JOHANNES KEPLER
UNIVERSITY LINZ

Author
Nikolaus von Zitzewitz
K12116426

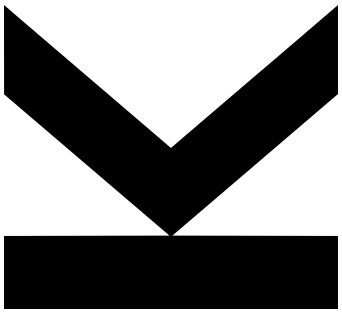
Submission
Machine Learning Institute

Thesis Supervisor
Univ.-Prof. Dr. **Sepp Hochreiter**

Co-Supervisors
Dr. **Thomas Adler**,
MSc. **Andreas Radler**

May 2025

Mamba in the Chomsky Hierarchy



Evaluating Mamba-SSM's potential using synthetic tasks

JOHANNES KEPLER UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Austria
www.jku.at
DVR 0093696

Abstract

Sequence modeling gives the foundation for state-of-the-art language models and time-series forecasting, yet transformers suffer quadratic compute and memory growth with sequence length. Mamba-SSM, a Selective State-Space Model, promises linear-time complexity by dynamically gating its state updates and using a diagonal-plus-low-rank parameterization. In this thesis, we present the first systematic evaluation of Mamba-SSM across fifteen synthetic benchmarks from the first three levels of the Chomsky hierarchy—regular, context-free, and context-sensitive grammars. We train three architectural variants (binary classifier, five-way classifier, generative) at hidden dimensions of 64, 128, and 256, using 4,000 pre-generated batches per epoch over three epochs, and compare against published RNN, xLSTM, and Transformer baselines. Our findings show that Mamba-SSM attains up to 100% accuracy on context-free tasks (Reverse String, Stack Manipulation), struggles near random on regular tasks like Parity Check, and yields good but still mixed results on context-sensitive problems (100% on Bucket Sort but 50% on Missing Duplicate).

Contents

List of Figures

List of Tables

1 Story Summary

1. **What is the central question?**
   How does Mamba-SSM perform, in terms of accuracy, on formal language tasks from the Chomsky hierarchy compared to other sequence modeling architectures such as Transformers (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, A. Gomez, et al. 2017), LSTMs (Hochreiter and Schmidhuber 1997), RNNs and a few other common architectures?

2. **Why is the question important?**
   Understanding the relative strengths of Mamba-SSM (Gu and Dao 2023) contributes to more informed model selection for sequence modeling tasks, particularly in contexts where computational efficiency and long-sequence generalization are critical. Given the increasing interest in replacing Transformer-based models with more efficient alternatives, a systematic evaluation of Mamba-SSM is timely and relevant.

3. **What evidence/data (variables) is needed to answer the question?**
   A benchmark comparison of Mamba-SSM's Gu and Dao 2023 performance on a suite of synthetic tasks representative of different language classes (regular, context-free, context-sensitive) from the Chomsky hierarchy (Delétang et al. 2023). This includes measured accuracies across tasks and against existing baselines (e.g., Transformer (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, A. Gomez, et al. 2017) and xLSTM (Beck et al. 2024)).

4. **What methods are used to get this evidence or data?**
   A total of 45 models were trained—three variants of Mamba-SSM (with 64, 128, and 256 hidden dimensions) were applied to 15 synthetic tasks derived from prior work by Delétang et al. 2023. The tasks are categorized by language class and were implemented using pre-generated datasets with controlled sequence lengths. Each

model was trained on Google Colab using a consistent training regime (3 epochs, batch size of 32, totaling 12,000 update steps).

5. **What analyses must be applied for the data to answer the central question?**
   Each model's performance was evaluated based on accuracy. These accuracies were then compared to published results from Transformer, LSTM, and other state-space architectures (Delétang et al. 2023; Beck et al. 2024). Deviations in performance across tasks and dimensions were also analyzed to identify consistent trends or limitations.

6. **What evidence/data (values for the variables) were obtained?**
   Mamba-SSM achieved near-perfect accuracy ($\geq$ 99%) on most context-free and context-sensitive tasks, including Reverse String, Stack Manipulation, Binary Addition, and Bucket Sort. However, it struggled on regular tasks like Parity Check and Cycle Navigation, often performing no better than random guessing.

7. **What were the results of the analyses?**
   Mamba-SSM outperformed Transformers (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, A. Gomez, et al. 2017) and traditional RNN-based models on nearly all tasks from the context-free and context-sensitive categories. Its performance was especially strong in tasks that required hierarchical reasoning or memory-based operations. Conversely, it underperformed on simpler regular tasks, consistent with prior findings on star-free language limitations in SSMs (Sarrof, Veitsman, and Hahn 2024).

8. **How did the analyses answer the central question?**
   The analyses demonstrate that Mamba-SSM is a highly competitive architecture for complex sequence modeling tasks, achieving better accuracy than Transformers (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, A. Gomez, et al. 2017) in most Chomsky hierarchy benchmarks (Delétang et al. 2023; Beck et al. 2024). However, its weaknesses on simpler regular languages highlight that its strengths are task-dependent.

9. **What does this answer tell us about the broader field?**
   Mamba-SSM presents a promising alternative to Transformer-based architectures for sequence modeling, especially when working with structured or hierarchical data. Its linear time complexity and strong empirical results suggest that state space

models could be key components of future scalable AI systems—though care must be taken to understand their limitations.

10. **Did the paper answer the question satisfactorily? Why (not)?**
    Yes. The thesis provides a comprehensive experimental evaluation of Mamba-SSM across multiple formal language tasks. Despite limited computational resources, the findings are consistent and robust, highlighting both the potential and the boundaries of Mamba-SSM as a modern sequence modeling architecture.

2 Introduction

The purpose of this thesis is to compare the new machine learning architecture Mamba-SSM (Gu and Dao 2023), using the Chomsky hierarchy, with other architectures. With the in total 15 synthetic tasks we evaluate the accuracies reached by Mamba-SSM with three different hyperparameter constellations. As Mamba-SSM is especially useful for sequence modeling just like other state-of-the-art architectures like transformers (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, A. N. Gomez, et al. 2023) or xLSTMs (Beck et al. 2024) and might therefore be a worthy extension of our modern endeavors in the field of machine learning. The main concepts necessary for understanding this thesis are the Chomsky hierarchy with its various synthetic problems and the Mamba-SSM architecture with its advantages as well as disadvantages compared to a few other currently prominent architectures.

2.1 Context and Motivation

In recent years sequence modeling has made great advances resulting in highly useful technologies like ChatGPT, Gemini and similar LLMs. As strong as these models are, there is still plenty of room for improvement. This is where Mamba-SSM comes into play. It is an attempt to continue this rapid progress by its new approach to sequence modeling through adapting state-space models and making them computationally efficient, while achieving a performance that can compete with the giants of AI like transformers or xLSTMs. However, since Mamba-SSM was published quite recently, its specific capabilities still need to be further explored. This thesis aims to contribute to this fundamental evaluation of Mamba-SSM using the framework of the Chomsky hierarchy to systematically explore its capabilities and limitations.

2.2 Problem Statement and Research Gap

A fundamental question for any new technological advancement is: What are its practical applications and limitations? This question is particularly relevant for Mamba-SSM, as its capabilities compared to existing architectures remain underexplored. This question needs to be answered in order to do further research and make educated decisions about whether Mamba-SSM is a good choice to solve a specific problem. With previous work on this, as done by Delétang et al. 2023 or Beck et al. 2024, this paper extends and verifies their results. Specifically some of the tasks used in the Chomsky hierarchy haven't been used yet to evaluate Mamba-SSM so this paper aims to close that gap to further solidify the scientific basis for future endeavors intending to use Mamba-SSM.

2.3 Research Objectives and Contributions

This thesis aims to extend the evaluation of Mamba-SSM's capabilities and provide insight into the strengths and limitations of this new architecture. By systematically evaluating Mamba-SSM using the Chomsky hierarchy we provide an understanding of what sequence modeling tasks Mamba-SSM is a useful choice for.

- **RQ1:** To what extent can Mamba-SSM solve different levels of complexity within the Chomsky hierarchy?
- **RQ2:** How does Mamba-SSM compare to other sequence modeling architectures such as Transformers and xLSTMs on these tasks?
- **RQ3:** What are the strengths and weaknesses of Mamba-SSM when evaluated with Chomsky hierarchy tasks?

To address these questions we evaluate Mamba-SSM on 15 different tasks based on the Chomsky hierarchy to build the foundation for future research and application of the architecture.

2.4 Hypotheses

This thesis explores the capabilities of Mamba-SSM by applying it to synthetic tasks from the Chomsky hierarchy. Based on theoretical insights and recent empirical findings by Beck et al. 2024, the following specific hypotheses are formulated.

- **Hypothesis:** Mamba-SSM will achieve similar or better accuracies in comparison to transformers on data generated with the code provided by Delétang et al. 2023
  Justification: As stated by Gu and Dao 2023 Mamba-SSM is an alternative to transformers but with a lower computational complexity of $O(n)$ instead of $O(n^2)$ and better performance on longer sequences. Therefore, it should achieve similar or higher accuracies than Transformers when applied to synthetic tasks in the Chomsky hierarchy.

2.5 Thesis Structure

This thesis is organized into six chapters:

- **Chapter 1: Story Summary** - Gives an overview about the central questions of the thesis.
- **Chapter 2: Introduction** — Presents the motivation, research gap, objectives, and scope of this work.
- **Chapter 3: Related Work** — Surveys prior research on sequence modeling, the Chomsky hierarchy, and the development of Mamba-SSM, Transformers, and xLSTM architectures.
- **Chapter 4: Methodology** — Details the experimental design: data generation (Chomsky-hierarchy tasks), model architectures (binary, multiclass, generative Mamba-SSM), hyperparameters, and training/testing protocols.
- **Chapter 5: Results** — Presents quantitative outcomes for all 15 tasks, including accuracy curves, heatmaps, and comparisons against results reported by Delétang et al. 2023 and Beck et al. 2024.

- **Chapter 6: Discussion** — Interprets findings in light of the research questions, analyzes performance differences across tasks and models, and examines strengths, limitations, and potential sources of discrepancy with prior work.

3 Related Work

Sequence modeling is a fundamental challenge in machine learning, with applications spanning language modeling, time-series forecasting, and bioinformatics. Traditional architectures like Recurrent Neural Networks (RNNs) and Long Short-Term Memory networks (LSTMs) (Hochreiter and Schmidhuber 1997) laid the groundwork, but Transformers (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, A. Gomez, et al. 2017) have become the dominant architecture due to their ability to capture long-range dependencies. However, Transformers suffer from quadratic time complexity with respect to sequence length, prompting research into more efficient alternatives.

One such alternative is the Selective State-Space Model (SSM) architecture, *Mamba-SSM*, introduced by Gu and Dao 2023. Mamba achieves linear time complexity $O(n)$ by gating state updates and utilizing a diagonal-plus-low-rank structure. It has demonstrated competitive performance across various modalities, including language, audio, and genomics.

Building upon this, the *Hyena Hierarchy* model proposed by Poli, Massaroli, et al. 2023 employs depth-wise long convolutions combined with gating mechanisms, achieving performance comparable to Transformers while scaling as $O(n \log n)$. Further advancements include *StripedHyena* and its successor *StripedHyena2*, which extend this approach to models with up to 7 billion parameters, using synthetic tasks as unit tests to guide training (Poli, Thomas, et al. 2024; Ku et al. 2025).

In parallel, the *RWKV* family of models revisits RNN architectures. RWKV-5, for instance, introduces matrix-valued hidden states, enabling competitive perplexity scores at strictly linear computational costs (Peng et al. 2024).

Despite these advancements, challenges remain. Studies have shown that even large-scale models struggle with certain structured tasks. For example, Higuchi et al. 2025 demonstrated that GPT-sized models have difficulty with probabilistic context-free grammars

(PCFGs). Similarly, Kallini et al. 2024 highlighted that some synthetic languages are effectively "impossible" for these models to learn reliably.

Theoretical analyses have further scrutinized the capabilities of SSMs. **merrill2024illusion** proved that SSMs, including Mamba, cannot express computations beyond the complexity class TC^0, limiting their ability to perform tasks like permutation composition. Jelassi et al. 2024 focused on the copying problem, concluding that while small Transformers can perfectly copy sequences of exponential length, models with fixed-size latent states, such as generalized SSMs, are fundamentally limited in this regard.

Conversely, some research offers a more optimistic view. **nishikawa2025provably** demonstrated that SSM layers combined with non-linear layers can match Transformers in dynamic token selection tasks. Hybrid models like *Zamba* (Glorioso et al. 2024), which interleave Mamba-SSM layers with Transformer self-attention layers, have achieved performance on par with similarly sized pure Transformers while offering faster inference.

Extensions of Mamba have also been explored. Domain-specific adaptations, such as *ClinicalMamba* for medical coding tasks (**clinicalmamba2024**), and multi-modal approaches like *Mixture-of-Mamba* (MoM) (**pioro2024moe**), further showcase the versatility and ongoing development of SSM-based architectures.

