

Falcon-H1: A Family of Hybrid-Head Language Models Redefining Efficiency and Performance

Falcon LLM Team

-  <https://huggingface.co/tiiuae>
-  <https://github.com/tiiuae/falcon-h1>

Abstract: In this report, we introduce Falcon-H1, a new series of large language models (LLMs) featuring novel hybrid architecture designs that are optimized for both high performance and efficiency across a broad spectrum of use cases. Unlike previous Falcon models, which were built solely on either Transformer or Mamba architectures, the Falcon-H1 series is based on a parallel hybrid architecture that combines the strengths of the Transformer-based attention mechanism with State Space Models (SSMs), known for their superior long-context memory and computational efficiency. We also systematically revisited nearly every aspect of model design, data strategy, and training dynamics—challenging several conventional practices in the domain. To support a wide range of deployment scenarios, the Falcon-H1 series is released in a rich set of configurations, including both base and instruction-tuned models at 0.5B, 1.5B, 1.5B-deep, 3B, 7B, and 34B parameter scales. Quantized versions of the instruction-tuned models are also available. In total, over 30 model checkpoints can be accessed via Hugging Face Hub.

Our comprehensive evaluations demonstrate that Falcon-H1 models consistently set new performance benchmarks through exceptional parameter and training efficiency. The flagship Falcon-H1-34B-Instruct rivals or outperforms leading models up to the 70B scale, such as Qwen3-32B, Qwen2.5-72B and Llama3.3-70B, despite being approximately half the size and trained on a fraction of the data. This parameter efficiency is even more pronounced at smaller scales, where our 1.5B-Deep model achieves performance competitive with state-of-the-art 7B–10B models, Falcon-H1-0.5B delivers performance on par with typical 7B models from 2024. These models demonstrate leadership across a wide range of tasks, including reasoning, mathematics, multilingual, instruction-following, and scientific knowledge. Combined with support for extended context windows of up to 256K tokens and multilingual coverage across 18 languages, Falcon-H1 models are well-suited for a wide array of applications. All Falcon-H1 models are released under a permissive open-source license¹, reinforcing our commitment to accessible, high-impact AI research and development.

Contents

1	Introduction	3
2	Architecture	4
2.1	Channel Allocation	5
2.2	SSM-Specific Parameters Ablations	7
2.3	Challenging Conventional Components	11

¹<https://falconllm.tii.ae/falcon-terms-and-conditions.html>

2.3.1	RoPE Base Frequency	11
2.3.2	Width–Depth Trade-offs	11
2.4	Tokenizer	13
2.4.1	Empirical Studies	13
2.4.2	Final Tokenizer Design and Implementation	17
3	Pretraining	17
3.1	Pretraining Data	17
3.1.1	Data Sources	18
3.1.2	Data Strategy	20
3.2	Training Dynamics	23
3.2.1	Training Stability	23
3.2.2	Effective Learning Rate and Effective Weight Decay	24
3.2.3	Maximal Update Parametrization (μP) with Tunable Multipliers	28
3.2.4	Other Aspects: Batch Scaling, Rampup, Warmup	31
3.3	Pretraining Infrastructure	33
3.3.1	Scaling Dynamics of Data Parallelism	33
3.3.2	Mixer Parallelism (MP)	34
3.3.3	Context Parallelism (CP)	36
4	Post-training	37
4.1	Post-training Data	37
4.2	Supervised Fine-Tuning (SFT)	37
4.3	Direct Preference Optimization (DPO)	38
5	Evaluation	39
5.1	Base Models	39
5.2	Instruct Models	44
5.3	Model Efficiency	52
6	Model Integrations	53
7	Conclusion	53
8	Authors	54
9	Acknowledgments	54
A	Languages used for training Falcon-H1 tokenizers	65
B	Scalar stochastic dynamics with weight decay	65
C	Tuning μP multipliers	67
D	Detailed evaluation results	68
D.1	Multilingual Evaluations - Base Models	68
D.2	Multilingual Evaluations - Instruct Models	73
D.3	Long-context Evaluations - Instruct Models	77

E Training Data	80
E.1 Synthetic English Data Topics	80
E.2 Programming Languages	80
E.3 Code Quality Classifier	81

1. Introduction

The rapid progress in the development of large foundation models—particularly large language models (LLMs)—has been driven by advances in architectural design, scaling strategies, and training paradigms. Beginning with Transformer-based architectures (Vaswani et al., 2017) and expanding through massive-scale pretraining and techniques such as supervised fine-tuning (SFT) and reinforcement learning from human feedback (RLHF)(Ouyang et al., 2022).

A key limitation of the vanilla Transformer lies in its quadratic complexity with respect to input sequence length. To address this, recent research has explored more efficient alternatives to traditional attention mechanisms, such as Multi-head Latent Attention (MLA), featured in the DeepSeek series (Liu et al., 2024a,b). In parallel, several novel architectures have been proposed to move beyond Transformers entirely, including Griffin (De et al., 2024), RWKV (Peng et al., 2023), Titans (Behrouz et al., 2024), and Mamba (Gu & Dao, 2023), which offer comparable or superior performance in certain tasks with greater computational or memory efficiency. These advances have given rise to a new class of hybrid models that combine attention mechanisms with state-space models (SSMs), taking advantage of their complementary strengths: attention excels at modeling long-range dependencies, while SSMs provide efficient sequence mixing. This hybrid paradigm has gained popularity in models such as Jamba (Lieber et al., 2024; Team et al., 2024), Samba (Ren et al., 2024), Zamba (Glorioso et al., 2024), and Hymba (Dong et al., 2024).

Building on these insights, we introduce Falcon-H1—an innovative series of large language models that feature a novel parallel hybrid architecture integrating Transformer-style attention with Mamba-based state-space models (SSMs). Distinct from other leading open-weight LLM series—including LLaMA (Grattafiori et al., 2024), Mistral (Jiang et al., 2023), Qwen (Yang et al., 2024a,b), and DeepSeek (Liu et al., 2024a,b), Falcon-H1 is explicitly architected around this hybrid design, harnessing the complementary strengths of both mechanisms to deliver faster inference, lower memory usage, and state-of-the-art performance across a wide array of benchmarks. The series includes both pre-trained and instruction-tuned variants across seven scales: 0.5B, 1.5B, 1.5B-deep, 3B, 7B, and 34B parameters. In addition to bfloat16-precision models, we also provide quantized versions in multiple precisions to support efficient deployment across diverse hardware environments. Notably, our flagship model, Falcon-H1-34B-Instruct, achieves competitive or superior results compared to the strongest open-weight models to date, such as Qwen3-32B, Qwen2.5-72B-Instruct and LLaMA3.3-70B-Instruct—despite being approximately half the size.

Below, we show the key features of Falcon-H1:

- **Innovative Hybrid Architecture:** We combine attention and Mamba-2 heads in parallel within our hybrid mixer block. Importantly, the amount of attention and mamba heads can be adjusted independently, allowing for an optimal attention and SSM ratio. This hybrid design enables faster inference, lower memory usage, and strong generalization across tasks.
- **Wide Range of Model Sizes:** The Falcon-H1 family includes base, instruction-tuned and quantized variants in various sizes—0.5B, 1.5B, 1.5B-deep, 3B, 7B and 34B—designed to meet the needs of diverse usages and deployment scenarios, from edge devices to large-scale systems.

- **Multilingual by Design:** Supports 18 languages out of the box, including Arabic (ar), Czech (cs), German (de), English (en), Spanish (es), French (fr), Hindi (hi), Italian (it), Japanese (ja), Korean (ko), Dutch (nl), Polish (pl), Portuguese (pt), Romanian (ro), Russian (ru), Swedish (sv), Urdu (ur), and Chinese (zh) — with scalability to 100+ languages, thanks to our multilingual tokenizer trained on diverse language datasets.
- **Compact Models, Big Performance:** Falcon-H1-0.5B delivers performance on par with typical 7B models from 2024, while Falcon-H1-1.5B-Deep rivals many of the current leading 7B–10B models. Each Falcon-H1 model is designed to match or exceed the performance of models at least twice its size, making them ideal for low-resource and edge deployments without compromising on capability.
- **256K Context Support:** Falcon-H1 models support up to 256K context length, enabling applications in long-document processing, multi-turn dialogue, and long-range reasoning, with exceptional long context performance and greater computational and memory efficiency, Falcon-H1 provides a great balance between performance and resource cost.
- **Robust Data and Training Strategy:** Falcon-H1 employs a redesigned training approach that maximizes the value of high-quality but limited data. Additionally, the training process scales smoothly across model sizes through a customized *Maximal Update Parametrization* (μP) recipe, specifically adapted for this novel architecture.

2. Architecture

Hybrid models combining SSM and attention mechanisms have emerged recently as a promising direction. The classical approach to integrating these components was through **sequential** designs (Team et al., 2024; Ren et al., 2024; Glorioso et al., 2024), where one module feeds into the other in series across layers. More recently, **parallel** designs (Dong et al., 2024) have emerged as an alternative integration strategy, where both modules see the same input and their outputs are fused/concatenated before the block projection. For Falcon-H1, we adopt the parallel formulation as shown in Figure 1. Our parallel hybrid design has the freedom to choose the ratio of attention and SSM channels, and we are able to keep a small share of attention heads for precision while SSMs handle most of the work.

However, since Mamba architecture is relatively new, their architectural hyper-parameters remain under-explored compared to well-established transformer designs. This necessitates a systematic investigation of design choices to optimize performance. We therefore perform detailed ablations and coarse grid searches on 300M to 1.5B parameter proxy models to understand the impact of key architectural decisions. We sweep critical settings and record both training loss and throughput metrics. These experiments inform the final configuration summarized in Table 1 and provide insights for the broader SSM research community.

Model	Params (B)	Layers	# Vocab	d_{model}	Heads (Q/KV, SSM)	d_{head} (Attn/SSM)	d_{state}	Context Len.	# Tokens
Falcon-H1-0.5B	0.52	36	32,778	1024	8/2, 24	64/64	128	16K	2.5T
Falcon-H1-1.5B	1.55	24	65,536	2048	8/2, 48	128/64	256	128K	3T
Falcon-H1-1.5B-Deep	1.55	66	65,536	1280	6/2, 24	128/64	256	128K	3T
Falcon-H1-3B	3.15	32	65,536	2560	10/2, 32	128/128	256	128K	2.5T
Falcon-H1-7B	7.59	44	130,048	3072	12/2, 24	128/128	256	256K	~12T
Falcon-H1-34B	33.6	72	261,120	5120	20/4, 32	128/128	256	256K	~18T

Table 1: Model architecture details of the Falcon-H1 series. Embedding and projection layers are untied for all the models.

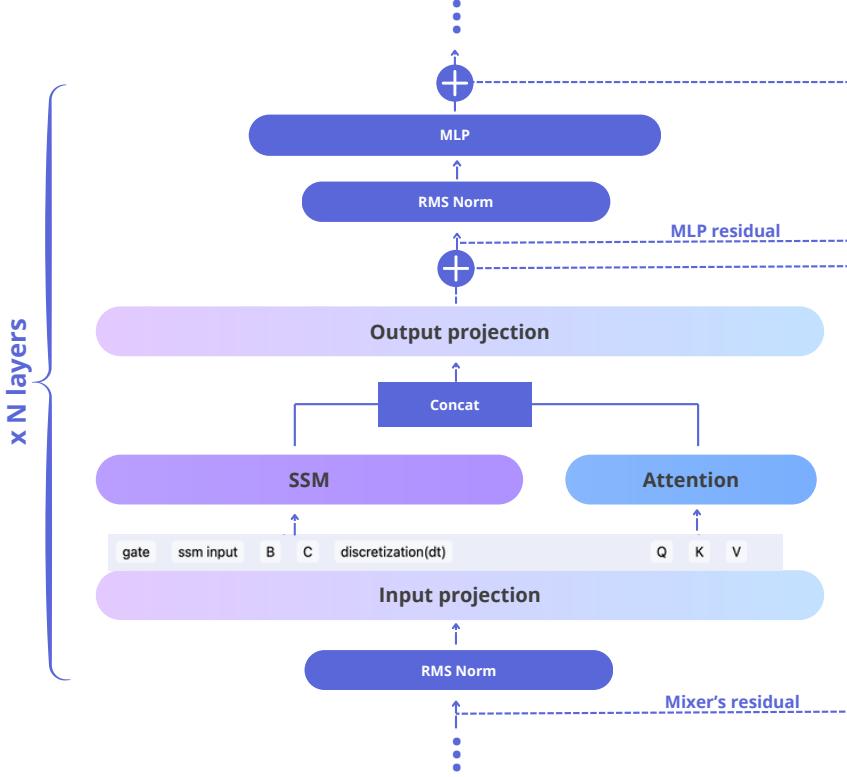


Figure 1: Falcon-H1 architecture. Attention and SSM run in parallel within each block; their outputs are concatenated before the block’s output projection. The number of SSM/Attention heads can be flexibly tuned.

2.1 Channel Allocation

A key aspect of our hybrid design is the concatenation of attention and SSM channels, which introduces an additional degree of freedom: the ability to independently vary the number of attention and SSM channels within each hybrid layer. This flexibility has not been explored in previous hybrid designs. For instance, (Dong et al., 2024) averages the outputs of attention and SSM channels, which requires both to have identical dimensions. To fully leverage this channel allocation flexibility, we conducted a systematic study of various allocation strategies—including different configurations of MLP channels and the positioning of the MLP block relative to the mixer.

Experimental settings. To provide a fair comparison across different channel allocation strategies, we adopted the following experimental design. Let d_{ssm} , d_{attn} , d_{MLP} denote the variable numbers of inner channels for the SSM, attention, and MLP blocks, respectively. The MLP channel dimension d_{MLP} can be varied freely without any constraints, as shown in Figure 1. For the attention and SSM blocks, we vary the number of query heads while keeping the head dimension, number of key-value (KV) heads, and number of attention groups fixed. Then, we divide the total available channels into 8 chunks, which can be freely allocated across the SSM, attention, and MLP modules. This results in the following parameterization:

$$d_{\text{ssm}} = \alpha_S \times 4096, \quad d_{\text{attn}} = \alpha_A \times 6144, \quad d_{\text{MLP}} = \alpha_M \times 4864, \quad (1)$$

where the chunk fractions $\alpha_S, \alpha_A, \alpha_M$ are chosen from

$$\alpha_S, \alpha_A, \alpha_M \in \left\{ \frac{1}{8}, \frac{2}{8}, \frac{3}{8}, \frac{4}{8}, \frac{5}{8}, \frac{6}{8} \right\}, \quad \alpha_S + \alpha_A + \alpha_M = 1. \quad (2)$$

In this experiment, the base amount of channels per chunk in (1) is set differently for the SSM, attention, and MLP blocks, with a ratio 4096 : 6144 : 4864 = 2 : 3 : 2.375. The reason behind this setting is to balance parameter count and efficiency of different allocations. Recall that the number of matrix layer parameters that scale with inner channels² are given by $3d_{ssm}d$, $2d_{attn}d$, $3d_{MLP}d$ for the respective blocks, where d is the hidden dimension of the model. Then, the ratio 2 : 3 : 2 of base channels would keep the number of parameters constant for different blocks. From this fixed parameter ratio, we slightly increased MLP base channels to take into account the lower computational cost of MLP compared to token mixing attention and SSM. This led to the adjusted ratio of 2 : 3 : 2.375, which we instantiate in practice as 4096 : 6144 : 4864 in this experiment.

In addition to channel allocations, another choice we have to make when assembling the hybrid model block is how to position SSM, attention, and MLP blocks with respect to each other. Similarly to parallel and sequential transformer design (Zhao et al., 2019; Chowdhery et al., 2022; He & Hofmann, 2024), we have the same choice of whether each pair of blocks should be processed in parallel or one after another. We have tested 3 configurations: fully parallel (**SAM**), semi-parallel (**SA_M**), and fully sequential (**S_A_M**), with the respective forward passes of a single model block l

$$\text{SAM: } \mathbf{r}_{l+1} = \mathbf{r}_l + \mathcal{F}_l^{\text{MLP}}(\mathcal{N}_l(\mathbf{r}_l)) + \mathcal{F}_l^{\text{attn}}(\mathcal{N}_l(\mathbf{r}_l)) + \mathcal{F}_l^{\text{SSM}}(\mathcal{N}_l(\mathbf{r}_l)) \quad (3)$$

$$\text{SA_M: } \mathbf{r}_{l+1} = \mathbf{r}'_l + \mathcal{F}_l^{\text{MLP}}(\mathcal{N}'_l(\mathbf{r}'_l)), \quad \mathbf{r}'_l = \mathbf{r}_l + \mathcal{F}_l^{\text{attn}}(\mathcal{N}_l(\mathbf{r}_l)) + \mathcal{F}_l^{\text{SSM}}(\mathcal{N}_l(\mathbf{r}_l)) \quad (4)$$

$$\text{S_A_M: } \mathbf{r}_{l+1} = \mathbf{r}''_l + \mathcal{F}_l^{\text{MLP}}(\mathcal{N}''_l(\mathbf{r}''_l)), \quad \mathbf{r}''_l = \mathbf{r}'_l + \mathcal{F}_l^{\text{attn}}(\mathcal{N}'_l(\mathbf{r}'_l)), \quad \mathbf{r}'_l = \mathbf{r}_l + \mathcal{F}_l^{\text{SSM}}(\mathcal{N}_l(\mathbf{r}_l)) \quad (5)$$

Here \mathbf{r}_l is the residual at the beginning of l 'th model block, $\mathbf{r}'_l, \mathbf{r}''_l$ are intermediate residuals in the middle of sequential model blocks; $\mathcal{F}_l^{\text{SSM}}, \mathcal{F}_l^{\text{attn}}, \mathcal{F}_l^{\text{MLP}}$ denote SSM, attention and MLP forward passes; and $\mathcal{N}_l, \mathcal{N}'_l, \mathcal{N}''_l$ are RMSnorms applied at the beginning of each block and shared for the blocks arranged in parallel to each other. Switching between these 3 block configurations almost does not change the parameter count in the model because SSM/attention/MLP blocks are rearranged as a whole without modifying the insides of each block. The only extra parameters come from the necessity to add new RMSnorm layers $\mathcal{N}'_l, \mathcal{N}''_l$ for each additional sequential computation within the model block.

We have compared all the configurations described above for a relatively deep model with $L = 60$ layers, hidden dimension $d = 1280$, resulting in approximately 1.2B parameters. All other training and architecture hyperparameters were identical, and we measured the loss after 70GT of training.

The results. Our channel allocation experiments have two parts, focusing on channel allocations and then on block arrangement.

First, for fully parallel **SAM** blocks arrangement we examined all 21 admissible $(\alpha_S, \alpha_A, \alpha_M)$ partitions, according to (2). The resulting loss values are plotted on figure 2 (left). We see a clear separation of magnitude between the impact of the number of attention channels and SSM \leftrightarrow MLP channel switching. Having more attention channels significantly degrades the performance, while SSM \leftrightarrow MLP channel switching has a noticeable but much weaker effect. We have also confirmed similar behavior for the **SA_M** block arrangement on a smaller number of runs.

²For attention and MLP all parameters that scale with inner channels are located in the matrix layers. However, Mamba2 SSM block contains a few vector-like parameters that scale with d_{ssm} , for example, the additional RMSnorm applied to inner channels after SSM computation. As the number of such extra parameters is tiny compared to the matrix layer ones, we neglect them when balancing parameter counts for different channel allocations.

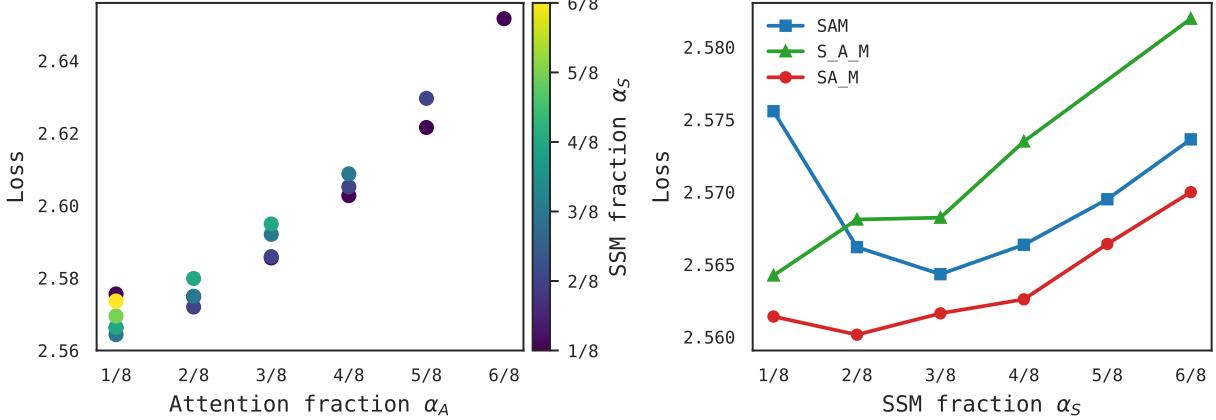


Figure 2: **(Left)**: The loss of fully parallel SAM hybrid block configuration for all possible $(\alpha_S, \alpha_A, \alpha_M)$ channel allocations according to (1),(2). **(Right)** The loss of all 3 considered block configurations SAM (3), SA_M (4), and S_A_M(5) for fixed optimal attention allocation $\alpha_A = \frac{1}{8}$ and varied SSM/MLP channel allocation.

For the second part of the experiment, we compare all three SAM, SA_M, S_A_M block arrangements while fixing attention channels to the minimum $\alpha_A = \frac{1}{8}$ and varying SSM/MLP channels so that $\alpha_S + \alpha_M = \frac{7}{8}$. The resulting loss values are plotted on figure 2 (right). We see that semi-parallel SA_M configuration provides the best results, $(\alpha_S, \alpha_A, \alpha_M) = (\frac{2}{8}, \frac{1}{8}, \frac{5}{8})$ being the optimal channel allocation. However, the dependence on SSM/MLP allocations is flat near optimum. Interestingly, as block configuration becomes more sequential SAM \rightarrow SA_M \rightarrow S_A_M, the optimal SSM fraction reduces as $\frac{3}{8} \rightarrow \frac{2}{8} \rightarrow \frac{1}{8}$. At the moment, we don't have an explanation of this behavior.

Based on the experiment results above, for Falcon-H1 models, we adopted SA_M block configuration with channel allocations roughly following 2 : 1 : 5 ratio, with slight deviation for different model sizes, which is possible thanks to flat dependence on SSM/MLP allocations near optimum.

2.2 SSM-Specific Parameters Ablations

We start by revisiting the Mamba2 design (Dao & Gu, 2024) of SSM block we employ for Falcon-H1 models. At the core of the block is a token mixing mechanism that maps an input sequence $x \in \mathbb{R}^L$ of length L to $y = \text{SSM}(x, \mathbf{B}, \mathbf{C}, dt | A_{\log}, D) \in \mathbb{R}^L$ via a recurrent mechanism involving hidden state $\mathbf{h} \in \mathbb{R}^{d_{\text{state}} \times L}$

$$\mathbf{h}_{t+1} = \bar{\mathbf{A}}_t \mathbf{h}_t + \mathbf{B}_t dt_t x_t, \quad y_t = \mathbf{C}_t^\top \mathbf{h}_t + D x_t. \quad (6)$$

Here \mathbf{B}_t determines the vector $\mathbf{B}_t dt_t x_t$ to be written into the SSM hidden state, $\bar{\mathbf{A}}_t \in \mathbb{R}$ determines the fraction of previous hidden state to be forgotten, \mathbf{C}_t is a reading projector, and the scalar $D \in \mathbb{R}$ controls the direct connection between input and output sequences, by passing the hidden state recurrent computation. The “time step” $dt_t \in \mathbb{R}$ controls both the writing intensity via $\mathbf{B}_t dt_t x_t$, and forgetting intensity via the parametrization of $\bar{\mathbf{A}}_t = \exp[-e^{A_{\log}} dt_t]$.

It is often convenient to view SSM operation (6) as implementing an linear sequence transformation with a causal “attention” matrix M_{ts}

$$y_t = \sum_{s \leq t} M_{ts} x_s, \quad M_{ts} = \mathbf{C}_t^\top \mathbf{B}_s dt_s \prod_{i=s+1}^t \bar{\mathbf{A}}_i + D \delta_{ts}, \quad \prod_{i=t+1}^t := 1. \quad (7)$$

The recurrence (6) describes the sequence transformation of a single SSM channel. Mamba2 organizes all the $d_{\text{ssm}} = d_{\text{head}}n_h$ channels first into n_h heads of dimension d_{head} , and then further unites the heads into n_g groups with some of the parameters shared within each group. Such organization into heads and groups is similar to grouped query attention (GQA) (Ainslie et al., 2023). Specifically, all the parameters $\mathbf{B}_t, \mathbf{C}_t, dt_t, \bar{A}_t, D$ are broadcasted along the head dimension, and $\mathbf{B}_t, \mathbf{C}_t$ are further shared within each group.

The key distinction of Mamba2 block compared to early SSM designs is that parameters defining recursion (6) are input dependent, allowing for much stronger expressivity. First, the input to the block $\mathbf{u} \in \mathbb{R}^{d \times L}$ goes through a linear transformation, then selectively through a causal depthwise 1D convolution $\text{conv1d}(\cdot)$, and finally through the SiLU activation function. Denoting concatenation with merged letters, we write the full input transformation as

$$\tilde{\mathbf{x}}\tilde{\mathbf{z}}\tilde{\mathbf{B}}\tilde{\mathbf{C}}\tilde{\mathbf{d}\mathbf{t}} = W_{xzBCdt}\mathbf{u}, \quad W_{xzBCdt} \in \mathbb{R}^{(2d_{\text{ssm}}+2n_gd_{\text{state}}+n_h) \times d}, \quad (8)$$

$$\mathbf{x}\mathbf{B}\mathbf{C} = \text{SiLU}(\text{conv1d}(\tilde{\mathbf{x}}\tilde{\mathbf{B}}\tilde{\mathbf{C}})), \quad \mathbf{d}\mathbf{t} = \text{Softplus}(\tilde{\mathbf{d}\mathbf{t}} + \mathbf{b}), \quad \mathbf{z} = \text{SiLU}(\tilde{\mathbf{z}}). \quad (9)$$

Here, $\mathbf{b} \in \mathbb{R}^{n_h}$ is a head-wise bias, and \mathbf{z} is the gate multiplied element-wise with SSM output $\mathbf{y}_g = \mathbf{y} \odot \mathbf{z}$, similarly to the gated MLP. Finally, \mathbf{y}_g goes into grouped RMSnorm³, followed by the output projection layers $W_o \in \mathbb{R}^{d \times d_{\text{ssm}}}$ to produce the output of Mamba2 block. We note that parameters A_{\log}, D, \mathbf{b} are, however, static learnable weights that are not input dependent.

In the remaining paragraph, we describe ablations for various dimensions of the Mamba2 block described above: head dimension $d_{\text{head}} = d_{\text{ssm}}/n_h$, the number of groups n_g , the recurrent state dimension d_{state} , the depthwise 1-D convolution kernel size, and the scan chunk size used by SSD algorithm (Dao & Gu, 2024) implementing the recursion (6).

State dimension vs. group count. Prior work shows that enlarging the SSM state size d_{state} consistently boosts accuracy, but at a non-trivial efficiency cost (Gu & Dao, 2023; Liu et al., 2024d; Stan & Rhodes, 2024; Mitra et al., 2025). Systematic studies of the accuracy–efficiency frontier remain sparse.

We therefore ran a two-dimensional grid search over the state dimension d_{state} and the number of groups n_g . To disentangle their effects from model size, we can fix the total number of parameters by fixing budget $B = d_{\text{state}} \times n_g$ while varying (d_{state}, n_g) . We swept five budgets $B \in \{4, 16, 64, 256, 1024\}$.

As shown in Fig. 3, validation accuracy rises almost exclusively with larger d_{state} ; varying n_g has only a marginal impact. Thus, the best configuration within any budget uses the smallest feasible n_g and the largest possible d_{state} . Conversely, training throughput deteriorates with increasing d_{state} , with a pronounced efficiency apex around $d_{\text{state}} = 16$.

Note that all experiments were conducted at a sequence length of 2048. Because longer sequences require a larger state to retain historical information, we take $(n_g, d_{\text{state}}) = (1, 256)$ for the final models as the best compromise. Since FALCON-H1-34B was trained with tensor parallelism (TP) = 4 and mixer parallelism (MP; see §3.3.2), we fixed the number of groups to $n_g = 2$ so that it is divisible by TP/2.

Head dimension d_{head} . To isolate the effect of the SSM head size d_{head} , we trained variants with $d_{\text{head}} \in \{16, 64, 256\}$ while keeping $d_{\text{ssm}} = d_{\text{head}}n_h$ so the parameter count stays roughly constant. We observe a $\leq 10^{-2}$ change in training cross-entropy, with a clear gain at larger heads (see Fig. 4a).

³grouped RMS normalization layer is required here to enable the usage of tensor parallelism (TP), which splits SSM channels across different devices.

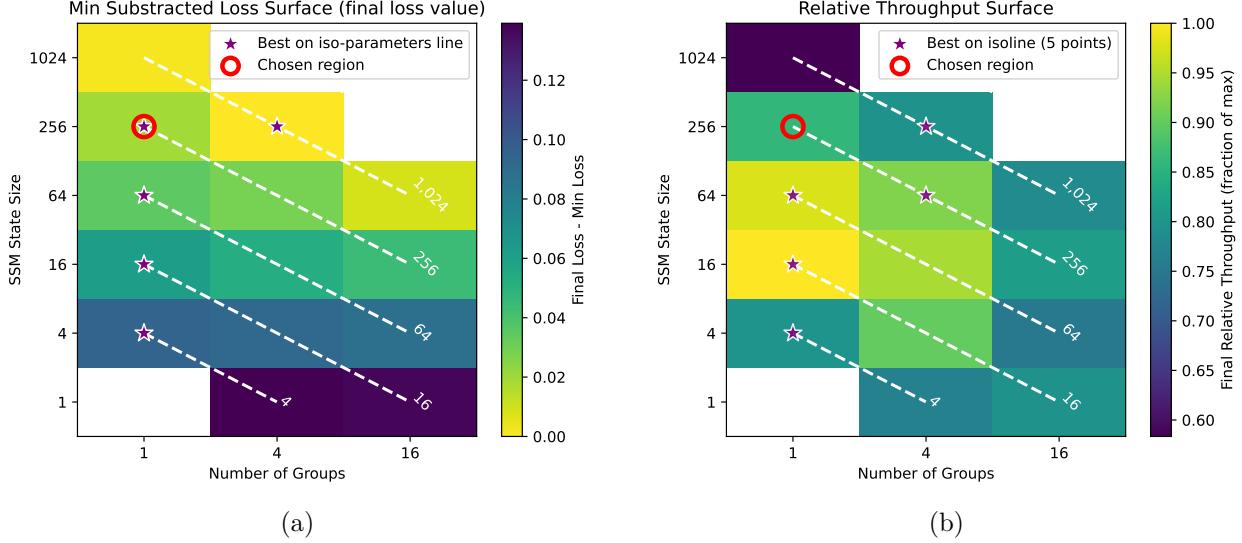


Figure 3: Hyperparameter optimization landscapes for SSM number of groups and state dimension size. **(a)** Loss surface showing performance relative to global minimum across number of groups and d_state size. **(b)** Relative throughput surface as fraction of maximum performance. Dashed lines indicate iso-parameter curves ($ng \times ds = \text{constant}$), implying constant total parameter count. Red stars mark optimal configurations for each computational budget, revealing distinct trade-offs between model quality and efficiency.

Throughput is more sensitive: $d_{\text{head}} < 32$ reduced GPU utilization, whereas $d_{\text{head}} \geq 64$ maintained optimal efficiency. To our knowledge, no prior work reports an explicit ablation of d_{head} for SSMs; existing sources simply pick values (e.g., 64 or 128) and discuss kernel limits (Dao, 2024a,b).

Larger head dimensions therefore yield a more favorable accuracy and efficiency.

Depthwise causal 1-D convolution (convdim). To our knowledge, no prior work reports an ablation over the depthwise causal Conv1d kernel size inside Mamba-style SSM blocks; existing papers simply fix $k = 4$ (Pei & others, 2025; Chao et al., 2024). The reference `causal_conv1d` CUDA kernel itself only supports {2, 3, 4} (Dao-AILab, 2023; Hoang & Mamba contributors, 2024). We therefore re-implemented the kernel to handle sizes up to 32 and swept {2, 4, 8, 16, 32}. Kernel size 4 indeed minimized validation loss, whereas both smaller and larger filters degraded accuracy, so we keep `kernel_size = 4` in the final models (see Fig. 4b).

