers to any configuration that achieved the highest percentage in at least one evaluation, *at any epoch*. The plots in Figures 3.3 and 3.4 demonstrate that these models exhibit varying performance across different epochs. As previously noted, results fluctuate over training: some models improve with additional epochs, others begin to overfit and degrade, while a subset displays unpredictable behavior whose underlying causes remain poorly understood. Nevertheless, at some point during training, these particular hyperparameter settings achieved notably high accuracy on the test set, a challenging evaluation benchmark since it consists exclusively of unfamiliar sentences. These test sentences are not only longer than those in the training set but may also contain novel recursive structures.

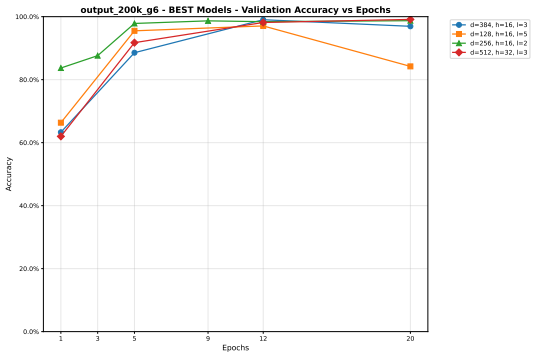
The comparison between validation and test set performance reveals striking differences. The validation set (comprising familiar, training-like data) shows largely predictable learning curves with recognizable growth patterns. In contrast, the test accuracy curves exhibit more erratic behavior. For instance, in dataset "*G*+rand" (Figure 3.4a), while some top-performing models like the dim=256, heads=16, layers=3 configuration follow validation trends closely, others display substantial divergence. Notably, the smallest configurations (e.g., dim=32/dim=64) demonstrate very unstable test performance.

3.5.2 In-Depth Model Analysis

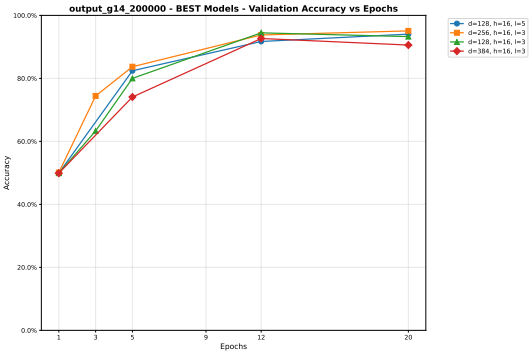
We now examine specific model configurations to better understand the patterns observed in Section 3.5.1. Our analysis focuses particularly on models demonstrating strong cross-dataset performance, as this may indicate genuine generalization capability across the SCFGs defined in Section 2.5.2.



(a) Validation accuracy trends for G+rand dataset showing mostly stable learning curves across different model configurations.

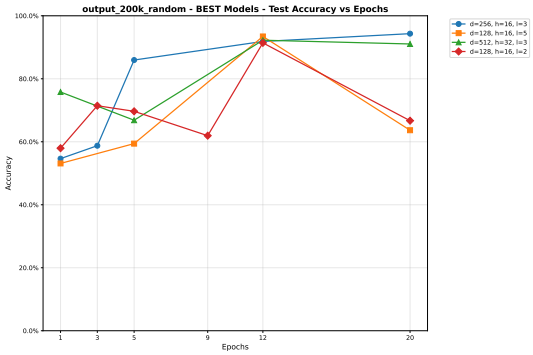


(b) Validation accuracy for G+G_6 dataset demonstrating more consistent performance patterns except for dim=128, heads=16, layers=5 which seems to underfit with longer training time.

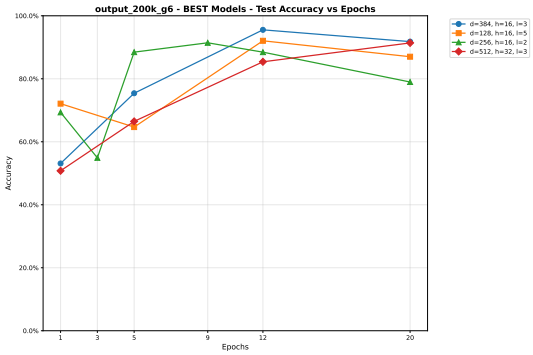


(c) Validation performance on $G_{14} + \text{rand}_{14}$ dataset showing comparable trends to other validation sets but with generally lower accuracy values — this is the hardest dataset.

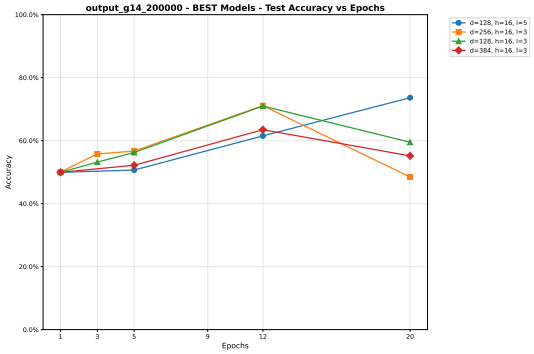
Figure 3.3: Validation accuracy trends for top models across the test-sets. Y-axis: accuracy, X-axis: epochs



(a) Test accuracy trends for G+rand dataset showing more erratic behavior compared to validation, with some models (e.g., dim=256, heads=16, layers=3) performing consistently while others (dim=128 configurations) exhibit instability.

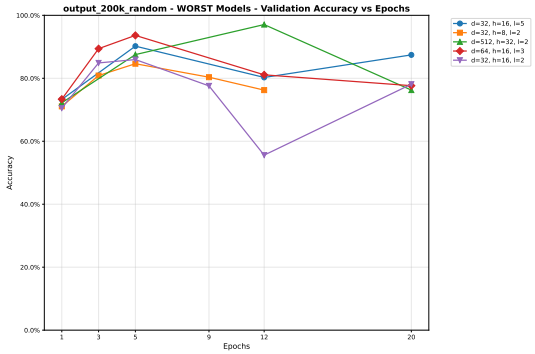


(b) Test performance on G+G_6 dataset revealing similar patterns of variance, with model performance diverging more significantly on unfamiliar test data.

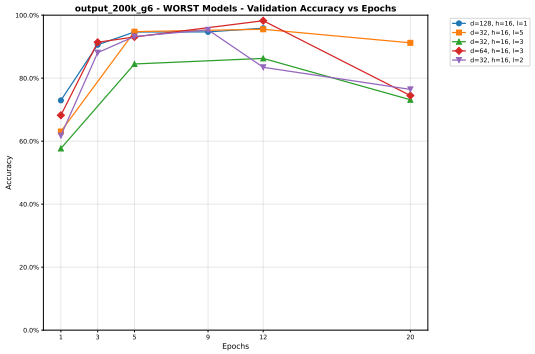


(c) Test accuracy for $G_{14} + \text{rand}_{14}$ dataset demonstrating not clear unpredictability nor performance fluctuations but generally much lower accuracy values than the "best models" found for the previous two datasets.

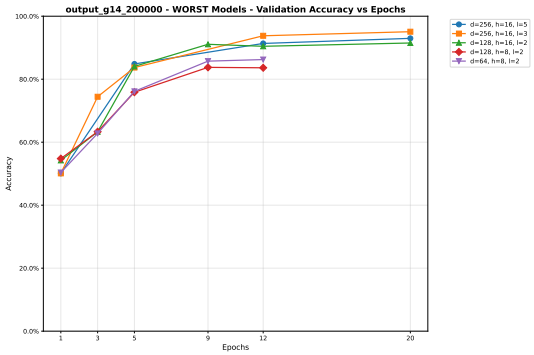
Figure 3.4: Test accuracy trends for top models. Y-axis: accuracy, X-axis: epochs



(a) Validation accuracy of worst-performing models on G+rand test dataset. Smaller configurations (dim=32/64) dominate this graph but validation curves remain relatively stable despite showing early signs of overfitting with increasing epochs, opposed to what is shown in the Figure 3.6.

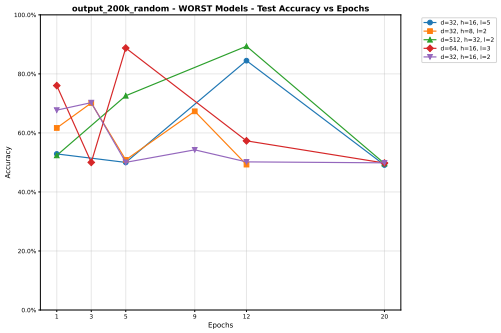


(b) Validation accuracy of worst-performing models on G+G_6 test dataset. Smaller configurations (dim=32/64) dominate also this graph but validation curves remain relatively stable, opposed to what is shown in the Figure 3.6.

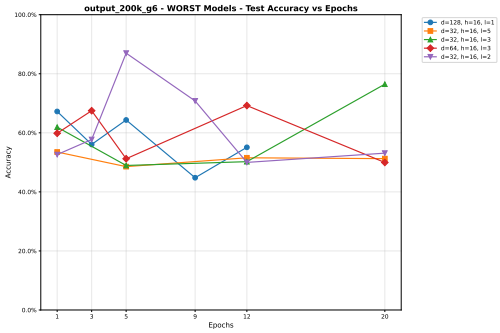


(c) Validation performance on $G_{14} + \text{rand}_{14}$ showing also some larger models (e.g., dim=256 and dim=128) among the worst performers. The validation curves are, also in this case, pretty reasonable.

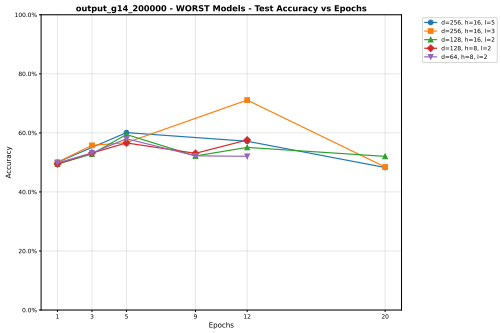
Figure 3.5: Validation accuracy trends for worst-performing models across test datasets, mostly smaller configurations. The y-axis shows the accuracy % and the x-axis the training epochs.



