How to Train Your HRM

Sam Olesker-Taylor, Erika Aranas, Michael Pearce & Luke Hudlass-Galley
Graphcore Research
Prudential Buildings, Wine St,
Bristol BS1 2PH, UK
{samot,erikaa,michaelp,lukehg}@graphcore.ai

Abstract

Hierarchical Reasoning Models (HRMs) are a recently proposed model architecture for solving complex reasoning tasks such as the Abstract and Reasoning Corpus (ARC-AGI) challenge: the objective is to learn an underlying transformation, demonstrated by example input–output pairs. The HRM learns transformations via supervised learning on the demonstration pairs. Each task involves an entirely new transformation, necessitating test-time training on the evaluation tasks.

We investigate training curricula for HRMs to compensate for limited test-time compute, focused on three stages: offline pre-training on available training data; test-time fine-tuning on evaluation tasks; test-time, per-task 'overfitting', in which a specialized model is trained for each task. Our results suggest that pre-training can offer early gains, which may not persist, and that fine-tuning on all tasks (training and evaluation) is optimal. The majority of test-time compute should be spent on fine-tuning, rather than overfitting—typically 2:1 or more.

1 Introduction

Large language models (LLMs) have recently seen remarkable success when solving complex reasoning problems through paradigms such as chain-of-thought and reinforcement-learning-based training schemes. However, the autoregressive nature of these models means they often struggle with accurately inferring logic and rule-based reasoning. This is highlighted by their performance in the **Abstract and Reasoning Corpus (ARC-AGI) challenge** (Chollet, 2019).

An ARC task is characterised by an *implicit transformation rule*. The goal is to infer this transformation from a collection (usually 2–4) of demonstration input–output pairs of grids, and evaluate the model's understanding of the transformation by validating it on a unseen test input grid. It is specifically designed to be difficult for LLMs: DeepSeek-R1, a model with 671B parameters, achieves under 16% accuracy on ARC-AGI-1, while GPT-4o attains less than 5% (ARC Prize). OpenAI's o3 attained 88% at its launch in late 2024, but at a cost of nearly $\$5\,000$ per task (Chollet, 2024b).

In contrast to conventional reasoning models, the recently-proposed **Hierarchical Reasoning Model (HRM)** has just 27M parameters, yet achieves 40% accuracy in ARC-AGI-1 (Wang et al., 2025). There is no in-context learning: each data-point is a *single* input–output pair, with its task identifier attached. The model is given an input grid and must predict the corresponding output. Training proceeds in a standard supervised manner, training on the demo pairs and evaluating on the test pairs.

The final objective is to correctly predict the test output for each task in the ARC evaluation set. The corresponding demo pairs are required to infer the appropriate transformation, so this falls into the framework of *test-time training*. A similar approach is used by other models tackling the ARC challenge (Barbadillo, 2024; Wu, 2024; Akyürek et al., 2025).

The official ARC challenge has strict compute requirements at submission time (ARC, 2025). The standard approach for HRM and related models collates all data into the test-time training (Wang et al., 2025; Jolicoeur-Martineau, 2025; ARC Prize Foundation, 2025). Given that training data is available pre-submission, we explore different curricula for training HRMs.

The remainder of the paper is structured as follows.

§2 We provide a high-level narrative of our investigations, the training curricula and datasets.

