Test-time Adaptation of Tiny Recursive Models

Ronan McGovern
Trelis LTD
Trelis.com

Abstract

Prior to the close of the 2025 ARC Prize competition, the leading open source approach - known as TRM, for Tiny Recursive Models - involved training a 7M parameter recursive neural network on augmented variants of ARC tasks. That approach scored approximately 7.8% on the public ARC AGI II evaluation set, but required a level of compute far in excess of what is allowed during the competition. This paper shows that, by starting from a tiny recursive model, pre-trained on public ARC tasks, one may gain a large enough head start for fine-tuning on competition tasks to be effective in the 12 hours allows on four L4 GPUs during a competition submission. Specifically, a model is pretrained on public tasks for 700k+ optimizer steps over 48 hours on 4xH100 SXM GPUs and obtains a 10% score on the public evaluation set. That model is then post-trained in just 12,500 gradient steps during the competition and reaches a score of 6.67% on semi-private evaluation tasks. Notably, such post-training performance is achieved by full-fine tuning of the tiny model, not LoRA fine-tuning or fine-tuning of task embeddings alone.

1 Introduction

1.1 Description and Brief History of the ARC Prize

The ARC (Abstract Reasoning Challenge) Prize is a competition to efficiently solve unseen abstract tasks using computers. ARC tasks are designed to be solvable by humans but challenging for programable approaches. The competition imposes a limit on the amount of compute that may be used to solve tasks, which is why approaches must not just generalise to unseen tasks but also solve them efficiently.

Historically, the leading approach to solving the first series of ARC tasks (ARC AGI I) was a brute force search - with important heuristics - through a library of operations (e.g. flip, rotate, repattern) [reference ice-cuber]. This was outperformed by a number of entries in the 2024 competition, involving pre-training a deep transformer model in the single billion parameter range on ARC training tasks prior to the competition, followed by post-training on the test-time tasks within sandboxed competition environment with limited compute. This brought scores on ARC AGI I from roughly 20-30% up into the 40%+ range.

ARC AGI II brought a series of more difficult tasks where both brute force using a library of operations or using a single billion parameter transformer with pre- and post- training resulted in near zero performance. At the time of the release of this new test set, leading large language models far exceeding the compute limits of the competition, also struggled to score beyond a few percent on the updated benchmark - although, by late 2025, reasoning language models were scoring in the low double digits.

1.2 Tiny Recursive Models as a Basis for Solving ARC AGI II Tasks

Compared to the 1B to 7B parameter deep transformer models of 2024, which used both pre-training and competition-time fine-tuning, the recursive model approach that emerged with the HRM paper [reference] took a number of new directions:

1. A smaller model size, well less than 100M parameters
2. The use of recursive layers rather than a single once through deep network

3. A focus on training from scratch on competition tasks, rather than first pre-training on ARC-like tasks

This approach scored 5% on ARC AGI II tasks, which, while low in absolute terms, constituted the leading open source approach at the time, and, was perhaps surprising for a model of such small size.

This HRM (Hierarchical Reasoning Model [11], hierarchical because it involves a hierarchy of recursive loops) was soon improved on in the Tiny Recursive Models (TRM) [2] paper by reducing the architecture to a single neural net of 7M parameters, plus task embeddings. Although involving fewer model parameters, this updated approach improved in performance, reaching 7.8% on ARC AGI II tasks.

While, at the time the leading approach, this TRM approach required pre-training on four H100 GPUS for over 48 hours - well in excess of the compute allowed in the ARC Prize 2025 competition. This gave motivation to the question of whether and how the tiny recursive model approach could be improved in compute efficiency, and valuable for a competition entry.

1.3 Fitting a Tiny Recursive Model Approach within Compute Requirements for the ARC Prize 2025 Competition

The ARC Prize 2025 competition allows for the use of four L4 accelerators for twelve hours. TRM pre-training on 1,120 tasks for 100k epochs takes roughly 48 hours on 4xH100 SXM GPUs. Accounting for a factor of roughly 8x between bf16 flops on a H100 SXM versus an L4 accelerator, there is only about 1/8 x 1/4 = 1/32th of the compute necessary to complete full pre-training on this model size. Cast in different terms, pretraining with the TRM paper's approach takes approximately 750k optimizer steps with a global batch size of 768, while the competition runtime allows for only about 15k optimizer steps at a batch size of 392, given that one must also allow time for inference/evaluation.