Chunk size (cs). The SSD kernel processes a long sequence in blocks of cs tokens. The efficiency scales favorably with cs until two limits emerge:

1. *Launch overhead.* Very small chunks ($cs < 64$) trigger many kernel launches and under-utilise the GPU.
2. *Memory pressure.* When $cs > 256$ the cross-chunk prefix-sum kernel (`dA_cumsum`) no longer fits in on-chip SRAM and becomes memory-bound, lowering the efficiency.

A broad plateau therefore appears at $cs \in \{128, 256\}$. This matches the implementation guidance in the Mamba official codebase where the Triton kernels are tuned for power-of-two values and default to $cs = 256$. We fix $cs = 256$ for all subsequent experiments to maximize throughput while retaining numerical stability.

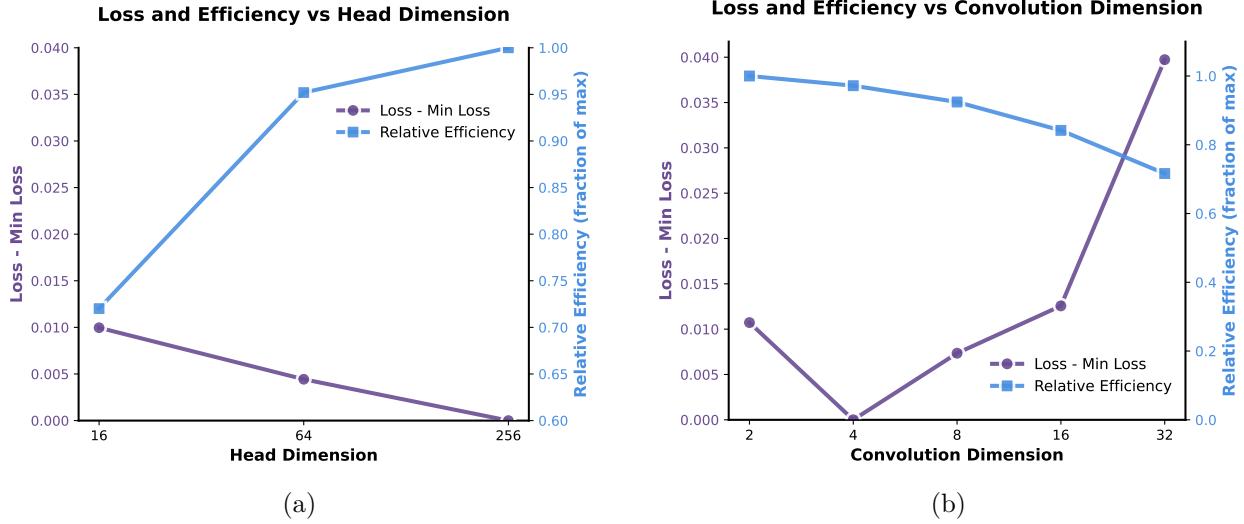


Figure 4: Model accuracy and computational efficiency across architectural dimensions. (a) Similar analysis for attention head dimensions, showing that larger head dimensions provide both better computational efficiency and lower loss. (b) Loss (purple line, left y-axis) and relative computational efficiency (blue line, right y-axis) as functions of convolution dimension. The analysis reveals an optimal trade-off at dimension 4, where the model achieves minimal loss while maintaining high efficiency.

Hidden State Resetting in Mamba. When several documents are concatenated inside one long sequence, the tail of document k ($0 \leq s < T_k$) leaks into the head of document $k+1$ through the product of \bar{A} in (7). This *cross-doc leakage* violates the independence assumption of language-model training and especially causes semantic contamination between unrelated contexts. For attention, a simple fix is a block-diagonal (segmented) mask that zeroes attention scores across document boundaries—often called *cross-document masking* (Grattafiori et al., 2024; Team et al., 2025).

For recurrent architectures, such cross-document bleeding can be avoided by resetting the hidden state at document boundaries. Let $r_t = 1$ signify that token t is the first token of a new document in the sequence. This binary indicator is derived from the data loader’s position tensor and marks document boundaries within a packed sequence.

To reset hidden-state on document boundary we need to force $\bar{A}_t = 0$ when t is at the document edge, we do so by injecting a large negative value -80 *channel-wise* into the SSM parameter vector \bar{A} before exponentiation at positions where $r_t = 1$:

$$\bar{A}_i = \exp[-e^{A_{\log}} \tilde{d}t_i + r_i \cdot (-80)] \approx \begin{cases} \mathbf{0}, & r_i = 1, \\ \bar{A}, & r_i = 0. \end{cases}$$

Thus, at a boundary, $\mathbf{h}_{t+1} = \mathbf{0} \cdot \mathbf{h}_t + \bar{B} \mathbf{x}_t = \bar{B} \mathbf{x}_t$, perfectly *resetting* the hidden state in a single step with no additional compute. Subsequent tokens use the standard transition weights \bar{A} . This resetting scheme does not add any compute or memory overhead while remaining gradient-safe because the -80 bias is constant, so gradients propagate normally through \exp and \tilde{A}_t . Moreover, it is numerically stable: $\exp(-80) \approx 10^{-35}$ lies above the FP16/BF16 underflow threshold ($\sim 10^{-45}$) yet empirically zeros the hidden state without training instabilities.

2.3 Challenging Conventional Components

2.3.1 RoPE Base Frequency

For Falcon-H1 models, we have used an unconventionally high value $b = 10^{11}$ of the RoPE base frequency. Below, we present our reasoning process to arrive at this value.

We have started the training of 7B and 34B models with a standard value $b = 10^4$ and sequence length $L_{\text{seq}} = 8192$. In the middle of the training, we increased sequence length to $L_{\text{seq}} = 16384$ but observed a drop in the model evaluations and revoked the change. This drop was unexpected since we did not change the data mixture, and the only effect of L_{seq} increase amounts to fewer training samples being cut. Suspecting RoPE base frequency as the main parameter that interacts with the training sequence length, we have increased the base frequency to $b = 10^6$. This change resulted in an immediate boost in model evaluations, and another boost after increasing sequence length again to $L_{\text{seq}} = 16384$.

To further investigate the impact of base frequency b , we ran a b sweep on Falcon-H1 0.5B model, depicted in Figure 5a. At smaller b , a few orders of magnitude around the model sequence length, the training loss $L(b)$ steeply depends on b , with lower values being extremely suboptimal. Earlier drop of evaluations on 7B/34B models when increasing L_{seq} can be interpreted as moving up the $L(b)$ curve due to the decrease of b relatively to L_{seq} . At larger b the curve $L(b)$ flattens and slowly increases, reflecting the NoPE (no positional embeddings) limit $b \rightarrow \infty$. Our chosen value $b = 10^{11}$ roughly corresponds to the optimum of $L(b)$ curve. We stress, though, that optimal b does not need to be estimated very accurately due to the flatness of $L(b)$ for large b . Applying $b = 10^{11}$ to 7B/34B models resulted in another increase of evaluation scores.

Finally, let us point out the advantage of using a large base frequency b when scaling the sequence either during continual pretraining or inference. RoPE assigns different frequencies $\theta_k = b^{-2k/d_{\text{head}}}$ to different dimensions k within query and key vectors. When the value of b is comparable to the original sequence length L_{seq} , further increase of L_{seq} requires reassigning frequencies within QK dimensions to allocate some space for smaller θ (or larger wavelengths θ^{-1}) needed to handle longer sequences. Different strategies of this reassigning, such as Position Interpolation (Chen et al., 2023), “NTK-aware” (bloc97, 2023a) or “NTK-by-parts” (bloc97, 2023b), improve performance on larger sequences but still deform the original assignment θ_k the model has adapted to during training.

Fortunately, using extremely large b during training leaves many dimensions k effectively unassigned, since the respective large sequences were never seen during training. In that case, no RoPE modifications are required when increasing sequence length beyond the training value, making sequence length extension for Falcon-H1 models extremely simple.

An interesting question is whether such large b values are optimal only for hybrid models, where SSM part can take care of short-range dependencies, or can also work for transformer models.

2.3.2 Width–Depth Trade-offs

Increasing the *depth* (number of layers) versus the *width* (hidden dimensionality) of a decoder-only model presents distinct trade-offs in expressivity and efficiency. Depth provides more sequential composition of nonlinear transformations, enabling hierarchical feature learning and multi-step reasoning that a shallow-but-wide model might require exponentially more neurons to emulate (Chen et al., 2024). In contrast, width expands the model’s capacity per layer, allowing it to encode more features in parallel. Depth thus increases representational power by adding layers of compositionality, while additional width increases the representational richness at each layer without increasing the number of sequential transformations.

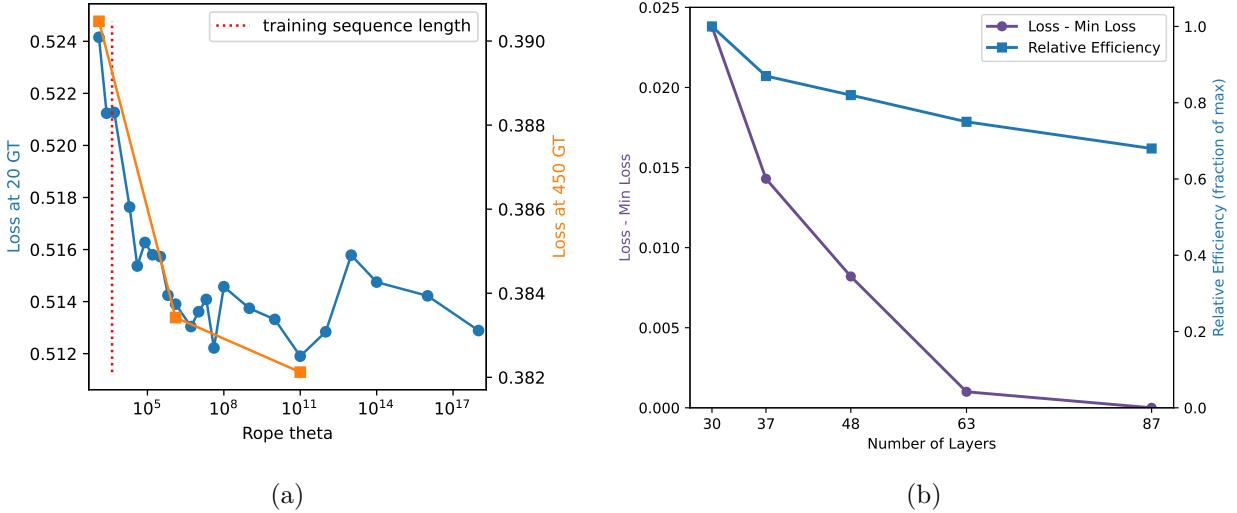


Figure 5: (a) Dependence of the training loss on RoPE base frequency b . The dotted line shows the training sequence length for a reference. First, we tried many base frequencies and measured the loss early in the training at 20GT. Then, we picked 3 characteristic base frequency values and measured the loss in a much later training stage at 450GT, with the results being roughly similar to the early measurement. (b) Dependence of the training loss on the number of layers for a fixed number of parameters.

From an optimization and efficiency perspective, these choices have significant implications. Deep stacks are more sequential, which hampers parallelism and can slow down training and inference: each added layer introduces an additional serial step that cannot be parallelized in time, leading to higher latency. Wider layers, on the other hand, perform more computation in parallel within a single layer (e.g., larger matrix multiplications), which modern hardware can exploit efficiently. Memory constraints also differ: very deep models must store activations for many layers during backpropagation (increasing memory usage proportional to depth). By contrast, widening a layer adds no additional sequential operations: each layer can encode more information and shortens the longest gradient path, alleviating bottlenecks. The trade-off is a higher peak-memory footprint. From a training point of view, very deep networks demand careful architectural tweaks to remain trainable, whereas very wide networks may hit other limits such as memory bandwidth or diminishing returns in utilization of parameters.

In practice, state-of-the-art LLM architectures balance width and depth to leverage the benefits of both. Modern decoder-only LLMs are typically built with dozens to ~ 100 layers, each with a very large hidden size, rather than choosing an extreme in one dimension. For example, LLaMA-3 (Grattafiori et al., 2024) scales from 8B to 70B parameters by increasing both depth and width layers with a 12 288-dimensional hidden size. This co-scaling strategy ensures the model has sufficient depth to compose complex behaviors and sufficient width to maintain high capacity and parallel throughput. In fact, many design heuristics keep the network’s depth and width growing in tandem with total size.

For the FALCON-H1 series, we revisited the width–depth trade-off under a fixed 1.5B parameter budget. We performed a joint sweep over hidden width d_{model} and depth L , scaling the learning rate inversely with width following a simple μP scaling (Yang et al., 2022) ($\eta \propto 1/d_{\text{model}}$) to stabilize training dynamics across configurations and omitted depth-scaling at this stage for simplicity.

We evaluated five architecture shapes: W1536L87, W1792L63, W2048L48, W2304L37, and W2560L30. Efficiency was measured in training giga-tokens per hour (gtok/h), and quality was assessed via pre-training cross-entropy.

As shown in Figure 5b, greater depth yielded consistently higher overall quality. The 87-layer extreme W1536L87 variant clearly outperformed the wider W2560L30 one. From our empirical studies, it even matched/outperformed 3.0B and 7.0B reference models twice to 5 times its size. This accuracy boost came at a cost: training throughput dropped by 25–30 % gtok/h and inference slowed by a comparable margin relative to the shallowest configuration. Because the depth-width balance remains under-explored, we are releasing two 1.5B variants—FALCON-H1-1.5B (width-balanced with 24 layers) and FALCON-H1-1.5B-DEEP (deeper with 66 layers) to foster further investigation of this trade-off.

2.4 Tokenizer

Considering the memory footprint and model performance across different model scales (Tao et al., 2024), we decided to train multiple tokenizers with different vocabulary size. Basically, we increase gradually the vocabulary size regarding model size as shown in Table 1. In this section, we show detailed studies and experimental results when building Falcon-H1 tokenizers.

2.4.1 Empirical Studies

In the literature, a tokenizer with a high compression rate (i.e., low fertility score) across languages—including non-Latin scripts—is widely regarded as essential. By encoding equivalent text with fewer tokens, such tokenizers improve both training and inference efficiency, requiring fewer iterations to generate the same amount of text. However, compression alone does not fully capture the factors influencing end-to-end LLM performance.

To address this, we conducted a series of experiments to investigate how various tokenizer training strategies affect both proxy metrics and downstream outcomes. This was primarily evaluated using the fertility score (compression rate) and the average number of bytes per vocabulary token (expressiveness). Beyond these proxy metrics, we also performed full-scale training experiments to directly observe the impact of design choices that might not be evident through compression or expressiveness alone. Specifically, we explored the impact of scaling tokenizer training data, regex splitting patterns, handling of punctuation and digits, and inclusion of L^AT_EX tokens.

Will scaling training data impact tokenizer performance? To investigate how scaling training data affects tokenizer performance, we conducted an experiments with six tokenizers trained on English text. We varied two factors: the training corpus sizes (1GB, 14GB, and 40GB) and the vocabulary size (65k and 135k). All other training parameters were held constant to isolate the impact of these variables. The results are summarized in Table 2.

The relationship between corpus size and performance is non-monotonic and is conditioned on the vocabulary size. For a 65k vocabulary, optimal performance is achieved with smaller corpora: the 1GB corpus yields the best compression, while the 14GB corpus maximizes bytes per token. Performance degrades at 40GB. Conversely, for a 135k vocabulary, the 14GB corpus surpasses both smaller and larger corpora on both metrics.

Conclusion: Simply increasing the training data volume does not guarantee a better tokenizer. Instead, there is an optimal range of data that depends on the vocabulary size.

	Vocab Size	Corpus Size	Fertility Score ↓	Bytes per Token ↑
65k		1GB	1.4350	8.2435
		14GB	1.4443	8.7274
		40GB	1.4907	8.3748
135k		1GB	1.3887	8.8912
		14GB	1.3344	9.2688
		40GB	1.3964	9.2577

Table 2: Tokenizer performance by corpus and vocabulary size. Best results for each vocabulary size are in **bold**. The arrows indicate the desired direction for each metric.

How important is the Splitting RegEx? The Splitting RegEx is a pattern rule used to break raw text into preliminary word-like units before applying the actual tokenization algorithm. In other words, it defines how the tokenizer initially segments text, influencing how efficiently it can compress data into tokens. For this analysis, we compared three publicly available splitting regex patterns used by different tokenizers: GPT-4o, LLaMA-3, and GPT-2. We trained all tokenizers with a fixed vocabulary size of 131k on the same dataset, and then measured both the fertility score and the average number of bytes represented in the vocabulary.

Splitting RegEx	Fertility Score ↓	Bytes per Token ↑
GPT-2	1.3466	9.1019
GPT-4o	1.3209	8.7924
LLaMA-3	1.3238	8.7003

Table 3: Impact of different splitting regex patterns on fertility score and average bytes per token. Best scores for each metric are in **bold**; arrows indicate the desired optimization direction.

From the results shown in Table 3, we observe that while the choice of splitting regex does have a measurable impact on both the fertility score and the average bytes per token, the differences between recent regex patterns such as GPT-4o and LLaMA-3 remain relatively small.

Conclusion: The differences between splitting regex patterns used by modern tokenizers are minor. For practical purposes, it is advisable to adopt a splitting regex from a well-established and up-to-date tokenizer.

Whether to apply Punctuation and Digits Splitting? Whether to apply punctuation and digit splitting remains an active topic of discussion in the community (Singh & Strouse, 2024), with notable impacts on domains such as mathematics and code. While metrics like fertility score and bytes-per-token are often used to evaluate tokenizers, we find they do not consistently predict downstream model performance. For example, as shown in Table 4, we trained two tokenizers on identical data—one applying individual digit splitting and one without. Although the latter achieves a better (lower) fertility score, most recent studies (Yang et al., 2024b; Grattafiori et al., 2024) adopt digit splitting as a standard practice, underscoring that such metrics alone may not fully capture tokenizer effectiveness.

To resolve this ambiguity, we conducted a controlled empirical study focused on downstream task performance. We trained three 1.8B Falcon-Mamba models (Zuo et al., 2024) on a shared

Splitting Setting	Vocab Size	Fertility Score ↓	Bytes per Token ↑
Individual digits	135k	1.3209	8.7924
No digits splitting	135k	1.2884	8.9688

Table 4: Fertility Score (Compression rate, digits excluded on the test set) and average number of bytes for different splitting regular expressions.

280GT dataset (including 40GT of decay stage). The models differed only in their tokenizer configurations: *Splits both digits and punctuation*; *Splits digits only*; *No specialized splitting for digits or punctuation*. As shown in Figure 6, despite some score fluctuations inherent to training dynamics, the results on the HumanEval code benchmark (Chen et al., 2021) indicate a clear trend: enabling both punctuation and digit splitting consistently leads to superior code generation performance. The benefits of punctuation splitting are further supported by a qualitative analysis of tokenization in non-Latin languages. In languages like Chinese and Japanese, punctuation marks are often full-width characters that can be incorrectly merged with adjacent words if not explicitly separated. Figure 7 illustrates this phenomenon. Without punctuation splitting, the tokenizer merges characters with punctuation, producing semantically incoherent units.

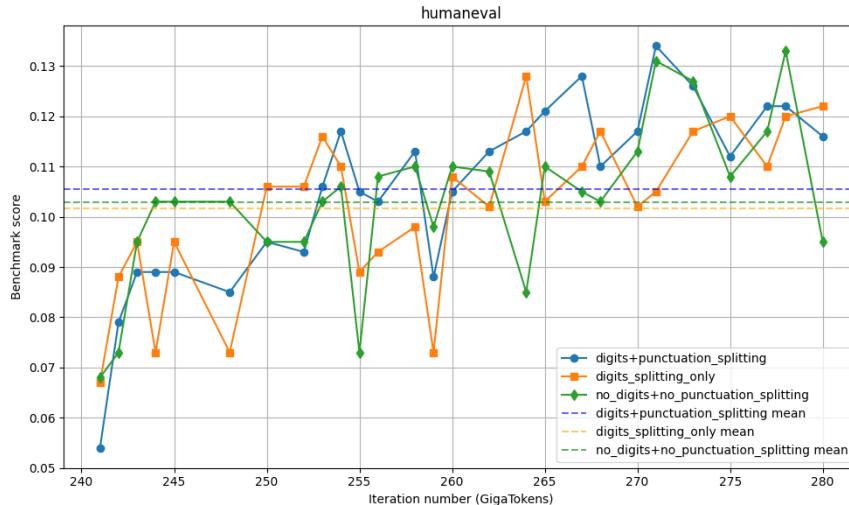


Figure 6: Model performance regarding different splitting strategies.

Without Punctuation Splitting (Falcon3 tokenizer)

Tokens are incorrectly merged with punctuation.

关闭此模式后，您将无法再看到它。如果您丢失了它，则必须创建一个新的。

With Punctuation Splitting (Falcon-H1 tokenizer)

Punctuation is correctly isolated, preserving word boundaries.

关闭此模式后，您将无法再看到它。如果您丢失了它，则必须创建一个新的。

Figure 7: Qualitative comparison for a Chinese sentence, with or without punctuation splitting.

Conclusion: Splitting both digits and punctuation seems to be the most effective strategy. This approach enhances performance on code and math tasks and ensures more robust, semantically meaningful tokenization across diverse languages. Moreover, our results underscore that tokenizer design should be guided by the actual model performance, rather than relying solely on proxy metrics such as fertility score.

How important are the common L^AT_EX tokens? Given the prevalence of L^AT_EX syntax in scientific and mathematical documents, we hypothesized that incorporating common L^AT_EX commands directly into the tokenizer’s vocabulary would enhance a model’s mathematical reasoning capabilities. The core principle is that representing frequent commands like `\frac` or `\sqrt` as single, atomic tokens simplifies the prediction task for the model, reducing the sequence length and compositional complexity of mathematical expressions.

To test this hypothesis, we curated a set of the most frequent L^AT_EX commands, mainly from the Overleaf documentation to ensure comprehensive coverage of formatting, referencing, and mathematical functions. We then conducted a controlled experiment by training two 1B-parameter Falcon-Mamba models (Zuo et al., 2024) on an identical, math-heavy data mixture of 280GT (including a 40GT decay stage). One model used our baseline tokenizer, while the other used a version where unused vocabulary slots were replaced with our curated set of L^AT_EX tokens. The models were evaluated on four different math benchmarks: MATH-Hard (Hendrycks et al., 2021b), gsm8k (Cobbe et al., 2021a), math_qa (Amini et al., 2019) and minerva-math. As shown in Figure 8, the model trained with the L^AT_EX-augmented tokenizer demonstrated a consistent and notable performance improvement across most benchmarks.

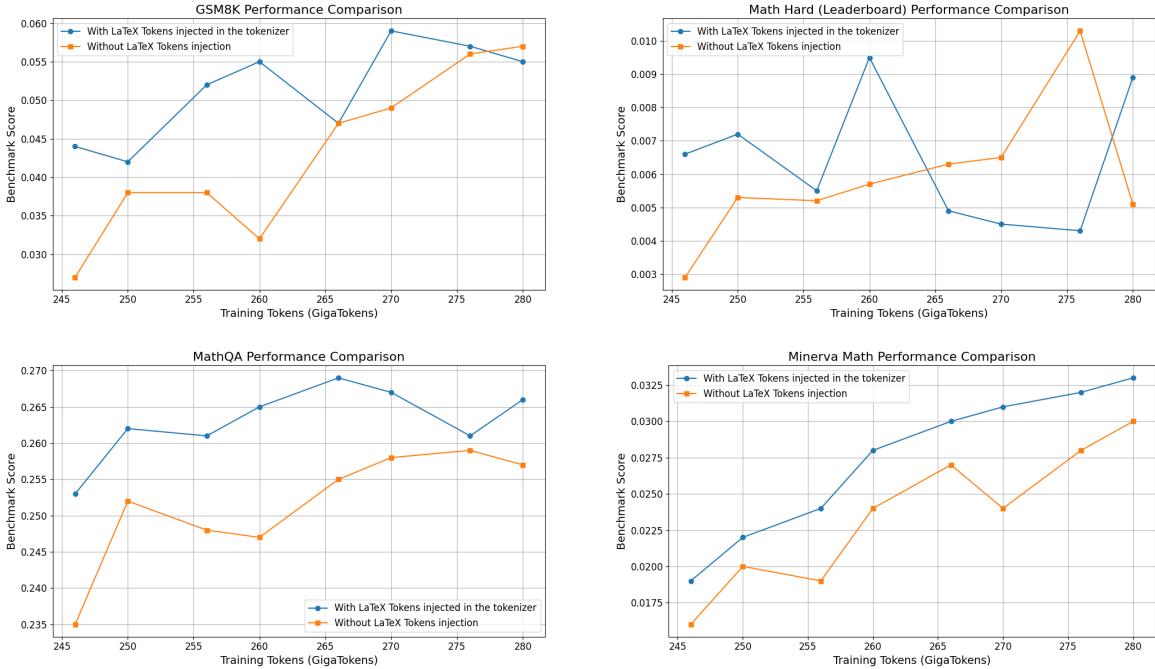


Figure 8: Performance on mathematical benchmarks for two 1B models during the training decay stage. The model trained with a tokenizer augmented with specialized L^AT_EX tokens (blue) consistently outperforms the baseline model (orange).

Conclusion: Enriching the tokenizer’s vocabulary with domain-specific tokens like L^AT_EX is an effective strategy for boosting model performance on specialized tasks. By providing a native vocabulary for mathematical syntax, we directly facilitate the model’s ability to process and reason about complex mathematical problems.

2.4.2 Final Tokenizer Design and Implementation

Our final tokenization strategy employs the Byte Pair Encoding (BPE) algorithm (Sennrich et al., 2016), trained on an extensive multilingual corpus covering over 121 languages ²⁷ to ensure broad linguistic support. We developed a suite of tokenizers with vocabulary sizes of 32K, 65K, 130K, and 261K, each tailored to a specific model scale (Table 1). A primary design principle is to scale the vocabulary size in proportion to the model’s overall architecture (Tao et al., 2024). This balance is critical for preventing the embedding layer from becoming disproportionately large, thereby maintaining computational efficiency during fine-tuning and inference, particularly in resource-constrained environments.

The design of these tokenizers incorporates key findings from our empirical studies. Specifically, we implement both digit and punctuation splitting, which we found essential for improving performance on code-related tasks and ensuring accurate segmentation in non-Latin languages. Furthermore, based on our experiments demonstrating improved mathematical capabilities, we manually inject common L^AT_EX commands directly into the vocabulary. To facilitate adaptability for downstream tasks, we reserve 1,024 special tokens across all tokenizers in the suite, providing end-users with the flexibility to customize the vocabulary for specific applications. A summary of the final trained tokenizers along with their vocabulary size is shown in the table 5.

Tokenizer name	Vocabulary size	Model
Falcon-H1-32k	32,768	Falcon-H1-0.5B*
Falcon-H1-65k	65,536	Falcon-H1-1.5B*, Falcon-H1-3B*
Falcon-H1-131k	131,048	Falcon-H1-7B*
Falcon-H1-262k	261,120	Falcon-H1-34B*

Table 5: List of available tokenizers trained for Falcon-H1

3. Pretraining

3.1 Pretraining Data

Capabilities of language models are known to come mainly from the training data, and that stays true for Falcon-H1 series. We have expanded our data corpus to more than 20 Teratokens, among which up to 18 Teratokens were used for training Falcon-H1 models. As shown in Table 1, each Falcon-H1 model comes with a different compute or token budget, considering their different learning capability and model capacity. The data mixtures are also designed differently across model scales. Noting that, although most Falcon-H1 models continued to show performance improvements toward the end of training, we chose to finalize training based on the allocated compute budget for each model.

The raw pretraining corpus was constructed from multiple sources, including web data, high-quality curated corpora, code, educational math content, and in-house synthetic data. We con-

ducted an extensive evaluation and studies of the data mixture, performing exhaustive tests and applying data optimization strategies. Notably, beyond the domain-specific or knowledge-intensive data typically emphasized in many LLM training pipelines, we observed that the organization of knowledge within the dataset has a significant impact on model performance. A strong correlation was found between factors such as training epochs, the number of unique high-quality data tokens, and the proportion of each data type within a training epoch. However, isolating and analyzing these factors independently is impractical, as they are interdependent and would substantially increase the complexity of the analysis during pretraining.

3.1.1 Data Sources

English Web data. Starting from FineWeb (Penedo et al., 2024a), we applied further quality filtering to improve knowledge density and training stability, using small language models as quality judges with carefully designed prompts. After processing the entire FineWeb dataset, we retained approximately 11T tokens.

Multilingual data. Apart from English, Falcon-H1 models support 17 languages (except the 0.5B model, which is English-only): Czech (cs), German (de), Spanish (es), French (fr), Hindi (hi), Italian (it), Japanese (ja), Korean (ko), Dutch (nl), Polish (pl), Portuguese (pt), Romanian (ro), Russian (ru), Swedish (sv), Urdu (ur), and Chinese (zh). The multilingual data corpus draws from diverse sources—mainly Common Crawl and a range of curated datasets.

For multilingual web data from Common Crawl, language identification was first performed at the HTML level using *pyclld2*, then refined post-extraction with *fasttext* and *trafilatura*. The data was segmented into five similarly sized partitions (4–13 dumps each), covering 2012 to mid-2024. All dumps from 2022–2024 were included, with older dumps sampled randomly due to lower multilingual content. Processing followed the heuristics-based pipeline of (Penedo et al., 2023), with language-specific tuning of Gopher Quality filtering, line-wise filtering, and stop words (Malarctic et al., 2024). A rule-based toxicity filter was applied using human-curated lists of offensive words. Native/proficient speakers rated each word (0: non-toxic, 1: context-dependent, 2: always toxic), and documents were filtered based on cumulative toxicity scores for which the formula was refined through human feedback. The processed data was re-partitioned (two to five parts per language) and deduplicated via MinHash at the part level. For languages with limited web data from above-mentioned data sources, we supplemented with additional public datasets: CulturaY (Thuat Nguyen & Nguyen, 2024) (Hindi, Korean, Urdu) and FineWeb2 (Penedo et al., 2024b) (Dutch, Romanian, Swedish, Korean, Urdu), processed through the same pipeline for consistency. The resulting multilingual web dataset for the 17 pre-trained languages totaled over 3,000 GigaTokens (GT), a portion of which was used during the pretraining phase.

To further enhance the multilingual data quality and diversity, we extract multilingual data from some highly curated data sources, including Wikipedia, academic preprints (arXiv, PubMed Central), online forums (Reddit, HackerNews, OpenSubtitles (Tiedemann, 2016), Ubuntu IRC, YouTube, StackOverflow), open-source books from Gutenberg (gut), public datasets like Europarl Corpus (eur), Gardian (Gardian, 2024), Makhzan (mak), and proprietary data.

Code data. Code data is widely considered as an important source for boosting a model’s general and reasoning capabilities. We have internally curated an extensive code corpus that contains file-level, repository-level, and High Quality (HQ) code splits. For the file-level split, we used an internally scraped and curated dataset, spanning 67 programming languages (listed in Appendix E.2). The data was sourced from GitHub repositories created up to May 2024 and from notebooks within the Meta Kaggle Code dataset ⁴. Notebooks were converted to scripts using

⁴<https://www.kaggle.com/datasets/kaggle/meta-kaggle-code>

JupyText⁵ after which all samples underwent heuristic filtering (Lozhkov et al., 2024). Language labeling followed the method in (Penedo et al., 2023), retaining files classified under one of 18 target languages, with a relaxed acceptance threshold (0.15), given that non-English content can typically appear in comments. Fuzzy deduplication was performed using MinHash and Local Sensitive Hashing (LSH) (Broder, 1997), computing 256 hashes per document with 5-grams and a Jaccard similarity threshold of 0.85. Personally Identifiable Information (PII), such as email addresses and IP addresses, was redacted using a pipeline inspired by DataTrove⁶ replacing tokens with standardized placeholders (e.g., >>EMAIL_ADDRESS<<, >>IP_ADDRESS<<).

The repository-level split containing long context code data was built with repository-level code samples (Guo et al., 2024; Hui et al., 2024), which were constructed by concatenating all source files from a given repository, enabling cross-file code comprehension. MinHash deduplication was performed at the repository level (prior to filtering), to preserve the logical structure of each repository. Files were concatenated in alphabetical order for effective duplicate detection, and later shuffled to mitigate bias and ensure a balanced representation.

For both file-level and repository-level code data, we curated a high-quality (HQ) split by applying code quality classifiers. For non-Python languages, we used a CodeBERT-based classifier⁷ covering 19 programming languages (see the full list in Appendix E.3), selecting samples that met a predefined quality threshold. For Python, we applied both the CodeBERT-based classifier and a specialized Python scorer⁸, retaining only samples that met the threshold for both classifiers. Additionally, we supplemented this corpus with the OpenCoder annealing corpus (Huang et al., 2024b), which includes algorithmic data, synthetic QA, and synthetic code snippet datasets⁹. We further preprocessed those data following the same methodology used for our file-level corpus.