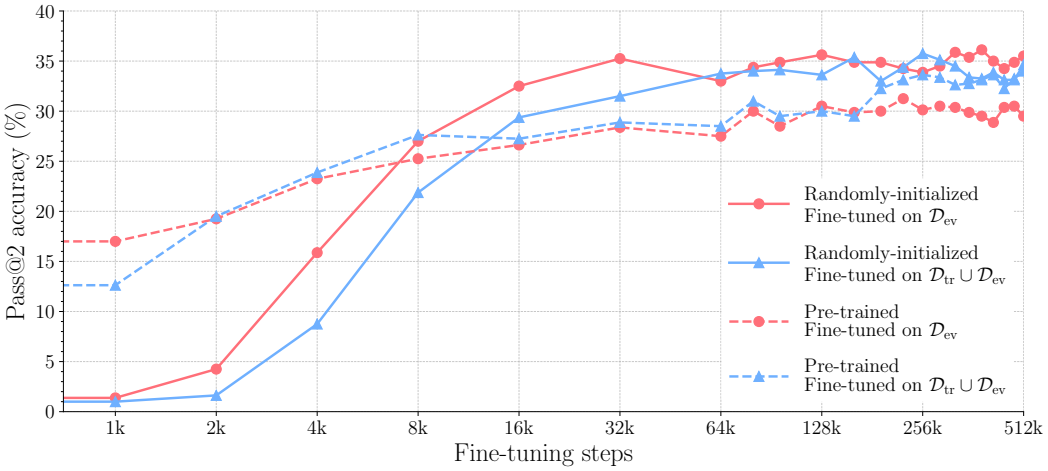

Figure 1: Starting from either a randomly initialized or a pre-trained model, fine-tuning is performed on either the entire dataset $\mathcal{D}_{tr} \cup \mathcal{D}_{ev}$ or just evaluation tasks $\mathcal{D}_{ev}$.

§3 We detail precisely the experiments and configurations, including a discussion of our results.

2 HRM Training Curricula

Test-time training, rather than just inference, is a popular strategy in the ARC challenge. The ARC dataset (Chollet, 2024a) provides a training and (public) evaluation split, denoted as $\mathcal{D}_{tr}$ and $\mathcal{D}_{ev}$, respectively. Each split has a demo and test input-output pairs. Performance is measured on the test pairs in the evaluation split—whose output grid must be hidden during any training. Each split consists of 400 tasks. For the avoidance of doubt, we treat the public evaluation split as if it were the private one: we cannot touch it before submission (test time). The public evaluation split is "roughly the same level of difficulty" as the private one (ARC Prize, 2025).

Previous works submit an untrained (randomly initialized) HRM, and perform test-time training on the entire dataset. Intuitively, this feels wasteful: why not pre-train on the training split, then perform test-time training on only the evaluation split? This way, each task is seen the same number of times as previously, but halves the compute required at test time. In this setting, the *order* is different though: instead of being mixed, all of $\mathcal{D}_{tr}$ is processed, then all of $\mathcal{D}_{ev}$.

Comparing the solid versus dotted curves in Figure 1 reveals a sizable initial boost from pre-training, which we attribute to meta-learning: learning to learn transformations. However, the ordering mattered for final performance: mixing $\mathcal{D}_{tr}$ and $\mathcal{D}_{ev}$ proved better than sequential training.

This raises the question of whether it is worth using the training split at all—after all, the model is never evaluated on it. Without pre-training and for a given number of steps, we observe that test-time training on only the evaluation dataset is better than using the entire dataset, as demonstrated by the two solid curves in Figure 1. Our interpretation is that training tasks may provide some value, but test-time compute budget is better spent on evaluation tasks most relevant to the final test set.

If the tasks from training don't really help those in evaluation, do the tasks inside evaluation help each other? We take this to the extreme: for each evaluation task, we initialize a new model and perform test-time training on only its demo pairs before evaluating on its test pair. The idea is to use the specific task's demo pairs, along with augmented versions, to overfit the model to the underlying transformation. The small size of HRMs enables this: even the longest individual overfitting run takes only 30 minutes on a single H100 GPU.

There are therefore three stages to training, for which we adopt the following terminology.

Pre-training: training the model before test-time on only $\mathcal{D}_{tr}$ (the training split). Generalization accuracy is not measured here, since no tasks in $\mathcal{D}_{ev}$ (the evaluation split) are considered.