Rather than train a model from scratch, an obvious approach is to instead fine-tune a model that has been pre-trained on public ARC tasks. There is no limit on the compute that can be used to pre-train such a model in advance of submission, although, the amount of publicly available ARC tasks, in particular tasks calibrated to ARC AGI II difficulty is very small. The question is, given competition tasks are new and unseen, and, given the model's capacity is small relative to pre-training and competition fine-tuning approaches of 2024, whether using a pre-trained TRM model can give a head start to fine-tuning at all?

1.4 Approach and Contribution

This paper demonstrates that starting from a pre-trained tiny recursive model significantly accelerates training on new, unseen tasks. The performance of such a fine-tuned model trends towards that of pre-training from scratch, although has not been found to quite reach or exceed the level of full pre-training performance - at least given the competition compute limits. Interestingly, the best performance was achieved by fully fine-tuning the pre-trained model, not by LoRA fine-tuning or by training the task id embeddings. Specifically, a pre-trained model achieving 10% on the ARC AGI II public evaluation split was capable of achieving 6.67% when fine-tuned on unseen semi-private tasks in the Kaggle competition environment.

Attempts were also made to improve the performance by using a stronger pre-trained model - trained on more and/or higher quality data for longer. These attempts were unsuccessful, although it remains possible - or even likely - that different choices of architecture, dataset or training hyperparameters could still lead to better results.

2 Methods

A recursive transformer model was pre-trained on ARC AGI II training tasks in close accordance to the Tiny Recursive Model paper [2]. During competition submissions, this pre-trained model was fully fine-tuned on the train example pairs of the test tasks. This fine-tuned model was used to predict test example outputs, using a majority voting method.

2.1 Pre-training

Three models were pre-trained. A first model model was trained almost exactly in line with the original

TRM paper. A second model was pretrained with an expanded pre-training dataset and for double the original number of epochs. A third model was then pre-trained with a smaller dataset, filtered for tasks matching ARC AGI II public evaluation split difficulty.

2.1.1 Original Paper Replication - 100k epochs

A TRM was trained in close alignment with the original paper, but with only 4 lower reasoning cycles instead of 6 reported in the paper. This deviation was unintentional and resulted from a commit in the TRM Github repository using that same value. Interestingly, other ablations by Xin Gao [1] and Konstantin Schuerholt also use this value of 4.

The data mix matched that of the original paper and included:

- ARC AGI II Training split [1000 tasks]: train + test example pairs
- Concept ARC split [160 tasks]: train + test example pairs
- ARC AGI II Evaluation split [120 tasks]: train example pairs only (test used for evaluation)

2.1.2 Extended Data for 200k Epochs

Following the heuristic that neural nets often improve in performance with longer pre-training, a second model was pre-trained for double the original number of epochs.

Following the heuristic of more higher in-distribution data helping the performance of neural nets, the dataset was expanded in two ways:

1. Inclusion of evaluation split test example pairs: The test example pairs from 100 of the 120 public ARC AGI II evaluation split were included in pre-training. Of the remaining 20 tasks, ten were used as an evaluation split, and ten were used as a post-training test split. Concretely, train example pairs from 110 tasks were included in pre-training and test example pairs from 100 tasks were included in pre-training. Test example pairs from 10 tasks were withheld for evaluation during pre-training, while train AND test example pairs from 10 tasks were withheld entirely for use in post-training.
2. Inclusion of 50 tasks from Simon Strandgaard's "tama" dataset [4]. These human-reviewed tasks cover concepts typical in ARC challenges and are of a difficulty somewhere between ARC AGI I and ARC AGI II. The hope for includiing this data was to broaden and enhance the pre-training dataset.

In sum, this meant pre-training on:

- ARC AGI II Training split [1000 tasks]: train + test example pairs
- Concept ARC split [160 tasks]: train + test example pairs
- ARC AGI II Evaluation split [110 tasks]: train example pairs from all 110 tasks and test example pairs from 100 tasks (the other ten serve as evaluation during pre-training)
- tama [50 tasks]: train + test example pairs

While there is the potential advantage of a higher quality, more in-distribution and larger pre-training dataset, there is a large trade-off in being able to assess performance with evaluation and test hold-out sets of only 10 tasks each, particularly when we are trying to assess performance with a granularity of one percent and where the anticipated score is in the region of 1-10 percent.

