Examining Cutting-Edge Neural Architectures for NLP: Mamba, RetNet, and Hyena

Advanced Machine Learning (AL_KSAIM_9_1)

MSc in Software Design with Artificial Intelligence

Technological University of the Shannon

A00326643: Rooshmica K R

A00326643@student.tus.ie

Table of Contents

Introduction

Natural Language Processing (NLP) is a fast-developing sub-domain of Artificial Intelligence (AI) that deals with enabling computers to read, write, and communicate in human language. NLP has changed drastically over the last couple of years, particularly because of the ubiquity of Transformer-based models and the pairing of huge datasets with large neural models. Over the past ten years, the area has seen radical change as researchers have created ever-more complex neural frameworks to tackle basic problems in linguistic data processing.
Three modern neural network architectures that have been put forth in the last five years are presented in this survey of the literature:

- **Mamba:** Selective state space model released in 2023
- **RetNet:** Retention networks introduced in 2023
- **Hyena:** A deep convolutional architecture introduced in 2023

This report explains their designs, breakthroughs, advantages, disadvantages, and performance on recent benchmarks. Each addresses particular issues in NLP for
instance, how to enable certain tasks to be more manageable, faster to process,
or deal with lengthy texts.

Modern Trends in NLP Neural Network Architectures

Recurrent neural networks (RNNs) and long short-term memory networks (LSTMs), two early NLP models, had trouble with lengthy sequences and sequential bottlenecks. Since 2017, NLP has been built on top of the Transformer architecture because of its parallelization and self-attention mechanism after Vaswani et al.'s 2017[1].

The following are recent developments in circumventing these limitations:

- **Linear-Time Sequence Models:** Unlike Transformers, which grows quadratically with sequence length, Mamba, RetNet, and Hyena all use effective models whose size grows linearly with sequence length.
- **Specialized processes:** Instead of using common attention processes, each architecture uses a unique mechanism for sequential data processing.
- **Hardware Optimization:** Implementation-specific elements that are tailored for hardware accelerators, such as GPUs, are increasingly considered in modern architectures.

Mamba

Overview

Mamba deviates from the Transformer model and was first presented by Gu et al. in December 2023[2]. It is a **State Space Model (SSM)**, which combines cutting-edge methods for increased efficiency with recurrent neural networks. Mamba was created to solve Transformers' quadratic complexity issue without compromising or lowering Transformer performance in lengthy sequences. Mamba introduces selective state space modelling, which enables the model to dynamically either retain or lose information based on the input content [2].

Mamba Architecture

Here is the Functional architecture of Mamba [3]

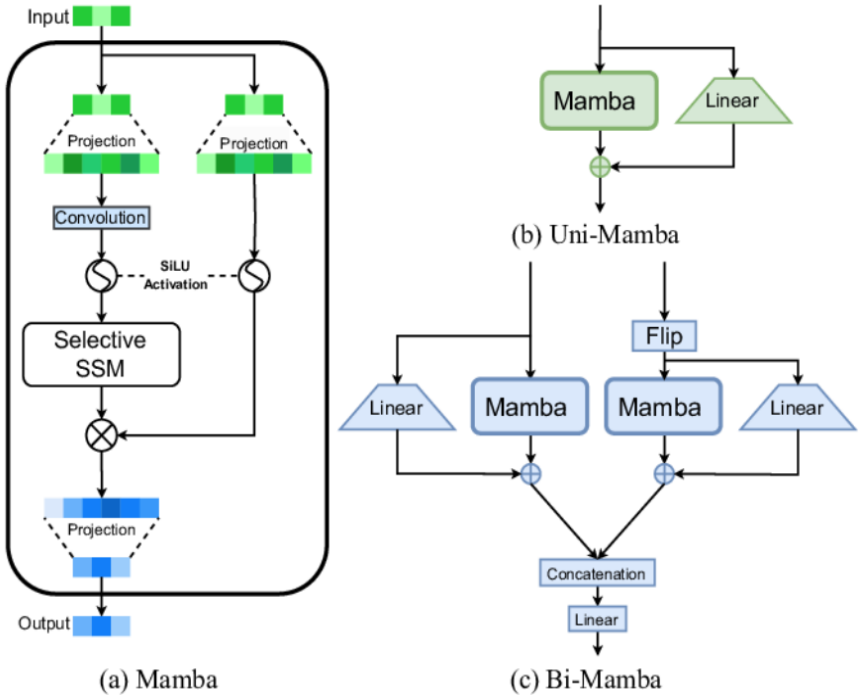


Figure 1: Mamba Functional Architecture [3]