Fine-tuning: test-time training using either $\mathcal{D}_{\text{tr}} \cup \mathcal{D}_{\text{ev}}$ (the standard HRM approach) or only $\mathcal{D}_{\text{ev}}$. Accuracy on $\mathcal{D}_{\text{ev}}$ is measured using the same trained model for all 400 tasks.

Overfitting: test-time training, loaded from a fine-tuned checkpoint, trained on demo pairs from a *single* $\mathcal{D}_{\text{ev}}$ task. Accuracy is the average of the 400 models' on their respective test pairs.

Each stage of training may be initialized by a checkpoint from a previous stage.

We do not constrain the pre-training compute budget. We report performance as a function of test-time (i.e., fine-tuning or overfitting) compute. The best training configuration for a given target performance is that which achieves it with minimal test-time compute.

3 Experiments and Configurations

3.1 Configuration

All our experiments are conducted with the default HRM framework, according to the official GitHub repository (Sapient Intelligence, 2025), with an exception around the batch size and learning rate.

HRM Architecture A single forward pass of an HRM comprises of repeated applications of 'high'- and 'low'-level transformer blocks. The details are not important to our investigations, so we defer the reader to the original paper (Wang et al., 2025). What is important is its *recurrence*: rather than exiting after a single forward pass, the model recurses, as many as 16 times in total.

Hyperparameters Pre-training and fine-tuning are performed on an $8\times$H100 GPU system with a global batch size of 3 072 with learning rate 4×10^{-4}. The original repo used batch size $768 = 3\,072/4$ and learning rate 1×10^{-4} (Sapient Intelligence, 2025). For overfitting, each model uses a single H100 with batch size $384 = 3\,072/8$ and learning rate $5 \times 10^{-5} = (4 \times 10^{-4})/8$.

Data Augmentation ARC tasks lend themselves naturally to data augmentation, and is widely applied (ARC Prize Team, 2025; Franzen et al., 2025; Barbadillo, 2024): rotating, flipping or changing the colors of an ARC task yields another legitimate task, with a functionally equivalent transformation. The original HRM implementation uses 1000 augmentations (Wang et al., 2025), but an ablation study by the ARC Prize Team found equivalent performance at 300 (ARC Prize Team, 2025). We create an augmented dataset for each task comprising 300 random augmentations.

Training versus Evaluation During training, a loss is calculated after *every* forward pass of the HRM, and the weights are updated by backpropagation. The model does not wait for a single datapoint to complete, which may recurse the HRM up to 16 times. During evaluation, no early exiting is permitted: each input grid receives 16 recursions of the HRM before exiting. We plan to explore the implications of the discrepancy between training and evaluation settings in future work.

Metrics A *step* is precisely one forward pass, since weights are updated after each. It is a fixed amount of compute, for a given batch size. Overfitting jobs require training 400 models, each of whose batch size is $1/8$ that of a fine-tuning one. Thus, we assume that one overfitting step is equivalent to $50 = 400/8$ fine-tuning steps. We report accuracy against fine-tuning-equivalent steps.

Following the ARC challenge rules (ARC, 2025), we report accuracy under pass@2. Multiple predictions are obtained by evaluating all 300 augmentations of the test input, then undoing the augmentation (e.g., swapping the colors back). The top two are chosen by vote counting.

3.2 Experiments

Pre-trained versus randomly-initialized We randomly initialized a 'seed checkpoint', which was used for all runs. When using a pre-trained checkpoint, this was checkpointed after 512k steps.

Fine-tuning Given an initial checkpoint, either the randomly initialized or pre-trained one, we perform (test-time) fine-tuning, either on the full dataset $\mathcal{D}_{\text{tr}} \cup \mathcal{D}_{\text{ev}}$ or just the evaluation split $\mathcal{D}_{\text{ev}}$. These four cases are illustrated in Figure 1, as well as the solid black lines in Figure 2.