(a) G+rand test performance of worst-performing models, showing high variance across configurations due to limited capacity (dim=32/64). Partial pattern memorization causes sporadic high scores but fails to ensure stable generalization across epochs.



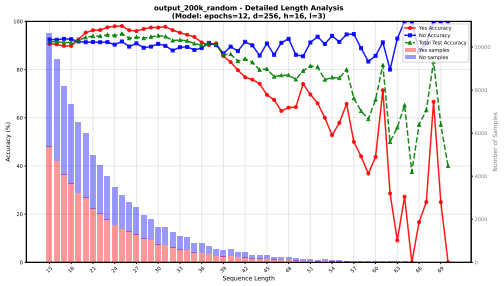
(b) G+G_6 test performance, exhibiting similar instability and high variance as Figure 3.6a due to poor generalization.



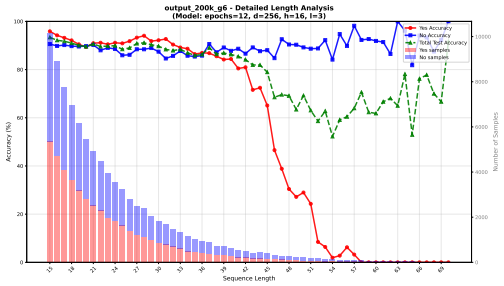
(c) $G_{14} + \text{rand}_{14}$ test performance, with most models stabilizing near 50% (random baseline) despite strong validation results. A dim=256 model peaks at 72% accuracy (12 epochs) but drops to 50% at epochs 1 and 20, indicating instability even in larger configurations.

Figure 3.6: Test accuracy trends for worst models. Y-axis: accuracy, X-axis: epochs

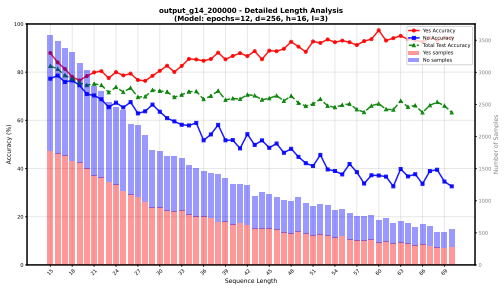
Configuration: dim=256, heads=16, layers=3, epochs=12 This configuration achieves the highest average accuracy across the three datasets. The accuracy performance per sentence length is shown in Figures 3.7a to 3.7c. For G+rand, the model maintains strong prediction capability (accuracy >90%) for sentences within 2.5–3 times the maximum length seen at training (consistent with prior research [45]), with a sharp decline beyond the breaking point at length 40. For G+G_6, it exhibits similar length-dependent accuracy patterns, with stabilized behavior at longer lengths, where acceptance and rejection rates diverge significantly beyond length 14, also breaking around length 40. For G_{14} + rand$_{14}$, a linear length-accuracy relationship emerges, where high acceptance accuracy at longer lengths likely reflects increased positive predictions rather than improved understanding, as rejection accuracy decreases linearly, suggesting prediction bias. This performance drop on longer sentences has minimal impact on overall accuracy due to their low representation in the dataset. This suggests that standard accuracy metrics offer limited insight into a model's true grammatical understanding, as they disproportionately reflect performance on frequent short sentences while masking difficulties with rarer but structurally important long sentences. The model achieves the best balance between acceptance and rejection accuracy across these datasets.



(a) G+rand test set performance.



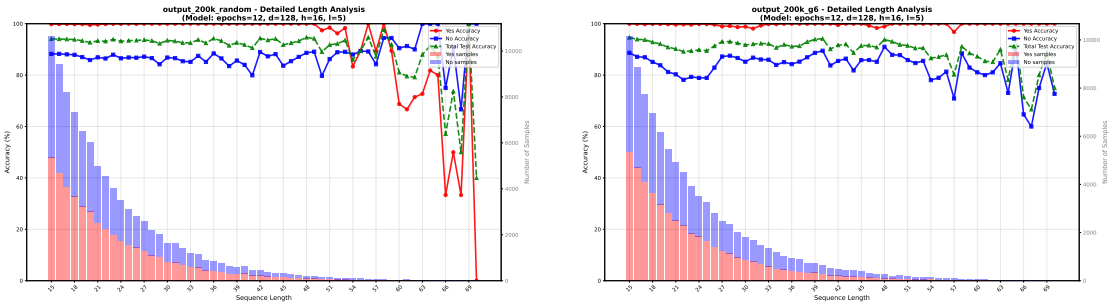
(b) G+G_6 test performance.



(c) G_{14} + rand$_{14}$ test performance.

Figure 3.7: Performance analysis across G+rand, G+G_6, and G_{14} + rand$_{14}$ test sets (dim=256, heads=16, layers=3, epochs=12). Y-axis: sentence length, left X-axis: accuracy (lines), right X-axis: number of samples per length bin (blue = rejected, red = accepted)

Configuration: dim=128, heads=16, layers=5, epochs=12 We analyze a top-performing model configuration on the G-based datasets: G+rand (using random noise) and G+G_6 (using antagonistic grammar G_6). This model achieves the highest average accuracy across both datasets, revealing distinct behavioral patterns compared to the best three-dataset model. For G+rand, as shown in Figure 3.8a, accuracy degrades with sentence length, with a breaking point at length 55, higher than the previous model's 40, indicating sustained predictive capability for longer sentences. For G+G_6, shown in Figure 3.8b, the model maintains stable accuracy across lengths, suggesting its configuration (dim=128, heads=16, layers=5) employs different strategies for random noise versus antagonistic grammatical patterns. It achieves near-perfect acceptance prediction (98–100%) and over 90% rejection accuracy for both datasets.

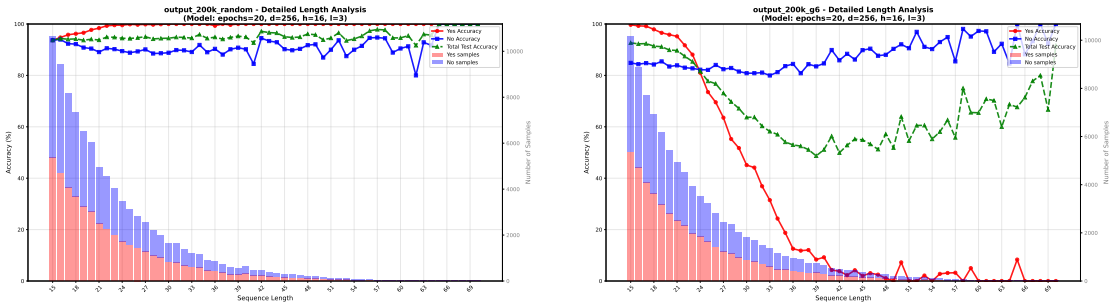


(a) G+rand test accuracy.

(b) G+G_6 test accuracy.

Figure 3.8: Performance analysis across G+rand and G+G_6 test sets (dim=128, heads=16, layers=5, epochs=12). Y-axis: sentence length, left X-axis: accuracy (lines), right X-axis: number of samples per length bin (blue = rejected, red = accepted)

Configuration: dim=256, heads=16, layers=3, epochs=20 As noted earlier, some models excel on certain datasets but perform poorly on others. This model, previously identified for top performance on G+rand and high accuracy on the challenging $G_{14} + \text{rand}_{14}$ dataset in Figure 3.4, unexpectedly shows poor accuracy on G+G_6. For G+rand, as shown in Figure 3.9a, it maintains stable performance across sentence lengths. However, for G+G_6, illustrated in Figure 3.9b, accuracy degrades rapidly with increasing sentence length, making interpretation challenging. These contrasting behaviors highlight the model's inconsistent generalization across datasets.



(a) G+rand test accuracy.

(b) G+G_6 test accuracy.

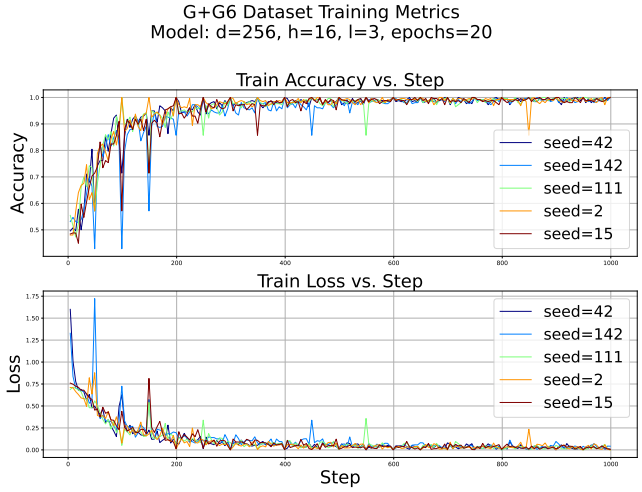
Figure 3.9: Performance analysis across G+rand and G+G_6 test sets (dim=256, heads=16, layers=3, epochs=20). Y-axis: sentence length, left X-axis: accuracy (lines), right X-axis: number of samples per length bin (blue = rejected, red = accepted)

3.6 Studies on Variance

As anticipated in Section 2.7.2 of Chapter 2, studying the variance of a model is crucial in scenarios where performance stability is a priority. Our previously described methodology for training, originally designed also with the intent to investigate the right number of training epochs to use, inadvertently provided insights into training variance by training identical configurations with varying epoch counts. This approach revealed how model performance fluctuates across training runs. However, in this section the focus shifts to testing variance, which captures the variability in model performance due to the randomness in test set sampling. To explore this, we propose the following experiments.