2.1.3 Filtered Hard Data for 500k Epochs

This third pre-trained variant aimed to test the heuristic that it can be better to train neural nets on a smaller amount of higher quality data for more epochs than on more mixed-quality data for fewer epochs.The same model and hyperparameters were used but training on 110 tasks from the ARC AGI II evaluation split and 120 hard tasks from the ARC AGI II training split. Training tasks were determined to be hard based on the ability of GPT-5-mini to write a python program that successfully solved all train and test example pairs. Specifically, any task for which GPT-5-mini could write a correct program, given approximately eight attempts, was filtered out. This left 137 remaining ARC AGI II train-

ing tasks, from which 120 were selected as a traininghard split. This filtering is somewhat arbitrary and skewed both because it filters based on a) python programming performance and b) LLM, not human, performance. The method was used because of prior unreported work attempting to solve ARC tasks by writing python programs. The choice of GPT-5-mini as a model was made because it was capable of solving only a few ARC AGI II evaluation tasks by writing python programs. As such, if a task can be solved through python program writing by GPT-5-mini this correlates with the task being too easy for ARC AGI II level tasks. And, the motivation for this "hard" dataset split was to have tasks more representative of the semi-private evaluation set on which performance is ultimately graded for the 2025 competition.

In sum, this meant pre-training on 230 tasks:

- ARC AGI II Training split, filtered with GPT-5-mini program writing for "hard" tasks [120 tasks]: train + test example pairs
- ARC AGI II Evaluation split [110 tasks]: train example pairs from all 110 tasks and test example pairs from 100 tasks (the other ten serve as evaluation during pre-training)

2.2 Post-training

Four approaches to post-training were conducted:

1. Full-fine tuning of a pretrained model.
2. Fine-tuning of embeddings only.
3. Full-fine tuning of a pretrained model, training only embeddings for the first quarter of optimizer steps.
4. LoRA fine-tuning of a pretrained model.

The fifth relevant approach – and null hypothesis – is to post-train on a randomly initialized model. For this baseline, we can look to results reported for the pre-training section above.

All post-training runs were conducted in the same manner and with the same hyperparameters as the pre-training runs, but with the following differences:

1. At the start of each pre-training run, the model was initialised from a pretrained model in the previous section.
2. Owing to lower total VRAM available on 4xL4s compared to 4xH100 SXMs, a global batch size of 392 was used instead of 768 for pre-training. Accordingly, the learning rate for the model trunk and for the embedding trunk were doubled (to 2e-4 and 2e-2, respectively).
3. Owing to the compute limitation at competition time, training was run for only 15,000 optimizer steps, rather than $\sim$750k in the original pre-training.
4. Post-training was conducted only on test tasks. No additional data was mixed in. For competition runs, this means running on semi-private test tasks, which can only be done by making a formal submission to the ARC Prize 2025 competition (noting that one submission is allowed daily).
5. TRM uses a majority voting method to make test output predictions, defaulting to a vote over 1,000 augmented versions of each task. Since compute time is limited during the competition, only 256 or 512 augmentations are used at evaluation time – although 1,000 are still used during pre- and post-training. The number of augmentations does not affect training time because epochs are defined as epochs over a given task and its variants (for a given task, a variant is sampled at random during each epoch).

2.3 Simulating and Measuring Post-training Performance

Simulating post-training performance is difficult, as there are only 120 public evaluation tasks for ARC AGI II, and - if one is to post-train on those tasks - those same tasks cannot be included in one's pre-trained model. Three workarounds were attempted, with varying degrees of success.

2.3.1 Post-training an ARC AGI I pre-trained model

To assess the effectiveness of post training one can first pre-train a model on ARC AGI I tasks, i.e. the train and test example pairs from the 400 task training split, and the test pairs from the 400 task evaluation split. Such a model was already available on

HuggingFace from Xin Gao [1] and so that model was used without re-training.

This pre-trained model - which has only six tasks that also appear in the ARC AGI II public dataset [see notes on this in Appendix ...] - can then be post-trained on the train example pairs of ARC AGI II training and then used to make predictions of the corresponding test example pairs. Note that, for post-training to be effective, it is not a necessary condition that this approach show positive results, because ARC AGI I tasks are significantly easier than ARC AGI II tasks. As such, a model pre-trained with ARC AGI I may be too weak to provide a head start to post-training on ARC AGI II level tasks. As it turns out, the ARC AGI I pre-trained model IS strong enough to meaningfully help with post-training for ARC AGI II grade tasks.