Math data. We used a combination of open-source datasets and in-house crawled or retrieved math data from the Web. The open-source datasets include Proof-Pile-2 (Azerbayev et al., 2023), FineMath (Liu et al., 2024c), InfiMM-WebMath-40B (Han et al., 2024), and OpenCoder FineWeb Math corpus (Huang et al., 2024a), largely consisting of math data extracted and filtered from Common Crawl. For the in-house math data, following a similar approach to (Shao et al., 2024), we first used a *fastText* classifier trained on OpenWebMath (Paster et al., 2023) to retrieve OpenWebMath-like pages. The classifier was then iteratively refined with extracted math data to expand coverage, targeting top math-related domains from Common Crawl. All math data underwent a decontamination process to remove overlaps with popular math benchmarks such as GSM8K (Cobbe et al., 2021a) and MATH (Hendrycks et al., 2021a).

Synthetic data. We used a combination of external open datasets and a large volume of in-house generated synthetic data. External sources include subsets of Nemotron-CC (Su et al., 2024) (`diverse_qa_pairs`, `extract_knowledge`) and Cosmopedia v2 (Ben Allal et al., 2024). From our experiments, we found that fully synthetic data generated without grounding on seed samples often lacks diversity and suffers from inconsistent quality. To address this, our in-house synthetic data was primarily created by rewriting curated raw data—including web, code, books, Wikipedia, arXiv, math sources, etc. We employed a diverse set of internal and external open models across various scales and architectures, considering the compute cost, data quality, and content diversity. Instead of focusing only on question-answer pairs, our generation strategies included enhancing writing style, increasing knowledge density, filtering redundant tokens, and applying iterative quality control to the generated samples. This rewriting process helped structure and formalize the

⁵<https://jupytext.readthedocs.io/>

⁶<https://github.com/huggingface/datatrove/blob/main/src/datatrove/pipeline/formatters/pii.py>

⁷https://huggingface.co/devngho/code_edu_classifier_v2_microsoft_codebert-base

⁸<https://huggingface.co/HuggingFaceTB/python-edu-scorer>

⁹<https://huggingface.co/datasets/OpenCoder-LLM/opc-annealing-corpus>

underlying knowledge, reduce noise, and ultimately improve training stability and efficiency.

Apart from rewriting raw samples, we also generate raw question-answer pairs from DeepMind’s mathematics dataset ([Saxton et al., 2019](#)), which are further enhanced with context-enriched questions, and correct chain-of-thought solutions generated in a way similar to STaR ([Zelikman et al., 2022](#)). To further increase the knowledge intensity of our pretraining corpus, we also generated synthetic textbooks using carefully constructed topic hierarchies extracted from Wikipedia, covering over 30K topics. Starting from 99 root categories (see Table 41 in Appendix), we crawled the Wikipedia category graph up to a depth of 5 (or until reaching article nodes), followed by deduplication and pruning of irrelevant topics. For each hierarchy, we first generated a table of contents, then created structured content for each unit—including comprehensive explanations, relevant examples, and diverse exercises.

Long context data. Long-context data naturally appears in sources such as web pages, repository-level code, books, and academic papers (e.g., arXiv). To enhance the model’s ability to handle extended sequences, we applied a range of restructuring strategies across target lengths of 32K, 128K, and 256K tokens. These include Fill-in-the-Middle (FIM), where random sections are removed from documents, and section reordering, where segments are shuffled and the model is tasked with reconstructing the original order—requiring a combination of long-context reasoning, memory, and coherence understanding. Additionally, we created a small set of synthetic long-context samples with question-answer pairs to further improve capabilities such as in-context learning, complex pattern retrieval, etc.

3.1.2 Data Strategy

Data validation. We carefully inspect each data source prior to injecting it into the training pipeline to obtain a detailed understanding of its quality and its impact on domain-specific tasks as well as on overall model performance. More specifically, to derive strong, interpretable signals for each data source, we train multiple 0.5B-scale models from scratch, subsequently evaluating their performance on carefully selected domain benchmarks. An alternative strategy involves training a single model from scratch and then modifying the data mixture during the learning rate decay stage to assess data quality with reduced computational costs ([Grattafiori et al., 2024](#)). However, in our experiments, this approach sometimes failed to yield clear or robust conclusions regarding data quality. This is primarily due to additional confounding factors, e.g., data distribution shift between the stable and decay stage, correlations in data used across these stages, and data mixtures within each of these stages, etc. To this end, we systematically train 0.5B models from scratch using either individual data sources or well-studied combinations of them. This enables us to isolate extra factors and examine multiple dimensions: absolute data quality, relative quality in comparison to existing datasets, interactions and correlations between different data sources and formats, and their respective impacts on various domain-specific tasks. Once the data is validated at the 0.5B scale, it will be passed to the models at medium and large model scales, being retained or adjusted based on the observed improvements in model performance.

Deterministic data loading and check-pointing. When adopting multiple data sources into training, we implemented a deterministic dataloader that reads samples sequentially from each source rather than sampling them randomly. This design offers several advantages: a) deterministic training behavior, enabling precise comparisons across different runs; b) flexible continual pretraining through coordinated model and data checkpointing across multiple data sources, without disrupting the loading order; c) dynamic data mixture updates during pretraining without needing to restructure data files or data classes; and d) improved control over multi-epoch training, allowing custom epoch sizes across diverse data sources on the fly - we define *one training epoch* as

a full pass over all unique tokens from a given data source.

Data mixture. Effective data organization strategies proved critical to the final performance of our models. These strategies include choices around the *data mixture*, *pre-training stages*, and *multi-epoch training*. In practice, these factors are tightly interconnected and require joint optimization, taking into account compute budgets, model scales, data source quality, and data volumes.

One common concern when reducing the proportion of web data is its potential impact on model generalization and knowledge diversity. However, we found this concern might be somewhat overstated. While popular web datasets like FineWeb (Penedo et al., 2024a) or Refined-Web (Penedo et al., 2023) offer broad topical diversity, their knowledge density is relatively low, even when enhanced by domain-specific filtering such as FineWeb-Edu (Penedo et al., 2024a) or FineFineWeb (M-A-P et al., 2024). With highly knowledge-intensive data sources and rewritten web samples, the raw web data can be significantly reduced without impacting model generalization and knowledge diversity.

Through extensive experiments on data quality and validation, we iteratively refined and converged on an optimal data mixture across the pretraining process. Starting with an initial mixture validated on smaller models, we progressively adjusted the mixture throughout the training by incorporating newly optimized and carefully evaluated data at various model scales. This process led to the final data configurations for the 34B and 7B models (Table 6), where web data accounted for only about 15% and 12.35%, respectively—substantially increasing the share of rewritten data (over raw web, code, curated sources, math data, etc.). For the smaller 3B, 1.5B, and 0.5B models, we maintained a constant data mixture throughout the whole pretraining stage, leveraging the dynamic data preparation carried out during the pretraining of the larger models.

Data Source	34B		7B		3B	1.5B	0.5B
	Start	End	Start	End	Mix	Mix	Mix
Raw data	99.47	43.45	81.07	42.26	39.70	23.20	11.50
Web	40.00	14.60	25.00	12.35	11.60	10.20	6.50
Curated	25.00	15.93	26.00	16.47	11.68	4.75	0.00
Code	20.00	10.05	20.00	10.74	14.00	8.00	5.00
Math	14.47	2.87	10.07	2.70	2.42	0.25	0.00
Rewritten data	0.23	52.05	10.56	53.04	56.80	69.80	75.50
Web & Curated	0.00	20.36	0.00	18.12	22.08	23.75	20.50
Code & Math	0.23	31.69	10.56	34.92	34.72	46.05	55.00
Synthetic data*	0.30	4.50	8.37	4.70	3.50	7.00	13.00
Total	100.00	100.00	100.00	100.00	100.00	100.00	100.00

* Fully synthetic samples, not derived from rewriting existing raw data.

Table 6: Falcon-H1 data mixtures across model sizes. For the 34B and 7B models, both start and end-of-training mixtures are shown. For the 3B, 1.5B, and 0.5B models, a single, static data mix was used. All values are in percent (%).

To mitigate the risk of overfitting or bias towards specific domain tasks and to preserve a balanced performance across diverse skill areas, we perform frequent checkpoint evaluations over diverse domain tasks throughout the pretraining stage. Additionally, whenever the data mixture changed, we perform frequent *vibe checks* on intermediate model checkpoints. This combined strategy ensures that the model maintains strong general capabilities while avoiding unintended

specialization or biases toward particular domains.

Data organization and scheduling. A primary challenge in large-scale pretraining is the profound imbalance between the vast quantities of web-scale data and the relative scarcity of high-quality, curated datasets. Specialized corpora like mathematical texts, while crucial for model capability, comprise a small fraction of the total training data corpus. To ensure these high-quality sources maintain their influence, we employ an aggressive up-sampling strategy enabled by multi-epoch training. This approach decouples the effective training data size from the raw token count of our finite, high-quality corpora.

This extensive reuse of data challenges the common practice of single-epoch training, often adopted to mitigate the risk of model memorization. However, our empirical investigation suggests that concerns about memorization may be overstated in large-scale regimes. By estimating the model’s *memorization window*—the temporal span over which it retains specific training examples—we found it possible to reuse high-quality samples multiple times without compromising generalization. As illustrated in Figure 9, this window can be approximated by analyzing the training loss on tokens seen at various points in the past. This finding is particularly relevant as most literature on memorization and catastrophic forgetting focuses on smaller models or shorter training durations.

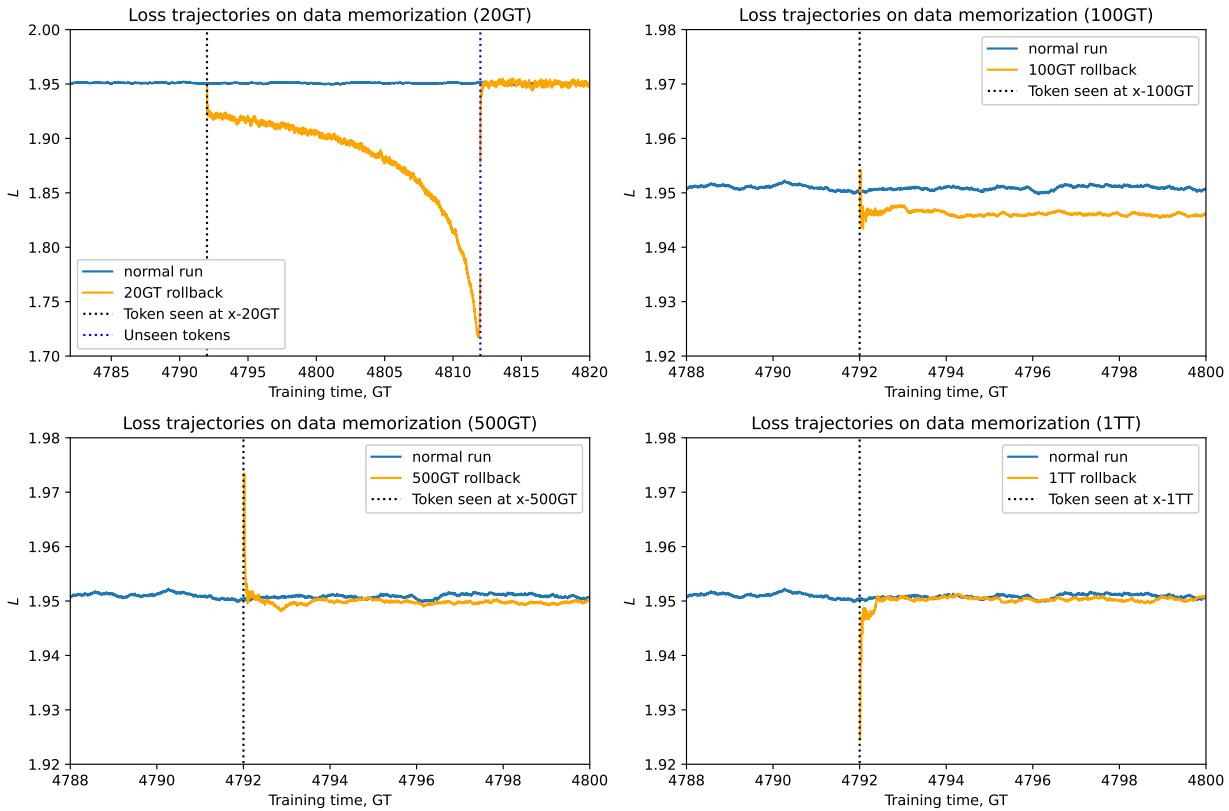


Figure 9: Model’s memorization window and loss trajectories

Beyond data composition, we found data scheduling to be a critical factor. Counter-intuitively, our experiments revealed that an "anti-curriculum" approach yielded superior results compared to conventional curriculum learning. In this paradigm, we introduce data of all complexity levels—from simple text to advanced mathematical problems—from the very beginning of training.

We hypothesize that this early and continuous exposure provides the model with a more effective learning trajectory to develop the internal features required to master complex tasks. Across models of various scales, our experiments confirmed that maintaining this optimized data mixture from the outset outperforms curriculum strategies that reserve high-quality data for later training stages. This finding holds when a sufficient volume of high-quality data is available.

Long-context training. We initiated pretraining with an 8K context window, with the majority of pretraining conducted at a 16K context length. An exception is Falcon-H1-0.5B, which was primarily trained at a 4K context due to its limited capacity for handling longer sequences. Thanks to our infrastructure optimizations, throughput degradation at 16K context was minimal. Moreover, we observed a performance boost compared to training with 4K or 8K context lengths. For long-context extension, we maintained the overall data mixture largely unchanged, but slightly increased the proportion of long-context samples within each data source. Following the learning rate decay phase, we continued training with a fixed minimum learning rate during long-context extension. We empirically found that performing this extension after the decay phase yielded no significant performance difference compared to extending it before decay. However, the post-decay approach was substantially more compute-efficient due to the reduced training throughput of long sequences. For Falcon-H1-34B and Falcon-H1-7B, we applied 60GT at both 32K and 128K context, and 25GT at 256K context. For Falcon-H1-3B and Falcon-H1-1.5B, the same token counts were used for both the 32K and 128K context stages.

3.2 Training Dynamics

3.2.1 Training Stability

During early experiments with Falcon-H1 we observed severe loss spikes from the beginning of the training. These spikes created two practical problems:

- (i) Spiky loss curves distort ablation results: a variant may look inferior simply because a spike coincides with the learning-rate decay, reversing the true ordering; and
- (ii) continuing with such spikes would force us to choose learning rates well below the optimum, which would slow convergence.

Eliminating the instability therefore became a prerequisite for any meaningful ablation.

Spike-like behaviour has been reported before in *Falcon2* (Malartic et al., 2024), *Falcon-Mamba* (Zuo et al., 2024), *Jamba* (Lieber et al., 2024), and *YuLan* (Hu et al., 2024b). Falcon2 attributed the issue to depth, whereas YuLan pointed to three potential triggers: exploding residual paths, unstable layer-norm statistics, and extreme attention scores.

A common practice is to employ *batch-skipping*: whenever the loss on a batch exceeds a user-defined multiple of the running median, that batch is dropped and the optimiser moments remain unchanged. This heuristic can suppress spikes caused by a handful of outlier examples, yet it is largely ineffective for *dynamics-induced* instabilities and only postpones the problem.

Isolating the Source. Our initial experiments were conducted on a *pure Mamba2* baseline. Loss spikes were already present in this setting, showing that the SSM pathway is *sufficient* to trigger the instability—although it may not be the sole contributor.

To test whether the spikes were driven by width or depth, we compared two vanilla Mamba2 variants trained under identical conditions:

- **Wide:** larger hidden dimension and more SSM heads, but fewer layers;

- **Deep:** smaller hidden dimension, comparable total parameter count, but more layers.

All other hyperparameters were kept fixed, and the learning rate was scaled as per μP scaling.

The wide model exhibited pronounced loss spikes, whereas the deep model trained smoothly. This finding implicates widthrelated dynamics inside the SSM—specifically the larger number of heads—as a primary driver of the observed instability, and motivated the analysis reported in the next sections.

Diagnosing the SSM Dynamics. We logged parameter statistics throughout training and found that spiking runs displayed a wider range of \tilde{dt}_t values. Recall from (6) that $dt_t = \text{softplus}(\tilde{dt}_t + b)$ controls both forgetting of and writing to SSM hidden state, as can be seen from (6). Therefore, large positive \tilde{dt}_t values have two antagonistic effects: linearly enlarging the information from the *current* token written into the hidden state while exponentially forgetting the information from the *previous* tokens.

Whenever the modeling objective requires both the recent token and its long-range context, gradient descent is pulled in opposite directions:

- increase \tilde{dt}_s so that dt_s increases and amplifies the contribution of token s through $\mathbf{C}_t^\top \mathbf{B}_s dt_s$ in (7).
- keep $\prod_{j=s'}^s \bar{A}_j$ with $s' < s$ from collapsing so that earlier information still reaches position t and propagates further.

Because these antagonistic signals arrive at different scales and at different optimization steps, the parameter overshoots and then over-corrects, producing the characteristic loss spikes.

Mitigation. We tested three interventions:

- Clip A_{\log} : *no effect*.
- Clip negative dt : *no effect*.
- Clip positive dt : **completely removed spikes**.

This confirms our previous hypothesis: spikes are caused by writing to the hidden state. However, we recognize that such clipping may restrict expressiveness. A softer alternative is to multiply the dt activation by a constant $0 < \alpha < 1$. This *attenuation* preserves the full parameter range while preventing early excursions into the unstable regime.

With attenuation enabled we can train Falcon-H1 at relatively high learning rates without observing any loss spikes. The attenuation factor is a part of the μP forward multipliers (see: 3.2.3) and will be tuned as well with other multipliers.

3.2.2 Effective Learning Rate and Effective Weight Decay

Parameter norms. To initiate the discussion of the joint effect of AdamW learning rate (LR) η and weight decay (WD) λ on the model training, we start with the behavior of parameter norms of matrix layers, to which WD was applied during training. Figure 10 (left) shows that parameter norms grow indefinitely with no WD $\lambda = 0$ while stabilizing at a constant level when $\lambda > 0$. The value of parameter norms after stabilizing $\|W\|$ depends on LR, WD, and also batch size B , raising the question of the precise form of this dependence.

Leaving the dependence of parameter norms on batch size to the future work, we have found that the dependence of $\|W\|$ on (η, λ) is well described, for all matrix layers W , by a simple scaling rule

$$\|W\| \propto \sqrt{\frac{\eta}{\lambda}}. \quad (10)$$

Moreover, if the norm is normalized in a mean fashion as $\|W\|^2 = \frac{1}{mn} \sum_{i=1}^n \sum_{j=1}^m W_{ij}^2$, its values for different layers W in the architecture are extremely close to each other, see figure 10 (left).

In fact, the scaling (10) is natural and is observed in a simple model of stochastic dynamics where Brownian expansion due to noise in the gradients dominates the attraction to the optimal parameter value. We provide such a toy model in section B and show that in the relevant regime of hyperparameters, the dependence weight norm on LR and WD has the functional form

$$\|W(\eta, \lambda)\|^2 = C_1 \frac{\eta}{\lambda} + C_2 \frac{1}{\lambda^2}, \quad (11)$$

where the constants C_1, C_2 are determined by the variance of the gradient noise and steepness of the loss function near the optimal value of the parameters. Consider the region of (η, λ) typically used in the model pertaining. One extreme scenario is large noise variance and a flat loss landscape, where the weight norm is mostly determined by the balance between weight decay contraction and Brownian expansion due to stochastic optimizer updates. Then, we have $C_1 \frac{\eta}{\lambda} \gg C_2 \frac{1}{\lambda^2}$ and the weight norms scaling is given by (10). The opposite extreme scenario is small noise variance and steep loss landscape, where the weight norm is mostly determined by the balance between weight decay contraction and attraction of the parameters to the optimal value W^* . Then, we would have $C_1 \frac{\eta}{\lambda} \ll C_2 \frac{1}{\lambda^2}$ and the weight norms scaling be $\|W\| \propto \lambda^{-1}$.

To determine which of the two above scenarios better describes the behavior of the norms during the actual model training, we performed a 2d sweep over (η, λ) values and measured the norms by the end of the constant LR stage of WSD schedule, just before LR decay. The results are depicted in the figure 10 and show that the scaling (10) indeed describes parameter norms very well, suggesting that the noise plays the dominant role in the dynamics of the matrix layers.

Effective learning rate. It is convenient to assign to both LR and WD a simple, intuitive role they play in the training process. Weight decay squeezes the learnable parameters and, therefore, controls their norms. Learning rate determines the speed of learning, and also the noise level in the presence of label noise (Liu et al., 2025). As a result, the final loss has a steep dependence on the LR, making it the most important parameter to be tuned.

This simple intuition does not hold for weight decay as we saw in the previous section: the combination $\sqrt{\frac{\lambda}{\eta}}$ controls the parameter norms instead of WD λ itself. Moreover, the LR intuition also does not hold, as we show in Figure 11 (Left): changing λ has a similar effect on the loss curve as changing η . We argue that simple LR and WD intuition can be kept if we switch original η, λ with *effective learning rate* (ELR) and *effective weight decay* (EWD) defined as

$$(ELR) \quad \eta_{\text{eff}} = \sqrt{\eta \lambda}, \quad (EWD) \quad \lambda_{\text{eff}} = \sqrt{\frac{\lambda}{\eta}}. \quad (12)$$

Essentially, the EWD definition was already verified in the previous section due to parameter norms scaling (10), equivalent to $\|W\| \propto \lambda_{\text{eff}}^{-1}$. Here we take $\|W\| \propto \lambda^{-1}$ as an intuitive functional form of parameter norm scaling that can be observed, for example, for minimizers of L_2 regularized loss with strong regularization λ , or in the weak noise scenario of (11).

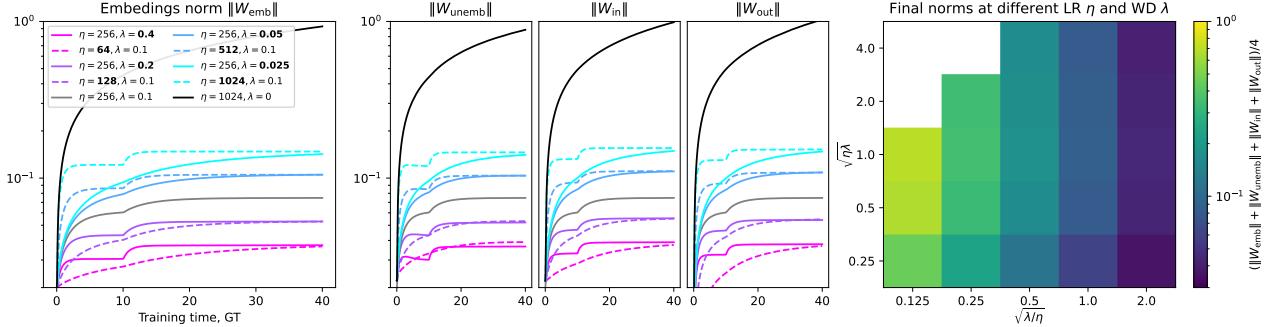


Figure 10: Behavior of weight norms of embedding W_{emb} , unembedding W_{unemb} , and input/output projections $W_{\text{in}}, W_{\text{out}}$ layers of pure Mamba2 model. **(Left)** Evolution of weight norms during training of 1B model. In addition to no WD run, we show WD sweep (solid) and LR sweep (dashed), both of which display similar impact on weight norms. Learning rate is measured in the units of 10^{-5} ; and the jump of norms at 10GT corresponds to batch size doubling within rampup. **(Right)** Average weight norm across all model layers at two-dimensional (η, λ) sweep on 300M model. The combinations $\sqrt{\frac{\lambda}{\eta}}$ and $\sqrt{\eta\lambda}$ are used as coordinate axes to demonstrate that weight norms are scaled as in (10). LR and WD are measured in the units of $\eta_0 = 256 \times 10^{-5}$ and $\lambda_0 = 0.1$. Empty cells correspond to diverged runs.

A heuristic way to obtain ELR definition (12) is to look at an optimizer part δW_t of a single step update $W_{t+1} = W_t - \eta \delta W_t - \lambda \eta W_t$ of a linear layer W , and define ELR as “meaningful” measure of the update strength associated with δW_t . We start with a proposition that relative magnitude of the parameters change $\frac{\|\eta \delta W_t\|}{\|W_t\|}$ is a more meaningful measure than the absolute magnitude $\|\eta \delta W_t\|$.

Then, recall that for Adam the update is given by a ratio $\delta W_t = \frac{G_{1,t}}{\sqrt{G_{2,t} + \epsilon}}$ of gradient moments $G_{1,t}, G_{2,t}$. An important empirical observation is that each component of the ratio has roughly the same magnitude around 0.1, almost independent from LR and WD used in the training. As a result, we can treat $\|\delta W_t\|$ as a constant, leading to simplified update strength measure $\frac{\|\eta \delta W_t\|}{\|W_t\|} \rightarrow \frac{\eta}{\|W_t\|}$. Next, assume that the parameter norms have already stabilized at their stationary state given by the scaling $\|W_t\| \propto \sqrt{\frac{\eta}{\lambda}}$. This further changes update strength measure to $\frac{\eta}{\|W_t\|} \rightarrow \sqrt{\eta\lambda}$, which is exactly our ELR definition (12).

Now, let us examine the properties of ELR on the noise level during training, that we can roughly measure as a difference of the loss just before and after learning rate decay $L_{\text{before LRD}} - L_{\text{after LRD}} = L_{\text{noise}}$. In the figure 11 (right) we plot L_{noise} for different values of ELR η_{eff} and EWD λ_{eff} . We see that keeping $\eta_{\text{eff}} = \text{const}$ and changing λ_{eff} has a weak effect on L_{noise} while keeping $\lambda_{\text{eff}} = \text{const}$ and changing η_{eff} affects the noise level strongly. We can roughly interpret this observation as the noise level being the function of only the effective learning rate $L_{\text{noise}}(\eta, \lambda) = L_{\text{noise}}(\eta_{\text{eff}})$. Essentially, this shows that ELR actually controls the noise level and closes our earlier statement that learning rate and weight decay intuition hold for EWD and ELR.

Finally, let us point out to an additional property of EWD and ELR that we call *log-scale orthogonality* and which is useful for (η, λ) sweeps often done within empirical tuning of training hyperparameters. A robust strategy for such sweeps is to use a log-scaled grid, for example, with the powers of 2: $(\eta, \lambda) \in \{\eta_0 2^i, \lambda_0 2^j\}_{i,j=0}^n$, which ensures a good balance between precision and coverage. Such experiment design naturally corresponds to the Euclidean metric, but in log-coordinates ($\log \eta, \log \lambda$). In this metric, the gradients of EWD and ELR turn out to be orthogonal,

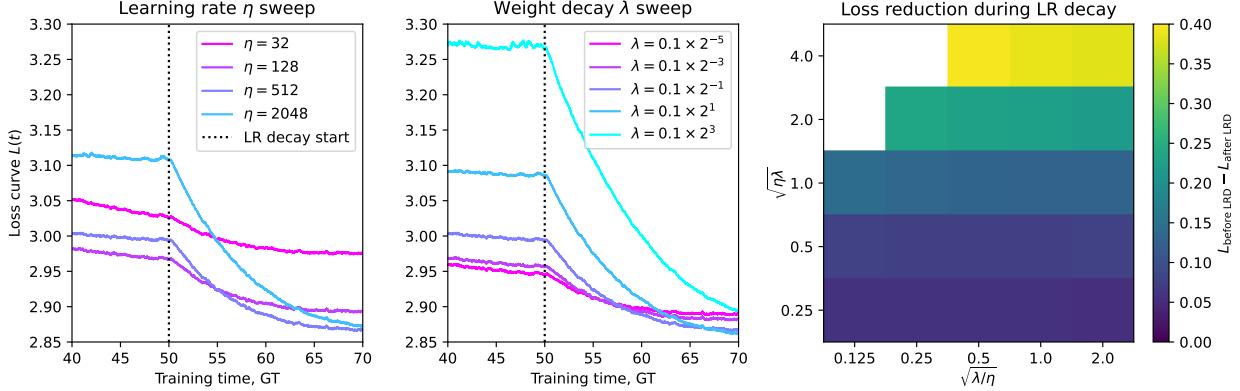


Figure 11: Effect of LR η and WD λ of the training loss curve of a 300M pure Mamba2 model. (**Left**) LR and WD sweeps on left and right subplots show that increasing (or decreasing) either LR or WD has a similar effect on the loss curve. (**Right**) Noise level measure as the loss gap before and after LR decay $L_{\text{noise}} = L_{\text{before LRD}} - L_{\text{after LRD}}$ at two-dimensional (η, λ) sweep. Again, the use of ELR and EWD for coordinate axis shows that the noise is mostly determined by $\eta_{\text{eff}} = \sqrt{\eta\lambda}$.

unlike in the standard Euclidean metric on the (η, λ) plane.

$$\nabla^{(\log)} \eta_{\text{eff}} \cdot \nabla^{(\log)} \lambda_{\text{eff}} = \frac{\partial \eta_{\text{eff}}}{\partial \log \eta} \frac{\partial \lambda_{\text{eff}}}{\partial \log \eta} + \frac{\partial \eta_{\text{eff}}}{\partial \log \lambda} \frac{\partial \lambda_{\text{eff}}}{\partial \log \lambda} = 0. \quad (13)$$

With the above orthogonality property, the sweeps on log-scale grids contain directions that maximally change the noise level (via ELR) while keeping the parameter norms constant (via EWD), and vice versa. This makes possible to effectively measure the effect of noise level and parameter norms on training dynamics, and we will utilize it for our μP sweeps in the next section.

To summarize, ELR and EWD defined in (12) have the following useful properties

- (A) **Parameter norms** $\|W\|$ are fully determined by EWD λ_{eff}
- (B) **Noise level** is fully determined by ELR η_{eff} .
- (C) **Log-scale orthogonality.** Directions of increase of ELR and EWD are orthogonal on $(\log \eta, \log \lambda)$ plane, allowing to separately and efficiently measure their effects within sweeps on log-scale grids.

We emphasize, however, that the above conclusions are “rough” approximations based on a limited set of experiments performed for Falcon-H1 training, and accurate investigation of these questions would be an exciting direction for future work.

Incorporating parameter norms scaling into the power scheduler. Recent work (Shen et al., 2024; Bjorck et al., 2025) noticed that for longer training durations T , the optimal learning rate of the WSD scheduler scales as $\eta_{\text{opt}} \propto T^{-b}$ with the exponent b roughly in the range $[0.3, 0.5]$. As a next step, (Shen et al., 2024) suggests to downscale the learning rate of the stable stages starting from specified activation time t_0 as $\eta(t) = \eta_0 \sqrt{\min(1, \frac{t_0}{t})}$, referred to as power scheduler (PS). Our internal tests support these conclusions with PS schedule achieving better loss than learning rate tuned WSD at different training token scales.

However, we also decided to incorporate our insights on the role of weight decay into the schedule. Specifically, let us relate the PS scaling of learning rate and weight decay to the scaling of ELR and EWD

$$\begin{cases} \eta(t) \propto t^{-\frac{1}{2}} \\ \lambda(t) \propto \text{const}(t) \end{cases} \implies \begin{cases} \eta_{\text{eff}}(t) \propto t^{-\frac{1}{4}} \\ \lambda_{\text{eff}}(t) \propto t^{\frac{1}{4}} \end{cases} \quad (14)$$

Now, we combine two observations. On the one hand, we observed that the final loss is much more sensitive to ELR than to EWD. In that case, we can mostly attribute the effectiveness of the PS scaling (14) due to the scaling of ELR rather than EWD. On the other hand, we still observed substantial gains from tuning weight decay μP multipliers, suggesting that it is beneficial to have tuned parameter norms of all the layers in the architecture. Then, if we would like to keep parameter norms at the “optimal” level throughout the whole training, we should not scale EWD, which controls the parameter norms. This brings us to *Effective Power Scheduler* (EPS) scaling

$$\begin{cases} \eta_{\text{eff}}(t) \propto t^{-\frac{1}{4}} \\ \lambda_{\text{eff}}(t) \propto \text{const}(t) \end{cases} \implies \begin{cases} \eta(t) \propto t^{-\frac{1}{4}} \\ \lambda(t) \propto t^{-\frac{1}{4}} \end{cases} \quad (15)$$

While we have observed an improved convergence speed of EPS over PS, a systematic study is required to properly incorporate weight norm control into the training schedule. In particular, EPS schedule we propose rests on the assumption that parameter norms should not be scaled during long training runs, which is not guaranteed to be the optimal choice.

3.2.3 Maximal Update Parametrization (μP) with Tunable Multipliers

Background. Maximal update parametrization (μP) was proposed in (Yang & Hu, 2022). It combines several ideas of how different factors affect the ability of the model to learn non-trivial internal representations as the model’s width d increases. One such factor is parameterizing the weight matrix W with a scalar multiplier m , so that the forward pass becomes $\mathbf{y} = mW\mathbf{x}$. Another is initialization variance and learning rate of different layers in the model’s architecture. The impact of these factors is formulated rigorously in the infinite width limit $d \rightarrow \infty$, where a unique scaling of main parameters with d is required to ensure nontrivial feature learning. The multiplier m , initialization variance σ^2 , learning rate η , and weight decay λ must scale with a certain powers of width d :

$$m = m_0 d^{-a}, \quad \sigma = \sigma_0 d^{-b}, \quad \eta = \eta_0 d^{-c}, \quad \lambda = \lambda_0 d^{-e}, \quad (16)$$

with specific values of scaling exponents (a, b, c, e) depending on the location of the layer W in the model architecture and the type of optimizer used, e.g. SGD or Adam. For example, for AdamW optimizer, μP scaling of hidden layers are $(a, b, c, e) = (0, \frac{1}{2}, 1, -1)$ while for output(LM head) layer $(a, b, c, e) = (0, 1, 1, -1)$. The subsequent work (Yang et al., 2023; Bordelon et al., 2023) also explores the scaling with model depth L to enable feature learning in the limit $L \rightarrow \infty$.

The main practical implication of μP is zero-shot hyperparameter (HP) transfer (μ Transfer) (Yang et al., 2022). Essentially, if one finds the optimal hyperparameters at a reference, typically small, model size d_{ref} , the optimal HPs at larger target size d could be roughly obtained from reference model values by using scaling rules (16). μ Transfer were previously applied for tuning and transferring LLM HPs in (Yang et al., 2022; Dey et al., 2023; Hu et al., 2024a; Dey et al., 2025; Liu et al., 2023c).

An important aspect of model parametrization is the presence of an exact symmetry transformation that rescales $(m, \sigma, \eta, \lambda)$ such that both the forward pass and the optimizer update stay

Architecture part	Forward pass	Shapes	Multiplier scaling
Full model	$m_{\text{unemb}} W_{\text{unemb}} \mathcal{N}^f((\mathcal{F}_L(\dots \mathcal{F}_1(m_{\text{emb}} W_{\text{emb}} X) \dots)))$	$W_{\text{emb}} \in \mathbb{R}^{d \times d_{\text{voc}}}$ $W_{\text{unemb}} \in \mathbb{R}^{d_{\text{voc}} \times d}$	$m_{\text{emb}} \propto 1$ $m_{\text{unemb}} \propto d^{-1}$
Hybrid block	$\mathcal{F}_l(\mathbf{r}_l) = \mathbf{r}'_l + \mathcal{F}_l^{\text{MLP}}(\mathcal{N}'_l(\mathbf{r}'_l))$ $\mathbf{r}'_l = \mathbf{r}_l + \mathcal{F}_l^{\text{attn}}(\mathcal{N}_l(\mathbf{r}_l)) + \mathcal{F}_l^{\text{SSM}}(\mathcal{N}_l(\mathbf{r}_l))$	$\mathbf{r}_l, \mathbf{r}'_l \in \mathbb{R}^{d \times L_{\text{seq}}}$	
SSM (Mamba2) block	$\tilde{\mathbf{x}} = m_x W_x \mathbf{r}$ $\tilde{\mathbf{B}} = m_B W_B \mathbf{r}$ $\tilde{\mathbf{C}} = m_C W_C \mathbf{r}$ $\mathbf{xBC} = \text{SiLU}(\text{conv1d}(\tilde{\mathbf{x}} \tilde{\mathbf{B}} \tilde{\mathbf{C}}))$ $\tilde{\mathbf{z}} = m_z W_z \mathbf{r}$ $\mathbf{dt} = \text{Softplus}(m_{dt} W_{dt} \mathbf{r} + b_{dt})$ $\mathbf{y}_{\text{SSM}} = \text{SSM}(\mathbf{x}, \mathbf{B}, \mathbf{C}, \mathbf{dt}) \odot \text{SiLU}(\tilde{\mathbf{z}})$ $\mathcal{F}_{\text{SSM}}(\mathbf{r}) = m_{\text{SSM}} W_{\text{SSM}} \mathcal{N}^{\text{SSM}}(\mathbf{y}_{\text{SSM}})$	$W_x \in \mathbb{R}^{d_{\text{ssm}} n_h \times d}$ $W_z \in \mathbb{R}^{d_{\text{ssm}} n_h \times d}$ $W_B \in \mathbb{R}^{d_{\text{state}} n_g \times d}$ $W_C \in \mathbb{R}^{d_{\text{state}} n_g \times d}$ $W_{dt} \in \mathbb{R}^{n_h \times d}$ $W_{\text{SSM}} \in \mathbb{R}^{d \times d_{\text{ssm}} n_h}$	$m_x \propto d^{-1}$ $m_z \propto d^{-1}$ $m_B \propto (d_{\text{state}} n_g)^{-1}$ $m_C \propto d^{-1}$ $m_{dt} \propto d^{-1}$ $m_{\text{SSM}} \propto (d_{\text{ssm}} n_h)^{-1}$
Attention block	$\mathbf{Q} = W_Q \mathbf{r}$ $\mathbf{K} = m_{\text{key}} W_K \mathbf{r}$ $\mathbf{V} = W_V \mathbf{r}$ $\mathcal{F}_{\text{attn}}(\mathbf{r}) = m_{\text{attn}} W_{\text{attn}} \text{GQA}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$	$W_{\text{attn}} \in \mathbb{R}^{d \times d_h^{\text{attn}} n_h^{\text{attn}}}$ $W_Q \in \mathbb{R}^{d_h^{\text{attn}} n_h^{\text{attn}} \times d}$ $W_K \in \mathbb{R}^{d_h^{\text{attn}} n_g \times d}$ $W_V \in \mathbb{R}^{d_h^{\text{attn}} n_g \times d}$	$m_{\text{attn}} \propto (d_h^{\text{attn}} n_h^{\text{attn}} d)^{-1}$ $m_{\text{key}} \propto d^{-2} (d_h^{\text{attn}})^{-\frac{1}{2}}$
MLP block	$\mathbf{y}_{\text{MLP}} = \text{SiLU}(m_{\text{gate}} W_{\text{gate}} \mathbf{r}) \odot (W_{\text{up}} \mathbf{r})$ $\mathcal{F}_{\text{MLP}}(\mathbf{r}) = m_{\text{MLP}} W_{\text{down}} \mathbf{y}_{\text{MLP}}$	$W_{\text{up}} \in \mathbb{R}^{d_{\text{MLP}} \times d}$ $W_{\text{gate}} \in \mathbb{R}^{d_{\text{MLP}} \times d}$ $W_{\text{down}} \in \mathbb{R}^{d \times d_{\text{MLP}}}$	$m_{\text{MLP}} \propto (d_{\text{MLP}} d)^{-1}$ $m_{\text{gate}} \propto d^{-1}$

Table 7: This table summarizes the location of all the forward μ P multipliers used in Falcon-H1 models, the rules to scale multipliers with model size, and the shapes of the relevant parameters or activations required for the scaling. Specific forms of scaling rule in the last column can be straightforwardly derived from μ P Desideratum, see, for example, (Yang et al., 2024c). GQA($\mathbf{Q}, \mathbf{K}, \mathbf{V}$) is Grouped Query Attention; SSM($\mathbf{x}, \mathbf{B}, \mathbf{C}, \mathbf{dt}$) is recurrent sequence transformation described in section 2.2; $\mathcal{N}, \mathcal{N}', \mathcal{N}^{\text{SSM}}, \mathcal{N}^f$ are RMS normalization layers.

unchanged. Specifically, for AdamW optimizer (with $\varepsilon = 0$), such scaling transformation with a parameter p is

$$m \rightarrow p^{-1}m, \quad \sigma \rightarrow p\sigma, \quad \eta \rightarrow p\eta, \quad \lambda \rightarrow p^{-1}\lambda. \quad (17)$$

This symmetry has a practical implication for μ Transfer as it allows to remove in (16) the scaling of either learning rate or forward multiplier (by choosing $p = (d/d_{\text{ref}})^{-a}$ or $p = (d/d_{\text{ref}})^c$). The motivation to remove one of the scalings comes from training infrastructure: multiplier m must be implemented directly in the forward pass of the model, while scaling of η, λ is typically implemented via optimizer parameter groups.

For Falcon-H1 models, we relocate the μ P scaling (16) from learning rate/weight decay to forward multipliers. For example, the original scaling $(a, b, c, e) = (0, \frac{1}{2}, 1, -1)$ for the hidden layers becomes $(a, b, c, e) = (1, -\frac{1}{2}, 0, 0)$. As a result, all the models in the series can be fine-tuned or continuously pretrained with the same learning rate and weight decay parameters.

Our approach. Our core idea is to augment μ Transfer scaling for transferring HPs across model sizes with tuning the μ P multipliers at the base model size.

To motivate the approach, let us rewrite the general scaling (16) for a model parameter $W^{(i)}$ from the point of view of the base model with width d_{ref}

$$m^{(i)} = m_{\text{ref}}^{(i)} \left(\frac{d}{d_{\text{ref}}} \right)^{-a}, \quad \sigma^{(i)} = \sigma_{\text{ref}}^{(i)} \left(\frac{d}{d_{\text{ref}}} \right)^{-b}, \quad \eta^{(i)} = \eta_{\text{ref}}^{(i)} \left(\frac{d}{d_{\text{ref}}} \right)^{-c}, \quad \lambda^{(i)} = \lambda_{\text{ref}}^{(i)} \left(\frac{d}{d_{\text{ref}}} \right)^{-e}. \quad (18)$$

The classical μ Transfer uses the Standard parametrization at the base model size, corresponding to no forward multipliers $m_{\text{ref}}^{(i)} = 1$ and use of global LR η and WD λ for all the layers, $\eta_{\text{ref}}^{(i)} = \eta$

Base model sizes	Forward multipliers	Matrix LR multipliers	Matrix WD multipliers	Vector LR multipliers
$L = 66$	m_{emb}	$2^{2.5}$	W_{emb}	2^2
$d = 1280$	m_{unemb}	2^{-5}	W_{unemb}	2^0
$d_h^{\text{ssm}} = 64$	m_{MLP}	2^{-2}	W_{in}	$2^{-0.5}$
$n_h^{\text{ssm}} = 16$	m_{attn}	2^{-1}	W_{out}	2^{-2}
$d_{\text{state}}^{\text{ssm}} = 128$	m_{SSM}	$2^{-1.5}$	W_{up}	$2^{-0.5}$
$d_h^{\text{attn}} = 64$	m_{gate}	$2^{-0.5}$	W_{gate}	$2^{0.5}$
$n_h^{\text{attn}} = 12$	m_{key}	2^{-2}	W_{down}	$2^{-0.5}$
$d_{\text{MLP}} = 3840$	m_x	2^{-2}		
	m_z	$2^{-1.5}$		
	m_B	$2^{-1.5}$		
	m_C	2^{-1}		
	m_{dt}	$2^{-1.5}$		

Table 8: Base model shapes and its μ P multipliers after tuning procedure described in section C. The notations for forward multipliers, as well as notations for most of the learnable parameters, were introduced in table 7. $W_{\text{in}} = W_{xzBCdt|QKV}$ is a merged weight matrix combining all the initial projections of both SSM and attention mixers, and $W_{\text{out}} = W_{\text{SSM}|attn}$ is similarly merged weight matrix that combines output projections of SSM and attention. We have merged these matrices in the context of μ P multipliers to reduce the total number of multipliers to be tuned. Finally, D and A_{\log} are learnable parameters of Mamba2 sequence transformation.

and $\lambda_{\text{ref}}^{(i)} = \lambda$. However, this strategy is somewhat contradictory: it uses specialized HPs at target size $d > d_{\text{ref}}$ that respect the limiting $d \rightarrow \infty$ feature learning scaling, while using global HPs for the base model as if d_{ref} were tiny and base model was very far from limiting $d \rightarrow \infty$ behavior. This assumption does not look reasonable in practice. For Falcon-H1 series, we have used the base model with $d_{\text{ref}} = 1280$ while the largest 34B model has $d = 5120$, just an $\times 4$ factor from the base model.

With such a premise, we have decided to individually tune HPs of the different layers to respect the limiting $d \rightarrow \infty$ tendency that could be affecting the model at d_{ref} . First, we group model parameters according to their role in the architecture, e.g. LM head or MLP down projection, to tune LR and WD multipliers $\eta_{\text{ref}}^{(i)}/\eta$ and $\lambda_{\text{ref}}^{(i)}/\lambda$. Second, we introduce a minimal set of forward multipliers that produce all possible transformations of the activations throughout the whole forward pass of Falcon-H1 architecture¹⁰. The minimal property of our set removes redundant multipliers that otherwise would be tuned in vain. For example, having both key and query multipliers $\mathbf{K} \rightarrow m_K \mathbf{K}$ and $\mathbf{Q} \rightarrow m_Q \mathbf{Q}$ is redundant because only their scalar product $\mathbf{Q}^\top \mathbf{K} \rightarrow m_Q m_K \mathbf{Q}^\top \mathbf{K}$ is used in attention and having only m_K is sufficient to cover all possible scalings of attention scores. Finally, we don't tune initialization variances $\sigma^{(i)}$ because the symmetry (17) reduces one degree of freedom in the HP set $(m, \sigma, \eta, \lambda)$, and our preliminary sweeps on initialization variances have shown weak dependence of the loss on $\sigma^{(i)}$. This resulted in 35 multipliers to be tuned for

¹⁰Formally, to have a fully complete set of forward multipliers we would also need to add multipliers to SSM inputs $\tilde{\mathbf{B}}$ and $\tilde{\mathbf{dt}}$. However, these multipliers are effectively taken care of by learnable parameters of SSM D and A_{\log} that can adapt to the right scale during training, unlike matrix parameters with non-adaptive norms as described in section 3.2.2.

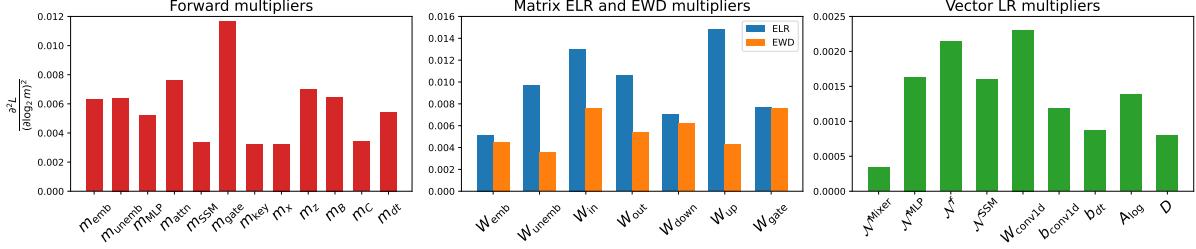


Figure 12: Sensitivity of the loss with respect to all 35 μ P multipliers we have tuned. The multipliers are organized into 4 groups as in table 8, with matrix layers ELR and EWD multipliers are shown in the middle plot side-by-side for comparison. All 4 multiplier groups have different magnitudes of the effect on the loss, and the y-axis limits were adjusted accordingly. ELR multipliers have the strongest impact, followed by forward multipliers, then EWD multipliers, and lastly LR multipliers of vector-like layers. Although the sensitivities w.r.t. to the last group are low, note that the final multiplier values are quite far from those of the matrix layers (see table 8), and, therefore, separating LRs of vector-like and matrix-like layers (as was done in our tuning) still yields significant performance improvement.

Falcon-H1 architecture.

The final values of tuned multipliers, as well as all base model shapes and global LR/WD, can be found in table 8, and the location of forward multipliers is detailed in table 7. We describe our procedure to tune multipliers in section C. It naturally leads to the estimation of the sensitivity of the loss on each multiplier around the optimum $\frac{\partial^2}{(\partial \log_2 m)^2} L$ that we display on figure 12 to provide intuition behind the impact of each of the 35 tuned multipliers.

3.2.4 Other Aspects: Batch Scaling, Rampup, Warmup

In this section, we briefly report interesting observations related to training dynamics that we have incorporated into Falcon-H1 training.

Batch scaling. First, following (Zuo et al., 2024) we have used batch scaling that scales the learning rate if the batch size is changed

$$\eta(b) = \eta_{\text{ref}} \sqrt{\frac{b}{b_{\text{ref}}}}, \quad (19)$$

where square root is a suitable scaling for Adam optimizer (Malladi et al., 2022). We have observed that scaling (19) better preserves optimal learning than no batch scaling at all. However, more careful studies are required for robust transfer of HPs with batch size, taking into account, for example, scaling of parameter norms with batch size to combine it with ELR/EWD picture discussed in section 3.2.2.

Rampup. At the beginning of the training, we have used the batch size rampup that linearly increases batch size over the specified duration, which we set around 50GT for different Falcon-H1 model sizes.

In figure 13 we considered 3 strategies and their impact on the training loss: no rampup, rampup without batch scaling, rampup with batch scaling (19). No rampup was observed to have the worst loss, while also amplifying training instabilities. Shortly after the rampup period, batch scaling

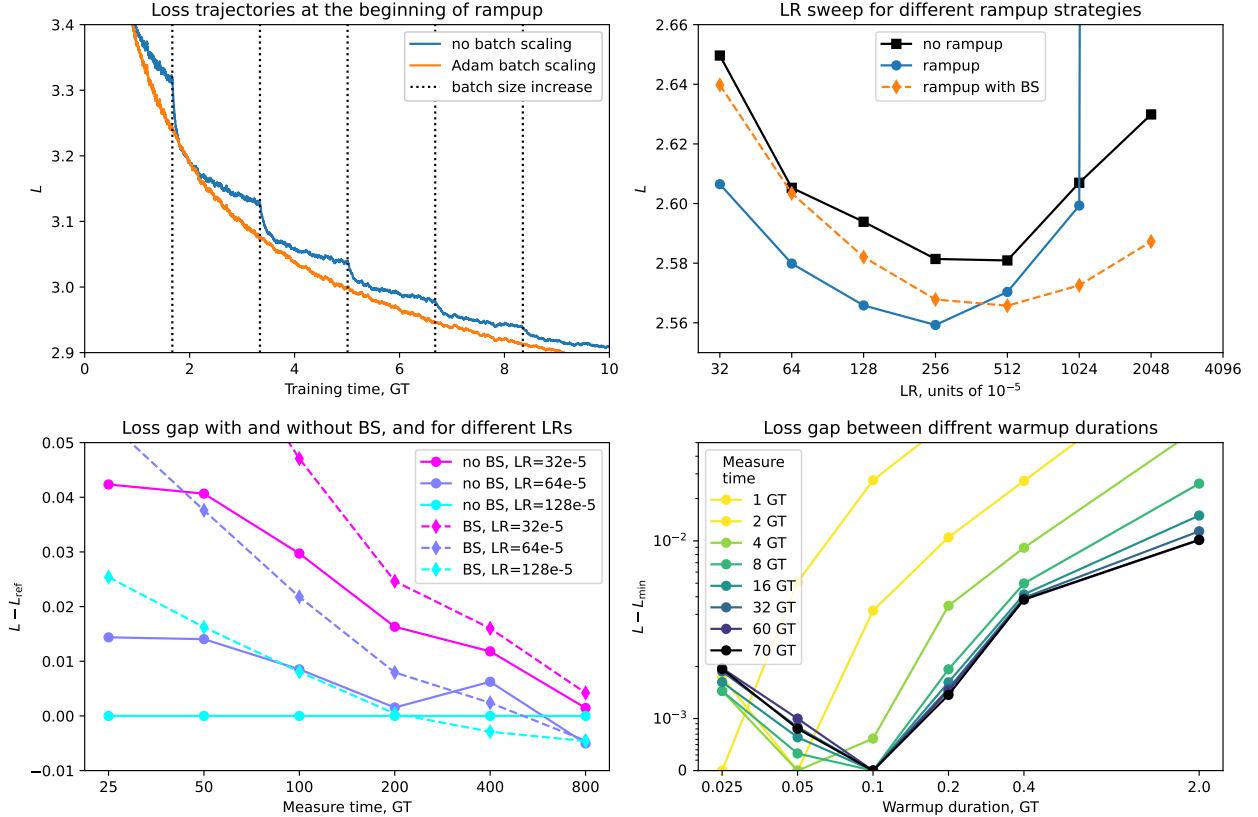


Figure 13: Rampup and warmup related observations on a 1.5B pure Mamba2 model. **(Top Left)** Train loss trajectories during rampup with and without batch scaling. No batch scaling run exhibits loss jumps at the moments of batch size increase. These jumps could be associated with a decrease in the noise level. **(Top Right)** Loss after learning rate decay at 70GT on a LR sweep for different rampup strategies. **(Bottom Left)** Evolution of the loss for runs with and without BS scaling and 3 different learning rates. To take into account the noise level, we measure the loss after LR decay, which is performed at several positions of the training trajectory. To be able to display losses at very distant moments of training, we subtract the loss of the reference run. We see that, for all considered learning rates, the runs with batch scaling have a better trend with longer training duration. **(Bottom Right)** Evolution of the loss gap between runs with different warmup durations. Unlike the previous plot, the loss is measured directly (without LR decay) because warmup duration does not seem to affect noise level, as can be seen from the last predecay measurement at 60GT and after decay measurement at 70GT being very close. We see that at early stages ($\lesssim 16\text{GT}$) loss vs warmup duration curve shifts with measurement time, reflecting that short warmup runs simply had more high LR steps at the beginning of the training. However, at later stages ($\gtrsim 16\text{GT}$) the curve stabilizes, suggesting a well-defined optimal warmup duration with long-lasting effect on the training.

showed worse loss than classical rampup without batch scaling, see figure 13 (top right). However, the gap between them closes at long training durations, and rampup batch scaling eventually outperforms its no batch scaling version, see figure 13 (bottom left). This latter trend is especially interesting because BS and no BS runs have exactly the same hyperparameters after the rampup period. One interpretation would be that batch scaling during rampup directs the training trajectory to a better region of parameter space, and the model continues to learn in this region

for a very long time.

Warmup. Another classical technique used at the beginning of training is to linearly increase LR to the target value in the earliest stages of the training, typically for a few gigatokens. In figure 13 (bottom right) we have tested the impact of warmup duration on the loss at a later training stage. Similarly to the rampup batch scaling, we observe that warmup duration has a long-lasting impact on the loss, with a relatively short optimal duration of 0.1GT.

3.3 Pretraining Infrastructure

Pretraining was conducted using *Mambatron*, our in-house distributed training framework. This framework is an evolution of *Gigatron*, the codebase previously used to train the Falcon model series, and has been specifically optimized to support the unique hybrid architecture of Falcon-H1. Building upon a foundation of standard 3-dimensional parallelism (3DP), we introduced two key innovations tailored for this hybrid design. First, we redesigned Context Parallelism (CP) to efficiently manage and scale the long sequence lengths inherent to the hybrid attention-SSM architecture. Second, we developed a novel strategy termed Mixer Parallelism (MP), which is specifically designed to parallelize computations across the distinct attention and SSM heads. This approach significantly accelerates both training and inference throughput. The Falcon-H1 model series was trained on a large-scale infrastructure comprising 4,096 NVIDIA H100 GPUs. To optimize resource utilization, we employed a dynamic node allocation strategy, enabling the simultaneous training of six Falcon-H1 models. The parallelism configurations for each model are detailed in Table 9.

Models	Batch Size	Context Len. Stage	DP	TP	PP	CP	MP
Falcon-H1-0.5B	4M	4K, 16K	64	1	1	1	✗
Falcon-H1-1.5B (1.5B-Deep)	4M	16K, 32K	256	1	1	1	✗
		131K	64	1	1	4	✗
Falcon-H1-3B	8M	16K, 32K	256	1	1	1	✗
		131K	64	1	1	4	✗
Falcon-H1-7B	8M	16K, 32K	256	2	1	1	✓
		131K	128	2	1	4	✓
		262K	64	2	1	8	✓
Falcon-H1-34B	26M	16K	448	4	2	1	✓
		32K	192	4	2	2	✓
		131K	48	4	2	8	✓
		262K	24	4	2	16	✓

Table 9: 5D Parallelism Configurations for Falcon-H1’s Training

3.3.1 Scaling Dynamics of Data Parallelism

While Data Parallelism (DP) is a fundamental technique for distributed training, its throughput scaling is not without limits. When scaling the number of DP workers (N_{DP}) while keeping the global batch size (B_g) constant, throughput gains diminish significantly once communication overhead outweighs computation. This occurs because, to maintain a fixed B_g , the number of gradient accumulation steps (K) per optimizer update must decrease inversely with N_{DP} , where $B_g = N_{DP} \cdot K \cdot B_\mu$ for a micro-batch size B_μ .

To formally analyze this behavior, we model the time for a single optimizer step as:

$$T_{\text{step}}(N_{\text{DP}}) \approx K \cdot t_{\mu} + t_{\text{sync}}(N_{\text{DP}}) \quad (1)$$

Here, t_{μ} is the constant time for a forward-backward pass on a single micro-batch, and t_{sync} is the latency of the gradient all-reduce operation, which can grow with N_{DP} due to network complexity. As N_{DP} increases, the computation term ($K \cdot t_{\mu}$) shrinks, while the communication term (t_{sync}) becomes the dominant component. Consequently, the overall throughput, given by B_g/T_{step} , deviates from ideal linear scaling. Substituting the terms, we get:

$$\text{Throughput}(N_{\text{DP}}) \approx \frac{B_g}{\frac{B_g}{N_{\text{DP}} B_{\mu}} \cdot t_{\mu} + t_{\text{sync}}(N_{\text{DP}})} \quad (2)$$

Theoretically, linear scaling could be maintained by increasing B_g proportionally with N_{DP} , keeping K constant. However, significantly changing the global batch size from the value determined during hyperparameter tuning can destabilize training dynamics, thus impacting model convergence and final performance. Therefore, our strategy involves a pragmatic trade-off. We cap the DP size at a value where communication overhead remains manageable, and increase the global batch size only up to a critical point that balances high throughput with stable model convergence. This ensures efficient hardware utilization without compromising the integrity of the training regime.

3.3.2 Mixer Parallelism (MP)

During pre-training, we developed a novel distributed training paradigm to boost the efficiency of training our largest models. Leveraging the parallel architecture of our decoder layers, where attention and Mamba layers are executed sequentially, we partitioned the Tensor Parallel (TP) world into two distinct groups: one dedicated to Mamba operations, the other to attention operations. This design allows these computations to run concurrently, followed by an all-reduce operation to synchronize their outputs. We refer to this strategy as *Mixer Parallelism* (MP). It significantly improves training throughput by optimizing both computational speed and memory efficiency while also boosting inference efficiency particularly for scenarios involving small batch sizes and sequence lengths.

Mixer Parallelism can be implemented using two distinct approaches. In *naive* Mixer Parallelism, predefined TP groups are assigned exclusively to a single mixer type (Attention or Mamba). In contrast, *interleaved* Mixer Parallelism distributes different mixer types across TP groups in an alternating fashion, achieving a more balanced distribution of computational overhead from the slower mixer layers. Figure 14 shows how different Mixer Parallelisms are implemented.

Training Efficiency with MP. To evaluate the efficacy of Mixer Parallelism for the model’s training, we conducted experiments using a 2B hybrid model, configured with a data parallelism of 4, a tensor parallelism of 4, and a context length of 2048. We measured the training throughput for a baseline without MP against both the naive and interleaved variants.

The results, summarized in Table 10, demonstrate the clear superiority of the interleaved MP strategy. By effectively balancing the computational load, interleaved Mixer Parallelism achieves a substantial 1.43x speedup over the baseline.

Inference Efficiency with MP. To evaluate the efficacy of Mixer Parallelism for inference scenarios, we conducted a comprehensive throughput analysis across a 3B and 7B parameters models. Based on its superior load-balancing properties, we focused exclusively on the *interleaved* MP variant for these experiments. Our evaluation was performed on a single node equipped with two NVIDIA H100 GPUs, configured with a Tensor Parallelism size of 2. We systematically varied two

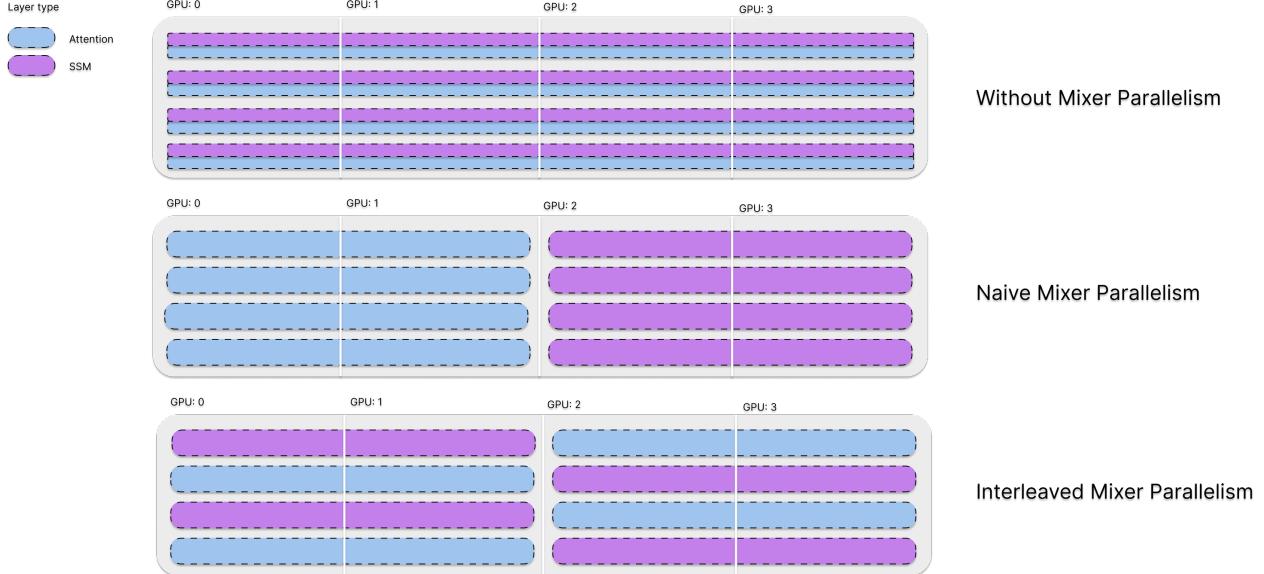


Figure 14: Diagram illustrating the Mixer Parallelism (MP) strategies. Each row represents a full decoder layer. Output projection and all reduce operations are applied at the end of each layer. (*Top*) Without MP, all GPUs compute both mixer types sequentially. (*Middle*) Naive MP statically assigns GPUs to one mixer type for all layers. (*Bottom*) Interleaved MP alternates assignments per layer, achieving better load balancing.

MP Variant	Throughput (Gtok/hr)	Speedup (ratio)
None (Baseline)	0.2339	1.00
Naive MP	0.2640	1.13
Interleaved MP	0.3343	1.43

Table 10: Training throughput and speedup comparison for different Mixer Parallelism variants.

key parameters to simulate diverse workloads: the batch size (from 1 to 128) and the number of generated output tokens (from 4096 to 32768). A constant prefill size of 8 tokens was used for all runs.

The implementation was integrated into a custom fork of the vLLM library (Kwon et al., 2023). We report in figure 15 throughput results of this experiment on all our model sizes. The dashed curves represent configurations using Mixer Parallelism, while matching colors indicate experiments with the same number of generated tokens.

The results show that Mixer parallelism significantly accelerates inference for low-latency scenarios characterized by small batch sizes and short generated sequences. Though this advantage diminishes and reverses for larger batches and longer generation sequences. This observation is consistent across all model sizes. We provide the community an easy way to test the implementation of Mixer parallelism in a vLLM fork ¹¹.

¹¹<https://tiiuae.github.io/Falcon-H1/deployment/>

VLLM Benchmark: Throughput vs. Batch Size

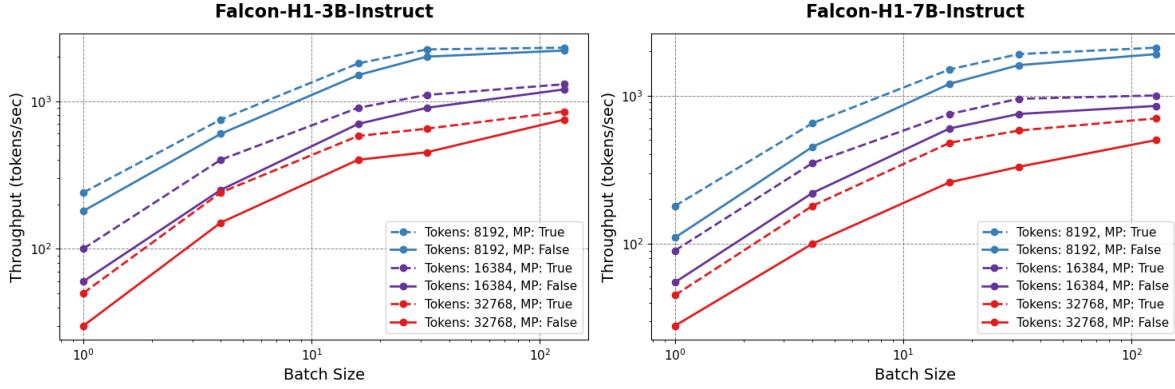


Figure 15: Throughput comparison for Mixer Parallelism across different model sizes (3B and 7B).

3.3.3 Context Parallelism (CP)

To keep GPU memory flat while scaling to long contexts, the model shards each sequence horizontally across the `cp_world` of J devices and lets every device work on one contiguous *chunk* of length Q .

Self-Attention chunks. For the attention blocks we reuse **RingAttention** (Liu et al., 2023a): each rank holds its local query–key–value slice and circulates K/V tensors around the ring so that the full score matrix is produced without ever materialising the whole sequence on one GPU. Memory is therefore $O(Q)$ per rank instead of $O(T)$ for the full length T .

SSM (Mamba-2) chunks. For the SSM layers we follow the chunk-wise **state-passing schedule** described in Section 8.2 of Mamba-2 (Dao & Gu, 2024) and illustrated in their Figure 7 (right). In words:

1. **Initial state.** Rank j waits for the final hidden state produced by rank $j - 1$ (rank 0 starts from zeros or an optional user-supplied context state).
2. **Local work.** Using that state and its own input slice \mathbf{x}_j , the rank runs the SSM kernel on its private chunk and produces the chunk’s output tokens \mathbf{y}_j and the *next* hidden state h_{j+1} . The computation is entirely local and no cross-rank dependency exists.
3. **State hand-off.** If a following rank exists, h_{j+1} is sent asynchronously to rank $j+1$; otherwise the pipeline finishes.

The only communication is a single tensor of shape $[B \times H \times d_{\text{state}}]$ per boundary—where B is the micro-batch size on the rank, H the number of SSM heads, and d_{state} the width of each state vector—so bandwidth remains *linear* in the number of GPUs.

CausalConv1D chunks. The preceding causal convolution stage is split in the same fashion: each rank gets the last $k-1$ timesteps from its left neighbour, performs its local depth-wise convolution, and forwards the $k-1$ boundary activations to the next rank.

4. Post-training

In this section, we detail the two-stage post-training procedure for our Falcon-H1 models, which consists of Supervised Fine-Tuning (SFT) followed by Offline Reinforcement Learning.

4.1 Post-training Data

Compared to the previous Falcon model series, we have significantly expanded the scope and quality of the supervised fine-tuning data. This expansion spans multiple domains and includes millions of high-quality samples. Notably, our efforts focus on improving the model’s capabilities in complex reasoning, mathematical and scientific problem-solving, instruction-following, and function calling. Some of the key improvements include:

- **Mathematical Problem-Solving:** We enhanced both the breadth and difficulty of mathematical problems by curating and rewriting high-quality solutions. Inspired by OpenMathInstruct-2 (Toshniwal et al., 2024) and AceMath (Liu et al., 2024e), we synthesized math problems across diverse sub-domains, carefully filtering out incorrect solutions. Additional synthetic examples were generated using data from the pretraining math corpus, ensuring high correctness and varying difficulty levels.
- **Scientific Problem-Solving:** We improved the model’s performance in scientific reasoning, particularly across STEM domains. Problem-solution pairs were extracted or synthesized from existing pretraining corpora, with subsequent refinements to ensure consistent formatting and data quality.
- **Conversational and Instruction-Following:** We enhanced the model’s conversational abilities and instruction-following by improving personalization, stylistic variety and multi-round long conversation capability.

The post-training corpus is based on both license-permissive open datasets and proprietary data sources. From open data, we partially adopted and refined the datasets such as OpenMathInstruct-2 (Toshniwal et al., 2024), Tulu3 (Lambert et al., 2024), Smoltalk (Ben Allal et al., 2024), and hermes-function-calling-v1 ("interstellarninja"). In many cases, questions and solutions were further optimized to improve clarity and correctness. Proprietary data includes high-quality extracted, refined, and synthetic sample pairs from internal high-quality sources on both STEM and non-STEM domains, along with human-annotated examples targeting specific skills. This hybrid strategy ensures diverse, accurate, and skill-aligned training data. We find that data quality and structure have a much greater impact on post-training performance than data volume alone. We carefully decontaminated the post-training data against popular benchmarks to prevent unintended data leakage and ensure fair evaluation.

4.2 Supervised Fine-Tuning (SFT)

We performed extensive data mixture and duration testing during SFT to balance the model’s performance across different domains. Compared to pretraining, SFT is more sensitive to data mixture. The resulting checkpoints that we released offer strong general performance but can be further adjusted for specific use cases.

Our SFT process is divided into two distinct stages: a primary stage with a 16k context length, followed by a long-context stage extending to 128k. The initial 16k stage was conducted for 3 GT,

succeeded by an additional 3GT for the long-context stage. During the total 6GT of SFT, we repeated different data sources for different amount of times (epochs) depending on their volume and weight in the mixture. The most repeated data source was Tulu3, with around 50% weight in the data mixture and around 3.5 epochs over the SFT duration. The other data sources were repeated for fewer than 2 epochs.

For the main 16k stage, we employed a Warmup-Stable-Decay (WSD) learning rate schedule. The decay phase followed an exponential profile, reducing the learning rate by a factor of eight to a minimum value of $\eta_{\min} = \eta/8$. The subsequent 128k long-context stage proceeded with a constant learning rate equal to this minimal value, η_{\min} . The 128k stage was omitted for our smallest 0.5B parameter model due to the inherent limitations in processing long sequences by models of that scale. The key hyperparameters for our SFT setup are summarized in Table 11.

Hyperparameter / Setting	Value / Details
Sequence Length	16k for the main stage; 128k for long context stage
Batch Size (b)	1 million tokens (MT)
Learning Rate (η)	128×10^{-6}
AdamW Parameters	$\beta_1 = 0.9$; $\beta_2 = 0.95$; no weight decay
Learning Rate Schedule	WSD: 50MT warmup, 1.5GT stable, 1.5GT decay
LR Decay	Exponential schedule from η to $\eta_{\min} = \eta/8$
Long Context Stage	+3GT with constant learning rate η_{\min}
Epochs per Data Source	$\lesssim 3.5$

Table 11: Hyperparameters for the Supervised Fine-Tuning (SFT) stage.

Finally, we note that we have used slightly different batch sizes, in the range of 0.25MT to 4MT, for different model sizes to meet the desired GPU allocation on the cluster and the job run time¹². Then, for a given batch size b , we have used square root batch scaling (19), taking the values from table 11 as reference values $b_{\text{ref}}, \eta_{\text{ref}}$. With such scaling, we have observed minimal impact of batch size on the final performance of the SFT model.

4.3 Direct Preference Optimization (DPO)

For the DPO stage, we utilized a fork of the AllenAI open-instruct repository¹³. Similar to the SFT stage, our data mixture was built upon Tulu3 (Lambert et al., 2024), supplemented by several other open-source and in-house preference datasets. We employed the standard DPO loss function (`dpo_loss_type: dpo_norm`). The hyperparameters for the DPO stage are outlined in Table 12.

Hyperparameter / Setting	Value / Details
Batch Size	256
Learning Rate	5×10^{-6}
Learning Rate Schedule	Linear decay to zero over 2 epochs, with a warmup ratio of 0.1
DPO Loss Parameter (β)	5
AdamW Parameters	PyTorch default ($\beta_1 = 0.9$, $\beta_2 = 0.999$)

Table 12: Hyperparameters for the Direct Preference Optimization (DPO) stage.

¹²For instance, the 128k long-context stage, being computationally intensive due to context parallelism, necessitated larger batch sizes to enable higher degrees of data parallelism (DP) for quicker job execution.

¹³<https://github.com/allenai/open-instruct>

An important aspect of our DPO strategy was the stopping criterion. Instead of training for the full two epochs, at which point the learning rate schedule concludes, we found that stopping at approximately one epoch yielded superior results. This approach was empirically determined to be more effective than either completing the full two-epoch schedule or using a linear scheduler that terminates after a single epoch.

5. Evaluation

To provide a rigorous and reproducible comparison, we benchmarked the Falcon-H1 series against a comprehensive suite of leading models, including Qwen3 (Yang et al., 2025), Qwen2.5 (Yang et al., 2024b), Gemma3 (Team et al., 2025), Falcon3 (Team, 2024), Llama3 (Grattafiori et al., 2024) and Llama4¹⁴. Our evaluation pipeline is built upon a foundation of established open-source frameworks: lm-evaluation-harness (Gao et al., 2024), evalchemy (Raof et al., 2025), evalplus (Liu et al., 2023b) and helmet (Yen et al., 2025).

Standardization and Reproducibility. Our methodology was standardized across all models to ensure a fair comparison. All evaluations were conducted within the same Docker environment to eliminate system-level variance. For the Qwen3 series, we disabled the "thinking mode" on all benchmarks to align its inference process with that of other models.

Framework-Specific Settings. To ensure stability, all evalchemy evaluations were pinned to a specific commit hash (f735e77). For relevant mathematical benchmarks within this framework, we standardized the number of generation turns to 16 and applied an identical system prompt across all models. Furthermore, all final math results were post-processed using Math-Verify (Kydliček) for consistent verification. For all other frameworks, we adhered to their default settings to maintain comparability with established results in the literature.

5.1 Base Models

For base models, we report in Table 13 the benchmarks and settings used for the evaluations. Basically, we check the model’s capabilities in five main domains: General, Math, Science, Code, and Multilingual. Noting that some benchmarks contains cross-domain tasks and do not fall neatly into a single category.

Falcon-H1-0.5B-Base. The results presented in Table 14 establish Falcon-H1-0.5B as a new benchmark for sub-1B parameter base models. Despite possessing the smallest footprint in the comparison, it leads on every Math, Science, and Code benchmark, often by substantial margins (e.g., GSM8k: 60.20 vs. 50.04; MATH-lvl5: 15.18 vs. 9.29). The model further confirms its strength in structured, knowledge-intensive reasoning by securing the highest scores on BBH and MMLU. Its performance on commonsense benchmarks like HellaSwag and Winogrande is competitive but surpassed by larger models in the 1B-1.6B class, suggesting a design that prioritizes deep reasoning capabilities over broad world knowledge. Ultimately, Falcon-H1-0.5B’s results demonstrate that with targeted design choices, even a 0.5B model can achieve highly competitive, non-trivial performance on complex tasks.

Falcon-H1-1.5B-Base and Falcon-H1-1.5B-Deep-Base. The evaluation results for our 1.5B-scale models, presented in Table 15, highlight the significant benefits of increased model depth. As discussed in Section 2.3.2, the Falcon-H1-1.5B-Deep variant, which features more layers while maintaining a similar parameter count, establishes itself as the clear state-of-the-art model in its class. Its performance is highly competitive, often rivaling that of current leading 7B to 10B models

¹⁴<https://huggingface.co/meta-llama/Llama-4-Scout-17B-16E>

Benchmark	Settings	Framework
General		
BBH (Suzgun et al., 2022)	logprobs, 3-shot	lm-eval-harness
ARC-C (Clark et al., 2018)	logprobs, 25-shot	lm-eval-harness
HellaSwag (Zellers et al., 2019)	logprobs, 10-shot	lm-eval-harness
Winogrande (Sakaguchi et al., 2021)	logprobs, 5-shot	lm-eval-harness
MMLU (Hendrycks et al., 2020)	logprobs, 5-shot	lm-eval-harness
Math		
GSM8k (Cobbe et al., 2021b)	strict match, 5-shot	lm-eval-harness
MATH lvl15 (Hendrycks et al., 2021a)	math verify, logprobs, 4-shot	lm-eval-harness
Science		
GPQA (Rein et al., 2023)	logprobs, 5-shot	lm-eval-harness
MMLU-Pro (Wang et al., 2024)	logprobs, 5-shot	lm-eval-harness
MMLU-stem (Hendrycks et al., 2020)	logprobs, 5-shot	lm-eval-harness
Code		
HumanEval (Chen et al., 2021)	pass@1	evalplus
HumanEval+ (Liu et al., 2023b)	pass@1	evalplus
MBPP (Austin et al., 2021)	pass@1	evalplus
MBPP+ (Liu et al., 2023b)	pass@1	evalplus
Multilingual		
Multi-Hellaswag (Dac Lai et al., 2023)	logprobs, 0-shot	lm-eval-harness
MGSM (Shi et al., 2022)	flexible extract, 8-shot native CoT	lm-eval-harness
Multi-MMLU (Dac Lai et al., 2023)	logprobs, 5-shot	lm-eval-harness

Table 13: Evaluation settings and benchmark sources for base models.

Tasks	Falcon-H1-0.5B	Qwen3-0.6B	Qwen2.5-0.5B	Gemma3-1B	Llama3.2-1.2B	Falcon3-1.6B
General						
BBH	40.22	<u>36.07</u>	32.62	30.26	30.72	35.24
MMLU	55.04	<u>52.64</u>	47.61	26.33	32.39	45.14
ARC-C	46.93	44.80	35.32	39.33	39.42	47.87
HellaSwag	56.30	53.51	51.79	<u>62.94</u>	65.73	62.30
Winogrande	59.43	60.54	56.83	<u>62.59</u>	62.75	61.17
Math						
GSM8k	60.20	<u>50.04</u>	34.80	2.20	7.05	34.95
MATH lvl15	15.18	<u>9.29</u>	4.23	1.21	0.98	3.40
Science						
GPQA	29.70	<u>29.11</u>	27.94	24.66	23.57	27.85
MMLU-Pro	30.04	<u>22.99</u>	18.98	11.31	11.80	16.11
MMLU-stem	57.12	<u>50.11</u>	43.74	27.59	30.19	40.06
Code						
HumanEval	35.98	<u>31.71</u>	29.27	6.71	18.90	10.37
HumanEval+	31.10	<u>27.44</u>	25.00	5.49	16.46	9.15
MBPP	52.12	<u>51.06</u>	40.74	12.70	35.98	12.43
MBPP+	43.39	<u>42.33</u>	34.66	9.52	29.89	9.52

Table 14: Performance of the 0.5B+ **Base** models.

such as Qwen2.5-7B (Yang et al., 2024b) and Falcon3-7B/10B (Team, 2024). This architectural choice proves particularly impactful for reasoning-intensive tasks; the deep model substantially outperforms its shallower counterpart and other peers on Science and Math benchmarks like MATH-lvl5 (+4.38 points) and MMLU-Pro (+5.54 points).

This strong performance continues into the Code and Multilingual domains, where Falcon-H1-1.5B-Deep consistently leads or is highly competitive. Although it is surpassed by Qwen3-1.7B (Yang et al., 2025) on specific benchmarks like HumanEval and GSM8k, its dominant overall profile underscores the efficacy of our deep architecture. The result also suggests that while benchmark categories are distinct, the underlying skills required to solve them are often shared. A strong, generalizable reasoning capability, which our deeper model demonstrates, appears to be a critical polyvalent skill that confers benefits across multiple domains.

Tasks	Falcon-H1-1.5B-Deep	Falcon-H1-1.5B	Qwen3-1.7B	Qwen2.5-1.5B	Gemma3-1B	Llama3.2-1.2B	Falcon3-1.6B
General							
BBH	52.37	<u>46.57</u>	43.05	40.55	30.26	30.72	35.24
MMLU	66.29	61.81	<u>62.46</u>	61.13	26.33	32.39	45.14
ARC-C	55.89	53.24	<u>55.72</u>	54.27	39.33	39.42	47.87
HellaSwag	69.72	66.76	67.09	<u>67.86</u>	62.94	65.73	62.30
Winogrande	67.09	65.59	<u>66.30</u>	64.56	62.59	62.75	61.17
Math							
GSM8k	<u>68.69</u>	52.01	70.74	63.00	2.20	7.05	34.95
MATH lvl5	24.77	<u>20.39</u>	16.39	8.84	1.21	0.98	3.40
Science							
GPQA	32.80	29.11	<u>29.45</u>	28.36	24.66	23.57	27.85
MMLU-Pro	41.07	<u>35.53</u>	33.81	28.72	11.31	11.80	16.11
MMLU-stem	67.43	<u>63.37</u>	61.53	54.93	27.59	30.19	40.06
Code							
HumanEval	<u>52.44</u>	50.00	67.68	35.37	6.71	18.90	10.37
HumanEval+	<u>46.34</u>	42.68	60.98	29.27	5.49	16.46	9.15
MBPP	70.90	65.08	<u>67.72</u>	60.05	12.70	35.98	12.43
MBPP+	60.32	55.03	<u>58.99</u>	49.47	9.52	29.89	9.52
Multilingual							
Multi-Hellaswag	50.36	<u>46.62</u>	46.47	42.89	46.14	41.61	31.42
Multi-MMLU	52.00	46.51	-	<u>48.09</u>	26.50	28.22	31.56
MGSM	60.33	<u>50.80</u>	-	45.13	-	4.73	9.40

Table 15: Performance of the 1B+ Base models.

Falcon-H1-3B-Base. At the 3B-4B parameter scale, Falcon-H1-3B showcases exceptional training efficiency. Despite being trained on only 2.5T tokens—an order of magnitude less data than the 36T tokens reportedly used for Qwen3 (Yang et al., 2025)—our model delivers a highly competitive performance profile, as shown in Table 16. While Qwen3-4B’s extensive training confers an advantage on many general, science, and coding benchmarks, Falcon-H1-3B’s resource-efficient approach enables it to achieve state-of-the-art capabilities in tasks like advanced mathematical reasoning. It secures leading scores on the challenging MATH-lvl5 benchmark (25.83) and the multilingual MGSM (64.00). Crucially, its performance had not yet plateaued at the conclusion of training, suggesting that its already strong results represent a conservative estimate of its full potential and that our targeted data strategy can achieve specialized excellence with significantly less computational cost.

Tasks	Falcon-H1-3B	Qwen3-4B	Qwen2.5-3B	Gemma3-4B	Llama3.2-3B	Falcon3-3B
General						
BBH	<u>53.17</u>	56.88	46.40	40.41	39.45	44.02
MMLU	<u>68.39</u>	72.92	65.56	59.41	55.94	56.77
ARC-C	<u>61.35</u>	64.33	56.57	58.36	51.02	55.12
HellaSwag	73.85	75.74	74.60	77.62	<u>76.39</u>	67.13
Winogrande	68.11	<u>72.30</u>	71.03	72.77	72.22	65.11
Math						
GSM8k	68.31	81.65	<u>74.60</u>	37.60	27.82	64.67
MATH lvl5	25.83	<u>24.47</u>	16.09	6.95	1.74	11.56
Science						
GPQA	<u>32.63</u>	34.90	28.44	29.78	28.78	29.78
MMLU-Pro	<u>40.58</u>	46.18	32.12	28.34	25.08	29.03
MMLU-stem	<u>69.55</u>	75.58	62.23	51.70	47.67	55.34
Code						
HumanEval	<u>59.15</u>	74.39	42.68	33.54	29.27	36.59
HumanEval+	<u>53.66</u>	68.90	35.37	28.05	26.22	31.71
MBPP	<u>71.43</u>	74.60	59.52	60.05	48.94	51.85
MBPP+	<u>57.94</u>	63.76	50.53	51.32	39.42	42.06
Multilingual						
Multi-Hellaswag	<u>55.15</u>	56.69	54.71	61.03	53.89	36.30
Multi-MMLU	<u>54.78</u>	64.91	55.13	52.57	45.45	38.31
MGSM	64.00	-	<u>59.93</u>	-	20.33	31.80

Table 16: Performance of the 3B+ Base models.

Falcon-H1-7B-Base. At the 7B+ parameter scale, as detailed in Table 17, Falcon-H1-7B establishes a new state-of-the-art benchmark, particularly in complex, knowledge-intensive domains. It demonstrates clear leadership in advanced reasoning by securing top scores on MMLU (77.38), MATH-lvl5 (34.67), and the challenging GPQA science benchmark (36.58). Furthermore, it excels in code generation tasks, leading on MBPP and MBPP+, and also achieves the best performance in multilingual mathematics on MGSM. This consistent top-tier performance underscores its exceptional capabilities in structured reasoning and specialized knowledge application. The competitive landscape at this scale also reveals distinct strengths among other models. The Qwen models, for instance, exhibit a strong aptitude for coding, with Qwen3-8B leading on HumanEval benchmarks. Meanwhile, Gemma3-12B, despite its larger size, primarily excels on commonsense reasoning tasks like HellaSwag and Winogrande. This distribution of results highlights a key finding: while larger models may show advantages in general commonsense tasks, the architectural and data choices of Falcon-H1-7B make it a superior model for high-value, reasoning-focused applications in science, math, and code, while also maintaining an advantage over its counterpart, Qwen3-8B, on general tasks requiring more world knowledge.

Falcon-H1-34B-Base. In the 34B parameter class, we evaluate Falcon-H1-34B against a highly competitive field that includes models up to the 70B scale, as well as the Llama4-scout-17B MoE model (109B). Qwen3-32B is not included since no base model checkpoint was released. As shown in Table 18, despite this challenging comparison, Falcon-H1-34B demonstrates state-of-the-art, parameter-efficient performance, distinguishing itself in specialized, complex domains, even when compared against significantly larger models.

It secures the top position on challenging reasoning benchmarks such as BBH (69.36) and

Tasks	Falcon-H1-7B	Qwen3-8B	Qwen2.5-7B	Gemma3-12B	Llama3.1-8B	Falcon3-7B	Falcon3-10B
General							
BBH	60.61	58.44	53.72	54.33	46.52	50.88	59.30
MMLU	77.38	<u>76.63</u>	74.17	74.23	65.17	69.98	73.22
ARC-C	65.19	67.75	63.91	<u>67.58</u>	57.68	62.71	67.49
HellaSwag	81.26	79.60	80.20	84.22	<u>81.97</u>	76.69	79.64
Winogrande	<u>79.01</u>	76.80	76.01	79.79	77.11	73.64	<u>79.01</u>
Math							
GSM8k	73.46	<u>83.02</u>	83.09	71.19	49.51	76.95	82.11
MATH lvl5	34.67	<u>28.85</u>	22.58	17.22	6.57	20.09	25.38
Science							
GPQA	36.58	<u>35.65</u>	32.30	34.56	31.46	35.07	35.40
MMLU-Pro	48.38	<u>48.25</u>	43.55	42.72	32.71	39.23	42.45
MMLU-stem	<u>77.20</u>	78.53	71.04	68.51	55.72	67.71	70.85
Code							
HumanEval	<u>67.68</u>	87.80	57.32	45.12	39.02	50.00	51.83
HumanEval+	<u>63.41</u>	82.32	48.78	36.59	31.71	43.29	44.51
MBPP	78.57	75.13	<u>76.72</u>	73.02	61.38	67.99	73.54
MBPP+	67.20	<u>64.02</u>	63.49	59.79	51.32	57.14	61.38
Multilingual							
Multi-Hellaswag	<u>65.16</u>	62.13	58.74	70.62	61.72	46.58	50.91
Multi-MMLU	67.55	68.71	64.07	67.14	53.58	-	53.17
MGSM	74.53	67.87	<u>71.07</u>	-	41.53	52.20	59.00

Table 17: Performance of the 7B+ Base models.

MATH-lvl5 (40.71). Furthermore, its leadership on the GPQA benchmark (42.70), all code generation tasks (HumanEval, HumanEval+), and multilingual mathematics (MGSM) underscores its exceptional and well-rounded capabilities in both reasoning and knowledge-intensive applications. The performance of other models at this scale reveals interesting trade-offs. The larger Qwen2.5-72B and Llama3.1-70B models show an advantage on general knowledge and commonsense reasoning tasks like MMLU, ARC-C, and HellaSwag. However, Falcon-H1-34B remains highly competitive, often securing the second-best score. This pattern reinforces a key finding from our smaller models: while increased scale can confer advantages on broad-knowledge tasks, the specialized architecture and data strategy of Falcon-H1 enable it to deliver superior, parameter-efficient performance on complex reasoning and code generation tasks. This positions Falcon-H1-34B as a leading model in its class and establishes it as a far more cost-efficient alternative to 70B+ models for developers seeking a powerful base for fine-tuning or specialized reasoning applications.

Overall Remarks of Falcon-H1 Base models. A key finding across our suite of base models is the achievement of state-of-the-art performance with remarkable training efficiency. Despite being trained on a modest 2.5T to 18T tokens, the Falcon-H1 series consistently challenges and often surpasses competitor models trained on substantially larger datasets. Our models establish clear leadership in complex, reasoning-intensive domains such as mathematics, science, and code generation across all evaluated scales. Notably, the performance of these models, particularly the smaller variants, had not yet plateaued at the conclusion of pretraining, indicating significant headroom for further optimizations with extended training. The resulting models exhibit a well-balanced performance profile, positioning them as powerful and efficient foundations for fine-tuning on specialized downstream applications.

Tasks	Falcon-H1-34B	Qwen2.5-72B	Qwen2.5-32B	Gemma3-27B	Llama3.1-70B	Llama4-scout
General						
BBH	69.36	<u>67.77</u>	67.45	61.60	62.78	61.71
MMLU	<u>83.46</u>	85.96	83.18	78.32	78.49	77.98
ARC-C	<u>71.25</u>	72.44	70.48	70.31	69.20	62.97
HellaSwag	85.68	<u>87.57</u>	85.13	86.19	87.78	84.01
Winogrande	82.72	<u>83.74</u>	82.32	82.40	85.32	78.93
Math						
GSM8k	76.50	<u>89.76</u>	90.14	81.35	80.52	83.24
MATH lv15	40.71	<u>38.14</u>	36.40	25.38	18.81	27.19
Science						
GPQA	42.70	<u>42.28</u>	39.68	35.82	36.49	35.99
MMLU-Pro	57.18	60.22	<u>58.05</u>	49.64	47.07	50.16
MMLU-stem	<u>83.82</u>	84.81	82.81	76.59	70.35	72.57
Code						
HumanEval	70.12	59.15	<u>59.76</u>	48.78	57.32	57.32
HumanEval+	64.63	51.22	<u>51.83</u>	40.85	50.61	48.78
MBPP	<u>83.33</u>	87.04	83.07	76.19	78.84	77.78
MBPP+	70.37	70.63	68.78	61.64	66.67	64.29
Multilingual						
Multi-Hellaswag	72.62	71.20	-	<u>74.01</u>	74.65	72.02
Multi-MMLU	<u>76.76</u>	78.54	-	72.41	71.10	72.38
MGSM	82.40	<u>82.20</u>	-	-	70.73	75.80

Table 18: Performance of the 34B+ **Base** models.

For the full evaluation results on the model’s multilingual capabilities, please refer to the dedicated Appendix section D.1, where we report each model’s performance by language.

5.2 Instruct Models

For instruction-tuned models, we report in Table 19 the benchmarks and settings used for the evaluations. For these models, we expanded the evaluation scope to cover a broader range of domains and tasks. While we provide a general categorization for clarity, it is important to note that some benchmarks are cross-domain and do not fall neatly into a single category. Falcon-H1-0.5B-Instruct model was evaluated exclusively on non-multilingual tasks, as it was trained only on English data. Long context benchmarks are only reported on 34B scale models for simplicity. For both the multilingual and long-context evaluations, this section reports the average scores for each task category. A detailed, per-language and per-task breakdown of these results is available in Appendix D.2 and Appendix D.3, respectively.

Falcon-H1-0.5B-Instruct. At the sub-1B parameter scale, the Falcon-H1-0.5B-Instruct model sets a new state-of-the-art benchmark, demonstrating a clear and consistent advantage in complex, reasoning-intensive domains as shown in Table 20. The model’s superiority is most pronounced in Math, where it achieves a sweeping dominance across all five benchmarks. Its performance on GSM8k (68.39) and MATH-500 (58.40) is particularly notable, substantially outperforming all competitors. This strength in structured reasoning extends to the Science and Code categories, where it secures top scores on the majority of tasks, including MMLU-Pro, HumanEval, and CRUXE-val. Furthermore, its leading performance on IFEval (72.07) confirms its exceptional ability to adhere to complex instructions. However, the competitive landscape is not uniform. Competitors

Benchmark	Settings	Framework
General		
BBH (Suzgun et al., 2022)	logprobs, 3-shot	lm-eval-harness
ARC-C (Clark et al., 2018)	logprobs, 0-shot	lm-eval-harness
TruthfulQA (Lin et al., 2021)	logprobs, 0-shot	lm-eval-harness
HellaSwag (Zellers et al., 2019)	logprobs, 0-shot	lm-eval-harness
MMLU (Hendrycks et al., 2020)	logprobs, 5-shot	lm-eval-harness
Math		
GSM8k (Cobbe et al., 2021b)	strict match, 5-shot	lm-eval-harness
MATH-500 (Lightman et al., 2023)	accuracy	evalchemy
AMC-23	average accuracy, 16 repetitions	evalchemy
AIME-24 (AIME)	average accuracy, 16 repetitions	evalchemy
AIME-25 (AIME)	average accuracy, 16 repetitions	evalchemy
Science		
GPQA (Rein et al., 2023)	logprobs, 5-shot	lm-eval-harness
GPQA_Diamond (Rein et al., 2023)	average accuracy, 3 repetitions	evalchemy
MMLU-Pro (Wang et al., 2024)	logprobs, 5-shot	lm-eval-harness
MMLU-stem (Hendrycks et al., 2020)	logprobs, 5-shot	lm-eval-harness
Code		
HumanEval (Chen et al., 2021)	pass@1	evalplus
HumanEval+ (Liu et al., 2023b)	pass@1	evalplus
MBPP (Austin et al., 2021)	pass@1	evalplus
MBPP+ (Liu et al., 2023b)	pass@1	evalplus
LiveCodeBench (Jain et al., 2024)	accuracy	evalchemy
CRUXEval (Gu et al., 2024)	pass@1, input & output average	evalchemy
Instruction Following & Others		
IFEval (Zhou et al., 2023)	inst & prompt avg accuracy	lm-eval-harness
Alpaca-Eval (Li et al., 2023)	LC winrate	evalchemy
MTBench (Bai et al., 2024)	turn 1 & 2 average	evalchemy
LiveBench (White et al., 2024)	global_average	evalchemy
Multilingual		
Multi-Hellaswag (Dac Lai et al., 2023)	logprobs, 0-shot	lm-eval-harness
MGSM (Shi et al., 2022)	flexible extract, 8-shot native CoT	lm-eval-harness
Multi-MMLU (Dac Lai et al., 2023)	logprobs, 5-shot	lm-eval-harness
Long Context		
HELMET-LongQA (Yen et al., 2025)	default	helmet
HELMET-RAG (Yen et al., 2025)	default	helmet
HELMET-Recall (Yen et al., 2025)	default	helmet

Table 19: Evaluation settings and benchmark sources for instruction-tuned models.

like Gemma3-1B and Qwen3-0.6B show an edge on certain coding (MBPP, LiveCodeBench) and preference-based instruction following (Alpaca-Eval, LiveBench) benchmarks. Similarly, on general commonsense tasks like HellaSwag and ARC-C, larger models in the comparison group hold an advantage. This specialized profile suggests that our fine-tuning strategy for Falcon-H1-0.5B-Instruct prioritizes deep, multi-step reasoning and precise instruction execution over performance on conversational or broad-knowledge benchmarks. This trade-off firmly establishes the model as the leading choice for applications requiring robust, complex problem-solving at an efficient scale.

Tasks	Falcon-H1-0.5B	Qwen3-0.6B	Qwen2.5-0.5B	Gemma3-1B	Llama3.2-1.2B	Falcon3-1.6B
General						
BBH	42.91	32.95	33.26	<u>35.86</u>	33.21	34.47
ARC-C	<u>37.80</u>	31.06	33.28	34.13	34.64	43.09
TruthfulQA	44.12	51.65	<u>46.19</u>	42.17	42.08	42.31
HellaSwag	51.93	42.17	52.38	42.24	<u>55.30</u>	58.53
MMLU	53.40	42.98	<u>46.07</u>	40.87	45.93	46.10
Math						
GSM8k	68.39	42.61	38.51	42.38	<u>44.28</u>	44.05
MATH-500	58.40	<u>46.00</u>	27.80	45.40	13.20	19.80
AMC-23	33.13	<u>27.97</u>	12.50	19.22	7.19	6.87
AIME-24	3.75	<u>2.71</u>	0.62	0.42	1.46	0.41
AIME-25	4.38	<u>1.67</u>	0.21	1.25	0.00	0.21
Science						
GPQA	29.95	26.09	26.85	<u>28.19</u>	26.59	26.76
GPQA_Diamond	27.95	25.08	24.24	<u>21.55</u>	<u>25.08</u>	31.31
MMLU-Pro	31.03	16.95	<u>18.73</u>	14.46	16.20	18.49
MMLU-stem	54.55	39.30	<u>39.83</u>	35.39	39.16	39.64
Code						
HumanEval	51.83	<u>41.46</u>	36.59	40.85	34.15	22.56
HumanEval+	45.12	<u>37.19</u>	32.32	37.20	29.88	20.73
MBPP	42.59	<u>56.08</u>	46.83	57.67	33.60	20.63
MBPP+	33.07	<u>47.08</u>	39.68	50.00	29.37	17.20
LiveCodeBench	<u>7.05</u>	9.78	2.94	5.09	2.35	0.78
CRUXEval	25.75	<u>23.63</u>	14.88	12.70	0.06	15.58
Instruction Following						
IFEval	72.07	<u>62.16</u>	32.11	61.48	55.34	54.26
Alpaca-Eval	10.79	9.59	3.26	17.87	<u>9.38</u>	6.98
MTBench	7.06	5.75	4.71	<u>7.03</u>	6.37	6.03
LiveBench	20.80	27.78	14.27	<u>18.79</u>	14.97	14.10

Table 20: Performance of the 0.5B+ **Instruct** models.

Falcon-H1-1.5B-Instruct and Falcon-H1-1.5B-Deep-Instruct. At the 1.5B parameter scale, the Falcon-H1 instruct models establish clear dominance, with both the deep and shallow architecture variants setting new state-of-the-art benchmarks. As shown in Table 21, Falcon-H1-1.5B-Deep-Instruct achieves comprehensive leadership across nearly all evaluated domains, securing the top position on the vast majority of General, Math, Science, Code, and Multilingual tasks, reaching a level of performance competitive with current state-of-the-art 7B models, like Qwen3-8B (Yang et al., 2025), Qwen2.5-7B (Yang et al., 2024a) (check Table 23). This sweeping success highlights the profound impact of our deep architectural design combined with instruction tuning. The performance gap is particularly striking in complex reasoning, where the model substantially outperforms all peers on benchmarks like GSM8k (82.34) and MATH-500 (77.80). The shallower Falcon-H1-1.5B-Instruct model also delivers an exceptionally strong performance, consistently securing the

second-best score across most benchmarks and often outperforming larger models like Qwen3-1.7B. Its leadership on Alpaca-Eval further underscores its well-rounded instruction-following capabilities. The overwhelming evidence from these evaluations confirms that the Falcon-H1-1.5B-Instruct models, particularly the deep variant, are leading choices for sophisticated, reasoning-driven tasks, delivering performance that often transcends their parameter scale.

Tasks	Falcon-H1-1.5B-Deep	Falcon-H1-1.5B	Qwen3-1.7B	Qwen2.5-1.5B	Gemma3-1B	Llama3.2-1B	Falcon3-1.6B
General							
BBH	54.43	<u>46.47</u>	35.18	42.41	35.86	33.21	34.47
ARC-C	43.86	42.06	34.81	40.53	34.13	34.64	<u>43.09</u>
TruthfulQA	50.48	<u>49.39</u>	45.98	47.05	42.17	42.08	42.31
HellaSwag	65.54	<u>63.33</u>	49.27	62.23	42.24	55.30	58.53
MMLU	66.11	<u>62.03</u>	57.04	59.76	40.87	45.93	46.10
Math							
GSM8k	82.34	<u>74.98</u>	69.83	57.47	42.38	44.28	44.05
MATH-500	77.80	<u>74.00</u>	73.00	48.40	45.40	13.20	19.80
AMC-23	56.56	<u>46.09</u>	43.59	24.06	19.22	7.19	6.87
AIME-24	14.37	<u>12.50</u>	11.25	2.29	0.42	1.46	0.41
AIME-25	11.04	<u>9.58</u>	8.12	1.25	1.25	0.00	0.21
Science							
GPQA	33.22	26.34	27.68	26.26	<u>28.19</u>	26.59	26.76
GPQA_Diamond	40.57	<u>35.19</u>	33.33	25.59	21.55	25.08	31.31
MMLU-Pro	41.89	<u>37.80</u>	23.54	28.35	14.46	16.20	18.49
MMLU-stem	67.30	<u>64.13</u>	54.30	54.04	35.39	39.16	39.64
Code							
HumanEval	73.78	<u>68.29</u>	67.68	56.10	40.85	34.15	22.56
HumanEval+	68.90	<u>61.59</u>	60.96	50.61	37.20	29.88	20.73
MBPP	68.25	<u>64.81</u>	58.73	<u>64.81</u>	57.67	33.60	20.63
MBPP+	56.61	<u>56.35</u>	49.74	56.08	50.00	29.37	17.20
LiveCodeBench	23.87	<u>17.61</u>	14.87	12.52	5.09	2.35	0.78
CRUXEval	52.32	<u>39.57</u>	18.88	34.76	12.70	0.06	15.58
Instruction Following							
IFEval	83.50	<u>80.66</u>	70.77	45.33	61.48	55.34	54.26
Alpaca-Eval	<u>27.12</u>	28.18	21.89	9.54	17.87	9.38	6.98
MTBench	8.53	<u>8.46</u>	7.61	7.10	7.03	6.37	6.03
LiveBench	36.83	34.13	40.73	21.65	18.79	14.97	14.10
Multilingual							
Multi-Hellaswag	53.14	<u>49.38</u>	37.89	42.93	41.77	39.78	32.04
Multi-MMLU	53.00	<u>48.06</u>	39.60	45.90	34.91	35.24	32.25
MGSM	60.00	<u>58.00</u>	52.40	45.20	-	29.73	15.33

Table 21: Performance of the 1B+ **Instruct** models.

Falcon-H1-3B-Instruct. At the 3B-4B scale, Falcon-H1-3B-Instruct emerges as a top-performing and highly versatile model, demonstrating clear strengths in reasoning, science, and instruction following (Table 22). It leads on the majority of General knowledge benchmarks like MMLU and BBH, and dominates the Science category. This shows that after instruction tuning, the model excels at applying its knowledge. Furthermore, its state-of-the-art scores on IFEval and MTBench highlight its superior ability to understand and follow complex instructions. While other models show specialized strengths, such as Qwen3-4B in mathematics, Falcon-H1-3B remains highly competitive across all areas. It also shows a distinct advantage in Code generation on MBPP

and excels in Multilingual tasks. This balanced and powerful performance across many different domains establishes Falcon-H1-3B-Instruct as a premier, all-around model in its class.

Tasks	Falcon-H1-3B	Qwen3-4B	Qwen2.5-3B	Gemma3-4B	Llama3.2-3B	Falcon3-3B
General						
BBH	53.69	<u>51.07</u>	46.55	50.01	41.47	45.02
ARC-C	49.57	37.71	43.77	44.88	44.88	48.21
TruthfulQA	<u>53.19</u>	51.75	58.11	51.68	50.27	50.06
HellaSwag	69.85	55.31	<u>64.21</u>	47.68	63.74	64.24
MMLU	68.30	<u>67.01</u>	65.09	59.53	61.74	56.76
Math						
GSM8k	84.76	<u>80.44</u>	57.54	77.41	77.26	74.68
MATH-500	74.20	85.00	64.20	<u>76.40</u>	41.20	54.20
AMC-23	<u>55.63</u>	66.88	39.84	48.12	22.66	29.69
AIME-24	11.88	22.29	6.25	6.67	<u>11.67</u>	3.96
AIME-25	<u>13.33</u>	18.96	3.96	<u>13.33</u>	0.21	2.29
Science						
GPQA	33.89	28.02	28.69	<u>29.19</u>	28.94	28.69
GPQA_Diamond	<u>38.72</u>	40.74	35.69	28.62	29.97	29.29
MMLU-Pro	43.69	29.75	<u>32.76</u>	29.71	27.44	29.71
MMLU-stem	69.93	<u>67.46</u>	59.78	52.17	51.92	56.11
Code						
HumanEval	<u>76.83</u>	84.15	73.78	67.07	54.27	52.44
HumanEval+	70.73	76.83	68.29	61.59	50.00	45.73
MBPP	79.63	68.78	72.75	<u>77.78</u>	62.17	61.90
MBPP+	67.46	59.79	60.85	<u>66.93</u>	50.53	55.29
LiveCodeBench	26.81	39.92	11.74	<u>21.14</u>	2.74	3.13
CRUXEval	<u>56.25</u>	69.63	43.26	52.13	17.75	44.38
Instruction Following						
IFEval	85.05	<u>84.01</u>	64.26	77.01	74.00	69.10
Alpaca-Eval	31.09	<u>36.51</u>	17.37	39.64	19.69	14.82
MTBench	8.72	<u>8.45</u>	7.79	8.24	7.96	7.79
LiveBench	36.86	51.34	27.32	<u>36.70</u>	26.37	26.01
Multilingual						
Multi-Hellaswag	58.34	43.12	50.81	<u>54.48</u>	50.93	37.51
Multi-MMLU	54.90	50.70	<u>52.90</u>	51.10	48.40	38.90
MGSM	<u>63.90</u>	68.90	57.30	-	62.20	42.10

Table 22: Performance of the 3B+ **Instruct** models.

Falcon-H1-7B-Instruct. At the 7B-12B parameter scale, Falcon-H1-7B-Instruct demonstrates a highly competitive and well-rounded performance profile, outperforming the larger Gemma3-12B model (Team et al., 2025) on a majority of benchmarks (Table 23). When compared to its direct counterparts like Qwen3-8B (Yang et al., 2025), its strengths in knowledge-intensive domains become particularly evident. The model leads on all four Science benchmarks and on key General reasoning tasks such as MMLU and ARC-C. It also exhibits strong practical skills, achieving top scores on HumanEval and HumanEval+ for code generation and leading across all three Multilingual benchmarks. While competitors like Qwen3-8B and Gemma3-12B show advantages in specific areas, particularly on several Math and preference-based benchmarks (e.g., Alpaca-Eval, LiveBench), the collective results highlight a key distinction. The broad and deep capabilities of Falcon-H1-7B-Instruct across science, reasoning, code, and multilingualism make it an exceptionally effective and

versatile model for a wide range of sophisticated, instruction-driven applications.

Tasks	Falcon-H1-7B	Qwen3-8B	Qwen2.5-7B	Gemma3-12B	Llama3.1-8B	Falcon3-7B	Falcon3-10B
General							
BBH	62.28	47.47	53.76	63.36	48.58	52.12	58.09
ARC-C	59.98	42.06	41.38	51.96	52.39	<u>54.35</u>	54.44
TruthfulQA	59.91	53.19	62.41	<u>61.02</u>	52.99	<u>55.58</u>	55.05
HellaSwag	75.92	60.56	63.40	<u>55.63</u>	71.28	71.81	<u>75.57</u>
MMLU	76.83	71.56	73.64	<u>72.50</u>	68.67	70.81	74.01
Math							
GSM8k	81.65	78.92	71.95	87.49	82.49	81.05	<u>85.06</u>
MATH-500	73.40	<u>83.80</u>	75.80	86.20	45.80	69.00	68.60
AMC-23	<u>56.72</u>	70.78	53.91	66.88	22.81	40.00	45.78
AIME-24	16.04	28.33	12.29	<u>22.50</u>	5.42	8.75	9.79
AIME-25	13.96	19.17	9.58	<u>18.75</u>	0.42	6.25	5.42
Science							
GPQA	36.33	25.84	31.79	<u>33.98</u>	32.72	31.21	33.39
GPQA_Diamond	56.90	<u>43.10</u>	33.00	37.71	31.31	37.21	34.68
MMLU-Pro	51.75	34.64	43.23	39.88	36.42	40.73	<u>44.05</u>
MMLU-stem	77.61	66.89	<u>69.36</u>	66.54	59.31	67.43	70.57
Code							
HumanEval	86.59	<u>84.75</u>	82.32	84.76	68.29	71.95	82.32
HumanEval+	81.10	<u>79.27</u>	73.78	75.61	61.59	65.85	75.00
MBPP	80.69	71.96	79.63	85.71	68.25	<u>77.25</u>	73.28
MBPP+	68.78	62.70	68.25	72.22	55.03	<u>65.87</u>	64.02
LiveCodeBench	<u>35.03</u>	45.60	32.68	30.92	15.85	12.72	19.77
CRUXEval	<u>66.51</u>	72.70	56.90	67.67	21.57	55.00	59.57
Instruction Following							
IFEval	85.35	<u>83.43</u>	75.25	81.51	77.04	76.59	78.84
Alpaca-Eval	40.23	46.13	29.48	<u>43.55</u>	25.48	27.56	24.31
MTBench	8.85	<u>8.74</u>	8.45	8.69	8.29	8.73	8.46
LiveBench	<u>45.74</u>	56.19	37.13	49.23	31.73	32.35	34.30
Multilingual							
Multi-Hellaswag	67.75	47.30	58.74	<u>66.53</u>	60.74	47.81	52.77
Multi-MMLU	67.83	50.44	61.20	<u>65.22</u>	55.53	50.62	53.67
MGSM	73.50	65.20	66.10	-	<u>70.70</u>	56.30	64.80

Table 23: Performance of the 7B+ **Instruct** models.

Falcon-H1-34B-Instruct. At the 34B scale, Falcon-H1-34B-Instruct demonstrates that exceptional performance does not require massive parameter counts. As shown in Table 24, our 34B model consistently competes with and often outperforms models twice its size. Its primary strength lies in its deep knowledge and reasoning, leading across the entire Science category and on general reasoning benchmarks like HellaSwag. Furthermore, its top score on MTBench confirms its high-quality conversational abilities. While larger models leverage their scale to gain an edge in mathematics and some coding benchmarks, this profile highlights Falcon-H1-34B’s remarkable parameter efficiency. It delivers state-of-the-art results in science and general reasoning while being significantly smaller, making it a powerful and cost-effective choice for knowledge-intensive applications.

Long-Context Capabilities of Falcon-H1. Given the efficiency benefits of Mamba-based architectures in long-context scenarios, a systematic evaluation of Falcon-H1’s capabilities in this area is crucial. We benchmarked Falcon-H1-34B-Instruct against several larger models using the

Tasks	Falcon-H1-34B	Qwen3-32B	Qwen2.5-72B	Qwen2.5-32B	Gemma3-27B	Llama3.3-70B	Llama4-scout
General							
BBH	<u>70.68</u>	62.47	72.52	68.72	67.28	69.15	64.90
ARC-C	61.01	48.98	46.59	44.54	54.52	63.65	<u>56.14</u>
TruthfulQA	65.27	58.58	<u>69.80</u>	70.28	64.26	66.15	62.74
HellaSwag	81.94	68.89	68.79	<u>73.95</u>	57.25	70.24	65.03
MMLU	<u>84.05</u>	80.89	84.42	82.80	78.01	82.08	80.40
Math							
GSM8k	83.62	88.78	82.26	78.47	<u>90.37</u>	93.71	<u>90.37</u>
MATH-500	<u>83.80</u>	82.00	83.60	82.20	90.00	70.60	83.20
AMC-23	<u>69.38</u>	67.34	67.34	68.75	77.81	39.38	69.06
AIME-24	23.75	<u>27.71</u>	17.29	17.92	27.50	12.92	27.92
AIME-25	16.67	<u>19.79</u>	15.21	11.46	22.71	1.25	8.96
Science							
GPQA	41.53	30.20	<u>37.67</u>	34.31	36.49	31.99	31.80
GPQA_Diamond	49.66	<u>49.49</u>	44.95	40.74	47.47	42.09	51.18
MMLU-Pro	58.73	54.68	<u>56.63</u>	56.35	47.81	53.29	55.58
MMLU-stem	83.57	81.64	<u>82.59</u>	82.37	73.55	74.88	75.20
Code							
HumanEval	87.20	90.85	87.20	<u>90.24</u>	86.59	83.53	85.40
HumanEval+	81.71	85.37	80.49	<u>82.32</u>	78.05	79.87	78.70
MBPP	83.86	86.24	89.68	87.83	<u>88.36</u>	88.09	81.50
MBPP+	71.43	71.96	75.40	<u>74.07</u>	<u>74.07</u>	73.81	64.80
LiveCodeBench	<u>49.71</u>	45.01	54.60	49.12	39.53	40.31	40.12
CRUXEval	<u>73.07</u>	78.45	75.63	73.50	74.82	69.53	68.32
Instruction Following							
IFEval	<u>89.37</u>	86.97	86.35	81.79	83.19	89.94	86.32
Alpaca-Eval	48.32	64.21	49.29	39.26	<u>56.16</u>	38.27	36.26
MTBench	9.20	9.05	<u>9.16</u>	9.09	8.75	8.98	8.98
LiveBench	46.26	63.05	<u>54.03</u>	52.92	55.41	53.11	54.21
Multilingual							
Multi-Hellaswag	74.55	58.39	<u>69.48</u>	65.90	69.64	62.30	64.64
Multi-MMLU	<u>77.76</u>	66.20	78.26	73.56	71.60	74.58	76.67
MGSM	76.33	71.80	72.33	73.60	77.87	83.87	86.87

Table 24: Performance of the 34B-scale **Instruct** models.

HELMET suite (Yen et al., 2025), with category-level results presented in Table 25 and a full breakdown in Appendix D.3. The findings highlight Falcon-H1’s strong, parameter-efficient performance, particularly in the demanding Retrieval-Augmented Generation (RAG) task at extreme context lengths. While remaining competitive with 70B-class models at shorter sequences, our model uniquely achieves the top score on RAG at 131k tokens (62.21), surpassing all competitors, including those more than twice its size. This suggests a superior ability to synthesize information from retrieved documents, a key capability for practical RAG systems. The results also reveal a nuanced performance profile. On tasks reliant on pure recall and long-form question answering (longQA), Falcon-H1 is surpassed by competitors like Qwen3-32B and Llama-3.3-70B at extreme context lengths. We attribute this performance gap not to architectural limitations but to our training data composition, which indicates substantial room for improvement with more curated long-context data. Crucially, despite this trade-off, Falcon-H1 still broadly outperforms the much larger Qwen2.5-72B-Instruct model across most long-context tasks. This, combined with its state-of-the-art RAG performance, positions Falcon-H1-34B as a highly effective and parameter-efficient choice for real-world, long-context systems.

Seq. Length	Falcon-H1-34B-Instruct	Qwen2.5-72B-Instruct	Qwen3-32B	Llama-3.3-70B-Instruct
HELMET-RAG				
8k	72.17	<u>72.21</u>	69.25	74.29
16k	81.46	80.42	77.92	82.33
32k	67.96	<u>70.08</u>	64.83	70.21
65k	67.08	63.25	61.96	69.08
131k	62.21	42.33	<u>57.08</u>	55.38
HELMET-Recall				
8k	100.00	100.00	100.00	100.00
16k	100.00	100.00	100.00	100.00
32k	97.50	98.38	100.00	<u>99.63</u>
65k	80.69	71.75	<u>96.50</u>	98.81
131k	56.63	38.81	86.13	<u>82.19</u>
HELMET-longQA				
8k	32.87	35.20	31.63	<u>33.67</u>
16k	34.64	<u>39.13</u>	35.68	39.75
32k	35.09	39.22	<u>41.15</u>	47.53
65k	32.45	36.71	<u>47.47</u>	48.57
131k	33.81	32.94	53.52	<u>46.06</u>

Table 25: HELMET performance metrics at various sequence lengths. The best result in each row is in bold, and the second-best is underlined.

Overall Remarks on Falcon-H1 Instruct models. The Falcon-H1 instruct series demonstrates a consistent pattern of state-of-the-art performance and exceptional parameter efficiency across all evaluated scales. Our smaller models, particularly the Falcon-H1-1.5B-Deep-Instruct, redefine the performance baseline in their class, delivering reasoning capabilities in math and science that are competitive with leading 7B and 10B models. This makes them powerful and efficient alternatives for resource-constrained and edge environments. As we scale up, the Falcon-H1-7B and 34B models establish themselves as leaders in knowledge-intensive and reasoning-focused domains. They consistently excel in science, code generation, and multilingual understanding, often outperforming competitor models with more than double their parameter count. Furthermore, the Falcon-H1-34B shows a distinct advantage in practical long-context applications, leading on the 131k RAG bench-

mark. While larger competitor models show an edge in certain math and preference-based tasks, the Falcon-H1 series consistently provides a superior balance of deep reasoning, broad knowledge, and high efficiency, positioning it as a premier choice for sophisticated, real-world applications.

5.3 Model Efficiency

We conducted a comparative evaluation of prefill (input) and generation (output) throughput between Falcon-H1-34B and Qwen2.5-32B¹⁵. All experiments were conducted on H100 GPUs using the vLLM framework (Kwon et al., 2023) with a tensor parallel size of 2. The performance of each phase was measured as follows:

- **Prefill throughput test:** Input sequence length was varied (2k to 262k tokens), while the output was fixed at 2,048 generated tokens per sequence with a batch size of 32.
- **Generation Throughput Test:** Input sequence length was fixed at 4,096 tokens with a batch size of 32, while the output generation length was varied (2k to 262k tokens).

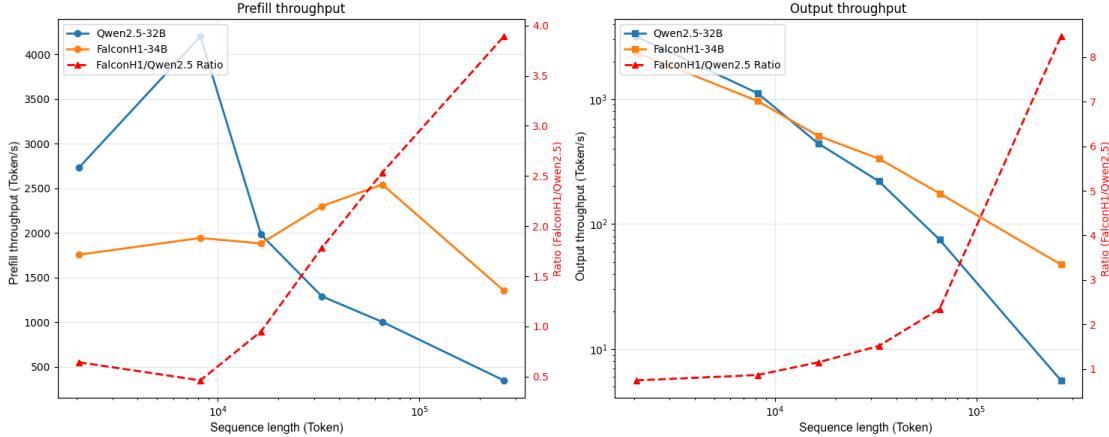


Figure 16: Model efficiency comparison between Falcon-H1-34B and Qwen2.5-32B.

As shown in Table 16, the results demonstrate the superior scalability of the Falcon-H1 hybrid architecture. While Qwen2.5-32B exhibits a marginal throughput advantage at shorter context lengths, Falcon-H1-34B becomes significantly more efficient as the context grows. At the longest sequence lengths tested, Falcon-H1-34B achieves up to a **4x improvement in input throughput** and an **8x speedup in output throughput**. This performance profile makes the model exceptionally well-suited for long-context input and generation use cases. The initial advantage of the Transformer-based model at short contexts is likely attributable to the highly mature optimizations of attention mechanisms within modern inference frameworks compared to current State-Space Model (SSM) implementations. As theoretically, Mamba-based or hybrid architectures are more efficient, we believe this gap highlights a promising direction for future work. We invite the community to contribute to optimizing SSM implementations, which we see as a critical step in advancing the next generation of efficient large language models.

¹⁵Experiments were conducted prior to the release of Qwen3; however, we anticipate no significant efficiency differences between Qwen2.5-32B and Qwen3-32B.

6. Model Integrations

To ensure broad accessibility and facilitate immediate adoption, Falcon-H1 is deeply integrated into the open-source AI ecosystem. Table 26 summarizes the key platforms and tools supported at the time of this report’s release. This list is continually expanding, with the most up-to-date information available on our project website¹⁶.

Category	Supported Tools and Platforms
General Usage	vLLM (Kwon et al., 2023), Hugging Face (<code>transformers</code> (Wolf et al., 2020), PEFT (Mangrulkar et al., 2022), TRL (von Werra et al.))
Fine-tuning	Llama-Factory (Zheng et al., 2024), OUMI (Oumi Community), Axolotl ^a , Unslot (Daniel Han & team, 2023)
Local Deployment	llama.cpp ^b , LM-Studio ^c , Jan ^d , Docker Model API ^e , Ollama ^f , [*] , Apple MLX ^g , [*]
Cloud Deployment	SkyPilot ^h
Quantization	AutoGPTQ (ModelCloud.ai & qubitum@modelcloud.ai, 2024)

^{*} Integrations validated internally; official support pending merge into main libraries at the time of the report release.

^a <https://github.com/axolotl-ai-cloud/axolotl> ^b <https://github.com/ggml-org/llama.cpp>

^c <https://lmstudio.ai/> ^d <https://github.com/menloresearch/jan>

^e <https://docs.docker.com/ai/model-runner/> ^f <https://github.com/ollama/ollama>

^g <https://github.com/ml-explore/mlx-lm> ^h <https://github.com/skypilot-org/skypilot>

Table 26: Key Ecosystem Integrations for the Falcon-H1 Series.

7. Conclusion

In this report, we introduced the Falcon-H1 series, a new family of models built on an innovative hybrid Mamba-Transformer architecture. Our design goal was to achieve state-of-the-art performance with exceptional resource efficiency, and our comprehensive evaluations confirm the success of this approach. A key advantage of Falcon-H1 is its ability to deliver superior performance while using significantly less training data—only 2.5T to 18T tokens—and offering up to 8x faster inference in long-context scenarios. This performance gain is particularly impactful at smaller scales, where our 1.5B-Deep model delivers capabilities competitive with leading 7B-10B models, making it ideal for edge deployments. At the larger end, our flagship 34B model challenges and often surpasses 70B+ competitors, particularly on knowledge-intensive tasks and practical applications like RAG at extreme context lengths. The success of this series is rooted in several key innovations: a flexible hybrid architecture allowing for an optimal attention-SSM ratio, a robust multilingual design, and a customized training strategy that maximizes the value of high-quality data. We also observed that the models’ performance had not yet saturated by the end of pretraining, indicating significant headroom for future gains. Ultimately, by providing a powerful, efficient, and versatile foundation for a wide range of applications, the Falcon-H1 series demonstrates a more sustainable and accessible path toward developing high-performance artificial intelligence.

Our future work will focus on several primary areas. First, we will prioritize data enhancement. We plan to iteratively refine both our base and instruction-tuned models by curating broader and more diverse high-quality datasets. Second, we will continue our architectural and algorithmic innovation, with a focus on effective knowledge compression and scaling to contexts beyond 256k.

¹⁶ <https://tiiuae.github.io/Falcon-H1/>

Finally, we will continue to scale our models strategically, exploring novel techniques to deepen their core reasoning capabilities. Through these efforts, we aim to continue developing models that are not only state-of-the-art but also fundamentally more efficient and accessible, thereby contributing to the sustainable advancement of artificial intelligence.

8. Authors

Core Contributors

Jingwei Zuo, Maksim Velikanov, Ilyas Chahed, Younes Belkada, Dhia Eddine Rhayem, Guillaume Kunsch*, Hakim Hacid

Contributors

- **Quantization and Integration:** Hamza Yous, Brahim Farhat, Ibrahim Khadraoui
- **Data & Evaluation:** Mugariya Farooq, Giulia Campesan, Ruxandra Cojocaru, Yasser Djilali, Shi Hu, Iheb Chaabane, Puneesh Khanna, Mohamed El Amine Seddik, Ngoc Dung Huynh, Phuc Le Khac, Leen AlQadi, Billel Mokeddem, Mohamed Chami, Abdalgader Abubaker
- **Platform & Infrastructure Support:** Mikhail Lubinets, Kacper Piskorski, Slim Frikha

*Individual who has departed from our team; work was conducted at TII.

9. Acknowledgments

We extend our sincere gratitude to the contributors and maintainers from the open-source AI community who were instrumental in integrating Falcon-H1 into the ecosystem. For their invaluable help with fine-tuning libraries, we thank Wing Lian for the support on Axolotl, Daniel Han for the support on Unsloth, Yaowei Zheng for the support on Llama-Factory, and the OUMI maintainers for their assistance. For their support with local deployment tools, we are grateful to Prince Canuma and Awni Hannun for their work on the Apple MLX integration, and to Georgi Gerganov, `compilade`, and Gabe Goodhart for their extensive efforts on llama.cpp and Ollama integration. We also thank the Jan maintainers for their help. Finally, for their crucial support on core libraries, we thank Arthur Zucker for his guidance on Hugging Face Transformers integration and the vLLM maintainers for their help in incorporating Falcon-H1 into their high-performance inference library.

References

- European Parliament Proceedings Parallel Corpus 1996-2011, Release v7. <https://www.statmt.org/europarl/>.
- Project Gutenberg. <https://www.gutenberg.org/>.
- Makhzan Dataset. <https://github.com/zeerakahmed/makhzan/>.
- AIME. AIME problems and solutions, 2025. URL https://artofproblemsolving.com/wiki/index.php/AIME_Problems_and_Solutions.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL <https://openreview.net/forum?id=hm0w0ZWzYE>.

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2357–2367, Minneapolis, Minnesota, 6 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1245. URL <https://aclanthology.org/N19-1245>.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*, 2023.

Ge Bai, Jie Liu, Xingyuan Bu, Yancheng He, Jiaheng Liu, Zhanhui Zhou, Zhuoran Lin, Wenbo Su, Tiezheng Ge, Bo Zheng, et al. Mt-bench-101: A fine-grained benchmark for evaluating large language models in multi-turn dialogues. *arXiv preprint arXiv:2402.14762*, 2024.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024.

Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, Thomas Wolf, and Leandro von Werra. Smollm-corpus, 7 2024. URL <https://huggingface.co/datasets/HuggingFaceTB/smollm-corpus>.

Johan Bjorck, Alon Benhaim, Vishrav Chaudhary, Furu Wei, and Xia Song. Scaling optimal LR across token horizons. In *The Thirteenth International Conference on Learning Representations*, 2025. URL <https://openreview.net/forum?id=WYL4eFLcxG>.

bloc97. Ntk-aware scaled rope allows llama models to have extended (8k+) context size without any fine-tuning and minimal perplexity degradation, 2023a. URL https://www.reddit.com/r/LocalLLaMA/comments/141z7j5/ntkaware_scaled_ropeAllows_llama_models_to_have/.

bloc97. Add ntk-aware interpolation "by parts" correction, 2023b. URL <https://github.com/jquesnelle/scaled-rope/pull/1>.

Blake Bordelon, Lorenzo Noci, Mufan Bill Li, Boris Hanin, and Cengiz Pehlevan. Depthwise hyperparameter transfer in residual networks: Dynamics and scaling limit, 2023. URL <https://arxiv.org/abs/2309.16620>.

Andrei Broder. On the resemblance and containment of documents. 06 1997. doi: 10.1109/SEQUEN.1997.666900.

Rong Chao, Wenze Ren, Wen-Yuan Ting, Hsin-Yi Lin, Yu Tsao, and Fan-Gang Zeng. An investigation of incorporating mamba for speech enhancement. *arXiv preprint arXiv:2405.06573*, 2024. URL <https://arxiv.org/abs/2405.06573>. Accepted to IEEE SLT 2024.

Lijie Chen, Binghui Peng, and Hongxun Wu. Theoretical limitations of multi-layer transformer. December 2024. URL <https://arxiv.org/abs/2412.02975>. arXiv:2412.02975v1 [cs.LG], 4 Dec 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Heben Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021.

Shouyuan Chen, Sherman Wong, Liangjian Chen, and Yuandong Tian. Extending context window of large language models via positional interpolation, 2023. URL <https://arxiv.org/abs/2306.15595>.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways, 2022. URL <https://arxiv.org/abs/2204.02311>.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021a.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021b.

Viet Dac Lai, Chien Van Nguyen, Nghia Trung Ngo, Thuat Nguyen, Franck Dernoncourt, Ryan A Rossi, and Thien Huu Nguyen. Okapi: Instruction-tuned large language models in multiple languages with reinforcement learning from human feedback. *arXiv e-prints*, pp. arXiv–2307, 2023.

Michael Han Daniel Han and Unsloth team. Unsloth, 2023. URL <http://github.com/unslotha/unslloth>.

Tri Dao. State space duality (mamba-2) part i: The model. <https://tridao.me/blog/2024/mamba2-part1-model/>, 2024a. “we also choose similar dimensions as modern Transformers, e.g. $P = 64$ or $P = 128$.”

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024b.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

Dao-AILab. causal-conv1d: Causal depthwise conv1d in cuda with a pytorch interface. <https://github.com/Dao-AILab/causal-conv1d>, 2023. Features: kernel size 2, 3, 4; supports fp32/fp16/bf16.

Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

Nolan Dey, Gurpreet Gosal, Zhiming Chen, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, and Joel Hestness. Cerebras-gpt: Open compute-optimal language models trained on the cerebras wafer-scale cluster, 2023. URL <https://arxiv.org/abs/2304.03208>.

Nolan Dey, Bin Claire Zhang, Lorenzo Noci, Mufan Li, Blake Bordelon, Shane Bergsma, Cengiz Pehlevan, Boris Hanin, and Joel Hestness. Don’t be lazy: Completep enables compute-efficient deep transformers, 2025. URL <https://arxiv.org/abs/2505.01618>.

Xin Dong, Yonggan Fu, Shizhe Diao, Wonmin Byeon, Zijia Chen, Ameya Sunil Mahabaleshwarkar, Shih-Yang Liu, Matthijs Van Keirsbilck, Min-Hung Chen, Yoshi Suhara, et al. Hymba: A hybrid-head architecture for small language models. *arXiv preprint arXiv:2411.13676*, 2024.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac’h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL <https://zenodo.org/records/12608602>.

Gardian. A curated research corpus for agricultural advisory ai applications, 2024. URL <https://huggingface.co/datasets/CGIAR/gardian-ai-ready-docs>.

Paolo Glorioso, Quentin Anthony, Yury Tokpanov, James Whittington, Jonathan Pilault, Adam Ibrahim, and Beren Millidge. Zamba: A compact 7b ssm hybrid model. *arXiv preprint arXiv:2405.16712*, 2024.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

Alex Gu, Baptiste Rozière, Hugh Leather, Armando Solar-Lezama, Gabriel Synnaeve, and Sida I Wang. Cruxeval: A benchmark for code reasoning, understanding and execution. *arXiv preprint arXiv:2401.03065*, 2024.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. Deepseek-coder: When the large language model meets programming – the rise of code intelligence, 2024. URL <https://arxiv.org/abs/2401.14196>.

Xiaotian Han, Yiren Jian, Xuefeng Hu, Haogeng Liu, Yiqi Wang, Qihang Fan, Yuang Ai, Huaibo Huang, Ran He, Zhenheng Yang, et al. Infimm-webmath-40b: Advancing multimodal pre-training for enhanced mathematical reasoning. *arXiv preprint arXiv:2409.12568*, 2024.

Bobby He and Thomas Hofmann. Simplifying transformer blocks. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=RtDok9eS3s>.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021a.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021b.

Scott Hoang and Mamba contributors. Clarification on how to interpret kernel size for conv1d (#523). <https://github.com/state-spaces/mamba/issues/523>, 2024. GitHub issue discussing the meaning and limits of the conv1d kernel size.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024a.

Yiwen Hu, Huatong Song, Jia Deng, Jiapeng Wang, Jie Chen, Kun Zhou, Yutao Zhu, Jinhao Jiang, Zican Dong, Wayne Xin Zhao, and Ji-Rong Wen. Yulan-mini: An open data-efficient language model, dec 2024b. URL <https://arxiv.org/abs/2412.17743>.

Siming Huang, Tianhao Cheng, Jason Klein Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, J. H. Liu, Chenchen Zhang, Linzheng Chai, Ruifeng Yuan, Zhaoxiang Zhang, Jie Fu, Qian Liu, Ge Zhang, Zili Wang, Yuan Qi, Yinghui Xu, and Wei Chu. Opencoder: The open cookbook for top-tier code large language models. 2024a. URL <https://arxiv.org/pdf/2411.04905.pdf>.

Siming Huang, Tianhao Cheng, Jason Klein Liu, Jiaran Hao, Liuyihan Song, Yang Xu, J. Yang, JH Liu, Chenchen Zhang, Linzheng Chai, et al. Opencoder: The open cookbook for top-tier code large language models. *arXiv preprint arXiv:2411.04905*, 2024b.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei

- Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-coder technical report, 2024. URL <https://arxiv.org/abs/2409.12186>.
- "Teknium" "interstellarninja". Hermes-function-calling-dataset-v1. URL <https://huggingface.co/NousResearch/hermes-function-calling-v1>.
- Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.
- Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.
- Hynek Kydlíček. Math-Verify: Math Verification Library. URL <https://github.com/huggingface/math-verify>.
- Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.
- Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. AlpacaEval: An automatic evaluator of instruction-following models. https://github.com/tatsu-lab/alpaca_eval, 5 2023.
- Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024.
- Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.
- Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*, 2021.
- Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024a.
- Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024b.
- Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context, 2023a. URL <https://arxiv.org/abs/2310.01889>.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b. URL <https://openreview.net/forum?id=1qvx610Cu7>.

Yan Liu, Renren Jin, Ling Shi, Zheng Yao, and Deyi Xiong. Finemath: A fine-grained mathematical evaluation benchmark for chinese large language models. *arXiv preprint arXiv:2403.07747*, 2024c.

Yue Liu, Yunjie Tian, Yuzhong Zhao, Hongtian Yu, Lingxi Xie, Yaowei Wang, Qixiang Ye, Jianbin Jiao, and Yunfan Liu. Vmamba: Visual state space model. *arXiv preprint arXiv:2401.10166*, 2024d. doi: 10.48550/arXiv.2401.10166. URL <https://arxiv.org/abs/2401.10166>. NeurIPS 2024 Spotlight.

Zhengzhong Liu, Aurick Qiao, Willie Neiswanger, Hongyi Wang, Bowen Tan, Tianhua Tao, Junbo Li, Yuqi Wang, Suqi Sun, Omkar Pangarkar, Richard Fan, Yi Gu, Victor Miller, Yonghao Zhuang, Guowei He, Haonan Li, Fajri Koto, Liping Tang, Nikhil Ranjan, Zhiqiang Shen, Xuguang Ren, Roberto Iriondo, Cun Mu, Zhiting Hu, Mark Schulze, Preslav Nakov, Tim Baldwin, and Eric P. Xing. Llm360: Towards fully transparent open-source llms, 2023c. URL <https://arxiv.org/abs/2312.06550>.

Zihan Liu, Yang Chen, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Acemath: Advancing frontier math reasoning with post-training and reward modeling. *arXiv preprint arXiv:2412.15084*, 2024e.

Ziming Liu, Yizhou Liu, Jeff Gore, and Max Tegmark. Neural thermodynamic laws for large language model training, 2025. URL <https://arxiv.org/abs/2505.10559>.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*, 2024.

M-A-P, Ge Zhang, Xinrun Du, Zhimiao Yu, Zili Wang, Zekun Wang, Shuyue Guo, Tianyu Zheng, Kang Zhu, Jerry Liu, Shawn Yue, Binbin Liu, Zhongyuan Peng, Yifan Yao, Jack Yang, Ziming Li, Bingni Zhang, Minghao Liu, Tianyu Liu, Yang Gao, Wenhui Chen, Xiaohuan Zhou, Qian Liu, Taifeng Wang, and Wenhao Huang. Finefineweb: A comprehensive study on fine-grained domain web corpus, December 2024. URL [<https://huggingface.co/datasets/m-a-p/FineFineWeb>] (<https://huggingface.co/datasets/m-a-p/FineFineWeb>).

Quentin Malartic, Nilabhra Roy Chowdhury, Ruxandra Cojocaru, Mugariya Farooq, Giulia Campan- san, Yasser Abdelaziz Dahou Djilali, Sanath Narayan, Ankit Singh, Maksim Velikanov, Basma El Amel Boussaha, et al. Falcon2-11b technical report. *arXiv preprint arXiv:2407.14885*, 2024.

Sadhika Malladi, Kaifeng Lyu, Abhishek Panigrahi, and Sanjeev Arora. On the SDEs and scaling rules for adaptive gradient algorithms. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL <https://openreview.net/forum?id=F2mhzjHkQP>.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. Peft: State-of-the-art parameter-efficient fine-tuning methods. <https://github.com/huggingface/peft>, 2022.

Saptarshi Mitra, Rachid Karami, Haocheng Xu, Sitao Huang, and Hyoukjun Kwon. Characterizing state space model (ssm) and ssm-transformer hybrid language model performance with long context length. *arXiv preprint arXiv:2507.12442*, 2025. doi: 10.48550/arXiv.2507.12442. URL <https://arxiv.org/abs/2507.12442>.

ModelCloud.ai and qubitium@modelcloud.ai. Gptqmodel. <https://github.com/modelcloud/gptqmodel>, 2024. Contact: qubitium@modelcloud.ai.

Oumi Community. Oumi: an Open, End-to-end Platform for Building Large Foundation Models. URL <https://github.com/oumi-ai/oumi>.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. Openwebmath: An open dataset of high-quality mathematical web text. *arXiv preprint arXiv:2310.06786*, 2023.

Yan Ru Pei and others. Let ssms be convnets: State-space modeling with optimal tensor contractions. *arXiv preprint arXiv:2501.13230*, 2025. URL <https://arxiv.org/abs/2501.13230>.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The refinedweb dataset for falcon llm: outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv:2306.01116*, 2023.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *arXiv preprint arXiv:2406.17557*, 2024a.

Guilherme Penedo, Hynek Kydlíček, Vinko Sabolčec, Bettina Messmer, Negar Foroutan, Martin Jaggi, Leandro von Werra, and Thomas Wolf. Fineweb2: A sparkling update with 1000s of languages, 12 2024b. URL <https://huggingface.co/datasets/HuggingFaceFW/fineweb-2>.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

Negin Raoof, Etash Kumar Guha, Ryan Marten, Jean Mercat, Eric Frankel, Sedrick Keh, Hritik Bansal, Georgios Smyrnis, Marianna Nezhurina, Trung Vu, Zayne Rea Sprague, Mike A Merrill, Liangyu Chen, Caroline Choi, Zaid Khan, Sachin Grover, Benjamin Feuer, Ashima Suvarna, Shiye Su, Wanjia Zhao, Kartik Sharma, Charlie Cheng-Jie Ji, Kushal Arora, Jeffrey Li, Aaron Gokaslan, Sarah M Pratt, Niklas Muennighoff, Jon Saad-Falcon, John Yang, Asad Aali, Shreyas Pimpalgaonkar, Alon Albalak, Achal Dave, Hadi Pouransari, Greg Durrett, Sewoong Oh, Tatsunori Hashimoto, Vaishaal Shankar, Yejin Choi, Mohit Bansal, Chinmay Hegde, Reinhard Heckel, Jenia Jitsev, Maheswaran Sathiamoorthy, Alex Dimakis, and Ludwig Schmidt. Evalchemy, 6 2025.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. *arXiv preprint arXiv:2311.12022*, 2023.

Liliang Ren, Yang Liu, Yadong Lu, Yelong Shen, Chen Liang, and Weizhu Chen. Samba: Simple hybrid state space models for efficient unlimited context language modeling. *arXiv preprint arXiv:2406.07522*, 2024.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*, 2019.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units, 2016. URL <https://arxiv.org/abs/1508.07909>.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Yikang Shen, Matthew Stallone, Mayank Mishra, Gaoyuan Zhang, Shawn Tan, Aditya Prasad, Adriana Meza Soria, David D. Cox, and Rameswar Panda. Power scheduler: A batch size and token number agnostic learning rate scheduler, 2024. URL <https://arxiv.org/abs/2408.13359>.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, et al. Language models are multilingual chain-of-thought reasoners. *arXiv preprint arXiv:2210.03057*, 2022.

Aaditya K. Singh and DJ Strouse. Tokenization counts: the impact of tokenization on arithmetic in frontier llms, 2024. URL <https://arxiv.org/abs/2402.14903>.

Matei-Ioan Stan and Oliver Rhodes. Learning long sequences in spiking neural networks. *Scientific Reports*, 14(1):21957, 2024. doi: 10.1038/s41598-024-71678-8. URL <https://www.nature.com/articles/s41598-024-71678-8>.

Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset, 2024. URL <https://arxiv.org/abs/2412.02595>.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

Chaofan Tao, Qian Liu, Longxu Dou, Niklas Muennighoff, Zhongwei Wan, Ping Luo, Min Lin, and Ngai Wong. Scaling laws with vocabulary: Larger models deserve larger vocabularies. *arXiv preprint arXiv:2407.13623*, 2024.

Falcon-LLM Team. The falcon 3 family of open models, December 2024.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Jamba Team, Barak Lenz, Alan Arazi, Amir Bergman, Avshalom Manevich, Barak Peleg, Ben Aviram, Chen Almagor, Clara Fridman, Dan Padnos, et al. Jamba-1.5: Hybrid transformer-mamba models at scale. *arXiv preprint arXiv:2408.12570*, 2024.

Huu Nguyen Thuat Nguyen and Thien Nguyen. Culturay: A large cleaned multilingual dataset of 75 languages, 2024.

Jörg Tiedemann. Finding alternative translations in a large corpus of movie subtitle. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pp. 3518–3522, 2016.

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data. *arXiv preprint arXiv:2410.01560*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformer Reinforcement Learning. URL <https://github.com/huggingface/trl>.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Siddartha Naidu, et al. Livebench: A challenging, contamination-free llm benchmark. *arXiv preprint arXiv:2406.19314*, 4, 2024.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Perric Cistac, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-Art Natural Language Processing. pp. 38–45. Association for Computational Linguistics, 10 2020. URL <https://www.aclweb.org/anthology/2020.emnlp-demos.6>.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024a.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024b.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Ying

Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Greg Yang and Edward J. Hu. Feature learning in infinite-width neural networks, 2022. URL <https://arxiv.org/abs/2011.14522>.

Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer, 2022. URL <https://arxiv.org/abs/2203.03466>.

Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. Tensor programs vi: Feature learning in infinite-depth neural networks, 2023. URL <https://arxiv.org/abs/2310.02244>.

Greg Yang, James B. Simon, and Jeremy Bernstein. A spectral condition for feature learning, 2024c. URL <https://arxiv.org/abs/2310.17813>.

Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. Helmet: How to evaluate long-context language models effectively and thoroughly. In *International Conference on Learning Representations (ICLR)*, 2025.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning. In *Conference on Neural Information Processing Systems*, 2022.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Guangxiang Zhao, Xu Sun, Jingjing Xu, Zhiyuan Zhang, and Liangchen Luo. Muse: Parallel multi-scale attention for sequence to sequence learning, 2019. URL <https://arxiv.org/abs/1911.09483>.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyuan Luo, Zhangchi Feng, and Yongqiang Ma. LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models. Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations). Association for Computational Linguistics, 2024. URL <https://arxiv.org/abs/2403.13372>.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Jingwei Zuo, Maksim Velikanov, Dhia Eddine Rhaiem, Ilyas Chahed, Younes Belkada, Guillaume Kunsch, and Hacid Hakim. Falcon mamba: The first competitive attention-free 7b language model. 2024. URL <https://arxiv.org/abs/2410.05355>.

A. Languages used for training Falcon-H1 tokenizers

#	Code	Language	#	Code	Language	#	Code	Language
1	af	Afrikaans	2	als	Swiss German	3	am	Amharic
4	an	Aragonese	5	ar	Arabic	6	arz	Egyptian Arabic
7	as	Assamese	8	ast	Asturian	9	av	Avaric
10	az	Azerbaijani	11	azb	South Azerbaijani	12	ba	Bashkir
13	bar	Bavarian	14	bcl	Central Bikol	15	be	Belarusian
16	bg	Bulgarian	17	bh	Bihari languages	18	bn	Bangla
19	bo	Tibetan	20	bpy	Bishnupriya	21	bs	Bosnian
22	bxr	Russia Buriat	23	ca	Catalan	24	ce	Chechen
25	ckb	Central Kurdish	26	cs	Czech	27	cv	Chuvash
28	cy	Welsh	29	da	Danish	30	de	German
31	dsb	Lower Sorbian	32	dv	Divehi	33	el	Greek
34	eml	Emiliano-Romagnol	35	eo	Esperanto	36	es	Spanish
37	et	Estonian	38	eu	Basque	39	fa	Persian
40	fi	Finnish	41	fr	French	42	fy	Western Frisian
43	ga	Irish	44	gd	Scottish Gaelic	45	gl	Galician
46	gn	Guarani	47	gom	Goan Konkani	48	gu	Gujarati
49	he	Hebrew	50	hi	Hindi	51	hr	Croatian
52	hsb	Upper Sorbian	53	ht	Haitian Creole	54	hu	Hungarian
55	hy	Armenian	56	ia	Interlingua	57	id	Indonesian
58	ie	Interlingue	59	ilo	Iloko	60	io	Ido
61	is	Icelandic	62	it	Italian	63	ja	Japanese
64	jbo	Lojban	65	jv	Javanese	66	ka	Georgian
67	kk	Kazakh	68	km	Khmer	69	kn	Kannada
70	ko	Korean	71	krc	Karachay-Balkar	72	ku	Kurdish
73	kv	Komi	74	kw	Cornish	75	ky	Kyrgyz
76	la	Latin	77	lb	Luxembourgish	78	lez	Lezghian
79	li	Limburgish	80	lmo	Lombard	81	lo	Lao
82	lt	Lithuanian	83	lv	Latvian	84	mai	Maithili
85	mg	Malagasy	86	mk	Macedonian	87	ml	Malayalam
88	mn	Mongolian	89	mr	Marathi	90	mrj	Western Mari
91	ms	Malay	92	mt	Maltese	93	mwl	Mirandese
94	my	Burmese	95	myv	Erzya	96	mzn	Mazanderani
97	nah	Nahuatl languages	98	nap	Neapolitan	99	nds	Low German
100	ne	Nepali	101	new	Newari	102	nl	Dutch
103	nn	Norwegian Nynorsk	104	no	Norwegian	105	oc	Occitan
106	or	Odia	107	os	Ossetic	108	pa	Punjabi
109	pam	Pampanga	110	pl	Polish	111	pms	Piedmontese
112	pnb	Western Panjabi	113	ps	Pashto	114	pt	Portuguese
115	qu	Quechua	116	ro	Romanian	117	ru	Russian
118	sv	Swedish	119	th	Thai	120	tr	Turkish
121	vi	Vietnamese						

Table 27: Language Codes and Corresponding Languages

B. Scalar stochastic dynamics with weight decay

To model the behavior of parameter norms $\|W\|^2 = \sum_{ij} W_{ij}^2$ we look at a single entry $W_{ij} = x$ and consider its evolution during training. Then, behavior of the typical values of x^2 could serve as a proxy for behavior of the parameter norms $\|W\|^2$.

The update of parameter x at iteration t using AdamW with learning rate η and weight decay λ can be written as

$$x_{t+1} = x_t - \eta A_t - \eta \lambda x_t. \quad (20)$$

Here $A_t = \frac{g_{1,t}}{\sqrt{g_{2,t} + \epsilon}}$ is the Adam update rule that uses moving average of the gradient g_1 and moving average (element-wise) squared gradient g_2 .

Generally, updates A_t depend on the dynamics of the rest of model beyond our selected parameter x , have stochastic nature w.r.t. randomness caused by sampling of the samples in a batch, and correlated across nearby iterations t due to moving averages used in the update rule. Yet, to get an intuitive picture of the role of AdamW parameters η, λ we consider a toy model of this update

$$A_t = h(x_t - x^*) + \xi_t, \quad (21)$$

where ξ_t is i.i.d. noise with zero mean and variance $\mathbb{E}[\xi_t^2] = \sigma^2$, and h describes steepness of the loss landscape for our chosen parameter x . Then, evolution (20) of x_t becomes a stationary linear stochastic equation.

Let us now track the evolution of the first two moments of x_t . Taking the expectation of (20) and its square we get the update of the moments

$$\mathbb{E}[x_{t+1}] = (1 - \eta h - \eta \lambda) \mathbb{E}[x_t] + \eta h x^* \quad (22)$$

$$\mathbb{E}[x_{t+1}^2] = (1 - \eta h - \eta \lambda)^2 \mathbb{E}[x_t^2] + 2(1 - \eta h - \eta \lambda) \eta h \mathbb{E}[x_t] x^* + (\eta h x^*)^2 + \eta^2 \sigma^2 \quad (23)$$

As $t \rightarrow \infty$, the dynamics of first and second moment converge to the stationary state $\mathbb{E}[x_t] \rightarrow x_\infty$, $\mathbb{E}[x_t^2] \rightarrow x_{\infty,2}$ that can be found by setting $\mathbb{E}[x_{t+1}] = \mathbb{E}[x_t] = x_\infty$ and $\mathbb{E}[x_{t+1}^2] = \mathbb{E}[x_t^2] = x_{\infty,2}$. With a direct calculation, we get

$$x_\infty = \frac{h}{h + \lambda} x^*, \quad (24)$$

$$x_{\infty,2} = \frac{\eta \sigma^2}{(\lambda + h)(2 - \eta \lambda - \eta h)} + x_\infty^2. \quad (25)$$

Now, let us try to map the toy model described above to the scenario of LLM training, where we find a substantial simplification of the second moment in the stationary state. Typical values of learning rate and weight decay used in pretraining are $\eta \lesssim 10^{-3}$ and $\lambda \approx 0.1$. These values imply that the product $\eta \lambda$ is very small, and we can drop it compared to the terms that are of the order of 1. It is also reasonable to assume that $h \ll 1$, which also implies $\frac{h}{\lambda} \ll 1$ for practical values of λ . Indeed, the steepness h roughly corresponds to quadratic approximation of the loss w.r.t. to the chosen parameter $L(x) \approx \frac{1}{2}h(x - x^*)^2$. Then, due to large number of parameters in the model, the sensitivity to a single parameter x should be relatively small, implying $h \ll 1$. Having two assumptions $\eta \lambda \ll 1$ and $\frac{h}{\lambda} \ll 1$ the second moment simplifies to

$$x_{\infty,2} \approx \frac{1}{2} \frac{\eta}{\lambda} \left(\sigma^2 + \frac{2(h x^*)^2}{\eta \lambda} \right). \quad (26)$$

The obtained result for $x_{\infty,2}$ has a clear interpretation as it fully separates signal and noise contributions. The first term describes the balance between weight decay contraction and Brownian motion expansion due to the noise in the updates. The second term describes the balance between, again, weight decay contraction, and attraction of the parameter x to its optimal value x^* . Importantly, these two terms have a different scaling w.r.t. learning rate η and weight decay λ .

Implication for the parameter norms. Recall that our parameter x was just a single entry of the parameter matrix W . Different entries would have its own values of σ, h, x^* but the same η, λ as those are global hyperparameters of the training algorithm. Then, we could write the dependence of parameters norm $\|W\|^2$ on learning rate and weight decay as

$$\|W\|^2 = \sum_{ij} W_{ij}^2 = \left(\frac{\sigma_{ij}^2}{2} \right) \frac{\eta}{\lambda} + \left(\sum_{ij} (h_{ij} x_{ij}^*)^2 \right) \frac{1}{\lambda^2} = C_1 \frac{\eta}{\lambda} + C_2 \frac{1}{\lambda^2}. \quad (27)$$

As the dependence of the parameter norms on η, λ can be accurately measured for the actual LLM training, we could check which of the two scalings more accurately describes the experimental data. As we have seen in section 3.2.2, the scaling $\|W\|^2 \propto \frac{\eta}{\lambda}$ very well captures the experiments values, implying $C_1 \frac{\eta}{\lambda} \gg C_2 \frac{1}{\lambda^2}$ for typical values of η, λ . This suggest that, at least for the most of the parameters, the noise in the updates dominates the attraction to the optimal values.

C. Tuning μP multipliers

To jointly tune all of our $M = 35$ multipliers, we have used a stagewise procedure comprised of micro-sweeps over each multiplier on each stage. With a slight abuse of notations, denote $\mathcal{M}_n = \{m_n^{(1)}, \dots, m_n^{(M)}\}$ the set of all μP multipliers at stage n . At each stage, we repeat the following steps:

1. Start the stage by running a training job with \mathcal{M}_n , and measure the final loss L_n that will serve as a baseline loss for the current stage.
2. For each multiplier $i = 1 \dots k$ run a log-scale microsweep: two training jobs with increased i 'th multiplier $m_{n,+}^{(i)} = pm_n^{(i)}$ and decreased $m_{n,-}^{(i)} = p^{-1}m_n^{(i)}$, and measure the respective losses $L_{n,+}^{(i)}$ and $L_{n,-}^{(i)}$. The scaling factor p is chosen to balance exploration and precision. We have used $p = 2$ at the beginning of the tuning to converge faster to the vicinity of the optimal set of multipliers, while switching to $p = \sqrt{2}$ towards the end of the tuning for the increased precision.
3. For each multiplier $m_n^{(i)}$ we manually inspected decreased, baseline, and increased losses $(L_{n,-}^{(i)}, L_n, L_{n,+}^{(i)})$ to pick the value of the next stage multiplier $m_{n+1}^{(i)}$. Most of the time we pick this value from $\{m_{n,-}^{(i)}, m_n^{(i)}, m_{n,+}^{(i)}\}$ depending on which of the respective losses is lower. However, sometimes we have also picked other values, for example, at earlier stages we could pick the value beyond maximal $m_{n,+}^{(i)}$ or minimal $m_{n,-}^{(i)}$ to increase exploration. Also, sometimes we picked an intermediate value, for example, $\sqrt{m_n^{(i)} m_{n,+}^{(i)}}$ if two losses $L_n, L_{n,+}^{(i)}$ turned out roughly the same and significantly better than $L_{n,-}^{(i)}$.
4. Construct the next stage set of multipliers $\mathcal{M}_{n+1} = \{m_{n+1}^{(1)}, \dots, m_{n+1}^{(M)}\}$ from the values picked on the step 3, then go to step 1 to start stage $n + 1$.

The above procedure is quite simple, and we expect it can be improved in the future to achieve a faster convergence to the optimal set of multipliers. However, the simplicity and manual elements of our procedure was important to build an intuition behind the roles of different multipliers in the training process, as we were also checking the whole training curve for spikes, the noise level (the loss difference before and after LR decay), and any possible anomalies in the training.

Another benefit of our procedure is the possibility to measure the sensitivity of the loss to each multiplier. Indeed, at the end of the procedure we obtain a set of multipliers that is close to optimality, and the dependence of the loss on the logarithm of each multiplier can be roughly approximated by a quadratic function $L(m) \approx \frac{a}{2}(\log m - \log m^*)^2 + L^*$. We fit our three loss measurements $(L_{n,-}^{(i)}, L_n, L_{n,+}^{(i)})$ with this quadratic dependence to estimate the sensitivity $a = \frac{\partial^2}{(\partial \log m)^2} L$, reported in Figure 12.

Effective learning rate and weight decay. Recall that we have 4 types of μP multipliers, grouped as in table 8

1. Forward multipliers $m^{(i)}$.
2. LR multipliers $\eta^{(j)}/\eta$ for matrix-like layers with enabled weight decay.
3. WD multipliers $\lambda^{(j)}/\lambda$ for the same matrix-like layers with enabled weight decay.
4. LR multipliers $\eta^{(k)}/\eta$ for vector-like layers with disabled weight decay.

Our tuning procedure in the space of multipliers has a coordinate-aligned structure, similar to coordinate descent. Therefore, it is beneficial for its convergence and interpretation to make different coordinates as independent as possible. We expect that forward multipliers and LR multipliers for vector-like layers are already quite independent from each other. However, as discussed in section 3.2.2, LR and WD multipliers for the same matrix-like layer are expected to be strongly coupled.

We aim to decrease the coupling by changing the coordinate axes to ELR and EWD (12), and tuning ELR multiplier $\sqrt{\eta^{(j)} \lambda^{(j)}} / \sqrt{\eta \lambda}$ and EWD multiplier $\sqrt{\frac{\lambda^{(j)}}{\eta^{(j)}}} / \sqrt{\frac{\lambda}{\eta}}$ instead of plane LR and WD multipliers. Specifically, for ELR micro-sweep at stage n we change LR and WD values as $(\eta_n^{(j)}, \lambda_n^{(j)}) \rightarrow (p\eta_n^{(j)}, p\lambda_n^{(j)})$, and for EWD micro-sweep as $(\eta_n^{(j)}, \lambda_n^{(j)}) \rightarrow (p\eta_n^{(j)}, p^{-1}\lambda_n^{(j)})$. Accordingly, in figure 12 we also report sensitivities $\frac{\partial^2}{(\partial \log \eta_{\text{eff}})^2} L$ and $\frac{\partial^2}{(\partial \log \lambda_{\text{eff}})^2} L$ instead of the raw LR/WD sensitivities.

Other settings, and applied scaling relations. The shapes of the model used for multiplier tuning are reported in table 7. As for the training job settings, we have used WSD learning rate schedule with 65GT of the stable stage, and 10GT of exponential decay that reduces LR 32 times. The global LR and WD values are $\eta = 256 \times 10^{-6}$ and $\lambda = 0.1$, sequence length 2048, and the global batch size of 4 million tokens (MT).

As we expect all the hyperparameters to be well-tuned by the end of multiplier tuning, it is essential to correctly transfer them to other model and training settings. Here are the scaling relations we have applied for the Falcon-H1 training.

- For the μP multipliers, we have the scaling version that scales forward multipliers, as described in table 7, while keeping LR and WD values unchanged. We did not use scaling of multipliers with model depth L , for simplicity and due to time constraints, but it can be directly applied following recent work (Yang et al., 2023; Dey et al., 2025).
- When changing the batch size, we have used the square root scaling of the learning rate (19).
- When changing the total training duration, we have applied our effective power scheduler (EPS) for the learning rate and weight decay of the stable stage of WSD, as described in section 3.2.2. The activation is the power scheduler should be set close to the duration of the stable stage our tuning jobs - 65GT. However, EPS can be activated later to ensure higher adaptivity in the later stages of training, for example, if curriculum learning or long context extension at the end of the training are used.

D. Detailed evaluation results

D.1 Multilingual Evaluations - Base Models

Task	Falcon-H1-1.5B-deep	Falcon-H1-1.5B	Qwen3-1.7B	Qwen2.5-1.5B	Gemma3-1B	Llama3.2-1.2B	Falcon3-1.6B
Multilingual Hellaswag							
Arabic (ar)	41.63	39.01	40.93	38.55	40.62	34.96	28.69
German (de)	50.44	46.55	47.18	43.48	45.56	41.47	30.82
Spanish (es)	58.52	54.69	54.20	51.98	52.22	48.13	37.49
French (fr)	57.25	53.68	53.07	50.30	50.93	46.07	37.45
Hindi (hi)	35.61	33.22	34.01	30.43	35.63	32.97	28.26
Italian (it)	54.39	50.21	50.53	46.36	49.11	44.23	31.84
Dutch (nl)	50.89	47.09	45.46	41.81	47.24	42.22	29.99
Portuguese (pt)	56.57	51.99	52.41	50.60	50.62	45.84	33.69
Romanian (ro)	48.17	43.74	42.86	35.71	43.46	39.04	29.63
Russian (ru)	51.20	48.04	48.16	45.31	45.77	42.12	29.08
Swedish (sv)	49.32	44.53	42.36	37.25	46.40	40.69	28.65
Average	50.36	<u>46.62</u>	46.47	42.89	46.14	41.61	31.42
Multilingual MMLU							
Arabic (ar)	43.92	40.17	-	42.67	26.30	26.03	27.29
German (de)	54.24	48.49	-	49.88	27.27	28.68	32.94
Spanish (es)	57.33	51.73	-	53.65	26.92	29.00	35.98
French (fr)	56.54	50.66	-	52.67	25.73	28.33	35.22
Hindi (hi)	40.69	37.72	-	34.47	26.04	27.24	27.93
Italian (it)	55.11	48.46	-	50.84	25.90	28.27	32.28
Dutch (nl)	53.21	46.54	-	48.31	26.98	29.00	30.67
Portuguese (pt)	57.04	51.53	-	53.23	27.09	28.61	33.99
Romanian (ro)	52.49	46.67	-	44.74	27.03	27.03	30.29
Russian (ru)	50.35	44.71	-	47.57	26.36	28.50	28.50
Swedish (sv)	51.92	44.53	-	45.16	25.90	28.78	29.76
Chinese (zh)	51.23	46.87	-	53.94	26.49	29.25	33.89
Average	52.00	46.51	-	<u>48.09</u>	26.50	28.22	31.56
Multilingual GSM (MGSM)							
German (de)	64.40	48.40	-	41.60	-	6.00	8.40
Spanish (es)	68.00	61.20	-	54.40	-	6.00	14.40
French (fr)	61.20	57.60	-	44.00	-	4.80	14.40
Japanese (ja)	42.80	31.60	-	34.00	-	2.80	2.00
Russian (ru)	64.40	54.00	-	44.40	-	3.60	3.60
Chinese (zh)	61.20	52.00	-	52.40	-	5.20	13.60
Average	60.33	<u>50.80</u>	-	45.13	-	4.73	9.40

Table 28: Performance comparison of **Base** models on multilingual tasks (1B-2B scale). All scores are percentages. The best average score is in bold, and the second-best is underlined.

Task	Falcon-H1-3B	Qwen3-4B	Qwen2.5-3B	Gemma3-4B	Llama-3.2-3B	Falcon3-3B
Multilingual Hellaswag						
Arabic (ar)	45.42	49.63	49.50	52.14	43.59	29.52
German (de)	56.29	58.09	57.29	61.36	54.74	35.01
Spanish (es)	62.98	64.69	60.15	67.09	61.75	48.95
French (fr)	62.52	63.76	59.32	66.62	59.99	48.33
Hindi (hi)	39.35	41.12	40.40	46.26	41.03	28.71
Italian (it)	58.90	61.86	58.34	64.44	57.78	36.45
Dutch (nl)	56.43	56.52	56.33	63.17	55.59	31.60
Portuguese (pt)	61.39	63.25	59.85	65.85	59.40	47.73
Romanian (ro)	53.09	53.68	52.76	60.48	50.63	31.21
Russian (ru)	55.48	57.39	55.47	59.76	53.99	31.26
Swedish (sv)	54.79	53.67	52.50	64.14	54.39	30.60
Average	55.15	<u>56.69</u>	54.71	61.03	53.89	36.30
Multilingual MMLU						
Arabic (ar)	46.98	57.70	49.50	47.14	39.24	29.77
German (de)	57.82	66.33	57.29	54.22	47.33	39.98
Spanish (es)	60.08	68.65	60.15	55.68	49.06	47.47
French (fr)	58.81	68.28	59.32	54.92	48.44	46.89
Hindi (hi)	42.53	52.64	40.40	45.28	37.50	29.95
Italian (it)	58.17	68.10	58.34	54.42	47.84	39.84
Dutch (nl)	56.48	65.99	56.33	53.69	46.96	36.11
Portuguese (pt)	59.37	68.79	59.85	54.97	48.45	46.56
Romanian (ro)	55.54	65.50	52.76	53.99	45.69	35.91
Russian (ru)	53.53	65.01	55.47	51.58	44.56	32.51
Swedish (sv)	54.98	64.78	52.50	53.61	45.76	34.70
Chinese (zh)	53.05	67.22	59.63	51.36	44.54	40.07
Average	54.78	64.91	<u>55.13</u>	52.57	45.45	38.31
Multilingual GSM (MGSM)						
German (de)	64.80	-	58.80	-	23.20	28.80
Spanish (es)	72.00	-	68.00	-	24.40	46.80
French (fr)	70.40	-	62.40	-	22.00	48.00
Japanese (ja)	51.20	-	45.60	-	12.80	8.80
Russian (ru)	64.00	-	62.00	-	16.80	14.80
Chinese (zh)	61.60	-	62.80	-	22.80	43.60
Average	64.00	-	<u>59.93</u>	-	20.33	31.80

Table 29: Performance comparison of **Base** models on multilingual tasks (3B-4B scale). All scores are percentages. The best average score is in bold, and the second-best is underlined.

Task	Falcon-H1-7B	Qwen3-8B	Qwen2.5-7B	Gemma3-12B	Llama3.1-8B	Falcon3-7B	Falcon3-10B
Multilingual Hellaswag							
Arabic (ar)	54.57	54.24	52.23	62.62	50.14	31.11	34.21
German (de)	66.28	64.09	60.58	71.81	63.13	44.52	51.18
Spanish (es)	72.71	70.30	68.82	76.82	69.71	67.57	71.36
French (fr)	71.73	68.86	67.89	75.32	67.95	66.96	70.25
Hindi (hi)	47.65	45.82	38.72	53.57	45.75	30.15	31.09
Italian (it)	69.71	66.86	63.57	74.01	66.37	51.34	56.61
Dutch (nl)	67.47	62.51	60.76	73.36	64.21	40.14	45.87
Portuguese (pt)	70.92	68.44	67.94	75.28	68.06	65.98	69.49
Romanian (ro)	64.65	59.64	49.32	70.74	59.48	38.09	42.91
Russian (ru)	64.52	62.39	61.60	68.95	60.25	37.55	43.17
Swedish (sv)	66.61	60.32	54.65	74.31	63.91	38.91	43.85
Average	<u>65.16</u>	62.13	58.74	70.62	61.72	46.58	50.91
Multilingual MMLU							
Arabic (ar)	60.93	62.06	58.19	61.96	46.98	-	35.33
German (de)	69.19	70.11	65.69	68.32	55.48	-	55.57
Spanish (es)	71.75	71.82	68.19	69.24	57.31	-	66.99
French (fr)	70.85	71.56	67.88	69.27	56.93	-	67.09
Hindi (hi)	57.06	57.92	49.65	59.03	43.93	-	35.81
Italian (it)	70.68	71.78	66.68	69.08	56.51	-	59.45
Dutch (nl)	69.74	70.06	65.73	69.04	55.54	-	51.94
Portuguese (pt)	70.93	71.67	68.48	68.96	57.30	-	66.98
Romanian (ro)	68.48	69.52	63.02	69.09	53.87	-	51.06
Russian (ru)	67.74	69.20	65.47	67.10	52.93	-	44.80
Swedish (sv)	67.82	68.71	62.92	68.44	54.11	-	49.14
Chinese (zh)	65.40	70.13	66.96	66.13	52.17	-	53.82
Average	<u>67.55</u>	68.71	64.07	67.14	53.58	-	53.17
Multilingual GSM (MGSM)							
German (de)	75.20	75.60	74.00	-	41.20	51.20	60.00
Spanish (es)	83.20	82.00	75.20	-	52.80	71.20	73.60
French (fr)	77.20	77.60	70.80	-	40.00	65.60	69.20
Japanese (ja)	62.00	58.00	59.60	-	28.00	24.00	34.80
Russian (ru)	76.40	81.20	74.40	-	44.40	38.80	49.20
Chinese (zh)	73.20	32.80	72.40	-	42.80	62.40	67.20
Average	74.53	67.87	<u>71.07</u>	-	41.53	52.20	59.00

Table 30: Performance comparison of **Base** models on multilingual tasks (7B-12B scale). All scores are percentages. The best average score is in bold, and the second-best is underlined.

Task	Falcon-H1-34B	Qwen2.5-72B	Gemma3-27B	Llama4-scout	Llama3.1-70B
Multilingual Hellaswag					
Arabic (ar)	62.95	63.59	66.36	64.08	65.19
German (de)	74.30	73.06	75.46	73.55	76.53
Spanish (es)	79.68	78.75	79.80	77.96	80.86
French (fr)	78.18	77.19	78.52	76.84	78.92
Hindi (hi)	54.53	54.08	57.31	55.99	58.46
Italian (it)	76.86	76.06	77.58	75.17	78.36
Dutch (nl)	75.54	73.93	76.77	74.71	77.65
Portuguese (pt)	78.26	77.68	78.87	76.77	80.37
Romanian (ro)	72.47	66.87	74.04	72.17	74.38
Russian (ru)	71.10	70.88	71.84	70.87	72.66
Swedish (sv)	74.93	71.08	77.60	74.10	77.78
Average	72.62	<u>71.20</u>	<u>74.01</u>	72.02	74.65
Multilingual MMLU					
Arabic (ar)	70.83	73.52	68.10	67.43	65.46
German (de)	78.23	80.28	73.31	73.78	72.10
Spanish (es)	79.65	80.63	74.56	74.33	73.33
French (fr)	79.67	81.03	74.13	74.60	73.45
Hindi (hi)	67.16	68.94	65.46	64.68	63.38
Italian (it)	79.46	81.03	74.12	74.22	73.79
Dutch (nl)	78.69	80.32	74.33	74.02	72.73
Portuguese (pt)	79.64	80.88	74.47	73.95	73.78
Romanian (ro)	78.42	79.03	73.97	73.60	72.52
Russian (ru)	76.65	79.08	71.96	72.55	71.28
Swedish (sv)	78.13	79.03	73.54	74.23	72.08
Chinese (zh)	74.60	78.78	70.94	71.27	69.39
Average	76.76	78.54	72.41	72.38	71.10
Multilingual GSM (MGSM)					
German (de)	82.40	83.20	-	76.00	71.20
Spanish (es)	85.60	86.00	-	77.60	79.60
French (fr)	80.00	77.60	-	72.40	69.20
Japanese (ja)	76.80	80.80	-	70.80	61.20
Russian (ru)	88.00	83.60	-	79.20	74.40
Chinese (zh)	81.60	82.00	-	78.80	68.80
Average	82.40	<u>82.20</u>	-	75.80	70.73

Table 31: Performance comparison of **Base** models on multilingual tasks (34B scale). All scores are percentages. The best average score is in bold, and the second-best is underlined.

D.2 Multilingual Evaluations - Instruct Models

Task	Falcon-H1-1.5B-deep	Falcon-H1-1.5B	Qwen3-1.7B	Qwen2.5-1.5B	Gemma3-1B	Llama3.2-1.2B	Falcon3-1.6B
Multilingual Hellaswag							
Arabic (ar)	44.22	41.51	36.19	39.20	37.65	33.38	29.49
German (de)	53.83	49.97	37.95	43.04	42.09	40.65	31.31
Spanish (es)	61.50	57.67	41.92	52.07	47.22	46.02	38.79
French (fr)	60.54	56.51	41.45	50.50	46.29	44.04	38.52
Hindi (hi)	37.35	34.98	30.68	30.24	33.42	32.89	28.50
Italian (it)	57.96	53.54	40.05	46.32	44.62	43.09	32.76
Dutch (nl)	52.94	49.98	36.89	41.76	40.11	39.19	29.67
Portuguese (pt)	60.01	55.80	41.64	50.89	45.66	44.22	35.22
Romanian (ro)	50.44	45.81	36.01	35.70	39.81	37.10	30.07
Russian (ru)	54.16	50.66	38.22	45.85	42.27	38.36	29.22
Swedish (sv)	51.58	46.80	35.80	36.67	40.39	38.60	28.88
Average	53.14	<u>49.38</u>	37.89	42.93	41.77	39.78	32.04
Multilingual MMLU							
Arabic (ar)	45.13	41.21	35.53	41.51	32.52	30.00	27.48
German (de)	55.02	50.57	43.92	48.04	36.11	36.75	33.91
Spanish (es)	57.85	53.09	45.40	51.42	36.25	38.29	37.49
French (fr)	57.47	51.91	44.90	51.08	36.06	37.66	36.87
Hindi (hi)	40.74	38.05	28.98	33.27	31.66	32.39	27.92
Italian (it)	55.94	50.56	45.63	49.04	35.54	37.01	33.71
Dutch (nl)	54.54	48.97	41.63	46.72	35.90	36.60	31.00
Portuguese (pt)	57.55	52.09	30.76	51.82	34.94	36.04	35.80
Romanian (ro)	53.83	48.32	40.63	43.32	34.92	32.81	30.97
Russian (ru)	51.93	46.84	40.59	45.89	35.37	35.27	29.12
Swedish (sv)	53.02	47.10	37.62	42.82	34.76	34.87	30.46
Chinese (zh)	51.53	47.16	50.38	52.95	33.66	37.68	33.60
Average	53.00	<u>48.06</u>	39.60	45.90	34.91	35.24	32.25
Multilingual GSM (MGSM)							
German (de)	61.20	61.20	50.80	42.00	-	32.40	15.20
Spanish (es)	70.00	71.20	60.80	49.20	-	33.60	24.40
French (fr)	50.40	57.60	45.20	46.80	-	34.00	24.00
Japanese (ja)	54.40	41.20	44.80	34.40	-	19.60	3.20
Russian (ru)	57.60	55.60	56.00	43.60	-	29.20	5.20
Chinese (zh)	66.40	61.20	56.80	55.20	-	29.60	20.00
Average	60.00	<u>58.00</u>	52.40	45.20	-	29.73	15.33

Table 32: Performance comparison of **Instruct** models on multilingual tasks (1B-2B scale). All scores are percentages. The best average score is in bold, and the second-best is underlined.

Task	Falcon-H1-3B	Qwen3-4B	Qwen2.5-3B	Gemma3-4B	Llama3.2-3B	Falcon3-3B
Multilingual Hellaswag						
Arabic (ar)	49.37	40.48	46.57	48.69	40.52	29.83
German (de)	59.27	44.51	51.41	54.22	52.90	36.47
Spanish (es)	66.48	46.99	60.17	60.26	58.80	51.19
French (fr)	65.88	46.82	60.09	59.55	56.92	50.98
Hindi (hi)	42.32	33.66	33.33	43.19	40.32	28.47
Italian (it)	62.44	45.45	55.11	58.01	55.04	38.47
Dutch (nl)	58.83	43.19	51.36	53.96	51.96	32.35
Portuguese (pt)	64.68	46.70	59.53	60.11	56.39	50.44
Romanian (ro)	55.55	40.99	42.35	53.95	47.12	32.15
Russian (ru)	59.04	42.83	53.42	54.26	49.12	31.22
Swedish (sv)	57.83	42.63	45.54	53.08	51.17	31.01
Average	58.34	43.12	50.81	<u>54.48</u>	50.93	37.51
Multilingual MMLU						
Arabic (ar)	47.57	41.65	48.21	46.24	41.03	30.39
German (de)	56.87	58.70	55.35	52.27	50.76	40.85
Spanish (es)	59.88	56.14	57.83	53.69	52.76	48.45
French (fr)	58.99	59.38	57.61	53.07	51.52	47.32
Hindi (hi)	42.42	36.39	38.45	44.27	40.09	30.17
Italian (it)	57.89	57.34	56.35	53.49	51.24	41.55
Dutch (nl)	55.99	56.59	54.34	52.30	50.31	37.47
Portuguese (pt)	59.29	26.74	58.13	52.54	52.25	47.96
Romanian (ro)	55.12	56.48	50.81	51.75	47.94	36.81
Russian (ru)	54.14	54.39	54.39	50.48	46.22	33.32
Swedish (sv)	55.30	53.83	50.23	52.03	48.60	33.88
Chinese (zh)	53.83	55.88	58.34	50.13	47.77	39.80
Average	54.90	50.70	<u>52.90</u>	51.10	48.40	38.90
Multilingual GSM (MGSM)						
German (de)	63.60	68.80	58.00	-	63.60	41.60
Spanish (es)	71.60	73.20	64.80	-	71.60	63.20
French (fr)	66.40	67.20	54.40	-	62.00	60.80
Japanese (ja)	59.60	65.20	51.20	-	51.60	14.00
Russian (ru)	54.00	65.60	54.40	-	62.80	20.40
Chinese (zh)	68.40	73.20	62.40	-	61.60	52.40
Average	<u>63.90</u>	68.90	57.30	-	62.20	42.10

Table 33: Performance comparison of **Instruct** models on multilingual tasks (3B-4B scale). All scores are percentages. The best average score is in bold, and the second-best is underlined.

Task	Falcon-H1-7B	Qwen3-8B	Qwen2.5-7B	Gemma3-12B	Llama3.1-8B	Falcon3-7B	Falcon3-10B
Multilingual Hellaswag							
Arabic (ar)	56.73	44.19	52.23	59.19	49.44	32.19	35.51
German (de)	69.51	48.34	60.57	67.64	62.29	46.58	52.95
Spanish (es)	75.94	51.06	68.81	72.53	68.58	68.74	74.00
French (fr)	75.08	52.35	67.89	70.86	66.79	68.14	72.71
Hindi (hi)	49.26	37.34	38.72	51.93	45.66	29.92	31.90
Italian (it)	72.32	49.67	63.57	69.86	64.55	52.72	59.14
Dutch (nl)	70.52	47.61	60.75	67.64	63.18	41.54	47.52
Portuguese (pt)	74.38	51.15	67.93	71.28	66.47	66.85	72.18
Romanian (ro)	66.25	44.87	49.32	66.45	59.21	39.65	44.17
Russian (ru)	66.26	47.25	61.60	65.17	59.17	39.60	44.80
Swedish (sv)	68.97	46.45	54.64	69.21	62.73	39.98	45.54
Average	67.75	47.30	58.74	<u>66.53</u>	60.74	47.81	52.77
Multilingual MMLU							
Arabic (ar)	60.44	49.11	56.23	59.40	47.17	35.34	36.26
German (de)	69.13	58.63	63.65	66.48	57.62	52.44	55.68
Spanish (es)	71.96	59.55	66.17	67.29	59.62	64.14	67.71
French (fr)	71.04	53.14	65.30	67.35	59.51	63.78	67.45
Hindi (hi)	56.52	36.61	46.15	57.41	45.31	34.33	35.24
Italian (it)	70.23	58.99	65.19	67.35	58.27	56.02	60.26
Dutch (nl)	69.83	57.66	63.05	67.02	57.01	49.28	52.79
Portuguese (pt)	71.55	23.60	64.44	67.67	58.80	63.80	67.27
Romanian (ro)	68.81	58.32	59.27	66.55	55.78	48.33	51.85
Russian (ru)	67.96	55.51	63.29	64.58	55.21	42.48	45.85
Swedish (sv)	68.65	43.71	60.43	66.31	56.57	46.79	50.02
Chinese (zh)	65.82	65.66	65.89	63.94	55.09	50.12	53.64
Average	67.83	50.44	61.20	<u>65.22</u>	55.53	50.62	53.67
Multilingual GSM (MGSM)							
German (de)	73.20	72.00	64.80	-	71.20	60.00	68.40
Spanish (es)	82.00	68.40	69.20	-	79.60	80.40	84.80
French (fr)	72.40	60.00	61.60	-	69.60	73.20	76.40
Japanese (ja)	70.80	61.60	64.80	-	56.00	30.80	45.20
Russian (ru)	67.60	60.80	61.60	-	76.40	36.00	46.80
Chinese (zh)	74.80	68.40	74.80	-	71.60	57.60	67.20
Average	73.50	65.20	66.10	-	<u>70.70</u>	56.30	64.80

Table 34: Performance comparison of **Instruct** models on multilingual tasks (7B-12B scale). All scores are percentages. The best average score is in bold, and the second-best is underlined.

Task	Falcon-H1-34B	Qwen3-32B	Qwen2.5-72B	Qwen2.5-32B	Gemma3-27B	Llama4-Scout	Llama3.3-70B
Multilingual Hellaswag							
Arabic (ar)	64.33	52.39	63.20	60.75	62.40	55.39	58.44
German (de)	76.96	59.48	70.93	67.63	70.52	62.57	64.68
Spanish (es)	80.39	62.87	76.03	72.68	74.94	69.10	69.59
French (fr)	80.56	62.45	75.19	72.60	73.97	68.13	68.62
Hindi (hi)	57.01	49.25	53.40	48.50	55.37	45.25	50.47
Italian (it)	79.07	61.38	73.61	70.21	73.64	65.44	66.97
Dutch (nl)	77.08	59.56	70.84	67.75	70.72	65.53	68.05
Portuguese (pt)	80.25	62.03	75.71	72.73	74.65	67.87	69.99
Romanian (ro)	73.95	57.83	66.06	60.83	69.93	62.80	63.34
Russian (ru)	73.92	56.58	69.53	66.75	68.31	59.93	64.55
Swedish (sv)	76.47	58.50	69.83	64.46	71.61	63.30	66.39
Average	74.55	58.39	69.48	65.90	<u>69.64</u>	62.30	64.64
Multilingual MMLU							
Arabic (ar)	72.08	65.71	73.67	68.17	66.30	68.70	70.51
German (de)	78.97	71.32	78.57	75.71	71.96	75.61	77.89
Spanish (es)	80.25	74.04	81.03	76.86	73.65	76.86	79.47
French (fr)	80.38	72.49	81.05	76.88	73.86	76.50	79.08
Hindi (hi)	68.34	56.77	69.07	61.39	63.95	67.37	68.22
Italian (it)	79.97	74.71	81.04	76.88	73.70	76.32	78.82
Dutch (nl)	79.56	67.34	79.74	75.58	72.54	76.00	78.70
Portuguese (pt)	80.49	31.56	81.25	76.21	74.24	77.06	78.99
Romanian (ro)	79.15	72.12	78.76	73.60	73.07	75.54	77.73
Russian (ru)	77.31	73.06	78.20	74.41	71.52	74.31	75.88
Swedish (sv)	78.85	69.09	78.51	73.54	72.82	76.11	78.10
Chinese (zh)	75.41	73.61	78.22	74.94	69.54	73.50	74.97
Average	77.76	66.20	78.26	73.56	71.60	74.58	76.67
Multilingual GSM (MGSM)							
German (de)	75.20	73.20	72.80	72.80	78.00	86.00	88.00
Spanish (es)	80.80	76.00	79.20	76.80	84.80	88.40	89.60
French (fr)	70.80	67.60	60.80	61.20	74.80	82.80	82.80
Japanese (ja)	78.40	69.20	74.80	81.20	76.80	78.00	84.40
Russian (ru)	68.00	68.00	63.20	68.40	72.40	84.40	88.40
Chinese (zh)	84.80	76.80	83.20	81.20	80.40	83.60	88.00
Average	76.33	71.80	72.33	73.60	<u>77.87</u>	83.87	86.87

Table 35: Performance comparison of **Instruct** models on multilingual tasks (34B scale). All scores are percentages. The best average score is in bold, and the second-best is underlined.

D.3 Long-context Evaluations - Instruct Models

Dataset	Metric	Falcon-H1-34B-Instruct	Qwen3-32B	Qwen2.5-72B-Instruct	Llama-3.3-70B-Instruct
alce_asqa	citation_prec	61.49	70.05	65.04	49.47
alce_asqa	citation_rec	61.90	63.37	69.52	66.00
alce_asqa	str_em	49.68	44.10	46.98	51.10
alce_qampari	citation_prec	23.67	29.01	28.36	20.48
alce_qampari	citation_rec	23.42	25.60	27.81	18.81
alce_qampari	qampari_rec_top5	28.80	29.20	29.40	37.00
banking77	exact_match	82.00	87.00	86.00	84.00
clinic150	exact_match	91.00	91.00	91.00	85.00
hotpotqa	substring_exact_match	66.67	69.33	67.67	71.33
infbench_choice	exact_match	55.00	46.00	61.00	50.00
infbench_qa	rougeL_f1	20.75	24.01	20.39	22.14
infbench_sum	f1	25.02	33.12	31.65	30.25
json_kv	substring_exact_match	100	100	100	100
msmarco_rerank_psg	NDCG@10	76.94	81.92	85.40	84.32
multi_lexsum	f1	32.06	34.17	34.07	31.84
narrativeqa	rougeL_f1	22.85	24.89	24.21	28.86
nlu	exact_match	74.00	79.00	83.00	83.00
nq	substring_exact_match	63.33	57.83	66.33	61.17
popqa	substring_exact_match	65.33	61.33	61.67	69.00
ruler_niah_mk_2	ruler_recall	100	100	100	100
ruler_niah_mk_3	ruler_recall	100	100	100	100
ruler_niah_mv	ruler_recall	100	100	100	100
trec_coarse	exact_match	69.00	69.00	72.00	63.00
trec_fine	exact_match	34.00	42.00	46.00	34.00
triviaqa	substring_exact_match	93.33	88.50	93.17	95.67

Table 36: Performance of HELMET tasks at sequence length 8192.

Dataset	Metric	Falcon-H1-34B-Instruct	Qwen3-32B	Qwen2.5-72B-Instruct	Llama-3.3-70B-Instruct
alce_asqa	citation_prec	49.85	71.23	55.52	48.14
alce_asqa	citation_rec	47.75	65.46	62.08	62.54
alce_asqa	str_em	52.15	46.20	43.77	50.17
alce_qampari	citation_prec	21.90	28.08	28.85	17.34
alce_qampari	citation_rec	20.91	25.96	27.88	15.64
alce_qampari	qampari_rec_top5	29.40	31.40	30.60	42.40
banking77	exact_match	86.00	90.00	91.00	91.00
clinic150	exact_match	95.00	96.00	94.00	95.00
hotpotqa	substring_exact_match	65.33	64.33	65.67	69.33
infbench_choice	exact_match	55.00	49.00	56.00	55.00
infbench_qa	rougeL_f1	28.40	31.48	32.13	32.46
infbench_sum	f1	24.92	34.91	34.08	32.11
json_kv	substring_exact_match	100	100	100	100
msmarco_rerank_psg	NDCG@10	66.15	70.24	74.89	76.01
multi_lexsum	f1	33.68	34.61	34.51	32.70
narrativeqa	rougeL_f1	20.52	26.57	29.27	31.78
nlu	exact_match	77.00	76.00	86.00	85.00
nq	substring_exact_match	57.00	56.50	67.33	61.50
popqa	substring_exact_match	67.50	60.83	62.33	64.33
ruler_niah_mk_2	ruler_recall	100	100	100	100
ruler_niah_mk_3	ruler_recall	100	100	100	100
ruler_niah_mv	ruler_recall	100	100	100	100
trec_coarse	exact_match	66.00	78.00	85.00	70.00
trec_fine	exact_match	43.00	51.00	51.00	39.00
triviaqa	substring_exact_match	93.00	86.50	93.67	95.67

Table 37: Performance of HELMET tasks at sequence length 16384.

Dataset	Metric	Falcon-H1-34B-Instruct	Qwen3-32B	Qwen2.5-72B-Instruct	Llama-3.3-70B-Instruct
alce_asqa	citation_prec	34.98	61.34	53.16	45.29
alce_asqa	citation_rec	25.52	56.04	57.80	55.78
alce_asqa	str_em	45.18	43.72	42.07	46.37
alce_qampari	citation_prec	12.13	25.43	24.18	11.12
alce_qampari	citation_rec	11.31	24.99	22.54	10.69
alce_qampari	qampari_rec_top5	16.80	31.40	27.60	45.00
banking77	exact_match	87.00	90.00	90.00	92.00
clinic150	exact_match	97.00	97.00	98.00	96.00
hotpotqa	substring_exact_match	60.67	57.67	64.33	66.00
infbench_choice	exact_match	56.00	62.00	57.00	68.00
infbench_qa	rougeL_f1	29.26	36.47	32.36	41.08
infbench_sum	f1	22.87	35.47	35.21	31.58
json_kv	substring_exact_match	99	100	98	100
msmarco_rerank_psg	NDCG@10	50.35	54.98	60.66	65.71
multi_lexsum	f1	34.62	32.59	36.73	33.27
narrativeqa	rougeL_f1	19.99	24.97	28.29	33.50
nlu	exact_match	84.00	80.00	82.00	87.00
nq	substring_exact_match	60.83	57.00	63.00	59.17
popqa	substring_exact_match	58.00	57.50	59.33	59.50
ruler_niah_mk_2	ruler_recall	97.00	100	100	100
ruler_niah_mk_3	ruler_recall	95.00	100	96.00	99.00
ruler_niah_mv	ruler_recall	99.00	100	99.50	99.50
trec_coarse	exact_match	64.00	64.00	90.00	70.00
trec_fine	exact_match	44.00	50.00	47.00	46.00
triviaqa	substring_exact_match	92.33	87.17	93.67	96.17

Table 38: Performance of HELMET tasks at sequence length 32768.

Dataset	Metric	Falcon-H1-34B-Instruct	Qwen3-32B	Qwen2.5-72B-Instruct	Llama-3.3-70B-Instruct
alce_asqa	citation_prec	10.91	47.02	31.50	40.17
alce_asqa	citation_rec	4.09	45.37	34.93	43.70
alce_asqa	str_em	18.98	45.68	44.18	41.12
alce_qampari	citation_prec	2.82	16.15	7.76	8.57
alce_qampari	citation_rec	1.75	14.03	6.09	8.07
alce_qampari	qampari_rec_top5	0.40	29.20	19.60	37.60
banking77	exact_match	92.00	92.00	93.00	96.00
clinic150	exact_match	96.00	98.00	97.00	97.00
hotpotqa	substring_exact_match	63.00	55.33	54.67	61.00
infbench_choice	exact_match	56.00	65.00	62.00	74.00
infbench_qa	rougeL_f1	23.32	42.56	18.15	38.74
infbench_sum	f1	21.59	38.51	33.99	30.11
json_kv	substring_exact_match	87	99	54	100
msmarco_rerank_psg	NDCG@10	30.60	37.83	40.71	41.04
multi_lexsum	f1	35.48	35.20	33.21	33.58
narrativeqa	rougeL_f1	18.03	34.85	29.98	32.97
nlu	exact_match	83.00	83.00	82.00	86.00
nq	substring_exact_match	57.67	55.00	56.50	60.83
popqa	substring_exact_match	51.67	52.50	52.50	59.00
ruler_niah_mk_2	ruler_recall	76.00	94.00	80.00	99.00
ruler_niah_mk_3	ruler_recall	66.00	97.00	54.00	97.00
ruler_niah_mv	ruler_recall	93.75	96.00	99.00	99.25
trec_coarse	exact_match	80.00	84.00	90.00	79.00
trec_fine	exact_match	60.00	64.00	55.00	52.00
triviaqa	substring_exact_match	96.00	85.00	89.33	95.50

Table 39: Performance of HELMET tasks at sequence length 65536.

Dataset	Metric	Falcon-H1-34B-Instruct	Qwen3-32B	Qwen2.5-72B-Instruct	Llama-3.3-70B-Instruct
alce_asqa	citation_prec	3.00	41.31	3.07	6.40
alce_asqa	citation_rec	0.53	37.40	2.76	4.80
alce_asqa	str_em	9.12	43.70	35.58	37.38
alce_qampari	citation_prec	2.77	11.31	0.00	1.53
alce_qampari	citation_rec	1.44	9.43	0.00	1.13
alce_qampari	qampari_rec_top5	0.40	21.00	4.40	18.20
banking77	exact_match	93.00	97.00	90.00	87.00
clinic150	exact_match	96.00	93.00	96.00	92.00
hotpotqa	substring_exact_match	56.00	47.67	30.67	43.67
infbench_choice	exact_match	62.00	75.00	61.00	58.00
infbench_qa	rougeL_f1	24.82	46.94	13.16	45.26
infbench_sum	f1	19.44	38.90	30.67	32.79
json_kv	substring_exact_match	49	82	15	87
msmarco_rerank_psg	NDCG@10	19.63	29.75	24.86	25.47
multi_lexsum	f1	34.18	34.52	32.48	32.59
narrativeqa	rougeL_f1	14.60	38.61	24.67	34.93
nlu	exact_match	87.00	89.00	85.00	82.00
nq	substring_exact_match	53.83	49.00	38.33	47.33
popqa	substring_exact_match	48.17	48.17	34.67	47.50
ruler_niah_mk_2	ruler_recall	43.00	84.00	43.00	66.00
ruler_niah_mk_3	ruler_recall	40.00	84.00	15.00	79.00
ruler_niah_mv	ruler_recall	94.50	94.50	82.25	96.75
trec_coarse	exact_match	66.00	85.00	92.00	66.00
trec_fine	exact_match	46.00	67.00	48.00	63.00
triviaqa	substring_exact_match	90.83	83.50	65.67	83.00

Table 40: Performance of HELMET tasks at sequence length 131072.

E. Training Data

E.1 Synthetic English Data Topics

Acoustics	Fields of finance	Patent law
Administrative law	Fields of history	Photonics
Algebra	Fields of mathematics	Physical chemistry
Analytical chemistry	Gases	Privacy law
Animal law	Genetics	Private law
Astrophysics	Geometry	Probability
Branches of biology	Health care	Property law
Branches of genetics	Health law	Public law
Branches of philosophy	Health research	Quantum mechanics
Branches of psychology	Health sciences	Religious law
Business law	Historical eras	Religious legal systems
Calculus	History by continent and topic	Sex laws
Chinese law	History by topic	Social law
Civil law (common law)	History of medical and surgical specialties	Specialist law enforcement agencies
Civil law legal systems	History of the United States	Statistics
Civil procedure	Housing law	Statutory law by topic
Civil rights case law	Human anatomy by organ	Subfields of chemistry
Combinatorics	Human anatomy by system	Subfields of computer science
Common law legal systems	Human physiology	Subfields of economics
Condensed matter physics	Immigration law	Subfields of physics
Contract law	Innovation economics	Tax law
Criminal law	Inorganic chemistry	Theory of relativity
Cultural history by period	Intellectual property law	Tort law
Decrees	International law	Types of accounting
Delegated legislation	Labour law	Types of marketing
Determinants of health	Legal terminology by type of law	Workplace
Edicts	Linear algebra	
Electromagnetism	Management theory	
Emergency laws	Marketing techniques	
Engineering disciplines	Medical diagnosis	
Entrepreneurship	Medical procedures	
Environmental law	Medical specialties	
Family law	Medical treatments	

Table 41: List of topics crawled from Wikipedia category tree starting nodes.

E.2 Programming Languages

The 67 included programming languages are: Agda, Assembly, Batchfile, BibTex, C, C#, C++, CMake, COBOL, Coq, CSS, Dart, Dockerfile, F, Fortran, Go, Haskell, HTML, Idris, Isabelle,

Isabelle, ROOT, Java, JavaScript, JSON, Julia, Kotlin, LabVIEW, Lean, Literate Agda, Lua, Makefile, Maple, Markdown, Mathematica, MATLAB, Nix, NumPy, Objective-C, Objective-C++, Octave, Pascal, Pep8, Perl, PHP, Pickle, PowerShell, Python, Q#, R, Ruby, Rust, SAS, Scala, Scilab, Shell, Swift, SystemVerilog, TeX, TypeScript, VBA, Verilog, VHDL, VisualBasic_.NET, XML, XSLT, YAML.

E.3 Code Quality Classifier

The code quality classifier we run over the multi-programming language corpus supports the following programming languages: Assembly, C, C#, C++, CSS, Dart, Go, HTML, Java, JavaScript, Kotlin, Lua, PHP, PowerShell, Python, Ruby, Rust, Shell, SQL. Then, these are the only programming languages included in the HQ code corpus.