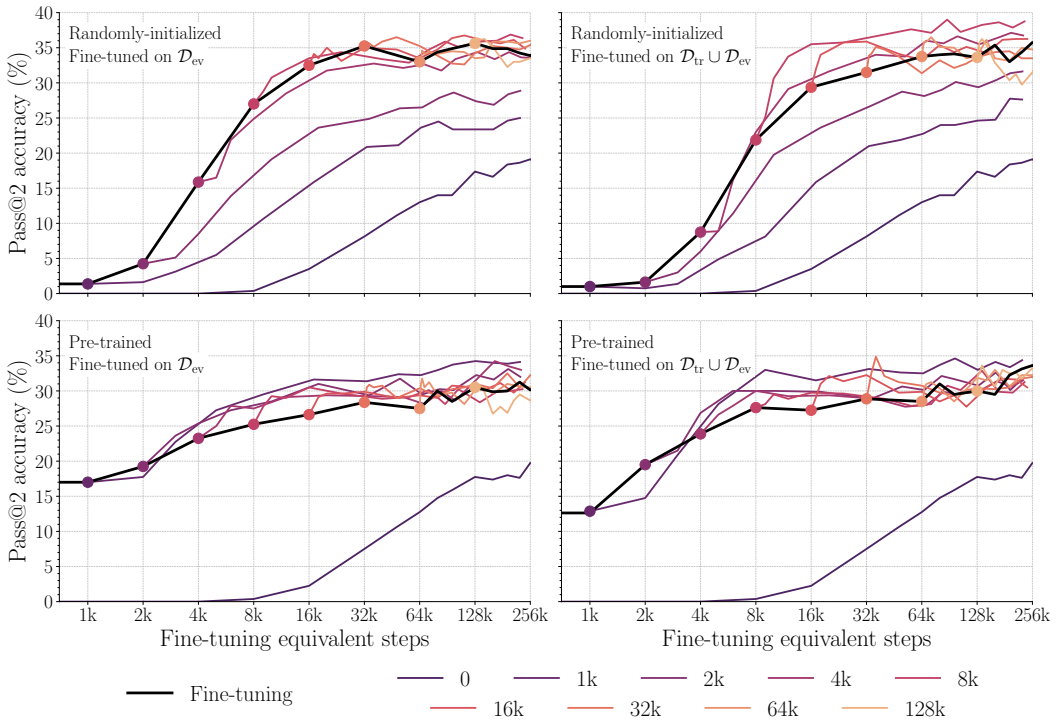

Figure 2: Pass@2 accuracies. All experiments are run up to the equivalent of 256k fine-tuning steps. The overfitting runs branch off from the 'trunk' fine-tuning run (black) at various checkpoints.

Overfitting For each of the four cases above, we checkpoint after 1k, 2k, 4k, 8k, 16k, 32k, 64k and 128k steps. For each of these, we initialize 400 models, one per evaluation task, and further train on that task's demo pairs. As above, a single overfitting step is equivalent to 50 fine-tuning steps. All cases are illustrated in Figure 2. The position on the horizontal axis indicates the *total* test-time compute, in units of fine-tuning steps—including both fine-tuning up to the checkpoint and any overfitting.

We want the model to overfit to the task's underlying transformation, and hence correctly predict the test output grid, not overfit to its exact demo pairs. The use of data augmentation encourages generalization to the test pair: instead of having a dataset of size 3, say, it is of size $3 \times 300 = 900$.

3.3 Interpretation of Results

We now discuss particular features of the results of these experiments, and hypothesize as to the underlying mechanisms explaining them.

Initial checkpoint The pre-trained model has better performance than the randomly-initialized model initially. However, its accuracy plateaus more quickly, and is soon overtaken. We hypothesize that pre-training is a type of meta-learning: pre-training teaches the model *how to learn*, which enables it to quickly pick up new transformations, leading to early gains. However, the transformations appearing in the evaluation split are fundamentally different to those in the training split. We also point out that the training split "consists of simpler tasks" than the evaluation split (ARC Prize, 2025). We attribute (part of) the degradation in final performance to this distribution shift.

Dataset choice The randomly-initialized accuracy curves have a similar qualitative shape, just offset. The blue line, which has training tasks as well as evaluation ones, lags behind the red one which only had evaluation. We hypothesize that this dilutes the 'quality' of the dataset: given enough steps, the same performance is attained, just more slowly—almost by a factor 10 to hit 35%.

Whilst the training split may provide some value if its compute were free, our results indicate that the evaluation tasks should be prioritized given a restricted test-time compute budget.

Overfitting curves Overfitting does not consistently help when there is no pre-training (first row in Figure 2)—remember, the branching point must be chosen without access to the test accuracy plots. With pre-training (second row), the picture is different. At least from 4k to 32k, the transition from fine-tuning to overfitting causes an initial spike. In the case of fine-tuning on $\mathcal{D}_{\text{ev}}$ only, the early checkpoints (1k–4k) maintain a benefit; we would say the same for $\mathcal{D}_{\text{tr}} \cup \mathcal{D}_{\text{ev}}$ too, were we to restrict to 128k steps. There are similar spikes on the top-right, but the gain is not consistently maintained.

We hypothesize that the focus on one specific evaluation task may help the model escape a local minimum corresponding to the training distribution. Future work will investigate this, including playing with variable learning rate—e.g., start high to escape the local minimum, then reduce for convergence. We also would try increasing the augmentation count just for the overfitting step, perhaps by as much as a factor 100, to really avoid overfitting to the specific demo pairs.

Finally, we caution that the overfit curves are noisy, and repetition is required. A model must be chosen without knowledge of the *test* statistics. So, even if statistical significance were established with further samples, it wouldn't indicate performance of a *given* overfitting run.

4 Conclusion

We investigate training curricula for HRMs on ARC involving three stages: pre-training, (test-time) fine-tuning, and (test-time) overfitting, to determine the most efficient use of test-time compute.

Using pre-trained weights eases the compute requirements at test time, and the early accuracy gains observed during the subsequent fine-tuning stage suggest that cross-task learning may be beneficial. However, in our setup, these gains were short-lived, and the final performance actually degraded versus a random initialization. Whether these gains can be sustained by adjusting the training configurations (e.g., adopting a variable learning rate) is worth further study.

Our experiments indicate that fine-tuning on all tasks ($\mathcal{D}_{\text{tr}} \cup \mathcal{D}_{\text{ev}}$) is optimal and that the majority of test-time compute should be spent on fine-tuning, rather than overfitting—typically 2:1 or more.

Given that inference dominates AI workload costs, our work is a step towards understanding the most compute-efficient training curricula for adapting to out-of-distribution reasoning tasks at test time.