2.3.2 Post-training an ARC AGI II pre-trained model, but with 10 public eval tasks withheld as a test

As an attempt at simulating post-training performance on an ARC AGI II pre-trained model, one can run a pretraining (as is described for the second, and third, models in the pre-training section) on ARC AGI II data but withholding 10 evaluation tasks as a test set for post-training.

This was done, but the signal from only ten test tasks is weak, noisy and not informative.

2.3.3 Post-training an ARC AGI II pre-trained model via a formal ARC AGI 2025 Submission

Given a positive result from pre-training an ARC AGI I model on ARC AGI II data, one can then take an ARC AGI II pretrained model (pretrained on data including ARC AGI II public eval training tasks) and submit that for post-training on the semi-private tasks in the competition. Of course, one cannot take an ARC AGI II pretrained model and easily simulate post-training because the train example pairs from evaluation tasks are already within the pre-training data. The assumption is that an ARC AGI II pre-trained model (i.e. trained on ARC AGI II public eval data) will be more in-distribution for the semi-private eval tasks than a model pre-trained only on what are much easier ARC AGI I tasks.

For the purpose of the competition, there were three pre-trained models submitted (the same three listed in the pre-training section): 1. The ARC AGI II pre-trained model. Here, post-training could not be simulated as it had been pre-trained on ARC AGI II evaluation train examples 2. An ARC AGI II pre-trained model (for 2x epochs, and with expanded data) - from which 10 tasks were withheld during pretraining. Here, post-training was simulated, but was too noisy to be informative given only 10 withheld tasks. 3. An ARC AGI II pre-trained model (for 500x epochs, and with a dataset filtered to include only 230 ARC AGI II difficulty tasks) - from which 10 tasks were withheld during pretraining. Again, post-training was simulated but too noisy to be informative.

However, since all three pre-trained model were submitted to the competition for pre-trainning, semi-private results ARE reported.

2.4 Post-training Methods

Four post-training methods are considered and evaluated on an ARC AGI I pre-trained model using ARC AGI II tasks. Based on those results, the chosen approach for competition submissions is to fine-tune embeddings for one quarter of post-training epochs and then fully fine-tuning for the remaining epochs.

2.4.1 Full fine-tuning

Full fine-tuning involved updating all model parameters, as in pre-training. The only differences were those listed above.

2.4.2 Fine-tuning of embeddings only.

In TRM, each task and any augmented variant of that task is assigned a task id and has it's own trainable embedding. At the start of post-training, the model sees a new set of tasks and variants, and so a new set of task ids and set of embeddings are initialised. These embeddings start the post-training process untrained, while the rest of the model parameters (attention, MLPs and heads) start in a pre-trained state.

As such, at the start of post-training, the task embeddings - which might be thought of as a description

of the transformation involved in a given task variant - are out of sync with the model's trunk. With this as motiviation, it appears reasonable to train only the embeddings to see whether they might be adapted by post-training in a manner that allows the model trunk to "execute" a program described by a newly trained task id. Were the training of embeddings sufficient, this might suggest that the trunk has the required tools to solve ARC tasks and the embedding need only be trained to signal to the trunk what tools should be used for the task at hand.

2.4.3 Fine-tuning of embeddings followed by full fine-tuning

As it turns out, post-training only of the embeddings results in a score of near zero on held out tasks. This motivates the idea of first training only the embeddings for some epochs (one quarter of total post-training epochs) followed by full fine-tuning for the remainder of epochs to allow for full fine-tuning.

2.4.4 LoRA fine-tuning of the trunk plus embeddings fine-tuning

Rather than train all parameters in the trunk (everything but the embeddings) one might instead train a set of low rank adapters for linear layers in the trunk. This reduces the number of parameters that are trainable. The motivation is potentially to avoid overfitting by training all parameters. However, TRMs are already low in parameter count, and the use of LoRA may reduce the ability of the model to adapt to the newly presented tasks. This appears to be the case.

3 Results

3.1 Pre-Training

All results for the TRM Paper ARC AGI II Replication Run are available at `https://wandb.ai/trelis/Arc2concept-aug-1000-ACT-torch`, see `pretrain_att_arc2concept_4`, and the associated pre-trained model checkpoint is released at [7].