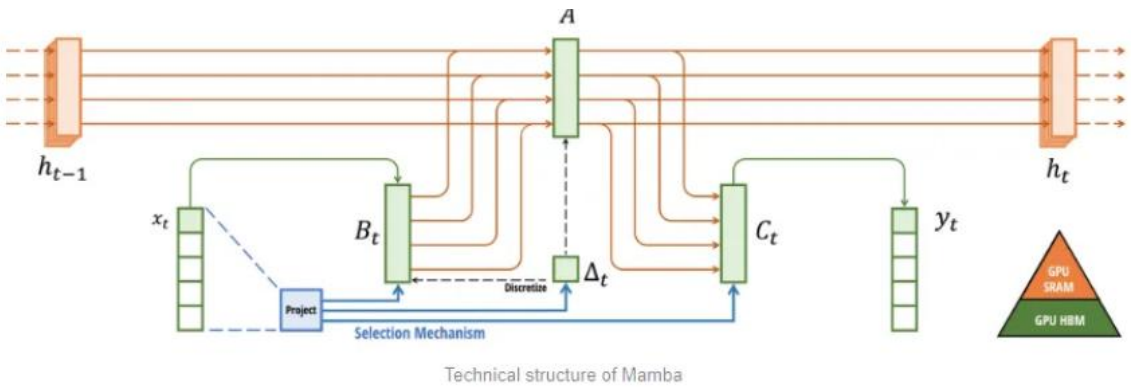


Figure 2: Mamba Technical Architecture [2]

Architecture Breakdown

Layer Types and Arrangement

Mamba is made up of alternating selective SSM layers and feedforward network (FFN) blocks [2]. A feedforward network and selective state space layer with sequential token processing is employed by the model, which also has a hidden state that changes according on the input sequence. The selected state space layer is used in place of the attention mechanism in Mamba, a decoder-only design that is comparable to GPT models [4].

Connectivity Patterns

Like Transformers, Mamba facilitates gradient flow by utilizing residual connections between layers. A selective method has been added to its structured state space sequence modelling (S4) component, which is the main innovation [2]. The selection process enables the model to preferentially focus on pertinent information by dynamically modifying the state development according to input content [4].

Activation Functions

The SiLU (Sigmoid Linear Unit, or Swish) activation function is the main one used by Mamba [2]

$$x * \text{sigmoid}(x) = \text{SiLU}(x)$$

To make the Gated MLP the well-liked "SwiGLU" form, we employ the SiLU/Swish activation mechanism. Finally, inspired by RetNet's use of a normalization layer in a comparable location, we also employ an optional normalization layer.

Learning Mechanisms

Using the AdamW optimizer and standard backpropagation, Mamba is trained [2]. Its learning strategy's main novelty is the hardware-aware algorithm that uses parallel scanning for training and sequential scanning for inference, enabling effective computation on contemporary hardware accelerators such as GPUs [3].

Specialized Components

- **Selective State Space Layer:** Mamba's core contribution is its selective state space approach that adaptively adjusts parameters according to input content, as opposed to fixed state transition parameters used in conventional SSMs for any input. This enables Mamba to selectively drop or retain data in its present state [2].
- **Hardware-Efficient Scanning Algorithm:** Through simultaneous model computing, Mamba's parallelization plan enables effective GPU training without interfering with sequential training. A specialized CUDA kernel makes this possible for both fast parallel scanning for training and sequential scanning for inference [2][3].

Key Contributions and Innovations

- **Selective Mechanism:** The most significant innovation of Mamba is its data-dependent parameter selection for the state space models, which allows it to change its behavior according on the input it is processing [2]. This enables the model to selectively store data on lengthy sequences.

- **Linear Scaling with Sequential Length:** Mamba scale linearly with sequence length, making it far more effective for processing lengthy documents than Transformers, which scale quadratically with sequence length [2][4].
- **Hardware-Efficient Implementation:** Mamba can be operated on current hardware thanks to the specialized CUDA kernels created for it, which allow for efficient parallel scanning during training and sequential scanning during inference [2].
- **Performance:** Mamba shows that state space models don't need attention mechanisms to perform as well as Transformer models on a variety of tasks without Attention. [2][5].

Strengths and Limitations

Strength

• **Effective Processing of Long Sequences:** Mamba can process significantly longer documents than Transformers with the same processing resources thanks to its linear scaling with sequence length [2][4].
• **Memory Efficiency:** For a given sequence length, the model uses a lot less memory than Transformers, which makes it more useful in settings with limited resources [2].
• **Comparable Performance:** Mamba outperforms Transformers of a similar size on a range of NLP tasks, despite disparities in architecture [2][5].
• **Adaptability:** Mamba can change its behaviour according to the input material according to the selected mechanism, which may increase its adaptability to various text types [2].

Limitations

- **Less Parallelizable:** This can affect throughput for short sequences because Mamba's sequential nature makes it intrinsically less parallelizable than Transformers during inference, even with hardware enhancements [3] [6].
- **Emerging Technology:** In contrast to the recognized Transformer ecosystem, Mamba has fewer optimized models, less community support, and a more recent architecture [5].
- **Limited Pre-training Exploration:** The knowledge and capacities of the current Mamba models may be limited because they were trained on fewer datasets than the largest Transformer models [2].
- **Specialized Hardware Requirements:** Although effective on GPUs, Mamba models' deployment possibilities may be restricted by the custom CUDA kernels needed for peak performance [2] [6].

Performance and Benchmarks

Mamba has proven to perform competitively in several benchmarks [2] [6]:
• **Language Modelling:** Mamba-2.8B outperformed Transformer models of a same size, achieving a perplexity of 7.08 on the C4 test.
• **Long-Context Understanding:**

Mamba achieved up to 5 times faster inference on long sequences than Transformers of equal size [2], outperforming attention-based models of comparable size on the Longaeval test. Additionally, it does well on standard NLP tasks like summarizing, answering questions, and classifying texts [6].

Model	Arch.	Layer	Acc.
S4	No gate	S4	18.3
-	No gate	S6	**97.0**
H3	H3	S4	57.0
Hyena	H3	Hyena	30.1
-	H3	S6	**99.7**
-	Mamba	S4	56.4
-	Mamba	Hyena	28.4
Mamba	Mamba	S6	**99.8**

Table 1: (**Selective Copying.**) Accuracy for combinations of architectures and inner sequence layers.

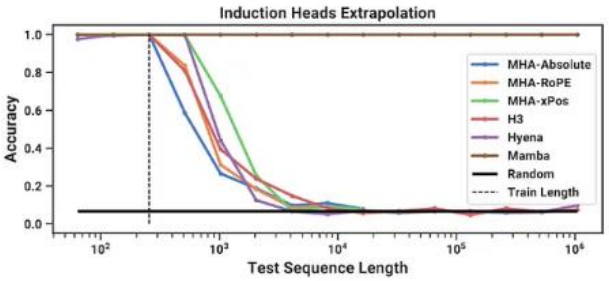


Table 2: (**Induction Heads.**) Models are trained on sequence length $2^8 = 256$, and tested on increasing sequence lengths of $2^6 = 64$ up to $2^{20} = 1048576$. Full numbers in Table 11.

Figure 3: Mamba Performance & accuracy comparing Hyena, H3 and S4 models [6]

Code Example

Example code from Hugging face[7]

```
from transformers import MambaConfig, MambaForCausalLM, AutoTokenizer
import torch

tokenizer = AutoTokenizer.from_pretrained("state-spaces/mamba-130m-hf")
model = MambaForCausalLM.from_pretrained("state-spaces/mamba-130m-hf")
input_ids = tokenizer("Hey how are you doing?", return_tensors= "pt")["input_ids"]

out = model.generate(input_ids, max_new_tokens=10)
print(tokenizer.batch_decode(out))
```

RetNet

Overview

Retention Networks (RetNet) introduced by Sun et al. in June 2023 [8] are a novel architecture that try to capture the strengths of both the Transformer and the RNN architecture. RetNet was introduced to remove the computational inefficiency of Transformers to process long sequences but continue to enable them to retain their parallel trainability as well as their performance level. The key innovation of RetNet is its "**retention**" mechanism, a powerful replacement for attention to allow the model to handle sequences of linear complexity and support both parallel and recurrent modes of computing [8] [9].

RetNet Architecture

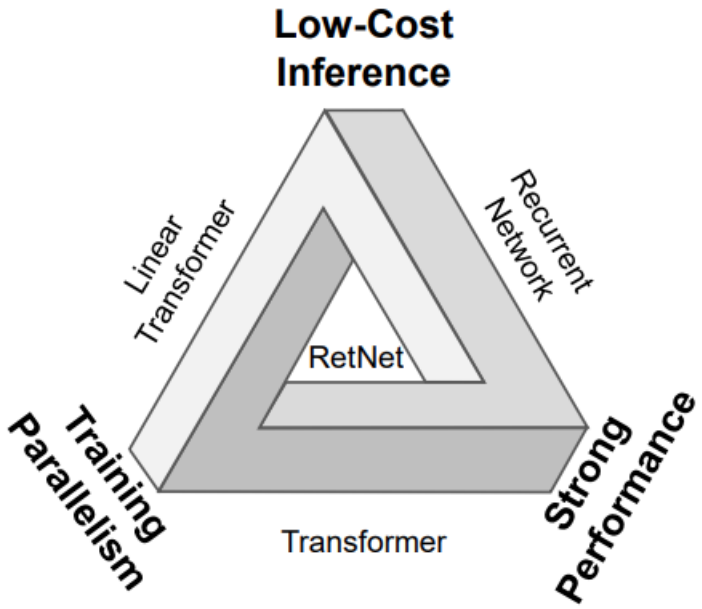


Figure 4: RetNet Architecture [8][10].

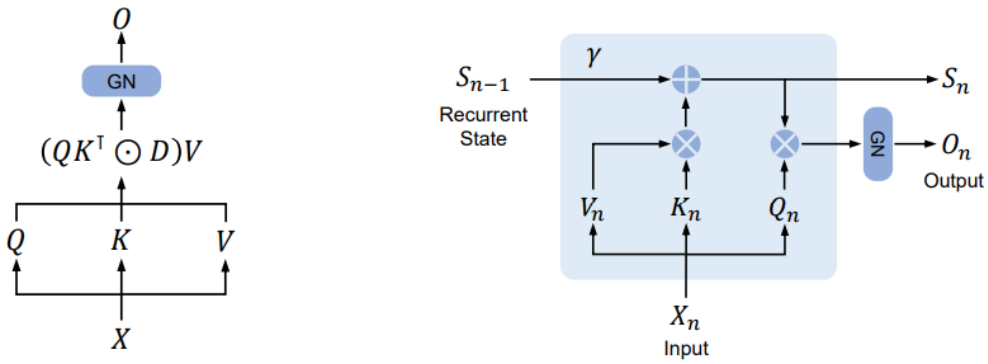
Architecture Breakdown

Layer Types and Arrangement

Like GPT models, RetNet has a decoder-only structure, but it substitutes a retention mechanism for the attention mechanism [8]. Several retention blocks, each comprising of feed-forward network (FFN) module and a multi-scale retention (MSR) module make up each RetNet block [8] [10].

Connectivity Patterns

RetNet uses residual connections to provide unhindered information flow between each feedforward and retention layer. RetNet can be calculated in three equivalent modes [9] [10]:



(a) Parallel representation. (b) Recurrent representation.

Figure 5: Parallel vs Recurrent representation of RetNet (GN stands for GroupNorm)[8][10]

1. **Parallel mode:** Effective for training, this mode computes all positions at once. Training parallelism can fully leverage GPU devices thanks to parallel representation [10]

$$\text{Retention}(X) = (QK^\intercal \odot D)V$$

where $\overline{\Theta}$ is the complex conjugate of Θ, and $D \in \mathbb{R}^{|x| \times |x|}$ combines causal masking and exponential decay along relative distance as one matrix. Similar to self-attention, the parallel representation enables us to train the models with GPUs efficiently.

2. **Recurrent mode:** sequential token processing that is effective for inference. Furthermore, the absence of key-value cache trickery significantly simplifies the implementation [10].

$$S_n = \gamma S_{n-1} + K_n^\intercal V_n$$
$$\text{Retention}(X_n) = Q_n S_n, \quad n = 1, \cdots, |x|$$

where Q, K, V, γ are the same as in Equation (5).

3. **Chunk wise recurrent mode:** This mode balances memory utilization and efficiency by processing token chunks in parallel with recurrence between chunks. While recurrently encoding the global blocks to conserve GPU RAM, we encode each local block in parallel for calculation performance [10].

$$\text{Retention}(X_{[i]}) = \underbrace{(Q_{[i]} K_{[i]}^\intercal \odot D) V_{[i]}}_{\text{Inner-Chunk}} + \underbrace{(Q_{[i]} R_{i-1}) \odot \xi}_{\text{Cross-Chunk}}, \quad \xi_{ij} = \gamma^{i+1}$$

where $[i]$ indicates the i-th chunk, i.e., $x_{[i]} = [x_{(i-1)B+1}, \cdots, x_{iB}]$.

Activation Functions

RetNet's feedforward layers mostly employ the Gaussian Error Linear Unit (GELU) activation function [8], defines as:

$$Y^l = \text{MSR}(\text{LN}(X^l)) + X^l$$
$$X^{l+1} = \text{FFN}(\text{LN}(Y^l)) + Y^l$$

where LN (·) is Layer Norm [BKH16]. The FFN part is computed as FFN(X) = Gelu (XW1) W2, where W1, W2 are parameter matrices.

Learning Mechanisms

A causal language modelling objective and the AdamW optimizer are used in conjunction with traditional backpropagation to train RetNet [8][9]. One of RetNet's unique training features is its ability to be deployed in recurrent mode yet trained in parallel mode. The model employs cosine learning rate schedule [9].

Specialized Components

By applying the retention mechanism across many heads with varying decay rates, **Multi-Scale Retention (MSR)** improves the model's ability to capture relationships across a range of temporal scales [8] [10]. An **exponential moving average (EMA)** of previous tokens, weighted by significance, serves as the retention mechanism. The retention technique is enhanced by RetNet's use of explicit positional encodings, or **XPos**, which allow for smooth transitions between parallel and recurrent modes [8][9].

Key Contributions and Innovations

- **Dual Computation Modes:** Combining the greatest features of RNNs and Transformers, RetNet's most notable innovation is its capacity to function in both parallel mode (for effective training) and recurrent mode [8][9].
- **Linear Scaling:** RetNet's memory and compute needs grow linearly with sequence length in both training and inference, in contrast to Transformers, which scale quadratically [8].
- **Retention Mechanism:** By drastically lowering computing costs and preserving competitive performance, the retention operation is an effective substitute for attention.
- **XPos Embeddings:** These specific positional embeddings provide for diverse deployment options by guaranteeing that the outcomes of the recurrent and parallel computations are equivalent [8][10].

Strengths and Limitations

Strengths:

- **Training Efficiency:** Model trains far more quickly than conventional RNNs since it can be trained in parallel, much like Transformers [8][9].
- **Inference Efficiency:** RetNet is very deployment-efficient in recurrent mode since it processes tokens in a sequential fashion with consistent memory usage, independent of sequence length [9].
- **Adaptable Deployment:** Depending on hardware limitations and sequence length requirements, deployment can be adjusted by switching between recurrent, parallel, and chunk wise recurrent modes [8][10].

Limitations:

- **Performance Gap:** On certain test tasks, RetNet performs marginally worse than Transformers of comparable size, despite being competitive [8][10].
- **Difficult Implementation:** Compared to ordinary Transformers, RetNet is more difficult to appropriately construct because to its numerous processing modes and particular embeddings [9].
- **Limited Pre-trained Models: Compared** to the vast Transformer model ecosystem, there are fewer pre-trained RetNet models accessible due to its more recent architecture [10].

Performance and Benchmarks

RetNet has shown encouraging results on several NLP benchmarks [8][9][10]:

- **Language Modelling:** On WikiText-103, RetNet-1B obtained a perplexity of 15.9, which is equivalent to a Transformer model of the same size but requires a lot less processing power.
- **Long-Context Understanding:** Compared to Transformers, which deteriorated more rapidly, Model demonstrated better performance retention as sequence length rose on long-sequence modelling benchmarks [8].
- **Inference Speed:** Model showed up to 11 times faster inference than Transformers in recurrent mode for lengthy sequences, with its advantage growing as the sequence length increased [8][9].

Architectures	Training Parallelization	Inference Cost	Long-Sequence Memory Complexity	Performance
Transformer	✔	$O(N)$	$O(N^2)$	✔✔
Linear Transformer	✔	$O(1)$	$O(N)$	✘
Recurrent NN	✘	$O(1)$	$O(N)$	✘
RWKV	✘	$O(1)$	$O(N)$	✔
H3/S4	✔	$O(1)$	$O(N \log N)$	✔
Hyena	✔	$O(N)$	$O(N \log N)$	✔
RetNet	✔	$O(1)$	$O(N)$	✔✔

Figure 6: Comparison of models from different angles. RetNet delivers good performance, linear long-sequence memory complexity, constant inference cost, and training parallelization [8].

RetNet fared better than Transformers on the PG-19 long-context language modelling benchmark at comparable parameter counts [8] [9].

Hyena

Overview

Hyena, which was first presented by Poli et al. in March 2023 [11], offers a novel method for sequence modelling that captures long-range relationships by utilizing long convolution filters with data-controlled gating. Hyena uses a fully convolutional technique that is intended for linear scaling with sequence length while utilizing simpler, more effective operations, in contrast to Transformer's self-attention or Mamba's state space modelling [11]

Hyena Architecture

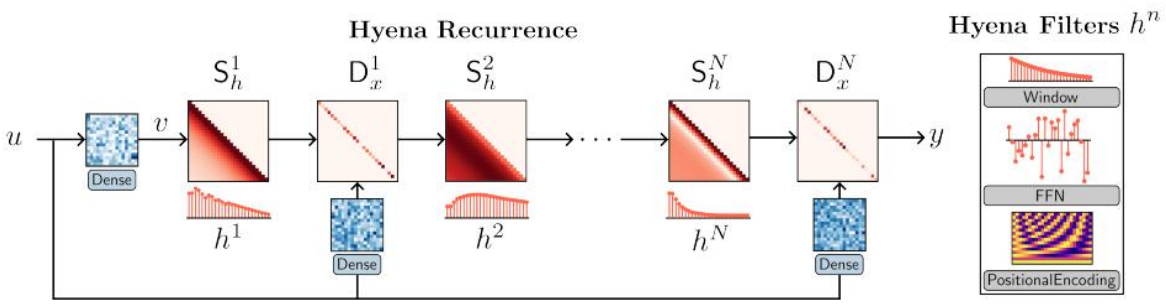


Figure 7: Hyena Architecture with Recurrence, filter and Matrices [12]

Architecture Breakdown

Layer Types and Arrangement

The design alternates between feedforward networks and Hyena operator blocks [11]. Every Hyena block includes:

- A Hyena operator with projections, data-controlled gating, and implicit long convolution filters.
- Normalization of the second layer and Neural network feedforward
- Hyena substitutes its specialized operator for attention processes, adhering to a decoder-only structure like to GPT models [11].

Connectivity Patterns

Hyena uses residual connections between layers, just like Transformers and other contemporary systems, to promote gradient flow. The fundamental change in Hyena is its implicit long convolution method, which uses filters with exponentially huge receptive fields and linear computational complexity to simulate very long-range dependencies [11] [12].

Definition 3.1 (Order–N Hyena Operator)**.** *Let $(v, x^1, \cdots, x^N)$ be projections of the input and let $h^1, \ldots, h^N$ be a set of learnable filters. The* Hyena$_N$ *operator is defined by the recurrence:*

$$\begin{aligned} z_t^1 &= v_t \\ z_t^{n+1} &= x_t^n (h^n * z^n)_t \quad n = 1, \ldots, N \\ y_t &= z_t^{N+1} \end{aligned} \tag{4}$$

Remark 3.1. *The time complexity of a* Hyena *recurrence is $\mathcal{O}(NL \log_2 L)$. The input-output map can be rewritten as*

$$y = x^N \cdot (h^N * (x^{N-1} \cdot (h^{N-1} * (\cdots))))$$

where each convolution is performed through the Fourier domain in $\mathcal{O}(L \log_2 L)$.

Figure 8: Hyena Operator representation [11]

Activation Functions

Hyena uses sine as the activation function [11].

Sinusoidal Activation[13] = $\sigma(\cdot) = \sin(\omega_a \cdot)$

Architectures Architectural hyperparameters for Hyena are shown in Table A.4. We use sine as an activation function for the FFN of Hyena filters.

Table A.4: Hyena architecture hyperparameters.

Size	depth	width	FFN width	filter FFN width	filter FFN depth	sine freq.
125M	12	768	3072	64	4	14
125M-slim	18	768	1536	64	4	14
153M	18	864	1728	64	4	14
355M	36	1024	2048	64	4	14
1.3B	36	2048	4096	64	4	14

Figure 9: Hyena Architecture using Sine as the activation function [11]

Learning Mechanisms

The model uses causal language modelling objectives on sequence problems and trains using standard back propagation with the AdamW optimization [11]. One of Hyena's unique features is its implicit parameterization of lengthy convolution filters, which enables effective learning of large receptive fields without the need for matching parameter changes [11][13].

Specialized Components

- By implicitly parameterizing filters with fewer parameters, Implicit Long Convolution allows for exponentially large fields of reception with linear computational complexity [11].
- Data-Controlled Gating determines gates from input data, enabling content-based dynamic information flow control [11].
- For very long sequences, recurrence and subsampling can be used to further increase efficiency [11][13].

Key Contributions and Innovations

- Very large receptive fields can be achieved without proportional parameter expansion thanks to implicit filter parameterization [11].
- O(n) complexity for long documents can be processed effectively with linear scaling [11][13].
- Operation Simplicity: Makes use of convolution operations that are well-optimized on contemporary hardware.
- Introducing increasingly potent variations (Hyena-S, Hyena, and Hyena-Hierarchy) with varying efficiency-expressivity trade-offs, the model hierarchy [11] [13].

Strengths and Limitations

Strengths:

- Efficiency of computation using convolution optimization and linear scaling [11]
- Despite using merely convolutions, long-range dependency modelling is effective.
- utilizing highly optimized convolution techniques for hardware-friendly implementation
- conceptual simplicity in comparison to attention mechanisms or state space models [13].

Limitations:

- Due to its recentness, there is little empirical validation across a variety of tasks.
- possibly less adaptable than attention in terms of identifying random patterns.
- fewer pre-trained models than methods based on Transformers [12].
- Implicit parameterization may result in preprocessing overhead [13].

Performance and Benchmarks

On several benchmarks, Hyena has shown encouraging performance [11][12][13]:

Language Modelling: Performs competitively on WikiText-103 with a far lower computational cost than similar models. At sequence length 8K, hyena operators are 100× faster than highly optimized attention, and at sequence length 64K, they are twice as fast [11].
Long-Range Arena: Handled long-range dependencies in a way that matched or surpassed attention-based approaches [12].
Scaling Properties: Good scaling with sequence length and model size [12][13].
Computational Efficiency: Superior perplexity over attention-based models for comparable computational resources [12].

Model	Acc @ 10	Acc @ 20	Acc @ 30	Acc @ 40	Loss @ 5B on The Pile
Conv1d	32	11	10	8	4.21
AFT-conv	55	21	12	10	3.57
H3	92	60	13	10	2.69
Transformer	100	100	92	82	2.59
Hyena	100	100	98	85	2.59

Figure 10: Hyena recall accuracy with vocabulary sizes of 10, 20, 30, and 40 in connection to test loss on The Pile following 5 billion tokens [11]

Comparative Summary

Computational Complexity:

- **Mamba:** O (L log (L)) for sequence length L [2]
- **RetNet:** O(N) linear complexity [8]
- **Hyena:** O (N log N) is the space complexity as stated in Remark 3.1 [11]

Architectures	Training Parallelization	Inference Cost	Long-Sequence Memory Complexity	Performance
Transformer	✔	$O(N)$	$O(N^2)$	✔✔
Linear Transformer	✔	$O(1)$	$O(N)$	✘
Recurrent NN	✘	$O(1)$	$O(N)$	✘
RWKV	✘	$O(1)$	$O(N)$	✔
H3/S4	✔	$O(1)$	$O(N \log N)$	✔
Hyena	✔	$O(N)$	$O(N \log N)$	✔
RetNet	✔	$O(1)$	$O(N)$	✔✔

Figure 11: Hyena Vs RetNet Vs Mamba Time and space complexity comparison [8]

For lengthy sequences, the authors showed that Hyena could match the most advanced Transformer language models while consuming significantly less compute [11]

Perplexity Comparison

- **Mamba** with S6 layer achieves best perplexity at 8.69
- **Hyena** architecture with S6 layer follows at 8.95
- Both outperform variants using S4 layers

Model	Arch.	SSM Layer	Perplexity
Hyena	H3	Hyena	10.24
H3	H3	S4 (complex)	10.30
-	H3	S4 (real)	10.34
-	H3	S6	**8.95**

Model	Arch.	SSM Layer	Perplexity
-	Mamba	Hyena	10.75
-	Mamba	S4 (complex)	10.54
-	Mamba	S4 (real)	10.56
Mamba	Mamba	S6	**8.69**

Figure 12: Hyena Vs Mamba Perplexity comparison [11]

Benchmark Scores

- **RetNet** shows strongest performance across multiple benchmarks:
  - Lowest perplexity on in-domain (26.05)
  - Best scores on GovReport (16.52) and SummScreen (24.46)
- **Hyena** performs well on PG22 (52.75) and QMSum (28.18)

Method	In-Domain	PG22	QMSum	GovReport	SummScreen
RWKV	30.92	51.41	28.17	19.80	25.78
H3	29.97	49.17	24.29	19.19	25.11
Hyena	32.08	52.75	28.18	20.55	26.51
Linear Transformer	40.24	63.86	28.45	25.33	32.02
RetNet	**26.05**	**45.27**	**21.33**	**16.52**	**22.48**

Figure 13: Hyena Vs RetNet Perplexity comparison [11]

By achieving linear scaling with sequence length, all three architectures overcome one of Transformer models' main drawbacks.

Conclusion

By tackling the efficiency constraints of attention mechanisms, Mamba, Hyena, and RetNet represent a substantial advancement in NLP design, surpassing Transformers. These models maintain competitive performance while achieving linear scaling with sequence length. Hyena's implicit convolutional filters effectively capture long-range dependencies with optimized hardware utilization, Mamba's selective state space technique allows for dynamic input focus, and RetNet's dual-mode computation offers remarkable deployment flexibility. Each has distinct operational advantages, demonstrating a move toward efficiency-driven, hardware-aware NLP design.

Instead of a single dominant paradigm, these developments point to a developing field with a variety of feasible architectural possibilities. Future studies will probably examine hybrid systems that capitalize on their complementing capabilities, performance on larger contexts and specialized domains, and scaling at greater parameter counts.

New Developments in Trends:

- Growing focus on hardware-aware design and computational efficiency
- Diversification of architecture utilizing essentially distinct mathematical techniques
- Increasing attention to trade-offs between training and inference for real-world implementation

Important Research Questions:

- When these systems are grown to hundreds of billions of parameters, how do they function?
- Could better models be produced by hybrid approaches that combine their complementary strengths?
- Beyond general language modelling, which architecture is most appropriate for certain domains?
- Which theoretical characteristics account for their individual advantages and disadvantages?

Reference

[1] A. Vaswani *et al.*, “Attention is All you Need,” in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2017. Accessed: Apr. 18, 2025. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[2] A. Gu and T. Dao, “Mamba: Linear-Time Sequence Modeling with Selective State Spaces,” May 31, 2024, *arXiv*: arXiv:2312.00752. doi: 10.48550/arXiv.2312.00752.

[3] “(PDF) Leveraging Joint Spectral and Spatial Learning with MAMBA for Multichannel Speech Enhancement,” ResearchGate. Accessed: Apr. 20, 2025. [Online]. Available: https://www.researchgate.net/publication/384074507_Leveraging_Joint_Spectral_and_Spatial_Learning_with_MAMBA_for_Multichannel_Speech_Enhancement

[4] D. Y. Fu, T. Dao, K. K. Saab, A. W. Thomas, A. Rudra, and C. Ré, “Hungry Hungry Hippos: Towards Language Modeling with State Space Models,” Apr. 29, 2023, *arXiv*: arXiv:2212.14052. doi: 10.48550/arXiv.2212.14052.

[5] “An Introduction to the Mamba LLM Architecture: A New Paradigm in Machine Learning.” Accessed: Apr. 20, 2025. [Online]. Available: https://www.datacamp.com/tutorial/introduction-to-the-mamba-llm-architecture

[6] A. C. IITM, “Zero to Mamba: An intuitive explanation to the Mamba Architecture,” Medium. Accessed: Apr. 20, 2025. [Online]. Available: https://medium.com/@aiclub.iitm/zero-to-mamba-an-intuitive-explanation-to-the-mamba-architecture-d52265b771ab

[7] “Mamba.” Accessed: Apr. 20, 2025. [Online]. Available: https://huggingface.co/docs/transformers/main/en/model_doc/mamba

[8] Y. Sun *et al.*, “Retentive Network: A Successor to Transformer for Large Language Models,” Aug. 09, 2023, *arXiv*: arXiv:2307.08621. doi: 10.48550/arXiv.2307.08621.

[9] Y. Sun *et al.*, “A Length-Extrapolatable Transformer,” Dec. 20, 2022, *arXiv*: arXiv:2212.10554. doi: 10.48550/arXiv.2212.10554.

[10] S. Chandra, “Retentive Networks (RetNet) Explained: The much-awaited Transformers-killer is here,” AI FUSION LABS. Accessed: Apr. 20, 2025. [Online]. Available: https://medium.com/ai-fusion-labs/retentive-networks-retnet-explained-the-much-awaited-transformers-killer-is-here-6c17e3e8add8

[11] M. Poli *et al.*, “Hyena Hierarchy: Towards Larger Convolutional Language Models,” Apr. 19, 2023, *arXiv*: arXiv:2302.10866. doi: 10.48550/arXiv.2302.10866.

[12] A. Lukyanenko, “Paper review: Hyena Hierarchy: Towards Larger Convolutional Language Models,” Medium. Accessed: Apr. 20, 2025. [Online]. Available: https://artgor.medium.com/paper-review-hyena-hierarchy-towards-larger-convolutional-language-models-e56b55232800

[13] M. Poli *et al.*, “Hyena Hierarchy: Towards Larger Convolutional Language Models,” in *Proceedings of the 40th International Conference on Machine Learning*, PMLR, Jul. 2023, pp. 28043–28078. Accessed: Apr. 20, 2025. [Online]. Available: https://proceedings.mlr.press/v202/poli23a.html