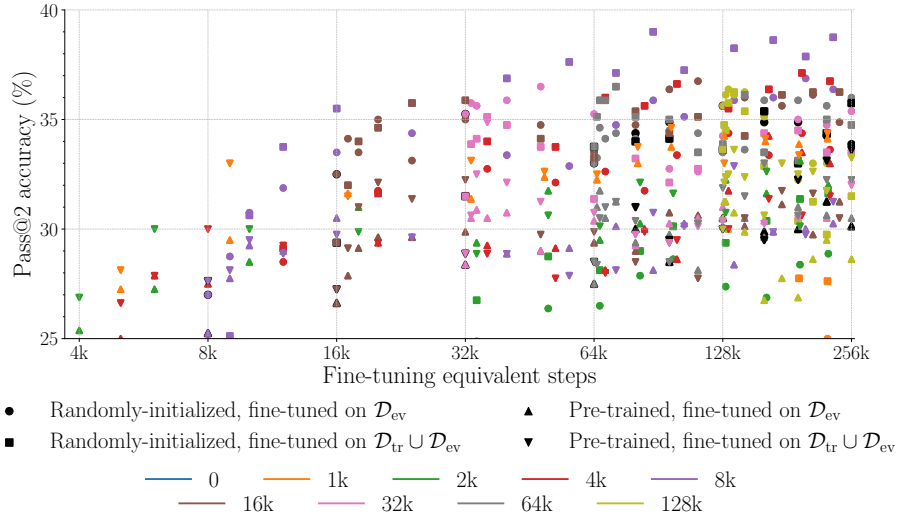

Figure 3: Pareto curve: accuracy vs test-time compute; pre-training is always free. The boundary always fine-tunes on the full $\mathcal{D}_{\text{tr}} \cup \mathcal{D}_{\text{ev}}$. The optimal curriculum depends on the compute available: for low compute, pre-train, then spend the majority on fine-tuning, with a small amount of overfitting; for high compute, do not pre-train, then spend about 8k on overfitting and the remainder on fine-tuning.

REFERENCES

ARC Prize 2025 Competition Rules. https://www.kaggle.com/competitions/arc-prize-2025/rules, 2025. Accessed: 2026-02-06.

Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. The Surprising Effectiveness of Test-Time Training for Abstract Reasoning. https://arxiv.org/abs/2411.07279, 2025. Accessed: 2026-02-08.

ARC Prize. ARC-AGI-1 Leaderboard. URL https://arcprize.org/leaderboard. Accessed: 2026-02-05.

ARC Prize. Official ARC Prize Guide. https://arcprize.org/guide, 2025. Accessed: 2026-02-06.

ARC Prize Foundation. Hierarchical Reasoning Model Analysis. https://github.com/arcprize/hierarchical-reasoning-model-analysis, 2025. Apache-2.0 licensed GitHub repository for analyzing Hierarchical Reasoning Model methods; accessed 2026-02-06.

ARC Prize Team. The Hidden Drivers of HRM's Performance on ARC-AGI, aug 2025. URL https://arcprize.org/blog/hrm-analysis. Accessed: 2026-02-05.

Guillermo Barbadillo. 2nd Place Solution for the ARC Prize 2024 Competition: Omni-ARC approach. https://www.kaggle.com/competitions/arc-prize-2024/writeups/guillermo-barbadillo-2nd-place-solution-for-the-ar, 2024. Accessed: 2026-02-06.

François Chollet. On the Measure of Intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

François Chollet. ARC-AGI: The Abstraction and Reasoning Corpus for Artificial General Intelligence. https://github.com/fchollet/arc-agi, 2024a. Apache-2.0 licensed repository containing ARC-AGI-1 task data and a browser-based interface; accessed 2026-02-06.

François Chollet. OpenAI o3 Breakthrough High Score on ARC-AGI-Pub, dec 2024b. URL https://arcprize.org/blog/oai-o3-pub-breakthrough. Accessed: 2026-02-05.

Daniel Franzen, Jan Disselhoff, and David Hartmann. The ARChitects - ARC Prize 2025 Technical Report. https://lambdalabsml.github.io/ARC2025_Solution_by_the_ARChitects/, 2025. Accessed: 2026-02-08.

Alexia Jolicoeur-Martineau. Less is More: Recursive Reasoning with Tiny Networks. *arXiv preprint arXiv:2510.04871*, 2025.

Sapient Intelligence. Hierarchical Reasoning Model Official Release. https://github.com/sapientinc/HRM, 2025. Apache-2.0 licensed repository containing code for the Hierarchical Reasoning Model; accessed 2026-02-06.

Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and Yasin Abbasi Yadkori. Hierarchical Reasoning Model. *arXiv preprint arXiv:2506.21734*, 2025.

William Wu. 4th Place Solution for the ARC Prize 2024 Competition. https://www.kaggle.com/competitions/arc-prize-2024/writeups/william-wu-4th-place-solution, 2024. Accessed: 2026-02-06.