Results for the extended training runs are available at `https://wandb.ai/trelis/Arc2-pretrain-final-ACT-torch`, and the released checkpoints are [8] and [9].

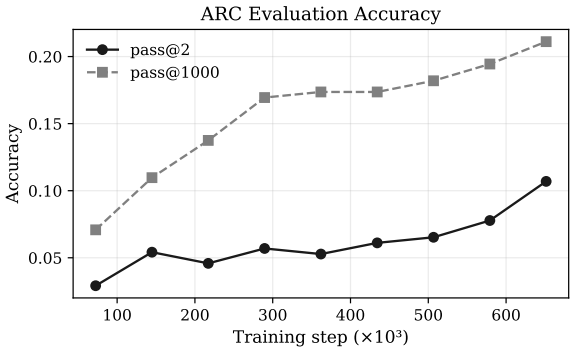

Figure 1: ARC evaluation pass@2 and pass@1000 accuracies throughout training for the `pretrain_att_arc2concept_4` replication run.

Select results are shown here from the replication rather than the extended runs, because the evaluation set of ten tasks in the extended runs is too small to provide statistically meaningful insights on performance.

Figure 1 shows the evolution of pass@2 and pass@1000 accuracy on the 120 ARC AGI II public eval tasks during the training run, reaching a final pass@2 score of 10%, which is higher than the originally reported score of 7.8% in the TRM paper, presumably associated with stochastic behaviour during training. Figure 2 shows the per-example-pair exact accuracy (i.e. percentage of training pairs the model gets correct within a training batch of 768 example pairs) over the course of training. It is clear that the model heavily overfits the data, although continued training gradually brings up the exact accuracy on the evaluation set, although still far below 100% by the end of training.

3.2 Post-Training

3.2.1 Post-training of the ARC AGI I Pre-trained Model

Post-training sweeps were conducted on the ARC AGI I checkpoint released by Xin Gao [1]. All experiments were logged in the `Arc-eval2-aug-1000-ACT-torch` Weights & Biases project [5]. Figure 3 compares eval-

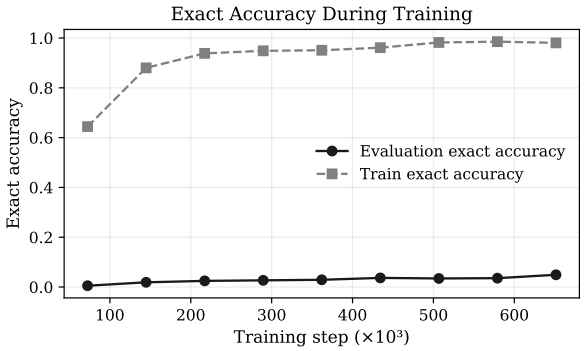

Figure 2: Evaluation and train exact accuracies throughout training for the `pretrain_att_arc2concept_4` replication run.

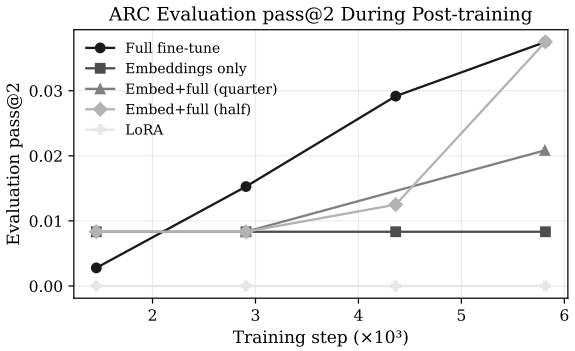

Figure 3: ARC evaluation pass@2 for the ARC AGI I post-training sweeps logged in [5]. Each curve shows the first 6k optimisation steps for a different adaptation strategy.

uation pass@2 during the first 6k optimisation steps for five adaptation strategies: full fine-tuning (`posttrain_aa1_aa2e`), embeddings-only (`posttrain_aa1_aa2e_fe`), embeddings-only followed by full fine-tuning after a quarter (`posttrain_aa1_aa2e_feq`) or half (`posttrain_aa1_aa2e_feh`) of the steps, and LoRA (`posttrain_aa1_aa2e_lora`).

Note that while only 6k optimizer steps were used for the above sweeps, competition submissions used either 12,000 or 15,000 optimizer steps.

Table 1: Semi-private evaluation accuracy achieved after post-training ARC AGI II checkpoints within the competition environment.

Pre-trained model	Accuracy (%)
TRM paper replication	6.67
Expanded data, 200k epochs	4.25
Filtered hard data, 1M epochs	1.27

3.2.2 Post-training of ARC AGI II pre-trained models during competition submissions

Simulating ARC AGI II checkpoints provides limited insight: the public evaluation set is already included in pre-training and the ten-task hold-out used for diagnostics offers too small a sample to estimate performance reliably. For completeness these diagnostics, recorded at [6], achieve zero pass@2 after post-training across all models.

Three pre-trained models were nevertheless submitted to the ARC Prize 2025 runtime for competition post-training. All submissions reused the configuration in Section 2, with the baseline TRM replication fine-tuned for 12.5k steps and the extended variants for 15k steps to stay within the twelve-hour budget. Table 1 summarises the resulting semi-private evaluation scores. Re-submitting the same model often yielded scores ranging from 3.33 to 6.67 depending on minor procedural tweaks (for example, adjusting the duration of frozen trunk updates or adding brief continued pre-training), underscoring the high variance inherent in the evaluation process.

4 Discussion

4.1 Pre-training from Scratch Does not Fit within Competition Compute Limits

Figure 1 shows the pass@2 and pass@10 performance of a pre-trained - from scratch - tiny recursive model evaluated on the ARC AGI II public evaluation set of 120 tasks. For the purpose of the ARC Prize 2025 competition, it is the pass@2 score that counts. Entrants are allowed to submit two output grid predictions per test example, and the best output grid is the one that counts.

If one were to naively approach the pre-training-

from-scratch approach of Fig. 1 in the competition, there would only be sufficient compute to conduct about 10-20k optimizer steps. This corresponds to the very left of the plot, where the pass@2 score nears zero. Granted, the compute required scales with epochs over the data and the number of tasks in the dataset. While the TRM paper trains on 1,180 tasks, one might consider pre-training - at competition time - on only the 240 competition tasks. This would reduce compute requirements by about 5x, although there are no guarantees of similar performance if the training data is reduced in diversity. Moreover, even a 5x reduction in the necessary optimizer steps per epoch with a smaller number of tasks - assuming one holds epochs constant - would result in needing 200k+ optimizer steps, still well beyond the 10-20k possible in the limited compute environment.

As such, the plot of pre-training performance in Fig. 1 tells us that pre-training from scratch requires too much compute for the competition environment, if one is to follow the recipe of the TRM paper. Of course, one could alternatively try to find a different model, dataset or pre-training recipe that performs as well and works with much less compute. Some effort was spent on this, and is described in 5, but it was generally hard to match or exceed performance of the original TRM design, not to mention doing so with less compute.

4.2 Fine-tuning a Pre-trained Model Significantly Accelerates Adaptation

Figure 3 is the fine-tuning analog of Fig. 1 for pre-training. Notice how the number of optimizer steps to achieve an improvement in performance is orders of magnitude lower when starting from a pre-trained model, than when pre-training from scratch.

The highest gains in accuracy are achieved by full fine-tuning OR by training only embeddings for a first portion of epochs (either half or one quarter), followed by full-finetunig for the rest. Fine-tuning embeddings only OR LoRA fine-tuning (which additionally includes fine-tuning of embeddings) lead to low accuracy. Note that there is a signicant level of noise involved in these results. For example, LoRA+embeddings fine-tuning looks inferior to fine-tuning embeddings only in the pass@2 metrics. However, looking at a broader set of results - available in Weights and Biases - reveals that LoRA+embeddings slightly outperforms embeddings-only when looking at pass@1000, a somewhat less noisy indicator of performance.

Clearly - and even with independent ARC tasks - it is possible to pre-shape the neural network in a manner that accelerates tuning on unseen tasks. This raises the question of what types of tasks are best used in pretraining to optimally shape the network for adaptation at competition time.

4.3 It remains unclear what diversity of pre-training tasks best shapes a network for post-training on unseen tasks

One heuristic in machine learning is that it can be better to train for longer on higher quality data than to expend that same amount of compute training for fewer epochs on a broader dataset of mixed quality.

Given ARC AGI II tasks are significant harder than ARC AGI I tasks, and given ARC AGI II's public evaluation set is closely calibrated in difficulty to the semi-private and private evaluation sets, it was worth pre-training only on ARC AGI II difficulty tasks, rather than the larger dataset of 1,000 ARC AGI II training tasks, most of which are much easier.

In practise, training for longer on this smaller but harder dataset (Filtered hard data, 1M epochs in 1) resulted in worse performance than pretraining on the larger training dataset ("TRM paper replication"). It is hard to know exactly why this is the case, and noise cannot be ruled out as a cause, but it is suggestive that the ARC AGI II training set and/or the Concept ARC split of 160 tasks bring useful diversity or relevant concepts to the pre-training.

The other pre-training approach shown in 1, "Expanded data, 200k" involved training for longer than the original TRM paper, including test examples from ARC AGI II public evaluation tasks, and including additional concepts from the tama dataset [4], all in the hope that added data diversity and training time might help with performance. Here, the difference in performance with the original TRM baseline (4.25% vs 6.67%) is likely within noise. Perhaps it is not surprising there is little improvement due to the expanded dataset because the dataset is only a

little larger than the original. "tama" concepts may overlap with concepts already present in ARC AGI II training data, and the inclusion of test examples from ARC AGI II evaluation data does not add new tasks, but only adds more examples per task. It is not obvious that training for longer would or would not have led to an improvement, but here it did not.

4.4 On the continued pre-training of pre-trained models

The trend of pass@2 and pass@10 scores in Fig. 1 suggests that simply training for longer could lead to improved results. There is an important nuance with TRM's design, whereby task embeddings are trained for an index of tasks. If one does not save that mapping of tasks to embeddings, then one has no choice but to reinitialise and re-train the embeddings. Unfortunately, for the replication run (although not for the expanded data or filtered hard data pretraining runs) the embedding to task mappings were not saved (which involves saving the task and task augmentation dataset), and strict continued pre-training was not possible.

Nonetheless, continued pre-training was tried for a further 72k optimizer steps (roughly 72% more steps) - with reinitialisation of embeddings - but this led to lower submission performance of 3.33% compared to 6.67%. As such, reinitialisation of embeddings may lead to partial model collapse and/or overfitting, perhaps as the embeddings learn more grid specifics rather than the general transformation.

4.5 On the use of augmentation-specific task embeddings

Both HRM and TRM assign a unique embedding not just to each task, but to every augmented variant (flips, rotations, re-colours) of each task. From the model's perspective, each task variant is an entirely separate task. The model may or may not learn that these tasks are related.

It is interesting to ask whether:

1. the model does learn that variants of the same task are related,

2. it would be more efficient to use the same task_id/embedding for all variants of a given task OR whether treating variants as independent tasks provides the model with some useful form of regularisation/generality during pre-training.

4.5.1 Using cosine similarity to measure the model's ability to encode related tasks

As a means of partially addressing (1), one might look at the cosine of the angle between task embeddings for i) variants of the same task and ii) between the base/original/unvaried form of different tasks. If the TRM does indeed learn to encode task variants similarly, perhaps one should expect a rise in the cosine between embeddings during pre- and post-training. With this as motivation, these measures were recorded during the pretraining of the TRM for 200k epochs on the extended dataset (extended to include 'tama' and ARC EVAL II evaluation test examples, see 2).

Figure 4 shows the evolution of the cosine of the angle between task embeddings of variants of the same task (averaged across tasks in a given batch), for training and evaluation sets. Note that the angle between task id embeddings is reported as zero at most steps because there is at most one variant of each task per batch of 768 example pairs. If one looks in greater granularity at the data, there are timesteps reaching cosine similarity of up to about 0.1 . This is more apparent in the training of the third model on the subset of hard tasks (not shown in these plots), since there are only 230 base task ids and every batch therefore includes more than one variant for each task. Nonetheless, the cosine similarity of task id embeddings for variants of the same task is low when measured on the training set. The cosine similarity is higher, and rising throughout training, when measured on the evaluation set. It is not obvious why training and evaluation sets diverge here and why the model might adapt to develop more similarity on evaluation example pairs.

Figure 5 illustrates the evolution - over the course of training - of the cosine of the angle between the base variants of different tasks within the same training batch. Although there is some gap between the cosine on training and evaluation examples, both rise together, although the measurement seems to asymptote more on the training examples. