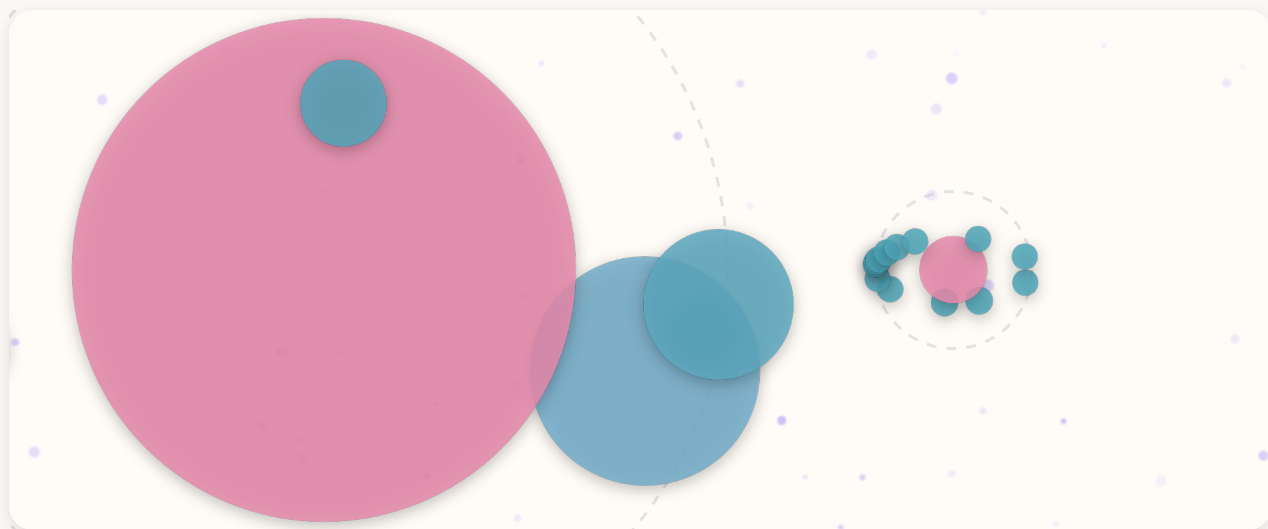


Falcon-H1-Tiny: A series of extremely small, yet powerful language models redefining capabilities at small scale



Falcon-H1-Tiny model series is a series of extremely small language models ($\leq 100M$ parameters) with remarkable end performance on specific target domains. We also release a State-Of-the-Art small reasoning model, Falcon-H1-Tiny-R-0.6B.

AUTHOR

[Falcon-LLM Team](#)

AFFILIATION PUBLISHED

[Technology Innovation Institute](#) Jan. 15, 2026

Introduction

Today, we are excited to introduce *Falcon-Tiny H1* series of models to the open-source community: a series of extremely small, remarkably powerful language models for everyone, covering popular use cases such as general chatbot assistance (general SFT model), multilingual, coding, agentic tasks as well as state of the art reasoning models for its size.

Beyond releasing models, we share to the community learnings around model architecture choices, packed with novel optimization algorithms and training data strategies, which were crucial to obtain the state-of-the-art results.






Throughout this release, we pave the way of a future that might rely on a series of tiny specialized models rather than bigger and more generalist models - for multiple viable scenarios, and release all model artifacts for the community to build new use cases, enhance existing specialized models or explore research ideas on top of them.

Our release includes:

- A concrete application of a novel model optimization paradigm which combines Learnable Multipliers (LRM) with Muon optimizer.
- A family of extremely small, state-of-the-art language models (90M parameters for English; 100M for multilingual), each trained separately on specific domains.
- A state-of-the-art 0.6B reasoning model pretrained directly on long reasoning traces, outperforming larger reasoning model variants.
- Key insights into pretraining data strategies for building more capable language models targeted at specific domains.

List of released artifacts

Below is the list of artifacts that is being open-sourced in this release. We also provide a brief description of each released model as follows:

- General usage English models:
- `falcon-h1-english` - **90M parameters** -
- `Falcon-H1-Tiny-90M-Base` : A base model trained on an English-heavy data mixture, similar to the Falcon-H1 pretraining setup.
- `Falcon-H1-Tiny-90M-Instruct-Curriculum` : A supervised fine-tuned (SFT) model initialized from the English base checkpoint. A lightweight DPO stage is applied on top.
- `Falcon-H1-Tiny-90M-Instruct` : A model pretrained from scratch using SFT data, followed by a lightweight DPO stage. This checkpoint can serve both as a base and an SFT model.
- `Falcon-H1-Tiny-90M-Instruct-Curriculum-pre-DPO` : As per the name, this model corresponds to `Falcon-H1-Tiny-90M-Instruct-Curriculum` before the DPO stage.
- `Falcon-H1-Tiny-90M-Instruct-pre-DPO` : same as above.

- General usage multilingual models - **100M parameters** -
 - `falcon-h1-multilingual` (100M):
 - `Falcon-H1-Tiny-Multilingual-100M-Base` ↗: a 100M language model pretrained on a mix of multilingual and high quality English data
 - `Falcon-H1-Tiny-Multilingual-100M-Instruct` ↗: Falcon-H1-Tiny-Multilingual obtained after applying a DPO stage on top of the sft checkpoint.
- Small Reasoning models - **600M and 90M parameters** -
- `Falcon-H1-Tiny-R-0.6B` ↗: state of the art 600M language model pretrained directly on Reasoning data, while doing a GRPO stage on top of it.
 - `Falcon-H1-Tiny-R-0.6B-pre-GRPO` ↗: Falcon-H1-Tiny-R checkpoint before the GRPO RL stage.
- `Falcon-H1-Tiny-R-90M` ↗: A 90M language models pretrained on the same data mixture as Falcon-H1-Tiny-R.
- Small Specialized models - **90M parameters** -
- `Falcon-H1-Tiny-Coder-90M` ↗: a powerful 90M language model trained on code data, which performs code generation and Fill in the Middle (FIM) tasks.
- `Falcon-H1-Tiny-Tool-Calling` ↗: a powerful 90M parameter model specialized in tool calling across all major patterns—parallel, sequential, and multi-turn function calling.

Motivations

For Falcon-H1 we have explored multi-epoch training on high-quality datasets and observed that it leads to improvement in model capability without apparent overfitting or other negative side effects. The key innovation to enable such safe data repetition is to estimate model's *memorization window* and ensure that the data is not repeated within this window. Such multi-epoch training opens new possibilities of building small specialized models by pretraining directly on the target specialized data from scratch. One of the main goals of Falcon-H1-Tiny project is to explore whether such *Anti-curriculum* strategy indeed leads to stronger specialized models.

Beyond data-centric approaches to enhance model capabilities, our prior experience highlights the importance of model hyperparameters, particularly architectural choices. For instance, variations in model depth can lead to radically different outcomes: Falcon-H1-1.5B-Deep achieves performance close to that of 7B-scale models, demonstrating

that architectural design alone can significantly impact model effectiveness beyond parameter count.

Separately, follow-up work on Falcon-H1 led us to explore optimization-related improvements. In particular, we investigated the interaction between MuP-based optimization, used in the main Falcon-H1 training, and the Muon optimizer. This line of work resulted in the development of Learnable Multipliers (LRM), which are employed in this project.

Together, these findings suggest that, in addition to data, architectural hyperparameters and optimization strategies represent two independent and powerful levers for enhancing model performance.

In this project, we deliberately fixed an upper bound on model size at a remarkably small scale (90M parameters) and focused on pushing performance to its limits by systematically exploring two axes: training data strategies and model architecture and optimization hyperparameter choices.

Covered sections

In this blogpost, we cover the following sections, which can be seen as independent sections between each other:

- *Learnable Multipliers (LRM) and Muon*: Brief introduction on the findings around our recent work – Learnable multipliers: freeing the scale of Language Model Matrix Layers – and how we applied it in our model series
- *Model architecture ablations*: A set of experiments and results which led to the final architectural choices for Falcon-H1-Tiny series.
- *Data strategy choice*: brief explanations on the intuitions behind data strategy choices and data forgetting window in Language Models.
- *Falcon-H1-Tiny-English*: In this chapter, we describe the entire lifecycle of building the Falcon-H1-Tiny-English variant, while comparing it to a classic “curriculum” approach, including a post-training stage with DPO algorithm.
- *Falcon-H1-Tiny-Multilingual*: Here, we detail the lifecycle of building the multilingual variant of Falcon-H1-Tiny, and how (and if) the conclusions drawn from the chapters before holds in a multilingual setting or not.

- *Falcon-H1-Tiny-R-0.6B and 90M*: First model from the series of specialized tiny models. We describe in this chapter, how we have a built SoTA reasoning Small Language Models by training exclusively the models on reasoning traces from scratch.
- *Falcon-H1-Tiny-Function-Calling*: Second series of specialized tiny models which focuses on function calling task. We detail in this section all the experiments and findings which led to the building of this model variant.
- *Falcon-H1-Coder*: Last series of specialized tiny models which is tailored to Python code generation and Fill-in-the-Middle task. In this chapter, we will cover all the experimental designs and results obtained for the coder variant of Falcon-H1-Tiny series.
- *Model usage*: We detail in this section how to easily get started with our models using the tools from the ecosystem.

Training Approach

Data Strategy

MEMORIZATION-AWARE REPETITION

A fundamental challenge in pretraining is the imbalance between large-scale lower quality data (e.g. web-scraped data, such as FineWeb) and higher quality curated data which we know is meaningful for some target domains and to enable specific capabilities to the model. Consider a situation with high-quality data source of D tokens representing a fraction p of overall data mixture. Then, the HQ source is exhausted after training for *epoch size* D_{ep} tokens

$$D_{ep} = \frac{D}{p}.$$

In a standard pretraining practice of single epoch training the data is not repeated, which puts an upper bound on HQ fraction $p \leq p_{ep} \equiv \frac{D}{T}$, where T is the total training tokens. During Falcon-H1 training we observed that the quality of the model is significantly held back by this constraint, as higher values of p would result in much better performance. This motivated us to repeat HQ data, which led us to investigate model memorization

abilities. We found that the model forgets training tokens seen long enough in the past, as shown in the figure below

Figure 9 from Falcon-H1 paper. We take a FalconMamba-7B checkpoint at late stage of training, and show it training samples seen before the checkpoint, recording the loss value on these seen tokens. As the loss value on new unseen tokens from the same training distribution is given by normal run (blue), the loss gap between normal run and data rollback curve (orange) shows the effect of memorizing individual training tokens. We see that the memorization degree decays with temporal delay of how long ago the training token was seen by the model.

Based on this forgetting behavior, we may (roughly) define *model memorization window* M as the number of tokens after which the model sufficiently forgets the tokens it has seen during training. For FalconMamba-7B shown on the figure, we can estimate $M \sim 100GT$, or $M \sim 500GT$ as a more conservative and safe estimate.

If a data source within a mixture has small enough epoch size $D_{ep} \lesssim M$, we can expect multiple repetitions of it to lead to a harmful memorization of the data. Conversely for large enough epoch size $D_{ep} \gtrsim M$ we can expect multiple repetitions to be harmless and does not lead to overfitting. This opens a possibility to significantly increase the upper bound on the fraction of high quality data by decoupling it from total training duration T

$$p \lesssim p_{mem}, \quad p_{mem} \equiv \frac{D}{M}.$$

Our hypothesis is that training with $p = p_{mem}$ allows for unlimited repetitions of HQ data without overfitting on the repeated data while taking full advantage of having HQ-rich data mixture. Falcon-H1-Tiny provides an implicit confirmation of this hypothesis, with $\gtrsim 100$ repetitions of SFT sources like ‘Tulu3’ during 800 GT of our SFT-pretraining. Yet, we stress that our current understanding of memorization-aware repetition is in an early stage, and requires systematic and rigorous investigation that we leave for future work.

ANTI-CURRICULUM

The option to safely repeat data during training by taking into account model’s memorization window opens up new possibilities to organize training data stages. The optimal choice can change significantly depending on the ratio between volume of high-quality data D and model’s memorization window M .

For Falcon-H1-Tiny we have considered two extreme strategies that we illustrate now on the example of chat and instruction-following SFT data. The first strategy is classic SFT finetuning: a long pretraining phase on a “general data mixture” is followed by a relatively short finetuning on the SFT mixture with, typically, up to 4 repetitions of SFT data. The second strategy is *SFT-pretraining* where we include SFT data into the pretraining mixture with maximal memorization-aware fraction $p = \frac{D_{SFT}}{M}$. Then, the training is performed in a single stage, following duration and other hyperparameters of pretraining stage from the first strategy.

Now, consider a scenario with a large model with, let’s say, 100B parameters and $M = 5000$ GT of estimated memorization window (based on a linear scaling from our FalconMamba-7B example), and an SFT dataset of 5 GT size. In this case, the maximal memorization-aware SFT fraction is $p = \frac{5GT}{5000GT} = 0.1\%$. It is then natural to expect that this fraction is insufficient for the model to focus on chat and instruction-following capabilities, and, therefore, the first strategy of classical SFT finetuning will be the optimal choice.

However, for small models, such as 100M Falcon-H1-Tiny, the situation changes dramatically. Linear scaling of the conservative 500 GT memorization window estimate for FalconMamba-7B leads to only 5 GT window size. The maximal SFT fraction then becomes $p = \frac{5GT}{5GT} = 100\%$ suggesting that we can even pretrain the model directly on SFT mix. Indeed, comparison between our

Falcon-H1-Tiny-90M-Instruct-Curriculum trained with SFT finetuning, and

`Falcon-H1-Tiny-90M-Instruct` trained with SFT-pretraining on a mixture with 25% of SFT data, reveals that SFT-pretraining can be a superior strategy to boost target capabilities (instruction-following measured by IFEval in this example).

Benchmarks	Falcon-H1-Tiny-90M SFT	Falcon-H1-Tiny 90M SFT DPO	Falcon-H1-Tiny-90M SFT-pretrain	Falcon-H1-Tiny-90M SFT-pretrain-DPO
IFEVAL	40.77	53.47	50.11	66.08

We arrive at a similar conclusion for another type of high-quality data: math reasoning traces. In this case we can pretrain on 100% of reasoning data from scratch, or do a classical reasoning SFT stage after general pretraining. The comparison between the two approaches on 90M model reveals a clear benefit of reasoning pretraining (no GRPO, best checkpoint along the training trajectory, data taken from [this figure](#)).

Benchmarks	Reasoning SFT	Reasoning pretraining
AIME24 pass@16, problems solved	3/30	6/30
AIME25 pass@16, problems solved	2/30	9/30
MATH500	0.2	0.4

Training algorithm and hyperparameters

For many of the training aspects we have followed our Falcon-H1 findings and recommendations detailed in [Zuo et al. \(2025\)](#) [↗]. Below is our main settings used, for example, for `Falcon-H1-Tiny-90M-Base` and `Falcon-H1-Tiny-90M-Instruct`. We use

- Learning rate (LR) 256e-5, weight decay (WD) 0.1, and batch size of 4 MT (million tokens).
- WSD learning rate schedule with 800GT total training duration, 100MT warmup duration, and x64 exponential decay over 100 GT.
- Power scheduler (square root LR decay) of learning rate starting from 100GT.
- Batch size rampup over first 40GT, during which we scale LR with square root of batch size

- Maximal Update Parameterization (μP) with multipliers for learning rate, weight decay, and forward multipliers. The 35 tuned values of multipliers are borrowed from Falcon-H1 and the transfer to smaller Falcon-H1-Tiny size was done through forward multipliers while keeping LR and WD fixed.

For some of our models we slightly deviate from this common setting, for example, `Falcon-H1-Tiny-Coder-90M` was pretrained on 315GT instead of 800GT. In such cases, we report the changes in the respective sections.

Now we proceed to the new additions to our training procedure compared to Falcon-H1.

MUON OPTIMIZER

In recent LLM training practice, the Muon optimizer has begun to emerge as a faster yet robust alternative to AdamW, which was previously the de facto default choice. We tested the Muon optimizer following the modifications proposed in [Liu et al. \(2025\)](#) [↗]: applying weight decay and scaling the update RMS norm to match AdamW values. We observe stable training with nearly the same optimal learning rate as AdamW, along with improved model evaluations. Accordingly, we use Muon for all falcon-h1-tiny models.

LEARNABLE MULTIPLIERS

In our recent work [Velikanov et al. \(2026\)](#) [↗] we argue that matrix layers of a model are trapped in a noise-WD with specific norm of matrix layers. This norm is dictated by training hyperparameters - LR and WD – instead of being learned from the data. To overcome the noise-WD equilibrium trap, we attach a learnable multiplier (LRM) to each row and column of weight matces in order to learn the norm of the respective row/column. This gives a substantial boost to downstream benchmarks as outlined in the table below (see the preprint for more details)

Table 1 from LRM paper, using Learnable Multipliers with Muon gives a considerable across most benchmarks

We validate the benefits of learnable multipliers on the Falcon-H1-Tiny architecture and model scale. Specifically, we compare two training runs: a Muon baseline and Muon with LRMs, trained for 200 GT, including 50 GT of exponential decay. From the results reported below, we observe performance improvements across most benchmarks, with tasks such as MMLU [Hendrycks et al. \(2021\)](#), BBH [Suzgun et al. \(2022\)](#), and GSM8K [Cobbe et al. \(2021\)](#) exhibiting up to a 20% relative gain from random values.

Evaluation benchmarks for a model that has been trained with LRM and without. There is a clear signal that LRM gives overall better performance.

Therefore, we confirm the benefit of learnable multipliers and adopt LRMs for all Falcon-H1-Tiny models. For more details about LRMs, please refer to [Velikanov et al. \(2026\)](#) ↗.

Model Architecture Ablations

For our models, we utilize the recent Falcon-H1 architecture, which combines parallel Mamba and Attention heads in the mixer block. We utilize the smallest vocab size that our tokenizer series support, i.e., 32768, and tie the embedding layer with the projection head. Our tokenizer contains manually injected most common LaTeX tokens, splits digits and punctuation to enhance the overall model performance on code and mathematic tasks as demonstrated in the Falcon-H1 paper. For this set of experiment, we are interested in identifying the key factors which will influence the end model performance, given a fixed “parameter budget”. Therefore, we fix the total number of model parameters to 90M and explore the impact of various model hyper parameters such as the dimensionality of mamba heads, attention heads, MLP factor, overall model depth and width.

EXPLORATION 1: DEPTH VS WIDTH

We start this ablation study by trying to answer a simple question - the depth vs width question. We construct 3 different model configurations

Model Candidates Configurations



We fix the data mixture by using a mixture close to the mixture of the final training stage of Falcon-H1, which was STEM focused and train the models for 200GT, which includes 50GT of decay stage.

We evaluate our tiny model candidates on leaderboard v1, and v2 tasks which includes:

- GSM8K, and Math-Hard for mathematical understanding and solving
- Hellaswag, Arc Challenge (easy), winogrande, truthful QA for basic english commonsense understanding
- MMLU and MMLU-Pro for scientific knowledge
- GPQA, BBH and MuSR for more advanced english commonsense (despite we know in advance these tasks will be difficult for a model of such size, and usually gives noisy evaluation results)

Evaluation results on the overall different models training across multiple benchmarks.

It seems that there are some signals that mid to deep models would give better performance by looking at hellaswag score where the difference is quite big. Although BBH seems much better with the shallow model, we decided here to prioritize english commonsense tasks as our final models will inheritly lag behind existing models due to

the nature of our data mixture which is very STEM-heavy. We decided to focus only on the difference between deep vs mid to determine if the tradeoff is worth it or not

Evaluation results on the overall different models training across multiple benchmarks.

The number of layers between the two model configurations (resp. 27 vs 50), is quite big and will severely impact the model's final throughput (we had roughly a 2x decrease for training throughput), while the gain on some STEM tasks (mainly MMLU and MMLU-Pro) is quite considerable, we decided to go for the mid architecture design in order to avoid ending up with an extremely low throughput. For our final model, we reduce the number of layers from 27 to 24 in order to match the number of parameters to be around 90M.

EXPLORATION 2: MLP FACTOR

We decided to explore the impact of inner model hyper-parameters with respect to the end model performance. As a proxy of the performance, we observe the loss function value between different training runs to conclude. For these models, we train each model using the same data mixture as the previous experiment, and train the models for a total of 70GT with a decay stage of 10GT. We utilize a classic WSD scheduler with an exponential decay factor of 64.

Impact of MLP dimension while compensating back with SSM dimensions

For this question, we defined 4 model candidates, representing different levels of compensation of SSM hidden dimension against MLP expansion factors. The configuration names are denoted in the comment above each declaration.

Model Candidates Configurations



Impact of MLP dimension while compensating back with SSM dimensions - lower is better

We concluded that increasing MLP size while compensating back with lower SSM size seems to lead to poor performance for tiny models - therefore we decided to stick for the baseline configuration for this question.

Impact of varying global hidden dimension while compensating back with MLP factor

Here, we tried 4 different variants on top of the baseline configuration to study the impact of varying the global hidden dimension while compensating back with MLP dimension.

Model Candidates Configurations



In a nutshell:

- `cfg2`: baseline - `d_ssm = 768`, `d_mlp = 768`, `hidden_size=512`
- `cfg5`: `d_ssm = 256`, `d_mlp = 2360`, `hidden_size=368`

- `cfg6`: `d_ssm = 256, d_mlp = 1700, hidden_size=448`
- `cfg7`: `d_ssm = 256, d_mlp = 1280, hidden_size=512`
- `cfg8`: `d_ssm = 256, d_mlp = 700, hidden_size=640`

Impact of varying global hidden dimension while compensating back with MLP factor - lower is better

Interestingly, between `cfg7` and `cfg8`, the `cfg7` seems slightly better, indicating that the “best” `hidden_size` might lie around 512 - All in all, it looks like `cfg2` seems to be the best configuration, with moderate `mlp_size` coupled with high `d_ssm` and moderate `hidden_size`. So it seems that :

- High-SSM setups descend faster than high-MLP/low-SSM variants. Therefore, SSM capacity is more valuable than large feed-forward width for a tiny model
- For fixed `d_ssm` value, there is a corresponding `hidden_size` to `mlp` ratio that is optimal.

EXPLORATION 3: ATTENTION CHANNELS

Impact of varying the number of attention heads

We study the impact of varying the number of KV heads in different scenarios. We compensate the remaining parameters into the MLP in order to match the number of parameters of the baseline configuration.

Model Candidates Configurations

Increasing KV heads while compensating back by decreasing the MLP dimension seems to give better results. However, there seem to be an optimal point in terms of ratio $\text{total_heads} / \text{kv_heads}$ from the results of `cfg11` which has a larger number of heads, larger MLP, but worse results than `cfg10`. Overall, our baseline still remains being the best configuration so far.

Falcon-H1-Tiny-English: curriculum vs pre-train with SFT data

In this section, we will cover our learnings and the decisions that led to our final artifacts for the Falcon-H1-Tiny-English track.

Curriculum model: final recipe

BASE MODEL

Following our conclusions from the previous sections, we pretrain our final base model on 800GT of data using our “best” model configuration covered in the model architecture section. We employ a WSD scheduler with a decay stage of 100GT and a batch size rampup of 20GT.

Our data mixture remains STEM focused, while we did an ablation on the proportion of web-data that we want to inject in the final mixture to boost the English commonsense tasks. We do not inject any SFT-style data into the final data mixture of the base model.

For that, we ran two training runs using different ratio of web-crawled data (i.e. fineweb and fineweb-edu), respectively representing 10% and 20% of the global data mixture. We conducted end-to-end evaluation of the checkpoints generated by these two training runs, and we include additional English commonsense benchmarks to better capture the effect of english web-data on the english commonsense capabilities of the model.

Ablation study on the web data ratio and its impact on the end performance of the model - increasing the web ratio from 10 to 20% doesn't hurt match the performance on STEM benchmarks while increasing the performance on english commonsense tasks.

Results showed that we obtained an overall tied result between the two runs, while the 20% web run gives an overall slightly better result on commonsense tasks while fairly preserving the model's capabilities on other benchmarks. Therefore, for the final training run, we decided to stick to 20% web data in our data mixture, to get an overall better performance on English commonsense tasks.

During the final stage, we also tried to increase the proportion of web-data during the decay stage using higher ratios, however we concluded that the increase in English commonsense tasks was not worth the degradation on other tasks.

SFT FINE-TUNING: EXTENSIVE SWEEPS ON SFT DURATION

For this model, we conducted end-to-end sweeps on the total SFT duration by measuring the impact of the SFT duration on the final model's performance.

We also coupled this experiment with a more conventional LR sweep at the same time to verify if LR adjustment is needed for SFT stage or not.

For the data mixture, we fix it to a mixture close to the final SFT mix we used for Falcon-H1. We conduct this double sweep on the training duration and LR values by running the following experiments:

Experiment Name	Learning Rate	Decay Duration	SFT total duration
1r-256-1gt-2gt	256e-04	1GT	2GT
1r-256-2gt-4gt	256e-04	2GT	4GT
1r-256-8gt-16gt	256e-04	8GT	16GT
1r-256-24gt-32gt	256e-04	8GT	32GT
1r-128-2gt-4gt	128e-04	2GT	4GT
1r-512-2gt-4gt	512e-04	2GT	4GT

Final results are showed in the plot below:

Ablation study on the web data ratio and its impact on the end performance of the model - increasing the web ratio from 10 to 20% doesn't hurt match the performance on STEM benchmarks while increasing the performance on english commonsense tasks.

For the learning rate, using `256e-4` seems to be the best option, while for SFT duration we decided to go for 10GT (best checkpoint from the run `1r-256-8gt-16gt`) as it offers

the most balanced performance (mainly with very good BBH, IFEVAL and GSM8K scores) across the target benchmarks. The evaluation results show that the model continues to learn up to that point, and the performance plateaus afterwards.

Anti-curriculum model: final recipe

To explore the anti-curriculum approach, we'll begin by systematically varying the amount of SFT data injected into the pretraining mix. Our goal is to identify an SFT ratio that preserves strong base-model performance while improving target SFT benchmarks, and to assess any unwanted effects such as a high memorization.

We start by preparing 5 data mixes that respectively contain 0, 25, 50, 75 and 100% of our SFT data mixture. The rest is assigned to the same mixture as our base model. Our SFT mix has the size of around 2 GT, resulting in $\{8, 4, 2.66, 2\}$ GT SFT epoch sizes for $\{25\%, 50\%, 75\%, 100\%\}$ variants.

We train the models on a total of 100GT, which includes 20GT of decay stage and re—evaluate the models on the same benchmarks as the previous experiments. We did not notice clear memorization artifacts, such as degradation of performance during training, even on SFT-100% with 2 GT SFT epoch size. However, the mixtures with mostly SFT data showed suboptimal performance overall, suggesting that pretraining data is essential to develop even instruction-following capabilities. We have found SFT-25% to give the best instruction following performance together with SFT-50%, while having best base model evaluation (without using the chat template) together with SFT-0%. Hence, SFT-25% model combines best of both worlds and can act as a base and instruct model at the same time.

Therefore, for the final recipe we decided to inject 25% of pure SFT data in the final mixture and allocate the remaining 75% to the mixture of the base model. We train the final model for a total of 800GT with a learning rate decay stage of 100GT.

SFT stage on top of Anti-curriculum model

After training the SFT-pretrain model, we asked ourselves the following question: should we perform an extra SFT stage on top of the pre-train SFT model?

We apply the same optimal learning rate found in the previous section and perform a SFT stage on top of the sft-pretrain checkpoint. We evaluate the trained model on the same evaluation benchmarks as the previous experiment and there was no clear signal on the

benefit of performing a SFT stage on top of the sft-pretrain checkpoint. Therefore, we concluded that having a single stage, `sft-pretrain`, is possible to obtain a model that can act simultaneously as a base and SFT model.

DPO for tiny models

Would DPO be effective for tiny models such as ours?

We have internally observed that for all our previous model scale (0.5B - 34B), a small DPO stage helps a lot to boost some key capabilities of the SFT models. Some very recent work such as [Jakimovski \(2025\)](#) showcases a nice convergence of DPO alignment after doing a SFT stage with reasoning traces on a 100M scale model. Let's explore a bit more about it using our models and see its impact on the end model performance.

We started the exploration by conducting a simple LR sweep for the DPO stage. We chose 4 different initial LR candidates for this purpose which are:

- `1e-05`
- `3e-07`
- `3e-06`
- `1e-06`

We trained the dpo models starting from sft-pretrain checkpoint for this first batch of experiments on 3 epochs and use cosine learning rate decay. To better assess the performance evolution, we perform end to end evaluations of our trained models on the same benchmarks we used for the pre-training jobs, to study its potential degradation on previous tasks, as well as these additional benchmarks which are more relevant to chat-style models:

- Alpaca [Li et al. \(2023\)](#) ↗
- LiveBench [White et al. \(2025\)](#)
- MT-Bench [Bai et al. \(2024\)](#) ↗

We observed that running the DPO algorithm for more than 1 epoch for our model led to a considerable performance degradation straight after the first epoch, despite the DPO reward being increased during the entire training run. From this first batch of experiment,

we decided to pick 2 LR candidates from this experiment (1e-06 and 3e-06) and train the models on 1 epoch only (as the choice of the number of epochs affects the training dynamic since we use cosine decay).

Evolution of evaluation results when doing DPO stage - key evaluation results such as IFEVAL increases drastically while other benchmarks scores are overall preserved.

Evolution of Instruction following benchmarks such as Alpaca, MTBench and LiveBench, during the DPO stage.

We observe an impressive boost of IFEVAL which goes from ~50 to 65+. We apply the same optimal learning rate for the curriculum SFT model and perform the same evaluations to obtain a clearer picture on the question of SFT pre-train vs SFT-curriculum.

End performances

In this section, we compare the end performances of our trained models between themselves and provide also a comparison against other open source models of similar size.

BASE MODELS

Below are the evaluation results of our Falcon-H1-Tiny-English base models. The evaluation benchmarks are divided into multiple categories:

- **English Commonsense:**
 - Anli [Nie et al. \(2020\)](#)
 - Arc Challenge (Easy) [Clark et al. \(2018\)](#)
 - BoolQ [C. Clark et al. \(2019\)](#)
 - Drop [Dua et al. \(2019\)](#)
 - Hellaswag [Zellers et al. \(2019\)](#)
 - Lambada [Paperno et al. \(2016\)](#) ↗

- WebQS [Berant et al. \(2013\)](#) ↗
- Winogrande [Sakaguchi et al. \(2019\)](#)
- Truthful QA [Lin et al. \(2022\)](#) ↗
- Sciq [Welbl et al. \(2017\)](#)
- Piqa [Bisk et al. \(2020\)](#)
- Race [Lai et al. \(2017\)](#) ↗
- **STEM:**
 - MMLU / MMLU-Var [Hendrycks et al. \(2021\)](#)
 - MMLU-Pro [Wang et al. \(2024\)](#)
 - GSM8K [Cobbe et al. \(2021\)](#)
 - Math-Hard [Hendrycks, Burns, Kadavath, et al. \(2021\)](#)
- **Instruction following:**
 - IFEVAL [Zhou et al. \(2023\)](#) ↗
- **Reasoning:**
 - BBH [Suzgun et al. \(2022\)](#)
 - MuSR [Sprague et al. \(2024\)](#) ↗
 - GPQA [Rein et al. \(2023\)](#)

Benchmark	Falcon-H1-Tiny-English-Base	Falcon-H1-Tiny-English-Instruct (SFT pre-training)	Mobile-LLM-140m-R1-base Zhao et al. (2025) ↗	Smol-LM-135M Allal et al. (2025) ↗
English Commonsense				
anli_r1	33.1	32.8	30.7	31.5
anli_r2	33.8	33.8	35.2	34.5
anli_r3	32.3	34.92	36	35.3
arc_challenge (acc)	24.4	23.89	22.6	28.1
boolq	58.96	50.94	53.7	60.5
drop (f1)	3.01	2.94	7.4	3.3
hellaswag (acc_norm)	37.52	36.53	33.7	43
lambada_openai (acc)	35.08	34.46	31.2	43
lambada_standard (acc)	25.57	25.42	24.5	35.5
webqs (em)	1.23	1.18	1.5	1.8
winogrande	51.3	50.59	52.2	52.6
truthful_qa_mc1	21.79	23.13	23.99	23.99
truthful_qa_mc2	36.42	38.26	40.88	38.77
sciq (acc_norm)	76.8	76.9	81.2	78.3
piqa (acc_norm)	65.13	63.32	63.6	68.4
race	31.96	29.66	28.8	30.9
<i>Avg</i>	35.523125	34.92125	35.448125	38.09125
STEM				

Benchmark	Falcon-H1-Tiny-English-Base	Falcon-H1-Tiny-English-Instruct (SFT pre-training)	Mobile-LLM-140m-R1-base Zhao et al. (2025) ↗	Smol-LM-135M Allal et al. (2025) ↗
MMLU	32.3	32.33	24.4	24.2
MMLU-var	29.42	28.37	27.9	30.5
MMLU-pro	7.18	7.95	1.8	1
GSM8k (flexible extract)	2.64	8.11	20.2	1.2
Math-Hard	2.47	1.43	1.4	1
Avg	14.802	15.638	15.14	11.58
Instruction following				
IFEVAL	30.07	32.45	18.6	18.4
Reasoning				
BBH	3.98	2.52	5.1	3.6
MuSR	2.73	2.57	4.8	9.8
GPQA	0	0	0	0

SFT MODEL EVALUATION

We evaluate our different checkpoints on roughly the same benchmarks as the base models, note these benchmarks are evaluated while applying the chat template to the input prompts (hence the different results between the two tables).