The *Extended Long Short-Term Memory* (xLSTM), introduced by Beck et al. 2024, enhances traditional LSTM architectures through matrix-based memory structures and exponential gating mechanisms. xLSTM achieves linear computational complexity and improved scalability, enabling competitive performance on language and time-series tasks (Beck et al. 2024).

In summary, while SSMs like Mamba offer promising alternatives to Transformers, especially in terms of efficiency, they face inherent limitations in expressivity. Hybrid models and architectural innovations continue to bridge these gaps, highlighting a dynamic and evolving landscape in sequence modeling research.

Importantly, this thesis extensively references Beck et al. 2024 and Delétang et al. 2023, whose results serve as critical baselines for comparing the accuracy of our proposed models. Both studies employ the Chomsky hierarchy as a structured framework to evaluate the capabilities of diverse sequence modeling architectures, thus providing essential context for interpreting and situating our empirical findings.

In the following sections, we provide an overview of the most relevant research for this paper with more detailed explanations.

3.1 The Chomsky Hierarchy

The Chomsky Hierarchy is a method to determine the complexity of a language in the context of formal grammars. It distinguishes between 4 levels with increasing complexity: regular (Type 3), context-free (Type 2), context-sensitive (Type 1) and recursively enumerable (Type 0) (Delétang et al. 2023). This paper covers tasks based on the first 3 of these complexity levels. The 4 levels are described as follows:

1. Regular languages are the simplest class in the Chomsky hierarchy and can be recognized by finite state automata (FSA). They do not require memory beyond the current state, meaning they cannot handle nested or recursive structures. Their grammar rules are strictly linear, allowing only simple patterns. Examples include sequences like a*b* (any number of a's followed by any number of b's). The regular problems used here are Cycle Navigation, Even Pairs, Modular Arithmetic (without brackets) and Pairity Check.

2. Context-free languages (CFLs) are more expressive than regular languages and can be recognized by pushdown automata, which use a single stack as memory. This allows them to handle nested structures, such as balanced parentheses or arithmetic expressions. Their grammar rules allow a single non-terminal to be replaced with a sequence of terminals and non-terminals. The Context-free problems used here are Reverse String, Modular Arithmetic (with brackets), Solve Equation and Stack Manipulation.

3. Context-sensitive languages (CSLs) require more computational power and are recognized by linear bounded Automata, a restricted type of Turing machine with a bounded tape. Unlike context-free languages, context-sensitive languages can enforce constraints between distant parts of a string, making them suitable for modeling complex syntactic structures. The Context-sensitive problems used here are Binary Multiplication, Binary Addition, Odds First, Bucket Sort, Missing Duplicate String, Compute Squareroot and Duplicate String.

4. Recursively enumerable languages are the most powerful class in the hierarchy and can be recognized by a Turing machine, meaning they can represent any computable function. Their grammars have no restrictions, allowing arbitrary transformations of strings. However, some problems in this class, such as the Halting Problem, are undecidable, meaning no algorithm can determine the answer in finite time. No problems of this levels are used here.

Synthetic Problems. To evaluate Mambas capabilities we use syntetic problems that are classified within the Chomsky hierarchy. In total we use 15 such problems from all classes of the Chomsky hierarchy but the recursively enumerable class. The generation of these task is based on the GitHub of the paper by Delétang et al. 2023. The various tasks are solved by one of three different Mamba-SSM model types: Binary classifier, multiclass (5 classes) classifier and a generative model. This is because current models generally do not reach this level yet. In the following each synthetic problem is given with an explaination as well as an example, of which all are found in the corresponding complexity levels.

Regular (Type 3):

Even Pairs: Given a binary sequence such as *ababaab* the model should determine if the number of *ab*'s and *ba*'s is even (true) or odd (false). In this case we have three times *ab* and two times *ba*, which results in a total of 5, resulting in the output false (or odd). The model used for this task is the binary classifier.

Modular Arithmetic (simple): Provided with a sequence of numbers in the set $\{0, 1, 2, 3, 4\}$ and operations $\{+, -, \cdot\}$, the model should compute the modulo 5 of the result. This variation of the task does not include brackets. Given the example $x = 3 + 2 \cdot 1$ the model should reach the result 0. The model used for this is the multiclass classifier.

Parity Check: Given a binary string such as *bbbaab* the model should determine if the number of *b*'s is eve (true) or odd (false). In the provided example the model should come to the result even (true). The used model for this is the binary classifier.

Cycle Navigation: Given a cycle of length 5 and a sequence of the letters $\{0, 1, 2\}$ corresponding to the actions 'stay', 'increase' and 'decrease', the model should determine the position after the commands of the sequence were executed. It always starts at position 0. Provided the example sequence 10211 the output of the model should be 2. The used model for this is the multiclass classifier.

Context-free (Type 2):

Stack Manipulation: Given a binary sequence such as *ababba* and a set of actions PUSH a, PUSH b and POP, the model should determine the binary sequence after a sequence of actions has been performed. If we apply the action sequence PUSH b, POP, PUSH b, we get the resulting binary sequence *ababbab*. If a POP action is performed on an empty sequence, nothing happens. For this task the generative model is used.

Reverse String: The task of this problem is to simply reverse a binary sequence like *bbaa* to the result *aabb*. Here the generative model is used.

Modular Arithmetic: This is just like Modular Arithmetic (simple) but with brackets added to the set of used symbols. Again the model should determine the modulo 5 of the result. Here again the multiclass classifier is used.

Solve Equation: Given an equation containing numbers of the set $\{0, 1, 2, 3, 4\}$, the operators $\{+, -, \cdot\}$, brackets and an unknown variable z, the model should determine the modulo 5 of the value of z where the equation is equal to 0. So for the example $4 \cdot (3 - z) = 0$ the model should give the output 3. For this task the multiclass classifier is used.

Context-sensitive (Type 1):

Duplicate String: Given a binary sequence such as *aabba* the task of the model is to generate the dupicated string. For the provided example this should be *aabbaaabba*. We use the generative model to solve this task.

Missing Duplicate: In this task a binary sequence is given where a token has been hidden. The input looks like *a_bba* (with $w = aab$) the output of the model should be *a*. The binary classifier is used for this task.

Odds First: Given a binary sequence such as *abbaab*, the result is the sequence of the tokens at the odd positions followed by the sequence of the tokens at the even positions. Therefore the models output should be *ababab*. The model trained on this task is the generative model.

Binary Addition: Given two binary numbers such as 0001 (1) and 1000 (8) the model should perform addition on these binary numbers and get to the result 1001 (9). For this the generative model is used.

Binary Multiplication: This task also uses two binary numbers such as 0001 (1) and 1000 (8) where the result given by the model should be 1000 (8). Again the generative model is used to solve this task.

Compute Sqrt: Here the model should compute the square root of a binary number such as 0100 (4) to which the corresponding result is 0010 (2). Here the generative model is used.

Bucket Sort: Given a sequence generated by using an alphabet of fixed size 5 the models task is it to generate the ordered sequence. So for the sequence 234021 the corresponding output should be 012234. For this task the generative model is used.

3.2 Transformer Architecture

The Transformer, introduced by Vaswani, Shazeer, Parmar, Uszkoreit, Jones, A. Gomez, et al. 2017, is a neural architecture designed for sequence modeling tasks such as translation, summarization, and language modeling. It departs from traditional recurrent and convolutional models by relying entirely on self-attention mechanisms, enabling efficient parallelization and modeling of long-range dependencies. This section presents a mathematically grounded overview of the core components of the Transformer.

Input Representation. Given a sequence of input tokens $\{x_1, x_2, \ldots, x_n\}$, each token is mapped to a d_{model}-dimensional embedding vector. To preserve the order of the sequence—which the Transformer lacks inherently— positional encodings are added to these embeddings. The resulting input matrix is $X \in \mathbb{R}^{n \times d_{\text{model}}}$. The positional encoding used in the original paper (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, A. Gomez, et al. 2017) is deterministic and defined as:

$$\begin{aligned} \text{PE}_{(i,2k)} &= \sin\left(\frac{i}{10000^{2k/d_{\text{model}}}}\right) \\ \text{PE}_{(i,2k+1)} &= \cos\left(\frac{i}{10000^{2k/d_{\text{model}}}}\right) \end{aligned} \tag{3.1}$$

where i is the token index and k is the dimension index. These sinusoidal encodings allow the model to learn relative position information.

Scaled Dot-Product Attention. Attention is the mechanism by which the model selectively focuses on relevant parts of the sequence. For each token, a query (Q), key (K), and value (V) vector are computed via learned linear projections:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V \tag{3.2}$$

where $W^Q, W^K, W^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ are learnable matrices. The attention output is computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{3.3}$$

This yields a weighted sum of the value vectors, with weights determined by the similarity between queries and keys.

Multi-Head Attention. To enable the model to jointly attend to information from different representation subspaces, the Transformer applies attention in parallel across multiple heads. For h heads:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{3.4}$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$ and $W^O \in \mathbb{R}^{hd_k \times d_{\text{model}}}$ is a final output projection.

Position-wise Feed-Forward Networks. After the attention mechanism, each token representation is passed through a feed-forward network (FFN) that is applied identically at every position. The FFN consists of two linear transformations with a ReLU activation:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{3.5}$$

where $W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$, $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$ and b_1, b_2 are biases.

Residual Connections and Layer Normalization. Each sub-layer in the Transformer (attention or FFN) is wrapped in a residual connection followed by layer normalization:

$$\text{LayerNorm}(x + \text{Sublayer}(x)) \tag{3.6}$$

This helps stabilize training and improve gradient flow in deep models.

Encoder and Decoder. The Transformer consists of an encoder and a decoder, each built from N identical layers. The encoder layer contains a multi-head self-attention block and an FFN. The decoder layer adds a masked multi-head self-attention mechanism before attending to the encoder output, followed by its own FFN.

Output Generation. The final decoder output is projected into the vocabulary space using a linear transformation followed by a softmax:

$$\text{Output} = \text{softmax}(xW^T + b) \tag{3.7}$$

This yields a probability distribution over the output vocabulary for each position in the sequence.

Conclusion. The Transformer architecture offers an efficient and scalable solution for modeling sequences. By replacing recurrence with self-attention and leveraging parallelism, Transformers achieve state-of-the-art performance in many domains.

3.3 Mamba-SSM Architecture

Mamba-SSM represents a significant advancement in sequence modeling by extending traditional State Space Models (SSMs) with a selective mechanism, resulting in a Selective State Space Model (SSSM) (Gu and Dao 2023). This architectural enhancement enables the model to process long-range dependencies more efficiently by dynamically selecting and retaining relevant state information. Unlike conventional SSMs, which apply fixed transformations to all input states, Mamba-SSM introduces a data-dependent selection process, allowing it to focus on the most relevant information at each time step.

This design not only improves computational efficiency—achieving linear time complexity O(n)O(n) compared to the quadratic complexity of Transformers—but also enhances the model's ability to generalize over long sequences. As a result, Mamba-SSM has shown promising performance in a variety of sequence modeling tasks, particularly in scenarios requiring efficient memory utilization and structured reasoning. The following sections provide a detailed examination of SSMs, the theoretical foundation of selective state spaces, and an analysis of how these modifications contribute to Mamba-SSM's capabilities.

SSM: State space models. Firstly, it is important to mention that this formulation of State Space Models (SSMs) is specifically used in the Mamba-SSM paper by Gu and Dao 2023 and might vary in different applications of SSMs. At each time step, the hidden state is updated according to a state transition matrix A, which determines how past information influences future states. The model also incorporates external input through an input matrix B, which transforms incoming data before integrating it into the system. Finally, an observation matrix C maps the internal hidden state to an output, allowing the model to generate predictions or process sequential data efficiently. In addition, there is Δ, which weighs A and B. A core strength of SSMs is, that their output can be computed using convolution, which enables parallel sequence processing (Gu, Goel, and Ré 2022). In mathematical notation, SSMs look like the following:

$$x_{t+1} = x_t + \Delta(Ax_t + Bu_t) \qquad y_t = Cx_t \tag{3.8}$$

- A: State transition matrix, representing the hidden state over time.

- B: Input projection matrix, which transforms the input before it is processed in the model itself.
- C: Observation matrix maps the hidden state to an output.
- Δ: Discretization step size enables stable updates when transitioning from continuous to discrete time.
- u_t: Input at time step t.
- x_t: Hidden state represents past information.
- y_t: Output given at time step t.

By using convolution, SSMs can be efficiently computed in the frequency domain (Gu, Goel, and Ré 2022):

$$y = (Ce^{\Delta A}B) * u \tag{3.9}$$

Despite convolution, large amounts of memory may be required for long sequences, since all hidden states for every time step need to be stored. Also, all states are treated equally which leads to misrepresentation of important and insignificant parts of the sequence. That way, it is difficult for SSMs to handle long-range dependencies. Furthermore, the linearity of the transformation $Ax_t + Bu_t$ limits its expressive power so that more complex hierarchical patterns as in natural language and other structured data cannot be recognized (Gu, Gupta, et al. 2022). Mamba-SSM introduces a new variation called selective state space models (SSSM).

Selective State Space Models. This new variation introduces selectivity by using a gating mechanism $G(x_t, u_t)$ that determines how much a new update should be applied (**dauphin2017language**; Gu and Dao 2023). It ensures that relevant states still receive the updates as needed but leaves unimportant states unchanged. This reduces computing time, since it only updates relevant hidden states, while ignoring insignificant ones. This enables the model to handle long-range dependencies more effectively (Gu, Goel, and Ré 2022). The new update of the hidden state therefore looks like this:

$$x_{t+1} = x_t + \Delta \cdot G(x_t, u_t) \cdot (Ax_t + Bu_t) \tag{3.10}$$

Furthermore, it adds a nonlinear activation function (like GELU or Swish) (Hendrycks and Gimpel 2023; Ramachandran, Zoph, and Le 2017) which increases its ability to represent more complex patterns in the data such as nested dependencies in natural language or long-range dependencies in time series . This changes the update step of the hidden state to this formulation:

$$x_{t+1} = x_t + \Delta \cdot G(x_t, u_t) \cdot \sigma(Ax_t + Bu_t) \tag{3.11}$$

The function $G(x_t, u_t)$ is a data-dependent gating function that basically decides if a hidden state is relevant or not and is learned during the training process. It depends on both the current hidden state x_t and the input u_t. If $G(x_t, u_t)$ is close to 0, a hidden state is not updated, if it is large, the update is stronger on the respective hidden state (Gu and Dao 2023). These adaptations maintain the ability to apply convolution and therefore keep it computationally efficient.

Parameterization of the State Transition Matrix A. Another key innovation of Mamba-SSM is its efficient parameterization of the state transition matrix A. Due to the high computational cost of parts such as $e^{\Delta A}$, which requires large amounts of memory and is slow, they use a diagonal-plus-low-rank structure to compute the exponentials of matrices (Gu and Dao 2023). This severely reduces computational complexity while still maintaining expressiveness and makes it scalable for long-range modeling.

Scan and FFT: How Mamba-SSM Enables Parallel Computation. Unlike RNNs, Mamba-SSM does not compute hidden states sequentially but instead uses a parallel scan to compute all state updates in parallel (Gu and Dao 2023). In this way, this reduces the core weakness of RNNs and severely reduce computational complexity . Additionally, Mamba-SSM applies FFT-based convolutions, which allow it to process sequence information in the frequency domain, further reducing computational cost while preserving long-range dependencies (Gu, Goel, and Ré 2022). These optimizations enable Mamba-SSM to

retain the expressive power of state-space models while achieving linear-time complexity, making it a highly scalable alternative to Transformers for sequence modeling.

4 Experiments

This chapter covers the details of our experimental setups. We explain the specifics of the datasets, as well as technical setup and the various models used in the experiments. It also elaborates on the training process in great detail.

4.1 Setup

For the setup we distinguish between the specifications of technical or hardware details, data, models and training to ensure reproducibility.

4.1.1 Technical Setup

All experiments and the full data generation process were performed with Google Colab. The models were trained using the T4 GPU, which is available on Google Colab at the time these experiments were run (April 2025).

4.1.2 Setup: Data

For the data we took a downscaled approach due to computational limitations in comparison to the xLSTM paper (Beck et al. 2024). We pre-generated the data and used a training, a validation and a test set. As for the training process the maximum sequence length is 40, we generated 100 batches per sequence length for the training data and 10 batches for the validation set. For the testing data, we also generated 10 batches per sequence length, which reached 41 to 500. So given the 100 batches per sequence length in the training data, we reach a total number of update steps per epoch of 4000.

4.1.3 Setup: Models

In total there are three experiments, each for a different set of hyperparameters, only differing in dimensionality, applied to the 15 different tasks of the Chomsky hierarchy. It should be noted that although for all 15 tasks the same hyperparameters are used, depending on the task, we apply the corresponding binary, multiclass, or generative model. All of the models are based on the Mamba-SSM architecture. The following table displays the various chosen hyper parameters for the experimental setups. All three experiments use 2 blocks and a dimensionality of 64, 128 and 256. Also, a dropout of 0.1 is applied, and the RMS-Norm is used before and after each Mamba-SSM block. All this is applied to a positional embedding.

4.1.4 Setup: Training

For each experiment, the model is trained over 3 epochs, which combined with the 4000 update steps gives us a total of 12000 update steps. Every 1000 batches the model is applied to the validation data and the losses as well as the accuracies are saved in plots. Some of these plots will later be used to give insights into the training process of various experiments. This differs from the approach of Beck et al. 2024 where 100k update steps were taken and the data was generated in the training itself. Due to our computational limitations we will take a scaled down approach and use epochs with pre-generated data. Additionally, we use gradient clipping with a global norm of 1.0 as well as the AdamW optimizer weight decay 0.1, $\beta_1 = 0.9$, $\beta_2 = 0.99$, and cosine LR schedule with 10% warm up.

4.2 Results

This section covers the results of the performed experiments based on the problems categorized in the Chomsky hierarchy. For reference the results are compared with the corresponding experiments performed by Delétang et al. 2023 and Beck et al. 2024. While there are deviations between the experimental setups of the two papers, it will be pointed out what paper a certain part of the setup is oriented by. Upfront, it can be said that compared to the results of Delétang et al. 2023 Mamba-SSM performs comparatively well,

while compared to the xLSTM paper of Beck et al. 2024 the results seem to deviate due to unknown reasons.

In the following the general results of the three experiments are presented with comparison to the results of Beck et al. 2024 and Delétang et al. 2023 respective papers. After that we go into detail with the experiments themselves and see if there are any peculiarities. We chose 3 different hyper parameter setups, only differing in the dimensionality, 64, 128 and 256, where 128 is the chosen dimensionality by Beck et al. 2024, to evaluate how low or high the models complexity has to be to perform well on the different tasks provided by the Chomsky hierarchy. But we will stick to having two Mamba-SSM-blocks for each models to preserve comparability with the other two relevant papers.

Comparison with Beck et al. 2024 results. First off, the xLSTM paper (Beck et al. 2024) only uses a subset of the synthetic tasks we used and two tasks we did not train our models on. Therefore, we only have a comparison with the following synthetic tasks: Bucket Sort, Missing Duplicate, Modular Arithmetic with and without brackets, Solve Equation, Cycle Navigation, Even Pairs and Parity Check. In the following table the naming convention for our experiments is Mamba_<hidden_dims>_<num_layers>. Note that each value is calculated depending on the random accuracy of each problem. So if a random accuracy would be at 0.20 for a classification with 5 classes and the model archives an accuracy of 0.75 (0.55 above random), the value in the table is calculated like $0.55/0.80 = 0.6875$ but rounded to two digits. The 0.80 is the difference between 1.00 and the random accuracy.

As visible in the heatmap 4.1, our results seem to slightly differ (or even strongly with Cycle Navigation) which we attribute to the difference in our training setups.

With respect to our **hypothesis** we get mixed results. As initially expected we get accuracies below Beck et al. 2024 results for the tasks Missing Duplicate, Modular Arithmetic with and without brackets, Solve Equation and Parity Check. In Bucket Sort our models outperformed Beck et al. 2024 models with close to perfect accuracies. But for Cycle Navigation we fall far behind with random accuracy while the model of Beck et al. 2024 achieves 0.86.

Comparison with Delétang et al. 2023 results. Since our experimental setup is mainly based on the results of the paper of Delétang et al. 2023, we also use all the Chomsky

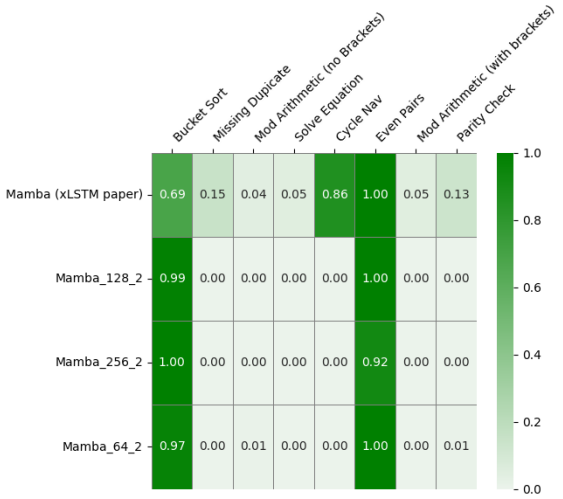


Figure 4.1: Experiment results compared with the paper of Beck et al. 2024.

hierarchy-based experiments. In the following is table 4.1 that extends the accuracy table of the paper by Delétang et al. 2023 with our results.

Table 4.1: Experiment results compared with the paper of Delétang et al. 2023.

Test Type	Task	RNN	Stack-RNN	Tape-RNN	Transformer	LSTM	Mamba_64_2	Mamba_128_2	Mamba_256_2
regular	Even Pairs	**100.0**	**100.0**	**100.0**	67.7	**100.0**	**100.0**	**99.6**	**96.5**
	Parity Check	**95.5**	**100.0**	**100.0**	50.4	**100.0**	50.5	50.4	50.5
	Cycle Navigation	**100.0**	**96.8**	**100.0**	33.9	90.0	20.4	19.8	20.3
	Modular Arithmetic (Simple)	**100.0**	**100.0**	**100.0**	23.1	**100.0**	19.6	19.7	20.2
dcf	Modular Arithmetic (brackets)	34.6	75.8	60.9	28.2	53.1	19.6	20.2	20.2
	Stack Manipulation	54.5	**93.8**	66.9	51.1	58.1	**90.0**	**99.3**	**100.0**
	Reverse String	60.6	**96.2**	**90.7**	56.1	59.5	**99.3**	**100.0**	**100.0**
	Solve Equation	31.3	46.1	37.7	23.5	37.1	20.1	20.1	20.2
cs	Duplicate String	50.2	51.6	**90.0**	52.8	56.5	**99.8**	**99.8**	**99.9**
	Missing Duplicate	51.7	53.6	66.1	53.8	53.1	49.2	50.1	50.2
	Odds First	50.4	51.5	**76.2**	52.7	54.6	**99.7**	**99.8**	**100.0**
	Binary Addition	49.5	50.9	**74.3**	51.7	54.8	**99.6**	**99.6**	**100.0**
	Binary Multiplication	49.6	51.4	54.5	50.4	52.9	**99.3**	**99.2**	**100.0**
	Compute Sqrt	53.9	56.1	55.6	51.5	57.3	**97.5**	**97.1**	**99.2**
	Bucket Sort	22.5	56.6	33.6	35.7	79.4	**97.8**	**98.9**	**99.8**

This shows Mamba-SSMs strong performance especially in context-sensitive tasks where all other model architectures seem to perform mostly poorly. On the other side it seems to underperform in most regular tasks which are easily solved by the classical models like RNNs and LSTMs. In context-free tasks it does perform very well with stack Manipulation and Reverse String, but does not get better than random for the rest of the tasks. All values above an accuracy of 90% are considered solved and are marked bold. Curiously,

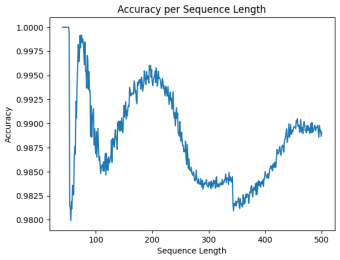


Figure 4.2: Bucket Sort.

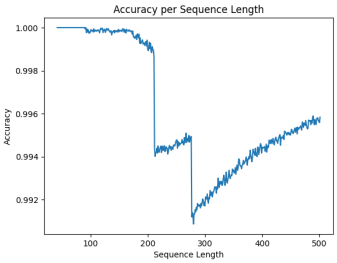


Figure 4.3: Binary Addition.

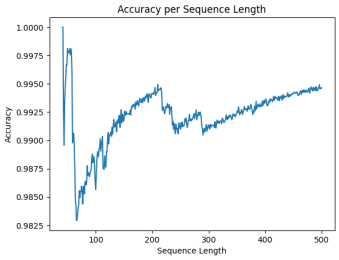


Figure 4.4: Binary Multiplication.

Figure 4.5: Accuracy vs. sequence length for three tasks.

while for all well-performing models increasing the number of dimensions improves the performance, the task even pairs is performing worse, although just a small bit.

Experiment 1: 128 hidden dimensions and 2 blocks As this set of hyperparameters matches the one used in the xLSTM paper (Beck et al. 2024) it gives the best comparability despite the highly reduced training efforts mentioned earlier. As seen in 4.1 its performance is higher than the Mamba-SSM-model used in the xLSTM paper for bucket sort and equal in even pairs. In all other tasks though it performs worse.

Inconsistent performances. As we evaluated the model performance on unseen sequence lengths from 41 to 500, we find in some plots very bumpy accuracies for increasing sequence lengths. Examples of this are provided in 4.5.

On these three tasks the Mamba-SSM-models are performing very well but we can see in the plots that their accuracies are behaving very strangely, even though they are still very high. While all of them seem to recover accuracy for higher sequence lengths, there are some sharp drops from one point to the next. Recent research by Ben-Kish et al. 2024 finds that this issue to generalize over unseen sequence lengths is based in Effective Receptive Field (ERF) Bias. This Bias describes that the generalization is roughly bounded by the maximum sequence length found in the training data. This causes Mamba-SSM to miss long-range dependencies. But this still does not explain the sudden bumps as well as the following slow recovery we witness in some of the accuracies over the sequence lengths.

Experiment 2: 64 hidden dimensions and 2 blocks. To cover also the more normal but still deviating results, a look at Stack Manipulation for the 64 hidden dimensions model is appropriate. In the plot we see the expected decline of accuracy for increasing sequence lengths.

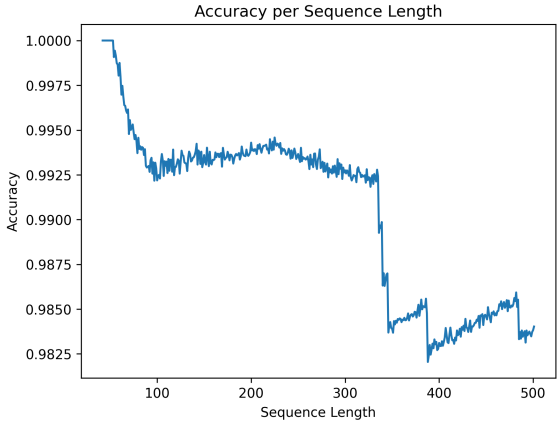


Figure 4.6: Task Stack Manipluation.

Experiment 3: 256 hidden dimensions and 2 blocks. This is the as expected best performing model generally but it also seems to have a few interesting results that might be able to give hints at challenges or potentials Mamba-SSM has. When we look at the accuracies of the task Duplicate String over the sequence lengths the plot shows a quite strange behavior. We would expect accuracy to start off high and then slowly drop but in fact it has sudden plummets and rises unexpectedly as visible in figure 4.7.

Although it has near perfect accuracy, these are still very unexpected results. Especially from a sequence length of about 150 to 200 the plateau seems to fill in a spot where we would expect gradual increase when comparing to the lengths before and after. Additionally, we again see the plummet early on and a steady increase instead of decrease in accuracy as it nears perfect accuracy again with longer sequences.

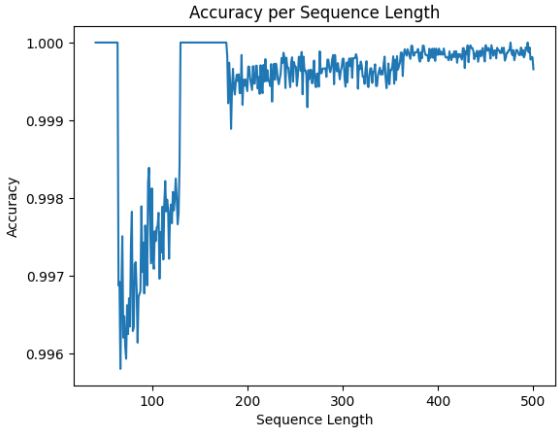


Figure 4.7: Task Duplicate String.

5 Discussion

As we compared our results both with the paper of Delétang et al. 2023 and Beck et al. 2024 we draw our conclusions from the tables 4.1 and 4.1. As both papers use slightly different tests to evaluate the performances of various models, and at the time Delétang et al. 2023 paper was published Mamba-SSM didn't exist yet, we look at the results in different subsections.

5.1 Performance Analysis

Comparison with Beck et al. 2024 In comparison with Beck et al. 2024 we have one deviation regarding our experimental setup. While Beck et al. used a maximum sequence length of 256, we stuck with the 500 from Delétang et al. 2023 for better generalization. Besides that we fully oriented the architecture of our Mamba-SSM model as well as the training process on the information we found in the paper of Beck et al. 2024. Also due to limited computational resources we had far less training on our models. This lead to worse performance but due to our precise imitation of all other factors we are certain that is simply due to less training. Mostly our models performed very well on the tasks where the Mamba-SSM of Beck et al. 2024 also performed well. Other than that, while the model of Beck et al. 2024 reaches slightly above random for many other tasks, our results didn't get better than random in those cases. But simply looking at Bucket Sort we can see that our models are performing better than the Mamba-SSM model of Beck et al. 2024. This might be due to overfitting on the Mamba-SSM model of Beck et al. 2024 we assume, which indicates the task Bucket Sort to be prone to this effect. Cycle Navigation on the other hand the Mamba-SSM model of Beck et al. 2024 is performing very well, while our models can't seem to learn anything. If this is simply our low computational resources or another reason is unclear.

Comparison with Delétang et al. 2023 As visible in table 4.1 the performance of Mamba-SSM is very good, especially in context-sensitive tasks. The only task where it doesn't get nearly perfect performance is the missing duplicate task. This counts for all 3 experiments with different choices of dimensionality. Since the picks for the tasks in this thesis are based on this paper, we have full comparability with the other models represented in the paper of Delétang et al. 2023. Also we took over the experimental setup and used the code from Delétang et al. 2023 GitHub to generate our data, which is why our maximum sequence length of 500 in the model evaluation and the general generation of data is best comparable.

In the context of our **research question 1** we see that Mamba-SSM performs exceptionally well with context-sensitive tasks and also context-free tasks but seems to fail for regular tasks. This shows that Mamba-SSM might be very well suited in context-free and context-sensitive problems but is not a good pick for regular tasks. Basically it performs better on more complex tasks according to the Chomsky hierarchy and rather poorly on simpler tasks. For **research question 2** we can see in our comparison with Delétang et al. 2023 that is outperforms transformers (and also many of the other architectures) within the limitations of the experiments. But when comparing it in the context of the experiments in the paper by Beck et al. 2024 it seems to perform just slightly better than other models like Llama (Touvron et al. 2023) while all of the models are heavily outperformed by xLSTM (Beck et al. 2024. With **research question 3** Mamba-SSM shows very good performance with, according to the categorization of the Chomsky hierarchy, more challenging tasks while underperforming on regular ones.

When comparing these results with our **hypothesis** we significantly outperform the transformers. While transformers do not much above random accuracies for context-sensitive tasks, our Mamba-SSM models reach near perfect performances for all tasks but Missing Duplicate. For regular and context-free tasks we can also say that we outperform transformers, although not as much as for context sensitive tasks.

5.2 Limitations and Future Work

Sudden changes in accuracy: As seen in many tasks (like 4.6 or 4.7) the accuracy sometimes varies widely from one sequence length to the other. Although for many cases

this mainly shows up for already high accuracies, the drops and plateaus when looking at the accuracies on all sequence lengths still persist. This might indicate some issue of Mamba-SSM to deliver consistent and smooth results and therefore offers a next step in future research to look into.

Performance asymmetry: Our models reach 98-100% accuracy on context-sensitive tasks such as Duplicate String and Binary Multiplication, yet hover near chance (50%) on Regular tasks like Parity Check. This suggests that Mamba's selective SSM excels in stack-like long-range dependencies, but lacks an inductive bias for global parity, mirroring observations in the paper of Sarrof, Veitsman, and Hahn 2024.

Scaling outlook: Under our resource-constrained setting (batch size 32, 12 k updates) the gap to the xLSTM study by Beck et al. 2024 remains; increasing to batch size 128 and 100 k updates, as in Beck et al. 2024, is a priority for future replication. Despite our computational limitations Mamba-SSM shows on par results with the task Even Pairs and even surpasses the Mamba-SSM model of Beck et al. 2024 (at 69%) with good 99% at same architectural setup.

Synthetic unit tests as predictors: The MAD pipeline (Poli, Thomas, et al. 2024) shows that micro-tasks can forecast scaling-law winners. Our results partly validate this claim: the same Regular-task weakness that MAD flags for pure SSMs re-appears in full-scale models. Integrating MAD scores into our hyper-parameter search could therefore save ≈ 4 GPU-days per experiment.

Limitations: We report single-seed results and evaluate only on synthetic languages; real-world corpora with hierarchical data (e.g., code, maths proofs) remain future work. Evaluating beyond a sequence length of 500 is still a task to be done.

Next steps: Two concrete avenues are (i) hybridising Mamba with a parity-aware sparse-attention head, and (ii) porting Mamba's data-dependent gating to RWKV-5's (Peng et al. 2024) matrix state, potentially combining the best of recurrent and SSM paradigms.

Bibliography

Beck, Maximilian et al. (2024). "xLSTM: Extended Long Short-Term Memory". In: *arXiv preprint arXiv:2405.04517* (cit. on pp. 1, 2, 4, 5, 6, 9, 20, 21, 22, 23, 24, 27, 28, 29).

Ben-Kish, Assaf et al. (2024). *DeciMamba: Exploring the Length Extrapolation Potential of Mamba*. arXiv: 2406.14528 [cs.LG]. URL: https://arxiv.org/abs/2406.14528 (cit. on p. 24).

Delétang, Grégoire et al. (2023). "Neural Networks and the Chomsky Hierarchy". In: *11th International Conference on Learning Representations* (cit. on pp. 1, 2, 5, 6, 9, 10, 11, 21, 22, 23, 27, 28).

Glorioso, Paolo et al. (2024). *Zamba: A Compact 7B SSM Hybrid Model*. arXiv: 2405.16712 [cs.LG]. URL: https://arxiv.org/abs/2405.16712 (cit. on p. 9).

Gu, Albert and Tri Dao (2023). "Mamba: Linear-Time Sequence Modeling with Selective State Spaces". In: *arXiv preprint arXiv:2312.00752* (cit. on pp. 1, 4, 6, 8, 16, 17, 18).

Gu, Albert, Karan Goel, and Christopher Ré (2022). "Efficiently Modeling Long Sequences with Structured State Spaces". In: *International Conference on Learning Representations (ICLR)* (cit. on pp. 16, 17, 18).

Gu, Albert, Ankit Gupta, et al. (2022). "On the Parameterization and Initialization of Diagonal State Space Models". In: *Advances in Neural Information Processing Systems* (cit. on p. 17).

Hendrycks, Dan and Kevin Gimpel (2023). *Gaussian Error Linear Units (GELUs)*. arXiv: 1606.08415 [cs.LG]. URL: https://arxiv.org/abs/1606.08415 (cit. on p. 18).

Higuchi, Rei et al. (2025). "When Does Metadata Conditioning (NOT) Work for Language Model Pre-Training? A Study with Context-Free Grammars". In: arXiv: 2504.17562 [cs.CL] (cit. on p. 8).

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Computation* 9.8, pp. 1735–1780 (cit. on pp. 1, 8).

Jelassi, Samy et al. (2024). *Repeat After Me: Transformers are Better than State Space Models at Copying*. arXiv: 2402.01032 [cs.LG]. URL: https://arxiv.org/abs/2402.01032 (cit. on p. 9).

Kallini, Julie et al. (2024). "Mission: Impossible Language Models". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*. Introduces a suite of "impossible" synthetic languages spanning the Chomsky hierarchy for probing LLM limits. Bangkok, Thailand, pp. XXXXX–XXXXX. URL: https://arxiv.org/abs/2401.06416 (cit. on p. 9).

Ku, Jerome et al. (2025). "Systems and Algorithms for Convolutional Multi-Hybrid Language Models at Scale". In: Presents *StripedHyena 2*, a convolutional multi-hybrid model with 2× throughput vs. linear-attention / SSM baselines. arXiv: 2503.01868 [cs.LG] (cit. on p. 8).

Peng, Bo et al. (2024). "Eagle and Finch: RWKV with Matrix-Valued States and Dynamic Recurrence". In: arXiv: 2404.05892 [cs.CL] (cit. on pp. 8, 29).

Poli, Michael, Stefano Massaroli, et al. (2023). *Hyena Hierarchy: Towards Larger Convolutional Language Models*. arXiv: 2302.10866 [cs.LG]. URL: https://arxiv.org/abs/2302.10866 (cit. on p. 8).

Poli, Michael, Armin W. Thomas, et al. (2024). "Mechanistic Design and Scaling of Hybrid Architectures". In: Introduces the *StripedHyena* hybrid with interleaved Hyena and attention blocks. arXiv: 2403.17844 [cs.LG] (cit. on pp. 8, 29).

Ramachandran, Prajit, Barret Zoph, and Quoc V. Le (2017). *Searching for Activation Functions*. arXiv: 1710.05941 [cs.NE]. URL: https://arxiv.org/abs/1710.05941 (cit. on p. 18).

Sarrof, Yash, Yana Veitsman, and Michael Hahn (2024). *The Expressive Capacity of State Space Models: A Formal Language Perspective*. arXiv: 2405.17394 [cs.CL]. URL: https://arxiv.org/abs/2405.17394 (cit. on pp. 2, 29).

Touvron, Hugo et al. (2023). *LLaMA: Open and Efficient Foundation Language Models*. arXiv: 2302.13971 [cs.CL]. URL: https://arxiv.org/abs/2302.13971 (cit. on p. 28).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan Gomez, et al. (2017). "Attention Is All You Need". In: *Advances in Neural Information Processing Systems* (cit. on pp. 1, 2, 8, 13).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, et al. (2023). *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762 (cit. on p. 4).