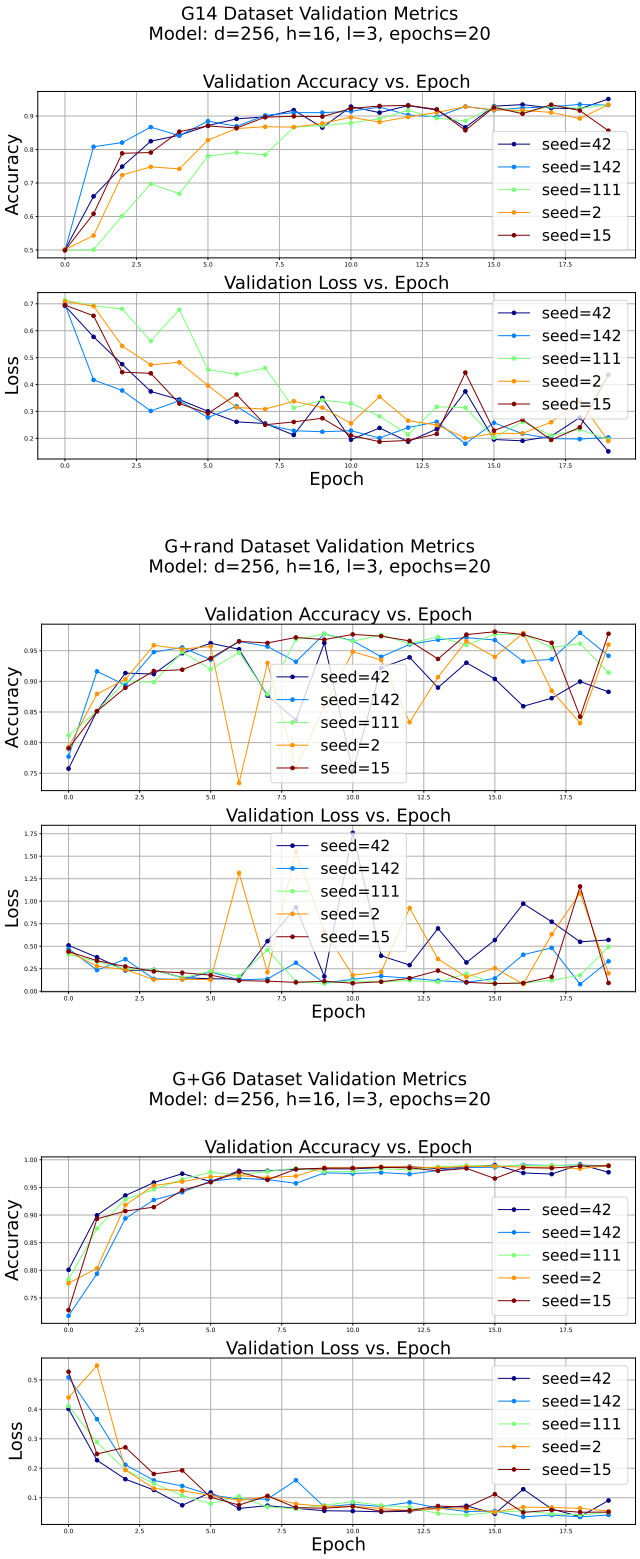
3.6.1 Analysis of Performance Variance Across Random Seeds for Train and Validation Sets

Let's start from an analysis of the test accuracies over 5 different seeds using two fixed configurations of the Transformer: dim=256, heads=16, layers=3, epochs=20. The results over the three different datasets are visible in Tables 3.3a to 3.3c.
As shown in Figure 3.10, the model exhibits variance across different random seeds, but the results are generally consistent. To investigate the presence of outlier results for certain seeds, we examine the training and validation curves in Figures 3.10a to 3.10c. These plots reveal that the learning curves are generally similar, with the exception of the more challenging G_{14} dataset, which consistently shows greater variance due to its increased learning complexity. In Figures 3.10b and 3.10c, we observe that both posi-



(c) Training metrics for G+G_6 dataset.

Figure 3.10: Training metrics across $G_{14} + \text{rand}_{14}$, G+rand, and G+G_6 datasets.



(c) Validation metrics for G+G_6 dataset.

Figure 3.11: Validation metrics across $G_{14} + \text{rand}_{14}$, G+rand, and G+G_6 datasets.

Seed	Test %	Loss
42	0.484	2.794
142	0.648	1.541
111	0.636	1.133
2	0.679	1.013
15	0.576	1.909

(a) Test metrics for $G_{14} + rand_{14}$ dataset.

Seed	Test %	Loss
42	0.943	0.144
142	0.881	0.468
111	0.918	0.273
2	0.939	0.191
15	0.684	1.401

(b) Test metrics for G+rand dataset.

Seed	Test %	Loss
42	0.818	0.883
142	0.931	0.242
111	0.897	0.273
2	0.942	0.181
15	0.942	0.172

(c) Test metrics for G+G_6 dataset.

Table 3.3: Test metrics for model (dim=256, heads=16, layers=3, epochs=20) across datasets, showing accuracy and loss variance by seed.

tive and negative peaks in the learning curves typically occur at similar steps across all seeds, indicating consistent behavior in the training dynamics. However, the $G+$rand dataset exhibits unusual behavior for certain seeds, notably seed=2 and seed=42, while other seeds (seed=142, seed=15, seed=111) maintain similar patterns. This suggests that variance starts happening in the validation set, to finally get higher on the test set — unseen and unfamiliar. For this specific configuration the increased variance is particularly evident in the $G+$rand dataset.

By examining the validation curves in Figures 3.11a to 3.11c, we observe that the behavior is generally consistent across various random seeds, particularly for the $G+G_6$ and, surprisingly, $G_{14}+\text{rand}_{14}$ datasets.

3.6.2 Analysis of Performance Variance Across Random Seeds for the Test Set

Let's analyze three different configurations trained using different random seeds—seed=42 ("Original") and seed=2 ("V2")—at initialization time of the model.

Configuration: dim=128, heads=16, layers=3, epochs=12 As shown in Table 3.4, the model exhibits high test variance for this configuration: $\sim 28\%$ on both G-based datasets and -5% on G_{14}-dataset. The tables reveal how training variance remains low, with validation variance around 2–3%, indicating stable performance on these sets. However, our previous guess of high test variance is again demonstrated. On output_200k_random we can see how "Original" did not learn (51%) while the new version "V2" reached higher results (around 80.5%); a similar pattern is observed on output_200k_g6, where "Original" scored 60.7% and "V2" achieved 88.5%. On the

more complex output_g14_200000 dataset, both models performed poorly, with "Original" at 71% and "V2" at 65.3%, showing a smaller variance of around 5%. The accuracy by sentence length for this configuration is shown in Figure 3.12 and Table 3.4, where we can see how the two models behave similarly on $G_{14} + \text{rand}_{14}$ dataset, while they show different behaviors on the other two datasets.

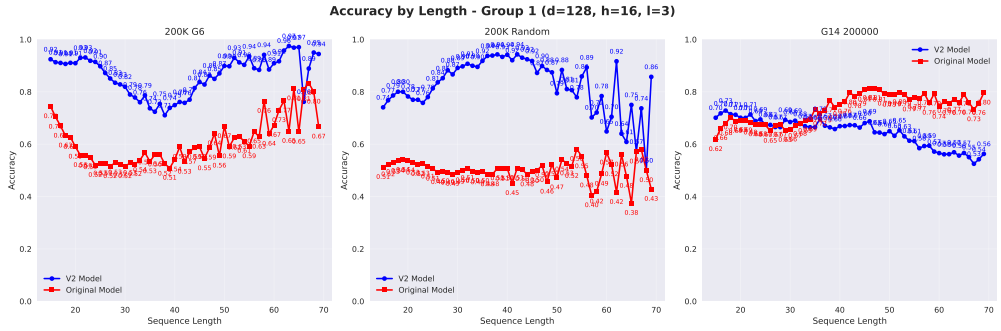


Figure 3.12: Accuracy by sentence length for small model (dim=128, heads=16, layers=3, epochs=12) across datasets. Y-axis: accuracy, X-axis: sentence length

Metric	V2	Original	Diff
train-acc	0.9775	0.9813	-0.0039
val-acc	0.8720	0.8940	-0.0220
test-acc	0.8052	0.5177	+0.2875

(a) Metrics for G+rand dataset.

Metric	V2	Original	Diff
train-acc	0.9896	0.9896	+0.0000
val-acc	0.9766	0.9755	+0.0011
test-acc	0.8847	0.6067	+0.2780

(b) Metrics for G+G_6 dataset.

Metric	V2	Original	Diff
train-acc	0.8894	0.9559	-0.0665
val-acc	0.9105	0.9446	-0.0341
test-acc	0.6534	0.7098	-0.0564

(c) Metrics for $G_{14} + \text{rand}_{14}$ dataset.

Table 3.4: Performance metrics for V2 and Original models across datasets, highlighting seed-induced variance.

Configuration: dim=128, heads=16, layers=5, epochs=12 By increasing the number of layers from 3 to 5, we obtain a model with 3.6 million parameters, instead of previous 2.4 million. This change significantly impacts the model's capacity and performance. We see the results in Table 3.5 showing again low variance on train, showing

0.3–6% of variance on validation and finally showing 2.6–10% of variance on test. We can visualize better how close test performances are in Figure 3.13.

Metric	V2	Original	Diff
train-acc	0.9771	0.9787	−0.0016
val-acc	0.9712	0.9058	+0.0654
test-acc	0.8339	0.9343	−0.1004

(a) Metrics for G+rand dataset.

Metric	V2	Original	Diff
train-acc	0.9853	0.9824	+0.0029
val-acc	0.9671	0.9706	−0.0035
test-acc	0.8536	0.9206	−0.0670

(b) Metrics for G+G_6 dataset.

Metric	V2	Original	Diff
train-acc	0.8831	0.9093	−0.0262
val-acc	0.8797	0.9172	−0.0375
test-acc	0.6411	0.6150	+0.0261

(c) Metrics for $G_{14} + \text{rand}_{14}$ dataset.

Table 3.5: Accuracy comparison for V2 and Original models across datasets, with difference (V2 − Original).

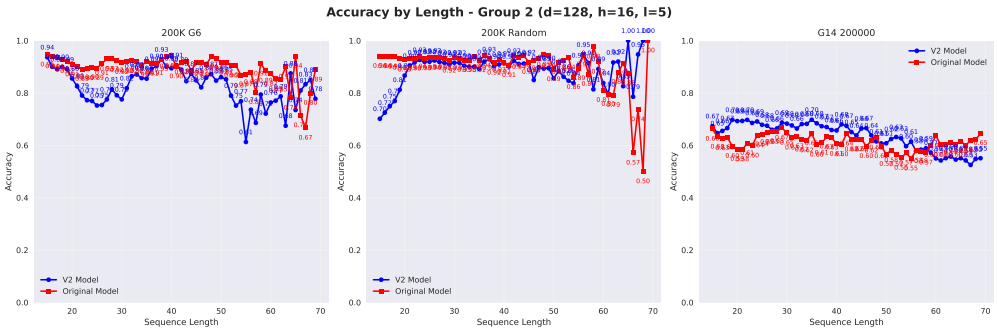


Figure 3.13: Accuracy by sentence length for model (dim=128, heads=16, layers=5, epochs=12) across datasets. Y-axis: accuracy, X-axis: sentence length.

Configuration: dim=256, heads=16, layers=3, epochs=12 This configuration, with 5.3 million parameters, was previously identified as a top performer on the $G + rand$ dataset (Figure 3.4). Consistent with prior models, it exhibits higher training variance (5.96%) on the G_{14} dataset but achieves stable performance on the $G + rand$ and $G + G_6$ datasets across both "Original" and "V2" seeds. Validation accuracy remains robust across all three datasets, while test performance is strong for $G + rand$ and $G + G_6$ but suboptimal for G_{14}, as expected. This model ranks among the most reliable across our

datasets. Table 3.6 details its overall performance, and Figure 3.14 illustrates stability across sentence lengths, which is consistent for both seeds.

Metric	V2	Original	Diff
train-acc	0.9739	0.9843	−0.0104
val-acc	0.9574	0.9808	−0.0234
test-acc	0.9141	0.9185	−0.0044

(a) Metrics for G+rand dataset.

Metric	V2	Original	Diff
train-acc	0.9838	0.9899	−0.0061
val-acc	0.9769	0.9820	−0.0051
test-acc	0.9079	0.8955	+0.0124

(b) Metrics for G+G_6 dataset.

Metric	V2	Original	Diff
train-acc	0.8859	0.9456	−0.0596
val-acc	0.8979	0.9379	−0.0399
test-acc	0.5641	0.7110	−0.1469

(c) Metrics for $G_{14} + \text{rand}_{14}$ dataset.

Table 3.6: Accuracy comparison for V2 and Original models across datasets (V2 − Original).

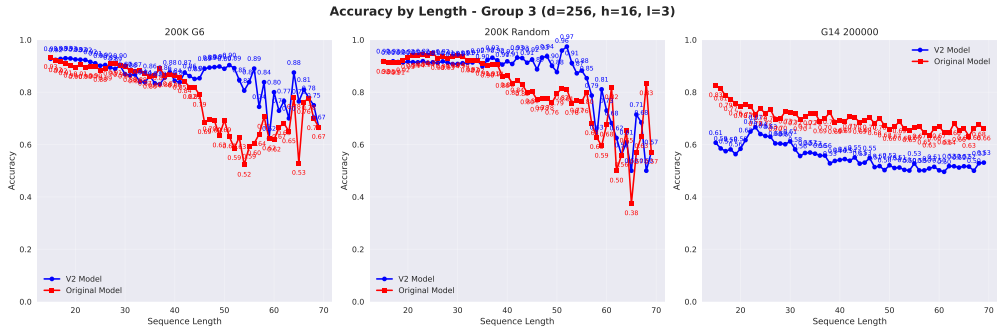


Figure 3.14: Accuracy by sentence length for model (dim=256, heads=16, layers=3, epochs=12) across datasets, showing variance. The x-axis shows sentence length, and the y-axis indicates accuracy per sentence length.

3.6.3 Bootstrap Resampling for Performance Analysis

To compare the performance of two Acceptor models with different dimensions and performances, a bootstrap analysis (see Section 2.7.2) was conducted using predictions from test set results stored in tab-separated text files [43]. The analysis loads predictions and true labels, converts them to binary format ("[Y]" as 1, "[n]" as 0), and computes *accuracy*, *precision*, *recall*, and *F1-score* (see Section 2.7.1) for each model. Bootstrap

resampling with 10,000 iterations is performed to estimate the variability of these metrics for the two models and computes the differences between them, with each iteration sampling approximately 59,000 unique indices from the test set, omitting around 34,000 indices due to replacement as explained in Section 2.7.2. Confidence intervals (95%) and two-tailed p-values are calculated to assess the significance of the differences between the models, while standard deviations provide insight into metric stability. We now examine the performance on the *G*+rand dataset using two models, both taken with the "Original" random seed:

- dim=256, heads=16, layers=3, epochs=12 (from Table 3.4a)
- dim=128, heads=16, layers=3, epochs=12 (from Table 3.6a)

The distribution of results obtained through the bootstrap sampling method developed by Koehn [43], visible in Figure 3.15, confirms the validity of the results and indicates that the larger model (dim=256) consistently outperforms the smaller model (dim=128). However, this comparison does not account for the variance across random seeds, which, as previously observed, can influence these outcomes.

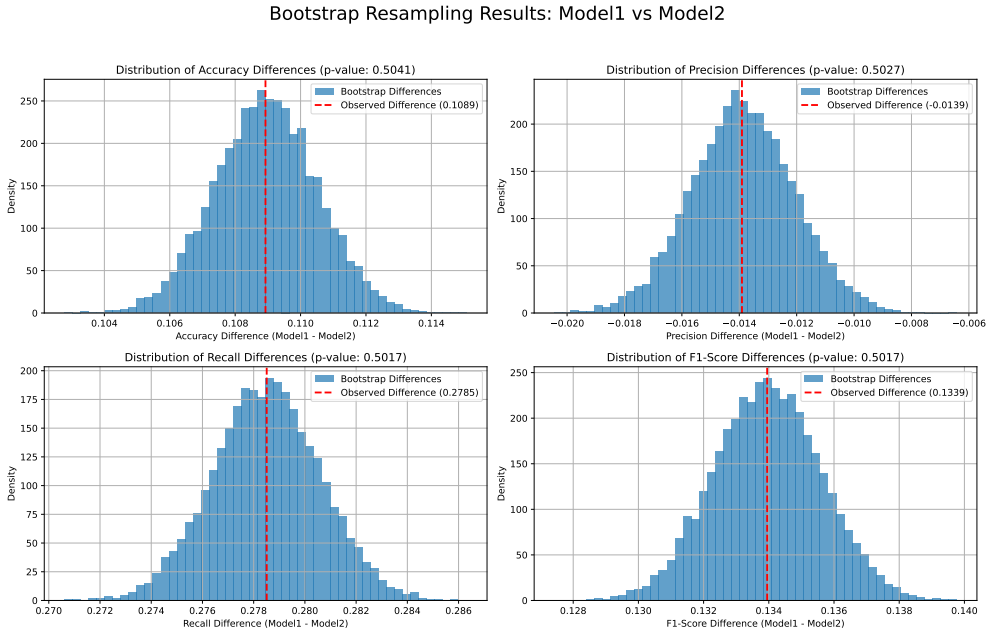


Figure 3.15: Bootstrap Resampling results for the two models: dim=256 and dim=128.

3.7 CONCLUSION

Our analysis indicates that training performance remains stable across different random seeds, reflecting consistent learning patterns on the training set. However, variance emerges in the validation set, particularly for datasets incorporating random noise, in contrast to sentences belonging to the real language generated from G. This variance is not uniform across all validation datasets but is pronounced in the presence of random noise, despite the validation set having sentence lengths similar to those in the training set.

Test set accuracies exhibit significantly greater variability, with notable performance differences. For instance, one random seed accurately predicts up to a sequence length of 55, while another is limited to length 30; but in most cases the performances across different random seeds are very similar if not equal (see Table 3.3). Bootstrap sampling, as developed by Philipp Koehn [43], supports these results as non-random. The observed variance, well-documented in the literature, is primarily attributed to the limited dataset size (200,000 samples) and the longer sentence lengths in the test set compared to the training set, which may amplify variability.

These findings align with recent research on length generalization, defined as the ability to extrapolate from shorter training sequences to longer test sequences. Zhou et al. [45] demonstrate that length generalization in Transformers is fragile and heavily influenced by factors such as data format, position encoding, random weight initialization, and training data order. Their study shows that while standard Transformers can extrapolate to sequence lengths up to 2.5 times the input length with the right configuration, performance remains sensitive to these factors, leading to significant variance across random seeds. This fragility in length generalization likely contributes to the variability observed in our test set results, particularly for longer sequences, underscoring the need for optimized data formats and position encodings to enhance model robustness.

4

Translation Experiment

"I want to talk the dialect of your people. It's no use of talking unless people understand what you say." [1]

The first experiment showcased an Encoder-Only Transformer's capability to produce an accurate target-language sentence from a given source sentence. Unlike the previous experiment shown in Chapter 3, which involved extensive analyses, modifications, and iterations, this experiment on machine translation inference is more focused: it centers on a single grammar, uses a Seq2Seq Transformer architecture and explores only five different configurations of the hyperparameters: this is a significant reduction compared to the over 200 configurations examined in the previous Chapter 3. To learn about **model architecture** and **model configuration** differences, please refer to Section 3.3.

4.1 Experimental Setup

This section details the experimental setup for the Machine Translation experiment, including the computational environment, model configurations, and key variations tested.

[1] Zora Neale Hurston, "Moses, Man of the Mountain", 1939

4.1.1 Environment and Infrastructure

The experiments were conducted on the CloudVeneto clusters provided by University of Padova, differing from the setup used in Section 3.1.

- **OS**: Ubuntu 6.8.0–60-generic #63-Ubuntu SMP PREEMPT_DYNAMIC, x86_64 architecture.
- **CPU**: AMD EPYC 7413 24-Core Processor, 8 CPUs, 48-bit physical and virtual addressing, with virtualization support (AMD-V).
- **RAM**: 7.8 GiB total.

4.1.2 Software and Dependencies

The experiments relied on the following Python (v. 3.11.11) packages:

- `pip` 23.3.1
- `fairseq` 0.12.2
- `torch` 2.6.0
- `transformers` 4.49.0
- `nltk` 3.9.1

4.2 Dataset Preparation

For this experiment, a single dataset is constructed, unlike the multiple datasets used in the prior experiment detailed in Section 3.2. This dataset is derived from grammar G, as defined in Section 2.5.2, and includes only positive examples from the datasets described in Sections 3.2.1 and 3.2.2. It consists of 107,000 source sentences in *train.src*, with approximately 100,000 additional sentences split between *valid.src* and *test.src*, paired with corresponding target sentences in *train.tgt*, *valid.tgt*, and *test.tgt*. Each line in the source files aligns precisely with the corresponding line in the target files. The dataset is split into source (.src) and target (.tgt) files of identical dimensions with aligned sentences to comply with the standard dataset construction requirements of the `fairseq` framework [23]. Unlike the previous experiment, the test set here excludes sentences longer than 14 tokens because the parsing algorithm described in Section 2.7

has a time complexity of $O(n^6)$, making it infeasible to parse 93,000 sentences of greater length, which could take days and 14 is the maximum of the training set sentence lengths.

4.3 Model Configurations

This experiment employs, as mentioned, only five model configurations, all created using the `fairseq` framework developed by *Meta* [23], rather than being manually crafted through `PyTorch` as we saw in Chapter 3. The model architecture is based on the Transformer model introduced by "Attention Is All You Need". To learn more about this model architecture see Section 2.2.
Each model consists of an encoder and a decoder, with the first model featuring a single layer for both components, the second model incorporating three layers and the third model incorporating six layers. Finally, the last two models, referred to as "6-layer-v1" and "6-layer-v2", also have six layers but with reduced embedding dimensions and feed-forward network (FFN) sizes to maintain a parameter count comparable to the 1-layer and 3-layer models, respectively. All models share the same embedding dimension of 512, number of attention heads (8), and dropout rate (0.3). The key differences among the models are summarized in Section 4.4.

4.3.1 Training Methodology

The experimental protocol employed consistent training parameters across all configurations:

- **Embedding dimension**: 512
- **Optimizer**: Adam ($lr = 1 \times 10^{-3}$, $\beta = (0.9, 0.98)$) with inverse square root scheduling and 4000-update warmup
- **Loss**: CE Loss
- **Gradient clipping**: 1.0 (norm)
- **Dropout**: 0.3
- **Loss function**: Label-smoothed cross-entropy (smoothing factor = 0.1)
- **Embedding sharing**: Input and output embeddings shared

- **Training checkpoints**: Saved in a designated directory for continuous monitoring and evaluation

This means that all following configurations were trained with the same hyperparameters, differing only in the number of layers (while varying the size of the FFN and embedding dimensions only for the last two models).

4.3.2 Model's Vocabulary

The Transformer does not employ a pre-given vocabulary enabling direct processing of the dataset as in the previous chapter, but constructs the vocabulary alone through the `fairseq-preprocess` phase, essential for the next steps: training and generation.

4.3.3 Model's Inference

The inference process is executed using the `fairseq-generate` command, which processes the dataset and applies the model checkpoint specified in the command. It uses a batch size of 128 and a beam search width of 5, with source and target languages set as `src` and `tgt`, respectively.
The generated results are output to `r/generate-test.txt`.

```
generate:
    fairseq-generate data-bin/
        --path data-bin/training_checkpoints/checkpoint.pt
        --batch-size 128
        --beam 5
        --source-lang src --target-lang tgt > r/generate-test.txt
    touch generate
```

4.4 Results and Analysis

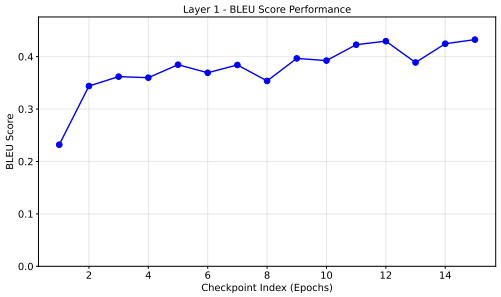
As demonstrated by prior studies [8], the computational capabilities of self-attention (attention is explained in Section 2.2), for processing hierarchical structures are limited and only emerge with increased layers or heads relative to input length. Given that this experiment involves a machine translation, meaning that it requires token generation, we anticipate suboptimal results. As explained in Section 2.5.1, numerous permutations of production rules exist, and even minor changes, such as inverting terminal positions in the yielded structure trees, can lead to a negative parse.

Identifying such errors is challenging due to: (1) multiple valid target language sentences corresponding to a single source sentence, with the gold target potentially differing significantly from the generated output; (2) the difficulty in pinpointing the exact error in the output, as it would require parsing all possible permutations of the generated sentence using deterministic SCFGs, which is computationally intensive.

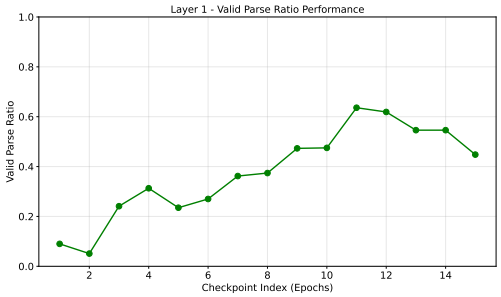
In the following paragraphs, we discuss the obtained results.

4.4.1 1-Layer Architecture

This 1-layer configuration (see Section 4.3.1 for hyperparameters and details) was trained for 15 epochs, with target sentence generation evaluated across all checkpoints. As shown in Figure 4.1, the BLEU score exhibited linear growth from 0.35 to a stable 0.4, peaking at 0.43, with no significant oscillations. The valid parse ratio, however, did not closely follow the BLEU score trends, reaching a peak of 63% at epoch 11. The invalid parse ratio mirrored this, with a minimum of 37%. These results suggest modest but stable performance for the 1-layer model, with limited alignment between BLEU scores and parse ratios. We observed better results than for the 6-layer configuration. This was unexpected, as deeper configurations typically outperform shallower ones in various tasks, therefore we conducted further experiments, discussed in Section 4.4.4, to investigate this anomaly.



(a) BLEU score across 15 checkpoints, showing linear growth from 0.35 to a stable 0.4, peaking at 0.43.



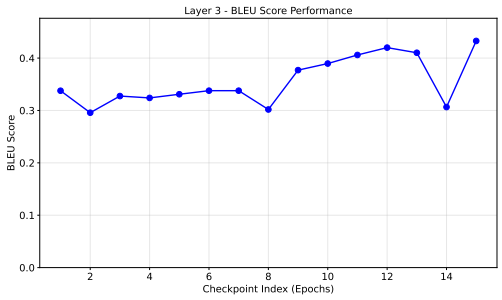
(b) Valid parse ratio across 15 checkpoints, not closely tracking BLEU scores, peaking at 63% at epoch 11.

Figure 4.1: MT performance metrics for a 1-layer model across 15 checkpoints.

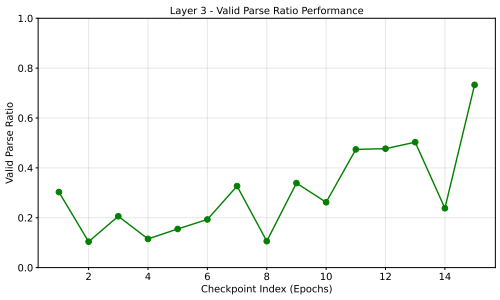
4.4.2 3-Layer Architecture

This 3-layer configuration (see Section 4.3.1 for hyperparameters and details) was trained for 15 epochs, with target sentence generation evaluated across all checkpoints.

As shown in Figure 4.2, the BLEU score exhibited linear growth from 0.35 to 0.43, with no significant oscillations. The valid parse ratio seems to follow the BLEU score trends, reaching a peak of 73.3% at epoch 15, but has much more oscillations than the previous model and has a very peculiar behaviour between epoch 14 and epoch 15 giving very different parse ratios. The invalid parse ratio mirrored this, with a minimum of 22%. These results suggest good but unstable performance for the 3-layer model for such a small amount of training epochs. Probably more training epochs would stabilize the valid parse ratio. Specifically to the last (and best) checkpoint, we observed better results than for the 1-layer configuration shown previously, as expected, since deeper configurations typically outperform shallower ones in various tasks.



(a) BLEU score across 15 checkpoints, showing linear growth from 0.35 to 0.43.

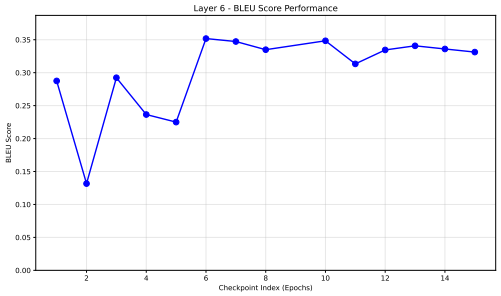


(b) Valid parse ratio across 15 checkpoints, not closely tracking BLEU scores, peaking at 78% at epoch 15.

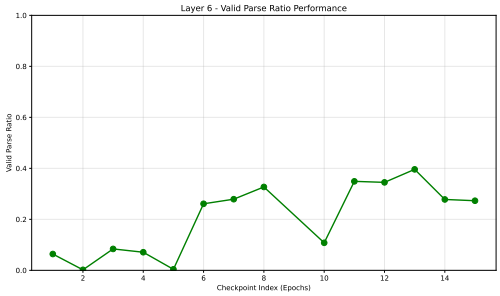
Figure 4.2: MT performance metrics for a 3-layer model across 15 checkpoints.

4.4.3 6-Layer Architecture

We initially hypothesized that the 6-layer configuration, with more parameters than other setups, would yield the best performance. However, the results did not meet expectations as previously mentioned in Section 4.4.1. We trained also this configuration for 15 epochs, generating target sentences using all intermediate checkpoints. As shown in Figure 4.3, the BLEU score oscillated between 0.15 and 0.35 for the first six epochs before stabilizing. The valid parse ratio, peaking at a maximum of 40%, showed improved stability after the initial epochs (except for checkpoint 10). These results indicate consistent but limited performance in machine translation for this configuration, but probably, as for the 3-layer configuration, more training epochs would improve the performance.



(a) BLEU score across 15 checkpoints, oscillating between 0.15–0.35 initially, stabilizing at 0.33.



(b) Valid parse ratio across 15 checkpoints, stabilizing with BLEU but peaking at 40%.

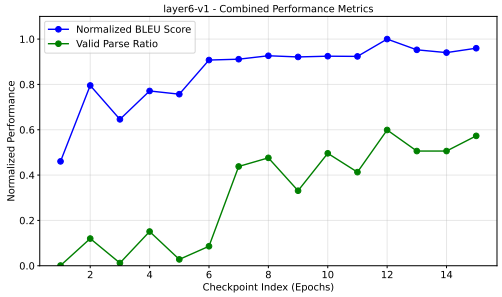
Figure 4.3: MT performance metrics for a 6-layer model across 15 checkpoints.

4.4.4 Variants for Layer 6

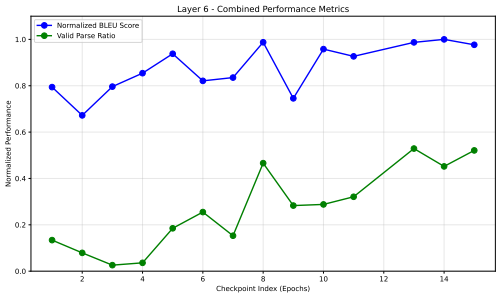
Our results were surprising because deeper configurations are typically expected to outperform smaller ones. To investigate this, we conducted an experiment on model depth (determined by the number of encoder and decoder layers) while adjusting the feed-forward network (FFN) and embedding layers to keep the total number of parameters close to those of the plain 1-layer and 3-layer configurations, which were also included in this experiment. For the 1-layer-like model, called "6-layer-v1", we set the embedding dimension to 208 (instead of 512) and the FFN layers in both the encoder and decoder to 832 (instead of the default 2000 used in other configurations). For the 3-layer-like model, called "6-layer-v2", we set the embedding dimension to 296 and the FFN layers to 1184. The total number of parameters in these models is roughly similar, with a difference of approximately $\pm 50,000$ parameters compared to the 7 and 14 million parameters in the other configurations. The results are very interesting, revealing that larger models (6-layer family) underperform compared to smaller models (1-layer and 3-layer), with the 3-layer model surpassing all others, including the two modified versions of the 6-layer configuration. This trends will be more thoroughly discussed in the next section.

4.5 Best Checkpoint Analysis

The analysis below involves selecting the best checkpoint for each model based on the number of valid parses and comparing all models at their optimal checkpoints. It references Figure 4.5, which illustrates:



(a) BLEU scores and Valid-parse ratio across 15 checkpoints for 6-layer-v1.



(b) BLEU scores and Valid-parse ratio across 15 checkpoints for 6-layer-v2.

Figure 4.4: MT performance metrics for a 6-layer-v1 and 6-layer-v2 models across all 15 checkpoints.

(1) Accuracy of correct length predictions for sentence lengths 1–14 (e.g., for a source sentence of length 4, the proportion of generated target sentences with an effective length of 4, given the 1-to-1 token correspondence) across all best-performing models, see Figure 4.5a.

(2) Percentage of valid parses for sentence lengths 1–14 across all best-performing models, see Figure 4.5b.

(3) Percentage of valid parses for sentence lengths 1–14 that differ from the golden target provided in the test set, across all best-performing models, see Figure 4.5c.

1-layer (checkpoint 11) For the 1-layer model (Checkpoint 11), visible between all models shown in Figure 4.5, the BLEU score reached 0.4227 at its peak. The model demonstrated consistent correct length predictions across sentence lengths, as seen in Figure 4.5a. The valid parse ratio was 63.6% (636/1000), with performance varying by sentence length (Figure 4.5b). Notably, 68.7% (437/636) of the valid parses differed from the gold target, indicating a significant portion of correct but non-target outputs (Figure 4.5c).

3-layer (checkpoint 15) For the 3-layer model (Checkpoint 15), the BLEU score was 0.4327, slightly higher than the 1-layer model. The valid parse ratio was 73.3% (733/1000), as seen in Figure 4.5b, with 72.4% (531/733) of valid parses differing from the gold target (Figure 4.5c). This suggests strong performance with a higher proportion of valid outputs compared to the 1-layer model.

6-layer (checkpoint 13) In contrast, the 6-layer model (Checkpoint 13), achieved a lower BLEU score of 0.3408. Correct length predictions were less consistent compared to the 1-layer and 3-layer models (Figure 4.5a). The valid parse ratio was substantially lower at 39.6% (396/1000), as shown in Figure 4.5b, with 55.6% (220/396) of valid parses differing from the gold target (Figure 4.5c).

6-layer-v1 (checkpoint 12) For the 6-layer v1 model (Checkpoint 12), the BLEU score was 0.4016. The valid parse ratio was 59.9% (599/1000), with 68.3% (409/599) of valid parses differing from the gold target.

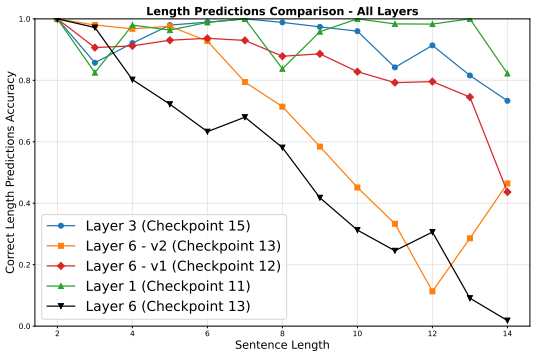
6-layer-v2 (checkpoint 13) For the 6-layer v2 model (Checkpoint 13), the BLEU score was 0.3733, with a valid parse ratio of 52.9% (529/1000) and 59.6% (314/529) of valid parses differing from the gold target. These results suggest that the 6-layer model struggled more with generating valid parses and was more sensitive to sentence length variations compared to the 1-layer and 3-layer models. The 6-layer v1 and v2 models performed better than the original 6-layer model but still fell short of the 1-layer and 3-layer models in terms of BLEU score and valid parse ratio. Overall, the 3-layer model outperformed all the other configurations in this translation task.

As we are handling synthetic languages, BLEU scores may be less reliable compared to natural languages due to structural differences, requiring cautious interpretation alongside other metrics [46].

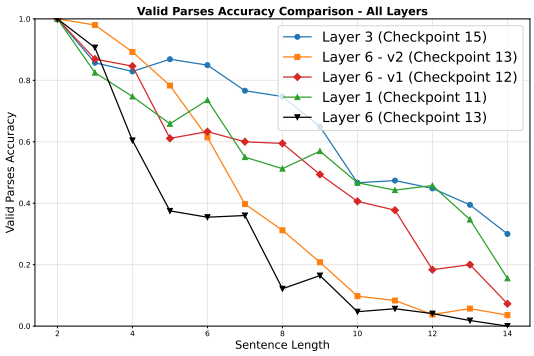
4.6 Metrics and Evaluation

To evaluate the performance of our models, we used two metrics to quantify differences between predicted and target sentences. The first, token differences, counts mismatched tokens between the two strings. Since each token is a single character in our setup, this metric directly reflects character discrepancies. The second, Levenshtein distance, a standard in natural language processing (NLP), measures the minimum number of single-character edits, insertions, deletions, or substitutions required to transform one string into another, ranging from 0 (identical strings) to the length of the longest string [47]. These metrics are visualized in Figure 4.6.

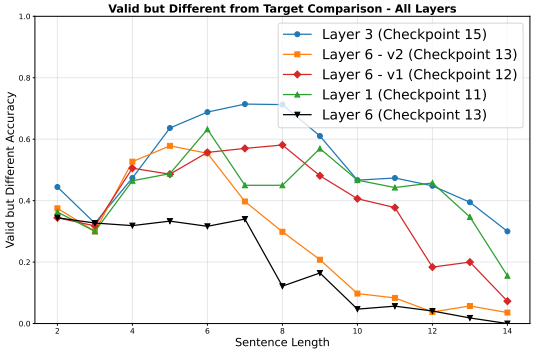
For checkpoint 11 (1-layer model), Figure 4.6a shows that most sentences exhibit small token differences between target and predicted outputs (red line), indicating high token similarity. However, order discrepancies increase with sentence length, suggesting that while the correct tokens are often used, their arrangement diverges, especially



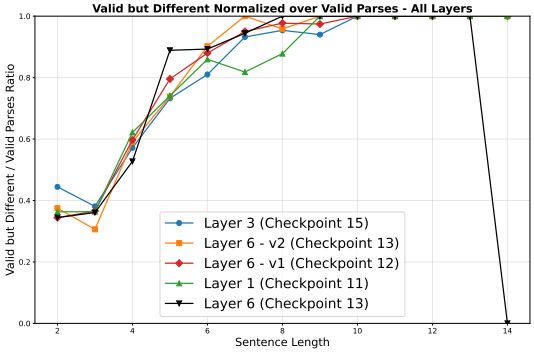
(a) Correct length predictions per sentence length.



(b) Valid parses per sentence length.



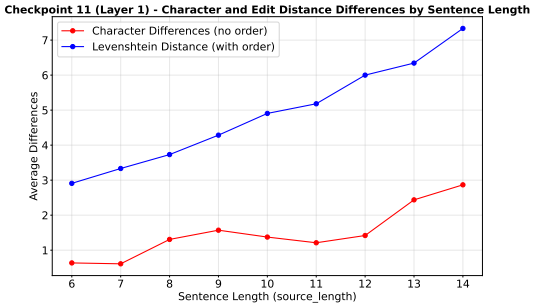
(c) Valid parses different from gold target per sentence length.



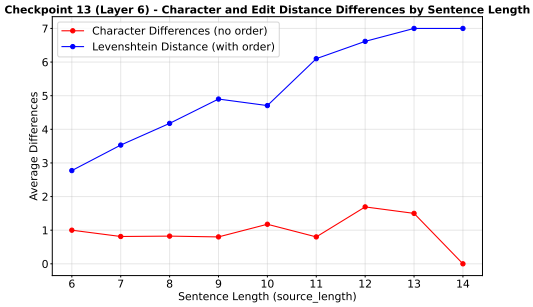
(d) Valid parses different from gold target per sentence length normalized over total valid parses.

Figure 4.5: Performance of all configurations taken at their best checkpoint. Y-axis: accuracy, X-axis: sentence length.

for longer sentences. In contrast, for checkpoint 13 (6-layer model), Figure 4.6b reveals slightly worse token differences, averaging 1 token mismatch until sentence length 12, rising to 2 thereafter. Order differences grow linearly, reaching 7 token differences for a sentence length of 14, indicating significant structural misalignment despite using similar tokens. These results, as depicted in Figure 4.6, underscore the 1-layer model's better token accuracy but highlight both models' challenges with token ordering as sentence length increases.



(a) For checkpoint 11 (1-layer model), most sentences show small token differences between target and prediction (blue line), with increasing order discrepancies as sentence length grows.



(b) For checkpoint 13 (6-layer model), token differences average 1 until length 12, rising to 2, while order differences grow linearly, reaching 7 token differences at length 14.

Figure 4.6: Token and order differences between target and predicted sentences for 1-layer (checkpoint 11) and 6-layer (checkpoint 13) models.

4.7 Conclusion

In conclusion, our models struggle to generalize over the SCFG G, defined in Section 2.5.2, and fail to fully grasp sentence formation in the synthetic languages. The 1-layer configuration achieved second best valid parse ratio of 63.6%, but Figure 4.5 reveals its limitations, with generalization decreasing linearly from 100% to $\sim$ 20% at the maximum sentence length (14), despite stable length predictions and a high BLEU score of 0.4324. The 3-layer configuration achieved the first best valid parse ratio of 73.3% going from 100% at minimum length to $\sim$ 36% at the maximum one.

Deeper 6-layer models struggle to identify patterns within the first 15 epochs, likely requiring extended training to improve performance, as supported by literature suggesting deeper models should perform better [48]. Alternatively, a new loss function may be necessary, as explored in prior work [49].

Analysis of invalid parses at length 14, using the parser described in Section 2.7, suggests errors often stem from simple token inversions. Even with correct words in near-identical order, precise terminal positioning is critical for SCFG compliance, rendering translations invalid if misaligned. While the high BLEU score indicates similarity to reference translations, its reliability for synthetic SCFGs is limited. Designed for natural languages, BLEU overlooks critical syntactic and semantic features in synthetic or structured languages, such as programming languages or the currently used SCFGs, reducing its effectiveness compared to natural language evaluation [50].

5 Conclusions

"Ask not whether an object falls because it is pulled from below or pushed from above. Ask how well you can predict the time it takes for the object to travel a certain distance, and how that time will vary from object to object and as the angle of the track changes. Moreover, said Galileo, do not attempt to answer such questions in the qualitative and slippery nuances of ***human language****; say it in the form of* ***mathematical equations****."* [1]

5.1 Contributions or Summary of Findings

In the first experiment, discussed in Chapter 3, we explored how certain models (specifically: dim=128, heads=16, layers=5, epochs=12 and dim=256, heads=16, layers=3, epochs=12) achieve very high accuracies of 92% and 96.7% in predicting what fits and what does not within the SCFGs. We noticed that training performance stays stable across different random seeds, showing consistent learning patterns on the training set. However, there is some small variation in the validation set, especially when datasets include random noise. On the test set, accuracies vary a lot, with one random seed predicting correctly up to a sequence length of 55, while another stops at length 30 and then loses performance. Using bootstrap sampling, we confirmed that this variability in the test set is not random. Our findings match research on length generalization, which shows that standard Transformers can handle sequences up to 2.5 times longer

[1] J. Pearl, *Causality: Models, Reasoning, and Inference*, New York: Cambridge University Press, pp. 401–428, 2009.

than the training set with the right setup. In fact, some of our configurations showed Transformers generalizing up to length 70, which is five times the maximum length of the training set sentences.

For the second experiment, presented in Chapter 4, we saw how the Transformer architecture can be used for machine translation tasks, even with synthetic languages built from SCFGs. The 3-layer architecture performed best, reaching a valid parse ratio of 73.3%, though its generalization dropped from 100% at the shortest length (2) to about 36% at the longest length (14). By analyzing invalid parses at length 14, we found that errors often come from simple token inversions, showing how important exact terminal positioning is for SCFG compliance. The high BLEU score suggests translations are similar to the reference, but it is less reliable for synthetic SCFGs since it misses key syntactic and semantic details compared to natural language evaluation.

5.2 Limitations and Future Work

As discussed in Sections 3.7 and 4.7, both experiments reveal limitations that need to be addressed and could lead to future research opportunities. In Chapter 3, we observed that nearly all models exhibit variability on the test set. However, applying the Bootstrap Method by Kohen [43] suggests that one model may perform significantly better than another, and this difference does not seem to stem from randomness. Despite this, we cannot draw firm conclusions yet, as further experiments are necessary.

Dataset Size One key limitation is the size of the datasets, which are considered too small. The $50-50$ split between accepted and rejected pairs could be tested with different ratios, such as $75-25$ or $70-30$, to explore their impact. Additionally, increasing the maximum length of generated sentences, for example, from 100 to 200 tokens, and expanding the Transformer's input range from sentences of 1–14 to 1–25 could help the model learn and generalize more effectively.

Syn-CYK Parser However, these improvements face structural challenges. Generating longer sentences and larger datasets requires significant computational power. As described in Section 2.7, the deterministic Syn-CYK Parser has a complexity of $\Theta(n^6)$, meaning that processing sentences of 200 tokens would demand 200^6 operations. This is highly time-consuming and limits the scope of the current research. Similarly, scaling the dataset from 200,000 to millions of samples would also require substantial time and resources.

"Harder" SCFGs Another critical limitation lies in the choice of the three grammars outlined in Section 2.5.2 for constructing the datasets in Section 3.2. These grammars lack theoretical significance and do not represent mathematically "harder" or "simpler" structures. The classification of grammars as "harder" or "simpler" is based only on the number of productions, nonterminals, and potential ambiguities, rather than rigorous theoretical evidence. For instance, there is no mathematical proof that G_{14} is more complex than G-based grammars. A thorough investigation into this issue, along with the development of larger datasets based on theoretically validated harder and simpler grammars, is essential for advancing this research and should be prioritized in future work.

Finding the Breaking Point In Chapter 3, in Section 3.5.2, we found that most models generalize well up to a sequence length of 35 (2.5 times the maximum length seen at training), consistent with existing research [45]. Many models, like the one with dim=256, heads=16, layers=3, epochs=12, struggle beyond length 50 (3.5 times the maximum length seen at training), while others, such as dim=128, heads=16, layers=5, maintain performance up to length 70 (5 times the maximum length seen at training). This shows a breaking point where generalization starts to decline varies across models. Future work should explore the specific factors, such as model architecture, training methods, or dataset features, that influence where this breaking point occurs. To do that, a first step might be to plot the accuracy of each model with same fixed dimension against the length of the sentences, to see if there is a pattern that can be exploited to predict the breaking point of a model.

Loss Function Another limitation of the current work is the use of Cross-Entropy (CE) Loss for training in both Chapter 3 and Chapter 4. In future work, we plan to explore a new loss function that incorporates the Syn-CYK Parser seen in Section 2.7, for example, to recognize generated translations that could be right even if they are distant from the single golden target given. This could involve Reinforcement Learning techniques to help models understand the structure of languages and identify valid translations, rather than focusing only on mimicking a single correct target translation. The current CE Loss does not account for language ambiguity, which allows multiple valid translations for a single source sentence, as it only considers the provided golden target. Other similar projects have already implemented more complex loss functions [49].

Negative Samples for MT Finally, in Chapter 4, we observed that the model often produces invalid parses, especially as sentence length increases. To address this, we could introduce negative samples during training, which are intentionally incorrect translations. This approach would help the model learn to distinguish between valid and invalid outputs, potentially improving its ability to generate syntactically correct translations. By exposing the model to a variety of incorrect examples, it can better understand the rules and structures of the languages involved, leading to more accurate and reliable translations. This experiment, which has not been conducted yet, could be a valuable direction for future research but requires significant changes to the current loss function and training process, requiring contrasting learning techniques.

References

[1] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released January 12, 2025. 2025. URL: https://web.stanford.edu/~jurafsky/slp3/.

[2] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[3] Alexander Rush. *The Annotated Transformer*. https://nlp.seas.harvard.edu/annotated-transformer. Accessed: 2025-08-21. 2018.

[4] Martin Popel et al. "Human translation quality is achievable with deep learning". In: *Nature Communications* 11.1 (Aug. 2020), p. 4296. ISSN: 2041-1723. DOI: 10.1038/s41467-020-18073-9. URL: https://www.nature.com/articles/s41467-020-18073-9.

[5] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *arXiv preprint arXiv:2010.11929* (2020). Presented at ICLR 2021. URL: https://arxiv.org/abs/2010.11929.

[6] Linhao Dong, Shuang Xu, and Bo Xu. "Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, pp. 5884–5888. DOI: 10.1109/ICASSP.2018.8462506. URL: https://ieeexplore.ieee.org/document/8462506.

[7] Tom B. Brown et al. "Language Models are Few-Shot Learners". In: *arXiv preprint arXiv:2005.14165* (2020). Presented at NeurIPS 2020. URL: https://arxiv.org/abs/2005.14165.

[8] Michael Hahn. "Theoretical Limitations of Self-Attention in Neural Sequence Models". In: *Transactions of the Association for Computational Linguistics* 8 (2020). Ed. by Mark Johnson, Brian Roark, and Ani Nenkova, pp. 156–171. DOI: 10.1162/tacl_a_00306. URL: https://aclanthology.org/2020.tacl-1.11/.

[9] William Merrill, Ashish Sabharwal, and Noah Schwartz. "Formal Language Theory Meets Modern NLP". In: *Proceedings of the 2021 ACL Workshop on Benchmarking: Past, Present and Future*. Association for Computational Linguistics, 2021, pp. 46–52. DOI: 10.18653/v1/2021.bppf-1.6. URL: https://aclanthology.org/2021.bppf-1.6.

[10] William Merrill, Michael Hahn, and Ashish Sabharwal. "The Computational Complexity of Transformers". In: *arXiv preprint arXiv:2205.08337* (2022). URL: https://arxiv.org/abs/2205.08337.

[11] Gail Weiss, Yoav Goldberg, and Eran Yahav. "Thinking Like Transformers". In: *Proceedings of the 9th International Conference on Learning Representations (ICLR)*. Presented at ICLR 2021. 2021. URL: https://openreview.net/forum?id=7S1W7M4CZ4.

[12] Tom Kocmi. *Exploring Benefits of Transfer Learning in Neural Machine Translation*. 2020. arXiv: 2001.01622 [cs.CL]. URL: https://arxiv.org/abs/2001.01622.

[13] Basu Dev Shivahare et al. "Survey Paper: Study of Natural Language Processing and its Recent Applications". In: *2022 2nd International Conference on Innovative Sustainable Computational Technologies (CISCT)*. 2022, pp. 1–5. DOI: 10.1109/CISCT55310.2022.10046440.

[14] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774.

[15] Yanchu Guan et al. *Intelligent Virtual Assistants with LLM-based Process Automation*. 2023. arXiv: 2312.06677 [cs.LG]. URL: https://arxiv.org/abs/2312.06677.

[16] Verónica Pérez-Rosas et al. "Automatic Detection of Fake News". In: *Proceedings of the 27th International Conference on Computational Linguistics*. Ed. by Emily M. Bender, Leon Derczynski, and Pierre Isabelle. Santa Fe, New Mexico, USA: Association for Computational Linguistics, Aug. 2018, pp. 3391–3401. URL: https://aclanthology.org/C18-1287/.

[17] John Hutchins. “The first public demonstration of machine translation: the Georgetown-IBM system, 7th January 1954”. In: (Jan. 2004). URL: https://web.archive.org/web/20071021224529/http://www.hutchinsweb.me.uk/GU-IBM-2005.pdf.

[18] Lei Huang et al. “A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions”. In: *ACM Transactions on Information Systems* 43.2 (Jan. 2025), 1–55. ISSN: 1558-2868. DOI: 10.1145/3703155. URL: http://dx.doi.org/10.1145/3703155.

[19] S. Leynas. “Expressivity of Transformers: Logic, Circuits, and Formal Languages”. In: *Proceedings of the 35th European Summer School in Logic, Language and Information (ESSLLI 2024)*. 2024. URL: https://sleynas.com/assets/pdfs/esslli2024/main.pdf.

[20] Jorge Pérez, Javier Marinković, and Pablo Barceló. “On the Turing Completeness of Modern Neural Network Architectures”. In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=HyGBdo0qFm.

[21] Gilad Yehudai et al. *When Can Transformers Count to n?* 2024. arXiv: 2407.15160 [cs.CL]. URL: https://arxiv.org/abs/2407.15160.

[22] Ashish Vaswani et al. “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017. URL: https://arxiv.org/abs/1706.03762.

[23] Myle Ott et al. “fairseq: A Fast, Extensible Toolkit for Sequence Modeling”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*. Ed. by Waleed Ammar, Annie Louis, and Nasrin Mostafazadeh. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 48–53. DOI: 10.18653/v1/N19-4009. URL: https://aclanthology.org/N19-4009/.

[24] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL]. URL: https://arxiv.org/abs/1810.04805.

[25] Zhilu Zhang and Mert R. Sabuncu. *Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels*. 2018. arXiv: 1805.07836 [cs.LG]. URL: https://arxiv.org/abs/1805.07836.

[26] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[27] Johnson Victor et al. "ps-CALR: Periodic-Shift Cosine Annealing Learning Rate for Deep Neural Networks". In: *IEEE Access* PP (Jan. 2023), pp. 1–1. DOI: 10.1109/ACCESS.2023.3340719.

[28] Adam Paszke et al. "Automatic differentiation in PyTorch". In: *NIPS-W*. 2017.

[29] Alex Graves. "Sequence Transduction with Recurrent Neural Networks". In: *CoRR* abs/1211.3711 (2012). arXiv: 1211.3711. URL: http://arxiv.org/abs/1211.3711.

[30] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper_files/paper/2014/file/5a18e133cbf9f257297f410bb7eca942-Paper.pdf.

[31] Markus Freitag and Yaser Al-Onaizan. "Beam Search Strategies for Neural Machine Translation". In: *Proceedings of the First Workshop on Neural Machine Translation*. Ed. by Thang Luong et al. Vancouver: Association for Computational Linguistics, Aug. 2017, pp. 56–60. DOI: 10.18653/v1/W17-3207. URL: https://aclanthology.org/W17-3207/.

[32] Maja Popović. "chrF: character n-gram F-score for automatic MT evaluation". In: *Proceedings of the Tenth Workshop on Statistical Machine Translation*. Ed. by Ondřej Bojar et al. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 392–395. DOI: 10.18653/v1/W15-3049. URL: https://aclanthology.org/W15-3049/.

[33] Kishore Papineni et al. "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Ed. by Pierre Isabelle, Eugene Charniak, and Dekang Lin. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://aclanthology.org/P02-1040/.

[34] Abhijit Agarwal and Alon Lavie. “Meteor, m-bleu and m-ter: Evaluation Metrics for High-Correlation with Human Rankings of Machine Translation Output”. In: *Proceedings of the Third Workshop on Statistical Machine Translation*. Columbus, Ohio, USA: Association for Computational Linguistics, 2008, pp. 115–118. URL: https://aclanthology.org/W08-0315.

[35] 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.

[36] P. M. Lewis and R. E. Stearns. “Syntax-Directed Transduction”. In: *J. ACM* 15.3 (July 1968), 465–488. ISSN: 0004-5411. DOI: 10.1145/321466.321477. URL: https://doi.org/10.1145/321466.321477.

[37] A.V. Aho and J.D. Ullman. “Syntax directed translations and the pushdown assembler”. In: *Journal of Computer and System Sciences* 3.1 (1969), pp. 37–56. ISSN: 0022-0000. DOI: https://doi.org/10.1016/S0022-0000(69)80006-1. URL: https://www.sciencedirect.com/science/article/pii/S0022000069800061.

[38] David Chiang. *An Introduction to Synchronous Grammars*. Tech. rep. University of Notre Dame, 2006. URL: https://www3.nd.edu/~dchiang/papers/synchtut.pdf.

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

[40] Daniel Gildea and Giorgio Satta. “Synchronous Context-Free Grammars and Optimal Parsing Strategies”. In: *Computational Linguistics* 42.2 (June 2016), pp. 207–243. DOI: 10.1162/COLI_a_00246. URL: https://aclanthology.org/J16-2002/.

[41] Liang Huang et al. “Binarization of Synchronous Context-Free Grammars”. In: *Computational Linguistics* 35.4 (Dec. 2009), pp. 559–595. DOI: 10.1162/coli.2009.35.4.35406. URL: https://aclanthology.org/J09-4009/.

[42] Ruibo Wang and Jihong Li. “Bayes Test of Precision, Recall, and F1 Measure for Comparison of Two Natural Language Processing Models”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Ed. by Anna Korhonen, David Traum, and Lluís Màrquez. Florence, Italy: Association

for Computational Linguistics, July 2019, pp. 4135–4145. DOI: 10.18653/v1/P19-1405. URL: https://aclanthology.org/P19-1405/.

[43] Philipp Koehn. "Statistical Significance Tests for Machine Translation Evaluation". In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Ed. by Dekang Lin and Dekai Wu. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 388–395. URL: https://aclanthology.org/W04-3250/.

[44] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. 2020. arXiv: 2001.08361 [cs.LG]. URL: https://arxiv.org/abs/2001.08361.

[45] Yongchao Zhou et al. *Transformers Can Achieve Length Generalization But Not Robustly*. 2024. arXiv: 2402.09371 [cs.LG]. URL: https://arxiv.org/abs/2402.09371.

[46] Ehud Reiter. "A Structured Review of the Validity of BLEU". In: *Computational Linguistics* 44.3 (Sept. 2018), pp. 393–401. ISSN: 0891-2017. DOI: 10.1162/coli_a_00322. eprint: https://direct.mit.edu/coli/article-pdf/44/3/393/1809172/coli_a_00322.pdf. URL: https://doi.org/10.1162/coli_a_00322.

[47] Frederic P. Miller, Agnes F. Vandome, and John McBrewster. *Levenshtein Distance: Information theory, Computer science, String (computer science), String metric, Damerau?Levenshtein distance, Spell checker, Hamming distance*. Alpha Press, 2009. ISBN: 6130216904.

[48] Andy Yang, Michaël Cadilhac, and David Chiang. *Knee-Deep in C-RASP: A Transformer Depth Hierarchy*. 2025. arXiv: 2506.16055 [cs.CL]. URL: https://arxiv.org/abs/2506.16055.

[49] Alexandra Butoi et al. *Training Neural Networks as Recognizers of Formal Languages*. 2025. arXiv: 2411.07107 [cs.CL]. URL: https://arxiv.org/abs/2411.07107.

[50] Shuo Ren et al. *CodeBLEU: a Method for Automatic Evaluation of Code Synthesis*. 2020. arXiv: 2009.10297 [cs.SE]. URL: https://arxiv.org/abs/2009.10297.

Acknowledgments

First of all I want to express my gratitude to my advisors, Prof. Giorgio Satta and Prof. Ondřej Bojar, for their invaluable guidance and expertise. Their mentorship has not only shaped this work but also profoundly enriched my learning experience, sparking inspiration and fostering my growth as a researcher.

Secondly I want to thank my family. To my parents, Vittoria and Carmine: thank you for always believing in me and supporting me. To my sister Eleonora: you will be great in your career as a physicist and I am proud of you. A special thanks to my girlfriend, Margarita, for always being there for me and keeping me going, coming to Prague and being part of this thesis journey — now it is my turn with you in Spain! Finally, a shout-out to my friends: our good times together made it easier.