Benchmarks	Falcon-H1-Tiny-90M-Instruct-Curriculum-pre-DPO	Falcon-H1-Tiny-90M-Instruct-Curriculum	Falcon-H1-Tiny-English-Instruct-pre-DPO (SFT pre-training)	Falcon-H1-Tiny-English-Instruct (SFT pre-training)	SmolLM2-135M-Instruct	S
STEM						
MMLU	26.28	24.81	26.57	27.31	24.64	2
MMLU-pro	1.58	1.82	5.38	7.04	1.14	1
GSM8k (flexible extract)	21.6	20.24	19.03	19.18	1.28	9
Math-Hard	3.37	3.6	3.19	2.03	1.49	1
<i>Avg</i>	13.2075	12.6175	13.5425	13.89	7.1375	9
English commonsense						
hellaswag (acc_norm)	35.28	35.63	36.21	36.43	40.21	4
arc_challenge (acc)	25.14	23.89	23.38	23.63	26.7	3
truthful_qa_mc1	25.45	24.47	23.01	25.82	25.82	2
truthful_qa_mc2	41.96	42.31	37.8	42.02	40.8	4
<i>Avg</i>	31.9575	31.575	30.1	31.975	33.3825	3
Reasoning						
BBH	4.64	4.13	5.01	2.51	4.7	4
MuSR	1.47	0.94	1.33	1.33	2.33	2
GPQA	2.56	4.51	1.09	0	0	0
<i>Avg</i>	2.89	3.1933333333	2.4766666667	1.28	2.3433333333	2
Instruction following						
IFEVAL	40.77	53.47	50.11	66.08	30.69	3

Benchmarks	Falcon-H1-Tiny-90M-Instruct-Curriculum-pre-DPO	Falcon-H1-Tiny-90M-Instruct-Curriculum	Falcon-H1-Tiny-English-Instruct-pre-DPO (SFT pre-training)	Falcon-H1-Tiny-English-Instruct (SFT pre-training)	SmolLM2-135M-Instruct	S
Alpaca Eval (win rate)	3.45	10.44	2.96	9.43	1.52	2
MT Bench (avg)	3.17	4.4	3.08	4.33	2.68	3
LiveBench (global avg)	10.51	12.4	11.09	15.69	8.25	1
Avg	14.475	20.1775	16.81	23.8825	10.785	1
Coding						
MBPP+	12.69	7.93	11.11	7.93	10.58	2
HumanEval+	9.76	7.31	9.76	7.31	-	-

Overall, the SFT-pretrain checkpoint seems better even after a DPO stage, but not with a huge margin compared to the SFT-curriculum. However, for key benchmarks of SFT models such as IFEVAL, the difference is clear (around 10 points).

Vibe checking the model

In addition to benchmark numbers, we *vibe-checked* the sft-pretrain model post-DPO. Since the model seems extremely good on instruction following, we decided to prompt the model using a few prompts from IFEVAL paper [Zhou et al. \(2023\)](#) [↗]. We use `llama.cpp`, and specifically use the `Q8_0` variant to stress test the quantized version of the model while simulating a real-world local deployment scenario. The model ends up having an impressively small memory footprint of ~90MB on the local machine after quantizing it to 8bit precision. We use the exact same sampling parameters between the two models when generating the responses. We have selected below few interesting prompts and display the generations from our model, compared to `HuggingFaceTB/SmolLM2-135M-Instruct` under the same constraints (llama.cpp + `Q8_0` quantization and identical sampling parameters).

WRITE A RUBRIC FOR PERFORMANCE REVIEW OF A SOFTWARE ENGINEER AND WRAP THE ENTIRE OUTPUT IN JSON FORMAT. YOU CAN USE MARKDOWN TICKS SUCH AS ```.

Response of Falcon-H1-Tiny on the prompt described in the subtitle

Response of SmoLLM2-135M on the prompt described in the subtitle

WHAT'S THE DIFFERENCE BETWEEN THE APPLE AND ALBANIA? ANSWER IN EMAIL FORMAT. YOUR RESPONSE MUST CONTAIN AT LEAST SIX PLACEHOLDERS WHICH SHOULD BE REPRESENTED BY SQUARE BRACKETS LIKE [NAME].

Response of Falcon-H1-Tiny on the prompt described in the subtitle

Response of SmoLLM2-135M on the prompt described in the subtitle

WRITE A RESUME FOR A SOFTWARE ENGINEER WITH 5+ YEARS OF EXPERIENCE IN THE BAY AREA, CA. IN YOUR RESPONSE, MAKE SURE TO INCLUDE AT LEAST 20 WORDS OR PHRASES IN ALL CAPITAL LETTERS.

Response of Falcon-H1-Tiny on the prompt described in the subtitle

Response of SmolLM2-135M on the prompt described in the subtitle

WRITE A COVER LETTER FOR A JOB IN A TECH COMPANY. MAKE SURE TO USE THE WORD "THE" ONCE OR LESS.

This is our favorite response !

Response of Falcon-H1-Tiny on the prompt described in the subtitle

Response of SmolLM2-135M on the prompt described in the subtitle

GIVE ME A 300+ WORD STARTUP PITCH FOR A COMPANY THAT PROVIDES A SERVICE THAT CREATES TRUSTS FOR PEOPLE ONLINE.

Response of Falcon-H1-Tiny on the prompt described in the subtitle

Response of SmolLM2-135M on the prompt described in the subtitle

Overall we feel that the model capabilities are quite impressive for its extreme small size.

Falcon-H1-Tiny-Multilingual: an extremely small language model trained on multilingual data

Motivations and Research Questions

Building on the architectural foundations and optimization strategies validated in our English experiments, we turned our attention to multilingual capabilities. Can a tiny model capture meaningful multilingual capabilities?

We used the base configuration from the English experiments (24 layers, 512 hidden dimension, etc.) while increasing the vocab size to 65k (thus, slightly increasing the model size to 100M), multi-epoch training strategy for high quality sources, Learnable Multipliers (LRM) multipliers paired with Muon optimizer for better performance, and the anti-curriculum insight that injecting high-quality data from the start outperforms traditional curriculum approaches. We argue that multilingual training may introduce new considerations that don't necessarily apply in English-only scenarios. Consequently, three questions drove our experimental design:

- Does the anti-curriculum advantage hold for multilingual models? In English, we demonstrated that mixing SFT data directly into pretraining beats the conventional pretrain-then-finetune pipeline. Would this finding generalize when juggling 17+ languages with vastly different resource levels?
- What is the optimal balance between English and multilingual content? Too much English may risk the model defaulting to English responses. Too little may sacrifice the instruction-following capabilities that high-quality English data provides. For a model with only 100M parameters, this tradeoff could be especially sharp.
- Can tiny models master multilinguality? The usual assumption suggests that strong multilingual performance requires massive scale LLM parameter budget. We'd like to challenge this assumption and see how this scales at extreme small scale.

Data Sources and Construction

MULTILINGUAL PRETRAIN CORPUS

Our multilingual pretraining data built directly on the Falcon-H1 infrastructure, which supports 17 languages beyond English: Czech, German, Spanish, French, Hindi, Italian, Japanese, Korean, Dutch, Polish, Portuguese, Romanian, Russian, Swedish, Urdu, and Chinese.

The backbone came from 3,000+ GT of Common Crawl multilingual extractions processed through the pipeline described in the Falcon-H1 report: pylc2 language identification, fasttext refinement, Gopher Quality filtering tuned per language, toxicity filtering with human-curated word lists, and MinHash deduplication. For languages with limited web coverage, we supplemented with additional high-quality sources for those specific languages.

We also included Wikipedia (20231101 snapshot) and a wide range of well-curated multilingual textbooks covering STEM and science topics across 9 languages.

After extensive internal sweeps on the distribution within multilingual sources, we settled on: 50% from the Common Crawl mix, 33% from Wikipedia, and 17% from textbooks. This distribution balanced breadth (web data) with quality and structure (Wikipedia and textbooks). For evaluating our models, we chose three multilingual benchmarks – hellaswag multilingual [Dac Lai et al. \(2023\)](#), M-MMLU and MGSM [Shi et al. \(2022\)](#).

PROPORTION OF ENGLISH SOURCES FOR PRETRAIN AND SFT CORPUS

Since our model needed to maintain strong English performance while adding multilingual capabilities, the English component of our data mixes drew from the same sources validated in earlier experiments.

The 20% web allocation (6% crawled + rewritten web) was based on our earlier finding that this ratio optimized English commonsense performance without sacrificing STEM capabilities.

For English SFT data, the mix is heavily dependent on high-quality instruction-following datasets covering well-curated problem solutions, single and multi-turn pairs targeting a wide range of capabilities including safety, instruction following, math, code, reasoning, and chat. Additional internal English and multilingual data sources offered further diversity in the SFT corpus for our final multilingual tiny model.

Training Strategies: Pretrain-SFT vs Curriculum

PRETRAIN-SFT (ANTI-CURRICULUM) APPROACH

Following our English model insights, we tested whether mixing SFT data directly into pretraining would work for multilingual scenarios. The final mix combined:

- 40% English pretrain data
- 20% English SFT data
- 20% multilingual pretrain data : 10% Common Crawl mix, 6.67% Wikipedia, 3.33% textbooks
- 20% multilingual SFT data: 19.5% from multilingual post-training datasets (without thinking traces), 0.5% from multilingual conversational data

We trained this model for 800 GT total using the same hyperparameters validated in our English experiments: learning rate value and schedule, batch size rampup over 20 GT, 100 GT exponential decay stage using the WSD scheduler (decay coefficient 64), and the same LRM + Muon optimization setup. This consistency allowed us to isolate the impact of multilingual data mixing without confounding from different training configurations.

CURRICULUM APPROACH

For comparison, we implemented a more traditional pipeline: first pretrain a multilingual base model, then finetune on SFT data.

Stage 1: Multilingual Base Pretraining

The base model pretrained on a mix of English and multilingual sources without any SFT data. For English, we used the same pretrain sources as above but respected the 20% web vs 80% STEM/code/reasoning ratio that worked well in earlier experiments. For multilingual, we maintained the 50/33/17 distribution (Common Crawl / Wikipedia / textbooks).

Stage 2: SFT Fine-tuning

After pretraining converged, we ran a supervised fine-tuning stage using a flexible class that let us sweep different English-to-multilingual ratios. The English SFT sources matched the pretrain-SFT mix but were rescaled to form a proper 100% base. Key parameters we could tune:

- `english_sft_weight`: percentage of English SFT data
- `multilingual_sft_weight`: percentage of multilingual SFT data

This parameterization let us systematically explore the English-multilingual balance.

Evaluation and Results

We evaluated our multilingual models on three key benchmark suites: STEM capabilities (MMLU, MMLU-pro, GSM8k, Math-Hard), English commonsense reasoning (hellaswag, arc_challenge, truthful_qa), reasoning (BBH, MuSR, GPQA), instruction following (IFEVAL), and multilingual capabilities (multilingual_hellaswag, multilingual_mmlu, multilingual_mgsm). We report results for the best-performing checkpoints from each training strategy and compare against similar-sized multilingual models (SmolLM2-135M, Gemma3-270m, etc...).

BASE MODELS COMPARISON

Benchmark Category	Metric	Falcon-H1-Tiny-101M	Mobile-LLM-140m	SmolLM2-135M	Gemma3-270m
STEM					
	MMLU	32.62	24.4	24.2	26.2
	MMLU-pro	9.48	1.8	1.0	0.8
	GSM8k (flexible)	8.04	20.2	1.2	1.5
	Math-Hard	2.10	1.4	1.0	1.1
	Avg	13.06	11.95	6.85	7.40
English Commonsense					
	hellaswag (acc_norm)	33.76	33.7	43.0	41.5
	arc_challenge (acc)	24.66	22.6	28.1	25.2
	truthful_qa	24.24	23.99	23.99	24.36
	Avg	27.55	26.76	31.70	30.35
Reasoning					
	BBH	2.95	5.1	3.6	2.2
	MuSR	3.60	4.8	9.8	7.8
	GPQA	3.20	0	0	0.1
	Avg	3.25	3.30	4.47	3.37
Multilingual					
	multilingual_hellaswag	29.71	28.36	28.99	31.93
	multilingual_mmlu	27.78	24.78	25.97	26.13
	multilingual_mgsm	2.13	2.06	1.53	1.93
	Avg	19.87	18.40	18.83	19.99

INSTRUCT MODELS COMPARISON

Benchmark Category	Metric	sft-pretrain (800GT)	Curriculum-SFT	Curriculum-SFT-DPO	SmolLM 135M Instru
STEM					
	MMLU	24.50	25.21	26.06	24.64
	MMLU-pro	2.61	2.02	2.37	1.14
	GSM8k (flexible)	10.61	12.96	13.12	1.28
	Math-Hard	3.31	2.89	3.31	1.49
	Avg	10.26	10.77	11.22	7.14
English Commonsense					
	hellaswag (acc_norm)	32.38	32.94	33.59	40.21
	arc_challenge (acc)	25.68	24.32	25.09	26.7
	truthful_qa	25.21	26.56	24.65	25.82
	Avg	27.76	27.94	27.78	30.91
Reasoning					
	BBH	3.93	3.40	4.01	4.70
	MuSR	2.08	5.03	4.75	2.33
	GPQA	2.44	1.59	2.56	0
	Avg	2.82	3.34	3.77	2.34
Instruction Following					
	IFEVAL	46.50	43.83	52.00	30.69
Multilingual					
	multilingual_hellaswag	29.67	29.27	29.77	28.79
	multilingual_mmlu	55.00	51.00	45.00	25.63
	multilingual_mgsm	4.07	4.73	4.67	1.40
	Avg	29.58	28.33	26.48	18.61

Key Findings

1. Anti-curriculum strategy in this case did not bring a clear boost for multilingual benchmarks: We observe minimal global changes between sft-pretrain and curriculum-SFT across nearly all benchmark categories. We hypothesize that the curriculum-SFT data injected from scratch was not high-quality enough to deliver meaningful improvements and may be fundamentally restricted by the 100M model size, which limits the capacity to absorb and benefit from additional training data.
2. Previous observation on dpo stage holds for tiny multilingual providing the main performance boost for instruction following capabilities.
3. Our multilingual capabilities remain competitive for a 100M model, achieving scores that are higher than models with same size but remains limited and modest in absolute terms (after we vibe-checked the model). Enhancing multilingual corpus quality, exploring multilingual data augmentation techniques, and deepening our understanding of multilingual knowledge retention at small scale will be key areas we'll be working on to improve these capabilities in future iterations.

Falcon-H1-Tiny-R: Paving the way for a new pretraining paradigm for reasoning models

Methodology

Building on our previous observations, we revisit the standard reasoning training pipeline. Reasoning models typically follow a three-stage recipe: (1) pre-training on general data, (2) supervised fine-tuning (SFT) on reasoning data, and (3) reinforcement learning (RL). Although some recent work such as [Olmo et al. \(2025\)](#) [↗] attempted a Zero-RL strategy which consists of doing an RL stage on top of a base model - overall a common practice within the community seems to lie around curriculum approaches to build reasoning models. Therefore, we decided to explore whether an anti-curriculum strategy can enhance the reasoning capabilities of small models, and if yes to what extent?

We first focus on merging the first two steps and directly pre-train on a reasoning data mixture (typically, data with complex problems solved with synthetically generated reasoning traces). Not only such an approach follows our anti-curriculum strategy but it should also allow the training of hybrid reasoning and general models by tuning the pre-

training data mixture. In this work, adhering to the philosophy of training highly specialized models, we focus exclusively on reasoning data.

Following this philosophy, we train two models (Falcon-H1-Tiny-R-0.6B and Falcon-H1-Tiny-R-0.09B). We apply Group Relative Policy Optimization (GRPO)

[Shao et al. \(2024\)](#) [↗] on Falcon-H1-Tiny-R-0.6B and release both pre-GRPO and post-GRPO checkpoints. This would also allow us to study the dynamics of RL algorithms such as GRPO on small scale models.

While we were skeptical about a hard “reasoning emergence” threshold above which reasoning suddenly appears, our post-GRPO Falcon-H1-Tiny-R-0.6B is competitive with much larger recent models (e.g., Qwen3-4B and Qwen3-8B [Yang et al. \(2025\)](#) [↗]). We also find that our techniques only push this threshold down to Falcon-H1-Tiny-R-0.09B: despite being strong for its size (surpasses Mobile-LLM-R1 series [Zhao et al. \(2025\)](#) [↗]), it still falls below our expected performance.

Inspecting the smaller model’s outputs, we find it is more prone to a repetition trap (repeating the same tokens - aligned with [Pipis et al. \(2025\)](#) [↗]). While a repetition penalty can mitigate this to some extent, understanding why this behavior emerges and how it interacts with optimization remains an open question.

Data Composition

Our training corpus is similar to the one from Falcon-H1R-7B [Team et al. \(2026\)](#) [↗].

Training strategy

During early Falcon-H1-Tiny-R experiments, training was noisy and we observed that most gains arrived during the learning-rate decay phase. We therefore use a WSD schedule with:

- **Warmup:** 100M tokens
- **Constant LR:** 500 GTok
- **Decay (stage 1):** exponential LR decay by a factor of 4 over 50 GTok
- **Decay (stage 2):** exponential LR decay by a factor of 256 over 350 GTok resulting in a total training budget of 900 GTok

The effect of this schedule is visible in [the figure below](#).

Results

We evaluate our models on the following benchmarks:

- AIME24 [Zhang & Math-AI \(2024\)](#)
- AIME25 [Zhang & Math-AI \(2025\)](#)
- LiveCodeBench v5v6 [Naman Jain \(2024\)](#)
- MATH-500 [Lightman et al. \(2023\)](#)

The figure below shows evaluation metrics as a function of total training budget (GTok) for each model/checkpoint family. We report both compute-scaled metrics (pass@16 / maj@16) and single-sample baselines (pass@1) for the key benchmarks.

Evaluation metrics vs GTok by model type. We report AIME24/AIME25 (pass@1, pass@16, maj@16), LCBv6 (accuracy), and Math500 (accuracy) across model variants and training budgets.

To further isolate the effect of *pretraining on reasoning* at tiny scale, the figure below focuses on the 90M regime and compares reasoning-pretrained runs against a classic “pretrain then SFT on reasoning” baseline. Across metrics, reasoning pretraining yields stronger results over all the evaluated metrics.

90M comparison: reasoning pretraining vs a classic “pretrain then SFT on reasoning” baseline. The orange run starts from a pre-trained checkpoint and begins with near-zero scores on the reasoning benchmarks, while reasoning pretraining scales better with budget across AIME24/AIME25 (pass@1 and pass@16) and Math500 (accuracy).

Metric glossary: pass@k = success rate with k sampled solutions; maj@16 = majority vote over 16 samples; acc@5 = top-5 accuracy; GTok = training budget in billions of tokens.

Key takeaways (compute-scaled metrics):

- Falcon-H1-Tiny-R-0.6B (post-GRPO) improves steadily with budget and becomes competitive with common 7B baselines at higher GTok.
- Falcon-H1-Tiny-R-0.09B improves with training but plateaus earlier, leaving a consistent gap vs the 0.6B model.
- Many of the largest gains arrive late, in the decay regime (see Training strategy).

We also summarize the remaining pass@1/accuracy benchmarks to pair with the plots.

Benchmark (metric)	Falcon-H1-Tiny-R-0.6B (post-GRPO)	Falcon-H1-Tiny-R-0.6B (pre-GRPO)	Falcon-H1-Tiny-R-0.09B	Qwen3-1.7B	OpenReasoning-Nemotron-1.5B	DeepSeek-R1-Distill-Qwen-1.5B	M
AIME24 (pass@1)	75.0	67.5	5.0	47.0	49.7	29.1	3
AIME25 (pass@1)	67.3	60.0	7.9	37.0	40.4	23.4	3
LCBv6 (accuracy)	39.0	35.0	4.5	29.8	28.3	19.9	2
Math500 (accuracy)	94.0	92.5	39.7	89.4	83.4	83.2	8

Performance and Scaling

We challenge the prevailing belief that strong reasoning is limited to large-scale models (e.g., 7B+ parameters). In our setting, a 0.6B model pre-trained specifically on reasoning can reach performance that is competitive with common 7B baselines on compute-scaled evaluation (pass@16 / maj@16), while remaining strong on broader reasoning-adjacent benchmarks.

Importantly, pass@k and majority-vote metrics should be interpreted as test-time scaling: they trade inference compute for higher success probability. Because a 0.6B model is much cheaper to sample than a 7B model, it can afford larger inference budgets at similar or lower cost, making these comparisons practically meaningful for latency/compute-constrained deployments. We keep this part open and invite the community to apply test time scaling on our models.

Finally, GRPO on small scale models can provide an additional boost on top of the pre-trained checkpoint (especially on AIME-style reasoning), but we found out that the gains are sensitive to optimization details (notably the GRPO learning rate), which we treat as a primary tuning knob.

Reinforcement Learning (RL): Observations

In our GRPO experiments, we found the models to be very sensitive to learning rate. With too small of a learning rate, training converges too slowly; with too large of a learning rate, we observe instability with the policy entropy diverging and the critic becoming noisy. In practice, we pick an intermediate value, and in the runs shown below we use **LR = 3e-6**.

GRPO phase

We run **60 GRPO steps** on the Falcon-H1-Tiny-R-0.6B checkpoint with a **generation context length of 32,000** tokens. This improves the post-GRPO checkpoint over pre-GRPO across benchmarks (see table above), e.g. AIME24 pass@1 **67.5** → **75.0**, AIME25 pass@1 **60.0** → **67.3**, LCBv6 accuracy **35.0** → **39.0**, and Math500 accuracy **92.5** → **94.0**.

Entropy dynamics during GRPO training.

Critic/value learning dynamics during GRPO training.

In addition to quality improvements, GRPO also shortens generations in this setting: the mean response length trends down from roughly **16k tokens** early in GRPO to roughly **8k tokens** by the end of the run.

Mean response length across GRPO steps at LR 3e-6.

Falcon-H1-Tiny-Function-Calling: A powerful function calling model in your hands

Motivation

Tool calling represents one of the most demanding capabilities for language models. Beyond understanding user intent, the model must recognize when a function should be invoked, think well about the situation then select, only when it is relevant, the appropriate tool from available options and generate syntactically correct arguments that match the expected schema. For a 90M parameter model, this seemed like an ambitious target - most successful tool-calling models operate at 0.5B+ parameters. We approached this with the same experimental rigor as our previous attempts. Does the anti-curriculum strategy (mixing tool calling data directly into pretraining) outperform traditional SFT? What percentage of tool calling data yields optimal results? And critically - can a model this small actually learn structured function calling without collapsing into degenerate outputs and produce coherent output?

Data Curation

We curated a diverse mixture of tool calling data designed to cover the full spectrum of real-world usage patterns:

- *Single-turn calls*: Simple one-shot function invocations with clear parameter extraction
- *Multi-turn conversations*: Extended dialogues where tool usage spans multiple exchanges, requiring the model to maintain context
- *Sequential tool chains*: Scenarios where one function's output informs the next call
- *Parallel tool calls*: Queries requiring multiple simultaneous function invocations
- *Relevance detection*: Examples where the correct response is to call a function from the available set of tools
- *Irrelevance handling*: Cases where the query should be answered directly without invoking any tool
- *Schema diversity*: Varying parameter types, optional fields, nested objects, and different API conventions
- *Edge cases*: Ambiguous queries, partial information, and malformed requests

This breadth of coverage ensures the model learns not just how to call functions, but critically when to call them — and when not to. One notable challenge we encountered involved training samples containing chain-of-thought reasoning traces, which we address in the following section.

The Reasoning Loop Problem

When we included data sources containing chain-of-thought reasoning traces in our training mix, the model exhibited a troubling behavior: infinite generation loops. Instead of producing clean function calls, it would spiral into repetitive text, often getting stuck on phrases or partial reasoning traces. This aligned with recent findings from “Wait, Wait, Wait... Why Do Reasoning Models Loop?” [Pipis et al. \(2025\)](#) [↗]. The paper demonstrates that smaller models, particularly those distilled from larger teachers, loop significantly more than their source models. The core insight: when the training distribution contains complex reasoning patterns that exceed the model's learning capacity, the model falls back to cyclic behaviors. Rather than learning to make progress through difficult reasoning steps, it learns the easier pattern of repetition. These data sources included relatively long chain-of-thought reasoning traces interleaved with tool calling examples.

For a 7B model, such traces provide useful scaffolding. For our 90M model, they created an overly complex learning target. The model couldn't compress the reasoning patterns into its limited parameter space, so it defaulted to the simplest available behavior: repeating tokens, a failure mode we clearly observed in the output generations during BFCL-v3 evaluation. Our solution was straightforward: we filtered out all reasoning and thinking content from the affected data sources, keeping only the direct tool calling examples. The improvement was immediate, the model began producing clean, structured function calls.

Training Strategies Comparison

We tested both approaches:

Curriculum-SFT: Standard supervised fine-tuning on the English SFT-pretrain model with varying percentages of tool calling data (20%, 30%, 40%, 50%, 75%, 85%). We perform an SFT stage with the same duration as the optimal SFT duration from the English Curriculum model.

Pretrain-SFT: Mixing tool calling data directly into pretraining alongside base and SFT data, maintaining a 2:1 ratio of base to SFT content that respects the previously drawn conclusions on SFT to pretraining data optimal ratio.

Interestingly, the results were different. Unlike our English experiments where pretrain-SFT showed clear advantages, for tool calling the two approaches performed nearly identically. BFCL [Patil et al. \(2025\)](#) scores tracked each other closely across configurations.

Our interpretation is that tool calling capability such as JSON syntax, tool schema matching, parameter extraction, appears bounded by the model's maximum feature learning capacity rather than by exposure timing or data repetition. More training time won't change what the model can fundamentally learn at this scale.

Scaling Tool Calling Data Percentage

BFCL performance scaled monotonically with tool calling data percentage that is being injected in the SFT mixture, though with diminishing returns - we SFT fine-tune various versions of Falcon-H1-Tiny-Tool-Calling starting from the English SFT-pretrain model, while varying the percentage of tool calling data in the SFT mixture:

Tool Calling %	Global BFCL v3 Score
20%	32.1%
50%	36.8%
75%	39.4%
85%	41.2%

We observed a consistent increase in the global score across increasing the percentage of tool calling data – on the tested ratios, the 85% configuration achieved peak performance across all other data configurations.

Final Results and Comparison

Our best model (85% tool calling data in SFT mixture) achieved:

Model	Size	Non-Live AST	Live AST	Relevance	Irrelevance	Multi-Turn	Global
Qwen3-0.6B	600M	71.79%	56.62%	75.00%	80.84%	3.62%	57.57%
Function Gemma	270M	48.40%	26.40%	61.10%	70.60%	0%	41.30%
Falcon-H1-Tiny-Function-Calling-90M	90M	36.06%	14.27%	94.44%	61.37%	0%	41.23%

In average, Falcon-H1-Tiny matches Function Gemma's global score while using 3x fewer parameters, however, the breakdown reveals an interesting trade-off: Our model underperforms on AST accuracy - the ability to generate syntactically perfect function calls. Where Function Gemma achieves 48.40% on non-live AST, we reach 36.06%. The gap is also consistent across live benchmarks.

However, we outperform on relevance detection: 94.44% versus 61.10%. Our model excels at recognizing when a function should be called versus when to respond directly. This suggests that the 90M, under our experimentation settings, capacity is better spent on understanding tool applicability than on memorizing schema details.

Vibe Checking the final model

Beyond benchmark numbers, we wanted to verify that the model actually works in practice. We ran Falcon-H1-Tiny-90M-Tool-Calling locally using llama.cpp on a MacBook with the GGUF-BF16 weights.

We post some recommendations below:

- Use the proper chat template with tools defined in the system prompt
- Temperature 0.1 is recommended for all Falcon-H1 models — higher temperatures lead to inconsistent outputs
- Tools should be provided in `<tools></tools>` XML tags as specified in the model's chat template

```
1 # Install llama.cpp
2 brew install llama.cpp
3
4 # Download the model
5 huggingface-cli download tiiuae/Falcon-H1-Tiny-90M-Tool-Calling-GGUF
6 \
7   Falcon-H1-Tiny-90M-Tool-Calling-BF16.gguf --local-dir ./
8
9 # Run with system prompt file containing your tools
10 llama-cli -m Falcon-H1-Tiny-90M-Tool-Calling-BF16.gguf \
   -cnv --temp 0.1 --system-prompt-file tools.txt
```

Simple prompts where the model answers correctly to user questions

The model correctly selected the appropriate function from a set of six available tools in each case, extracted the relevant parameters from natural language, and produced clean JSON output.

Simple prompt where we ask the model a general knowledge question together with a tool calling question - the model is able to not call the tool when not needed.

The model is also able to not use tool calling when not needed when the answer can be retrieved from general knowledge.

Key Findings

- *CoT is toxic for tool calling models at this scale.* Chain-of-thought reasoning traces in training data cause repetition loops rather than improved reasoning. Small models

lack the capacity to compress complex reasoning patterns. Focusing the model on pure function calling datasets helps the model to perform better in function calling at this extreme small scale.

- *SFT matches pretrain-SFT for structured capabilities.* Tool calling performance is bounded by model capacity, not exposure timing. Targeted SFT achieves the same results as full retraining.
- *Higher tool calling data in the SFT mix is better.* Using curriculum strategy, higher tool calling data percentages maintain general capability with proportional tool calling gains, until reaching a certain upper bound.
- *Relevance over accuracy.* Our Tiny model excels at knowing when to call functions rather than achieving perfect schema compliance. Improving AST accuracy while maintaining relevance detection likely requires higher-quality training data with cleaner schema examples - an area where further tool calling data research and generation could yield gains on both fronts.
- *Competitive parameter efficiency is achievable.* Matching 270M model performance at 90M parameters demonstrates that careful data curation can partially compensate for scale.

Future Work

- Increasing multi-turn capabilities; We observed that across small models, the multi-turn scores were relatively low compared to Qwen-0.6B. A future direction could be to study further why small models struggle in this category and how this can be mitigated.
- More specialization within tool calling categories? For some viable use cases, a fixed set of available tools could be enough to complete the desired tasks. Given the small memory footprint of the tiny models, users could simply fine-tune the tiny base models to comply to their use case. A future direction could be to properly study to which more granular capabilities these tiny models are upper bounded and can match the performance of much larger models.
- Understanding CoT influence on tiny models. Following the analysis from “Wait, Wait, Wait... Why Do Reasoning Models Loop?” [Pipis et al. \(2025\)](#) ↗, we plan to investigate whether specific conditions exist under which tiny models can handle chain-of-thought content without falling into repetition loops. Understanding the precise mechanisms, whether related to sequence length, reasoning complexity, or

training dynamics, could unlock approaches that allow tiny models to benefit from CoT data insights and potentially improve BFCL scores.

Falcon-H1-Tiny-Coder: A small but remarkable python pair programmer in your pocket

Does the conclusions we found above hold for other domains such as coding?

For this model we wanted to demonstrate whether it is possible to reach, or even nearly reach the performance of a larger model in the coding domain while restricting ourselves into a specific programming language (Python) and two relatively simple coding tasks which are:

- Code generation (HumanEval and MBPP ([Austin et al., 2021](#) ↗; [Chen et al., 2021](#) ↗)), as well as their extended variants [Liu et al. \(2023\)](#) ↗
- Fill-in-the-Middle [Bavarian et al. \(2022\)](#) ↗

Following [Bavarian et al. \(2022\)](#) ↗ and [Hui et al. \(2024\)](#) ↗, while being aligned with our previous intuition of pre-training using the target skill and domain from scratch, we inject FIM data from the beginning during the pre-training stage.

FIM format

Following the same learnings from [Bavarian et al. \(2022\)](#) ↗ and [Hui et al. \(2024\)](#) ↗, we decide to go for psm format and use the following prompt format:

```
1 | <|prefix|>{prefix}<|suffix|>{suffix}<|middle|>
```

Therefore, we inject the tokens `<|prefix|>`, `<|suffix|>` and `<|middle|>` in our tokenizer by replacing special reserved tokens with them.

Construction of our FIM data

We constructed our FIM data by applying a Structure-Aware FIM data generation following the methodology from [Gong et al. \(2025\)](#) ↗. Inside the constructed FIM data,

we also inject randomly splitted code snippets in order to make the model robust for random infilling tasks as well.

First attempts

During our first attempts of training Falcon-H1-Tiny-Coder, despite trying multiple different variations of data mixture candidates and data formats, we couldn't get any signal from HumanEval-FIM benchmark results and thought that the model performance was purely bounded by its extreme small size.

Initially, we were looking at two variations of HumanEval-FIM which are:

- HumanEval-FIM *single-line*
- HumanEval-FIM *random-span-light*

The second benchmark gave better signals. In addition to that, we ran benchmark results on HumanEval+ and MBPP+ which gave pretty good signals on all our runs. Therefore we were confident our model was learning how to code properly, but we needed to figure out what went off for FIM tasks and fix it. After doing multiple iteration of local tests and inspecting the generations of the model on HumanEval-FIM, we noticed the following behavior:

The model seemed to predict a complete new function when trying to autocomplete from a new line directly

When prompting the model from an indentation:

The model seemed to correctly predict the continuation of the function when prompting it with an indentation

After inspecting the prompts from HumanEval-FIM, we noticed that the indentation is not included right after the prefix. We could have “hacked” the FIM template to manually inject an indentation after the prefix, however this will be unpracticable in real-world usecases (i.e. what if we are already inside an indented line?), therefore we fixed our FIM data to construct some samples where the model has to predict indentations 50% of the time, as well as constructing samples where indentations are added after the prefix as well.

On masking and un-masking non FIM tokens

We also explore the impact of masking and not masking non-FIM tokens during pre-training. Analogously to a SFT stage, we wanted to know whether masking non-FIM tokens would lead to overall better performance or not, as this technicality is not specified in [Bavarian et al. \(2022\)](#) [↗]. In [Hui et al. \(2024\)](#) [↗], it is not explicitly stated but it seems that no masking is applied since FIM data is used both for learning FIM task and next-token prediction. We confirmed this by running two experiments, using the same data source while one applying loss masking on prompts and the other not masking anything in the data. We train two Falcon-H1-Tiny-90m separately on these data mixes on a total of 80GT, which includes a decay stage of 20GT.

For the data mixes, we allocate 80% of the mix for the FIM data to maximize the signal on FIM tasks, and 10% for pure code data and the rest of 10% evenly splitted between crawled web-data (fineweb and fineweb-EDU [Lozhkov et al. \(2024\)](#) [↗] [Penedo et al. \(2024\)](#) [↗]) and math sources.

Results showed that un-masking gives clearly better results, which might be correlated by the amount of effective trained tokens since the amount of un-masked tokens is much lower than the amount of total tokens. However, we can conclude that given a specific token and compute budget, it is more efficient to train on un-masked data to get overall better end-performance.

Final data and training strategies

Given the experimental results shared above, we fix the data mixture to the un-masked version of the mixture described in the previous section, which consists of 80% of FIM data and the remaining 20% evenly splitted between FineWeb-EDU and math data sources.

We train the end models on 315 GT which includes a learning rate decay stage of 80GT. Given the relatively small amount of available FIM data in our data mixture, we also explore the impact of training the models with dropout to simulatenously study the impact of common regularization techniques in a extreme data repetition scenario. Therefore we trained two models trained under the exact same settings, while one run has dropout being activated after all linear projections (except the final projection head), with a dropout probability of 0.1.

Evolution of various Python coding benchmarks (HumanEval-FIM, HumanEval, MBPP, ...) over the number of training steps

There is a clear signal on the effect of aggressive data repetition on HumanEval-FIM benchmark, which is mitigated by adding dropout. We relatively lose some points on other tasks when applying dropout but given the 'boost' we get on HumanEval-FIM, we chose the dropout checkpoint as our final checkpoint.

End performance

Due to the lack of very small coder models which also support FIM (Fill-in-the-Middle), we evaluate our model against the smallest model from Qwen2.5-Coder series

[Hui et al. \(2024\)](#) ↗.

Coder	Falcon-H1-Tiny-Coder-90M	Qwen-2.5-Coder-0.5B
HumanEval+ (@1)	14.63	23.17
HumanEval (@1)	16.46	27.44
MBPP (@1)	41.26	54.76
MBPP+ (@1)	34.92	48.67
HumanEval-FIM (@1)	22.66	72.95
HumanEval-FIM (@10)	40.75	91.38
HumanEval-FIM-RS (@1)	30.96	31.76
HumanEval-FIM-RS (@10)	56.7	56.7

Example usage

Below we explain how to use the model on your local laptop using [“Continue”](#) VS Code plugin and llama-server from llama.cpp as a backend to serve the tiny coder model.

First of all, download one of our GGUF models from HuggingFace in your local setup, you can achieve this by using the hf cli tool. For example:

```
1 | hf download tiiuae/Falcon-H1-Tiny-Coder-GGUF Falcon-H1-Tiny-Coder-Q8_0.gguf --local-dir ./
```

Then, make sure to install Continue plugin from VS Code extension marketplace as well as installing llama.cpp on your laptop.

After that, open Continue extension, go to “Models” then click on the add button to add a new model. This should display this dialog box.

Simply select the underlined 'config file' in order to proceed to modify the Continue config file

Carefully select the underlined “config file” to modify the Continue config file and add the configuration of the downloaded model on it by making sure to align on the FIM template.

```
1 name: Local Assistant
2 version: 1.0.0
3 schema: v1
4 models:
5 - name: tiny-fim
6 provider: llama.cpp
7 apiBase: http://localhost:8080
8 model: ./Falcon-H1-Tiny-Coder-Q8_0.gguf
9 roles:
10 - autocomplete
11 defaultCompletionOptions:
12   temperature: 0.8
13   maxTokens: 1024
14   autocompleteOptions:
15     disable: false
16     maxPromptTokens: 256
17     maxSuffixPercentage: 0.8
18     prefixPercentage: 0.8
19     onlyMyCode: true
20 promptTemplates:
21   autocomplete: <|prefix|>{{{prefix}}}<|suffix|>{{{suffix}}}
                <|middle|>
```

The API address is set to the default one which is being created when llama-server is launched.

In a separate terminal, launch `llama-server`:

```
1 llama-server -m ./Falcon-H1-Tiny-Q8_0.gguf
```

Model usage

The models are available to use with a wide range of tools from the ecosystem, below is a non-exhaustive list of supported tools:

llama.cpp

It is possible to use our models with the latest version of llama.cpp. Refer to the commands below to get started:

```
1 | brew install llama.cpp
2 | pip install huggingface_hub
3 | hf download tiiuae/Falcon-H1-Tiny-90M-Instruct-GGUF Falcon-H1-Tiny-90M-Instruct-GGUF-Q8_0.gguf --local-dir ./
4 | llama-cli ./Falcon-H1-Tiny-90M-Instruct-GGUF-Q8_0 -cnv
```

ollama

ollama also supports Falcon-H1 architecture, therefore you can refer to the command below:

```
1 | ollama run hf.co/tiiuae/Falcon-H1-Tiny-90M-Instruct-GGUF:Q8_0
```

Apple MLX

It is also possible to use our models with Apple MLX ([Hannun et al., 2023 ↗](#)) - an inference and training engine tailored for Apple Metal devices.

```
1 | pip install mlx_lm
2 | mlx_lm.chat --model tiiuae/Falcon-H1-Tiny-90M-Instruct
```

Hugging Face transformers

You can also use our models with `transformers serve` ([Wolf et al., 2020 ↗](#))

```
1 | transformers serve tiiuae/Falcon-H1-Tiny-90M-Instruct
```

vLLM & sglang

While it will likely be an overkill to do so, the models are also supported on deployment-server oriented tools such as:

- vLLM ([Kwon et al., 2023](#))

```
1 | vllm serve tiiaae/Falcon-H1-Tiny-90M-Instruct \
2 |   --tensor-parallel-size 1 \
3 |   --data-parallel-size 1
```

- `sglang` ([Zheng et al., 2024 ↗](#))

```
1 | python -m sglang.launch_server \
2 |   --model tiiaae/Falcon-H1-Tiny-90M-Instruct \
3 |   --tensor-parallel-size 1
```

All our models can be found in this Hugging Face collection.

Future Work

We believe that our work will bring valuable insights and added values to the open-source community. From pre-training data strategies to studying the impact of different optimization algorithms – while checking its universality across multiple key domains (general chat, agentic, multilingual, code, reasoning) - we hope this release will inspire future work and research around extreme small-scale language models.

Below is a non-exhaustive list of potential future directions:

- Model merging – what is the impact of model merging for extreme small-scale models? Would a potential future paradigm of training LLM be training multiple extreme small-scale models and merge them to combine their capabilities?
- Stronger work on data repetition and forgetting window: We need to deeply study this question as multiple signals from our work demonstrated the effectiveness of multi-epoch training in the context of language models and get back to Deep Learning fundamentals (e.g. Dropout, and other regularization techniques as shown in the final experiment on the coder variant).
- Quantization at an extremely small scale: What would be the impact of aggressively quantizing tiny models? We internally *vibe-tested* a 4-bit quantized version of our tiny coder model and observed very satisfying results. Properly studying the impact of quantization on this extreme small scale will give us better signals on the impact of quantizing models on this scale.

- Merging RL with pre-training: Current RL pipelines are slow, and we have seen that the earlier a skill is seen by a model the better it is. While it is straightforward to merge Pre-training and reasoning SFT, the next step would be to merge RL phase as well with these two steps.
- Scaling up slightly more? There might be a sweet spot in terms of model capabilities and size and 90M-100M scale might not be ideal for some usecases such as multilingual. Properly studying scaling laws for small scale models on a maximum number of domains will help us to determine a clearer picture of what we can achieve with small language models.

Acknowledgements

This work would not have been possible without the collaboration, through technical discussions, contributions and feedback from some members of the Falcon-LLM Team which includes the following (sorted in last name alphabetical order):

Abdalgader Abubaker, Reda Alami, Ali Almansoori, Omar Saif AlKaabi, Hamza Alobeidi, Leen AlQadi, Shaikha Alsuwaidi, Ahmed Alzubaidi, Mohamed Alyafeai, Pasquale Balsebre*, Younes Belkada*, Basma Boussaha, Iheb Chaabane, Ilyas Chahed*, Slim Frikha, Shi Hu, Puneesh Khanna, Abhay Kumar*, Mikhail Lubinets*, Suhail Mohamad, Dhia Eddine Rhaiem*, Mohamed El Amine Seddik, Maksim Velikanov*, Jingwei Zuo*, Hakim Hacid

* Core Technical Contributors

Citation

For attribution in academic contexts, please cite this work as

Falcon-LLM Team (2026). "Falcon-H1-Tiny: A series of extremely small, yet powerful language models redefining capabilities at small scale".

BibTeX citation

```
@misc{falcon_h1_tiny,  
  title={Falcon-H1-Tiny: A series of extremely small, yet powerful language models  
  redefining capabilities at small scale},  
  author={Falcon-LLM Team},  
  year={2026},  
}
```

Reuse

Diagrams and text are licensed under [CC-BY 4.0](#) with the source available on [Hugging Face](#), unless noted otherwise. Figures reused from other sources are excluded and marked in their captions (“Figure from ...”).

References

1. Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., & Schulman, J. (2021). Training Verifiers to Solve Math Word Problems. *arXiv Preprint arXiv:2110.14168*. [↑](#)
2. Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., & Steinhardt, J. (2021). Measuring Massive Multitask Language Understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*. [↑](#)
3. Liu, J., Su, J., Yao, X., Jiang, Z., Lai, G., Du, Y., Qin, Y., Xu, W., Lu, E., Yan, J., Chen, Y., Zheng, H., Liu, Y., Liu, S., Yin, B., He, W., Zhu, H., Wang, Y., Wang, J., ... Yang, Z. (2025). *Muon is Scalable for LLM Training*. <https://arxiv.org/abs/2502.16982> [↑](#)
4. Suzgun, M., Scales, N., Schärli, N., Gehrmann, S., Tay, Y., Chung, H. W., Chowdhery, A., Le, Q. V., Chi, E. H., Zhou, D., & Wei, J. (2022). Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them. *arXiv Preprint arXiv:2210.09261*. [↑](#)
5. Velikanov, M., Chahed, I., Zuo, J., Rhaiem, D. E., Belkada, Y., & Hacid, H. (2026). *Learnable Multipliers: Freeing the Scale of Language Model Matrix Layers*. <https://arxiv.org/abs/2601.04890> [↑](#) back: [1](#), [2](#)
6. Zuo, J., Velikanov, M., Chahed, I., Belkada, Y., Rhayem, D. E., Kunsch, G., Hacid, H., Yous, H., Farhat, B., Khadraoui, I., Farooq, M., Campesan, G., Cojocar, R., Djilali, Y., Hu, S., Chaabane, I., Khanna, P., Seddik, M. E. A., Huynh, N. D., ... Frikha, S. (2025). *Falcon-H1: A Family of Hybrid-Head Language Models Redefining Efficiency and Performance*. <https://arxiv.org/abs/2507.22448> [↑](#)
7. Allal, L. B., Lozhkov, A., Bakouch, E., Blázquez, G. M., Penedo, G., Tunstall, L., Marafioti, A., Kydlíček, H., Lajarín, A. P., Srivastav, V., Lochner, J., Fahlgren, C., Nguyen, X.-S., Fourrier, C., Burtenshaw, B., Larcher, H., Zhao, H., Zarka, C., Morlon, M., ... Wolf, T. (2025). *SmolLM2: When Smol Goes Big – Data-Centric Training of a Small Language Model*. <https://arxiv.org/abs/2502.02737> [↑](#)
8. Bai, G., Liu, J., Bu, X., He, Y., Liu, J., Zhou, Z., Lin, Z., Su, W., Ge, T., Zheng, B., & Ouyang, W. (2024). MT-Bench-101: A Fine-Grained Benchmark for Evaluating Large Language Models in Multi-Turn Dialogues. *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 7421–7454. [10.18653/v1/2024.acl-long.401](https://doi.org/10.18653/v1/2024.acl-long.401) [↑](#)
9. Berant, J., Chou, A., Frostig, R., & Liang, P. (2013). Semantic Parsing on Freebase from Question-Answer Pairs. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 1533–1544. <https://aclanthology.org/D13-1160> [↑](#)

10. Bisk, Y., Zellers, R., Bras, R. L., Gao, J., & Choi, Y. (2020). PIQA: Reasoning about Physical Commonsense in Natural Language. *Thirty-Fourth AAAI Conference on Artificial Intelligence*. [↑](#)
11. Clark, C., Lee, K., Chang, Ming-Wei, Kwiatkowski, T., Collins, Michael, & Toutanova, K. (2019). BoolQ: Exploring the Surprising Difficulty of Natural Yes/No Questions. *NAACL*. [↑](#)
12. Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., & Tafjord, O. (2018). Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge. *ArXiv, abs/1803.05457*. [↑](#)
13. Cobbe, K., Kosaraju, V., Bavarian, M., Hilton, J., Nakano, R., Hesse, C., & Schulman, J. (2021). *Training Verifiers to Solve Math Word Problems*. [↑](#)
14. Dua, D., Wang, Y., Dasigi, P., Stanovsky, G., Singh, S., & Gardner, M. (2019). *DROP: A Reading Comprehension Benchmark Requiring Discrete Reasoning Over Paragraphs*. [↑](#)
15. Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., & Steinhardt, J. (2021). Measuring Mathematical Problem Solving With the MATH Dataset. *arXiv Preprint arXiv:2103.03874*. [↑](#)
16. Jakimovski, B. (2025). *Can Tiny Language Models Reason?* [↑](#)
17. Lai, G., Xie, Q., Liu, H., Yang, Y., & Hovy, E. (2017). RACE: Large-scale ReAding Comprehension Dataset From Examinations. In M. Palmer, R. Hwa, & S. Riedel (Eds.), *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (pp. 785–794). Association for Computational Linguistics. [10.18653/v1/D17-1082](#) [↑](#)
18. Li, X., Zhang, T., Dubois, Y., Taori, R., Gulrajani, I., Guestrin, C., Liang, P., & Tatsunori B. Hashimoto. (2023). AlpacaEval: An Automatic Evaluator of Instruction-following Models. In *GitHub repository*. GitHub. https://github.com/tatsu-lab/alpaca_eval [↑](#)
19. Lin, S., Hilton, J., & Evans, O. (2022). TruthfulQA: Measuring How Models Mimic Human Falsehoods. *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 3214–3252. [10.18653/v1/2022.acl-long.229](#) [↑](#)
20. Nie, Y., Williams, A., Dinan, E., Bansal, M., Weston, J., & Kiela, D. (2020). Adversarial NLI: A New Benchmark for Natural Language Understanding. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. [↑](#)
21. Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., & Fernández, R. (2016). *The LAMBADA dataset: Word prediction requiring a broad discourse context*. <https://arxiv.org/abs/1606.06031> [↑](#)
22. Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., & Bowman, S. R. (2023). *GPQA: A Graduate-Level Google-Proof Q&A Benchmark*. [↑](#)
23. Sakaguchi, K., Bras, R. L., Bhagavatula, C., & Choi, Y. (2019). WinoGrande: An Adversarial Winograd Schema Challenge at Scale. *arXiv Preprint arXiv:1907.10641*. [↑](#)
24. Sprague, Z., Ye, X., Bostrom, K., Chaudhuri, S., & Durrett, G. (2024). *MuSR: Testing the Limits of Chain-of-thought with Multistep Soft Reasoning*. <https://arxiv.org/abs/2310.16049> [↑](#)
25. Team, G., Kamath, A., Ferret, J., Pathak, S., Vieillard, N., Merhej, R., Perrin, S., Matejovicova, T., Ramé, A., Rivière, M., Rouillard, L., Mesnard, T., Cideron, G., bastien Jean-Grill, Ramos, S., Yvinec, E., Casbon, M.,

- Pot, E., Penchev, I., ... Hussenot, L. (2025). *Gemma 3 Technical Report*. <https://arxiv.org/abs/2503.19786> ↑
26. Wang, Y., Ma, X., Zhang, G., Ni, Y., Chandra, A., Guo, S., Ren, W., Arulraj, A., He, X., Jiang, Z., Li, T., Ku, M., Wang, K., Zhuang, A., Fan, R., Yue, X., & Chen, W. (2024). *MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark*. ↑
27. Welbl, J., Liu, N. F., & Gardner, M. (2017). Crowdsourcing Multiple Choice Science Questions. *NUT@EMNLP*. ↑
28. White, C., Dooley, S., Roberts, M., Pal, A., Feuer, B., Jain, S., Shwartz-Ziv, R., Jain, N., Saifullah, K., Dey, S., Shubh-Agrawal, Sandha, S. S., Naidu, S. V., Hegde, C., LeCun, Y., Goldstein, T., Neiswanger, W., & Goldblum, M. (2025). LiveBench: A Challenging, Contamination-Free LLM Benchmark. *The Thirteenth International Conference on Learning Representations*. ↑
29. Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., & Choi, Y. (2019). HellaSwag: Can a Machine Really Finish Your Sentence? *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. ↑
30. Zhao, C., Chang, E., Liu, Z., Chang, C.-J., Wen, W., Lai, C., Cao, S., Tian, Y., Krishnamoorthi, R., Shi, Y., & Chandra, V. (2025). *MobileLLM-R1: Exploring the Limits of Sub-Billion Language Model Reasoners with Open Training Recipes*. <https://arxiv.org/abs/2509.24945> ↑
31. Zhou, J., Lu, T., Mishra, S., Brahma, S., Basu, S., Luan, Y., Zhou, D., & Hou, L. (2023). *Instruction-Following Evaluation for Large Language Models*. <https://arxiv.org/abs/2311.07911> ↑ back: [1](#), [2](#)
32. Dac Lai, V., Van Nguyen, C., Ngo, N. T., Nguyen, T., Dernoncourt, F., Rossi, R. A., & Nguyen, T. H. (2023). Okapi: Instruction-tuned Large Language Models in Multiple Languages with Reinforcement Learning from Human Feedback. *arXiv E-Prints*, arXiv:2307. ↑
33. Shi, F., Suzgun, M., Freitag, M., Wang, X., Srivats, S., Vosoughi, S., Chung, H. W., Tay, Y., Ruder, S., Zhou, D., Das, D., & Wei, J. (2022). *Language Models are Multilingual Chain-of-Thought Reasoners*. ↑
34. Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., & Cobbe, K. (2023). Let's Verify Step by Step. *arXiv Preprint arXiv:2305.20050*. ↑
35. Naman Jain. (2024). LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. *arXiv Preprint*. ↑
36. Olmo, T., :, Ettinger, A., Bertsch, A., Kuehl, B., Graham, D., Heineman, D., Groeneveld, D., Brahman, F., Timbers, F., Ivison, H., Morrison, J., Poznanski, J., Lo, K., Soldaini, L., Jordan, M., Chen, M., Noukhovitch, M., Lambert, N., ... Hajishirzi, H. (2025). *Olmo 3*. <https://arxiv.org/abs/2512.13961> ↑
37. Pipis, C., Garg, S., Kontonis, V., Shrivastava, V., Krishnamurthy, A., & Papailiopoulos, D. (2025). *Wait, Wait, Wait... Why Do Reasoning Models Loop?* <https://arxiv.org/abs/2512.12895> ↑
38. Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., & Guo, D. (2024). *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models*. <https://arxiv.org/abs/2402.03300> ↑
39. Team, F. L., Chaabane, I., Khanna, P., Mohmad, S., Frikha, S., Hu, S., Abubaker, A., Alami, R., Lubinets, M., Seddik, M. E. A., & Hacid, H. (2026). *Falcon-H1R: Pushing the Reasoning Frontiers with a Hybrid Model for Efficient Test-Time Scaling*. <https://arxiv.org/abs/2601.02346> ↑

40. Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., ... Qiu, Z. (2025). *Qwen3 Technical Report*. <https://arxiv.org/abs/2505.09388> ↑
41. Zhang, Y., & Math-AI, T. (2024). *American Invitational Mathematics Examination (AIME) 2024*. ↑
42. Zhang, Y., & Math-AI, T. (2025). *American Invitational Mathematics Examination (AIME) 2025*. ↑
43. Patil, S. G., Mao, H., Cheng-Jie Ji, C., Yan, F., Suresh, V., Stoica, I., & E. Gonzalez, J. (2025). The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. *Forty-Second International Conference on Machine Learning*. ↑
44. Pipis, C., Garg, S., Kontonis, V., Shrivastava, V., Krishnamurthy, A., & Papailiopoulos, D. (2025). *Wait, Wait, Wait... Why Do Reasoning Models Loop?* <https://arxiv.org/abs/2512.12895> ↑ back: [1](#), [2](#)
45. Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., & Sutton, C. (2021). *Program Synthesis with Large Language Models*. <https://arxiv.org/abs/2108.07732> ↑
46. Bavarian, M., Jun, H., Tezak, N., Schulman, J., McLeavey, C., Tworek, J., & Chen, M. (2022). *Efficient Training of Language Models to Fill in the Middle*. <https://arxiv.org/abs/2207.14255> ↑ back: [1](#), [2](#), [3](#), [4](#)
47. Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., ... Zaremba, W. (2021). *Evaluating Large Language Models Trained on Code*. <https://arxiv.org/abs/2107.03374> ↑
48. Gong, L., Cheung, A., Elhoushi, M., & Wang, S. (2025). *Structure-Aware Fill-in-the-Middle Pretraining for Code*. <https://arxiv.org/abs/2506.00204> ↑
49. Hui, B., Yang, J., Cui, Z., Yang, J., Liu, D., Zhang, L., Liu, T., Zhang, J., Yu, B., Lu, K., Dang, K., Fan, Y., Zhang, Y., Yang, A., Men, R., Huang, F., Zheng, B., Miao, Y., Quan, S., ... Lin, J. (2024). *Qwen2.5-Coder Technical Report*. <https://arxiv.org/abs/2409.12186> ↑ back: [1](#), [2](#), [3](#), [4](#)
50. Liu, J., Xia, C. S., Wang, Y., & Zhang, L. (2023). Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation. *Thirty-Seventh Conference on Neural Information Processing Systems*. <https://openreview.net/forum?id=1qvx610Cu7> ↑
51. Lozhkov, A., Ben Allal, L., von Werra, L., & Wolf, T. (2024). *FineWeb-Edu: the Finest Collection of Educational Content*. Hugging Face . <https://doi.org/10.57967/hf/2497> ↑
52. Penedo, G., Kydliuček, H., allal, L. B., Lozhkov, A., Mitchell, M., Raffel, C., Werra, L. V., & Wolf, T. (2024). The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale. *The Thirty-Eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*. <https://openreview.net/forum?id=n6SCKn2QaG> ↑
53. Hannun, A., Digani, J., Katharopoulos, A., & Collobert, R. (2023). *MLX: Efficient and flexible machine learning on Apple silicon* (0.0). <https://github.com/ml-explore> ↑
54. Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., & Stoica, I. (2023). Efficient Memory Management for Large Language Model Serving with PagedAttention. *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*. ↑

55. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., ... Rush, A. M. (2020). Transformers: State-of-the-Art Natural Language Processing. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45. <https://www.aclweb.org/anthology/2020.emnlp-demos.6>↑
56. Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Barrett, C., & Sheng, Y. (2024). *SGLang: Efficient Execution of Structured Language Model Programs*. <https://arxiv.org/abs/2312.07104>↑

made with love with [research article template](#)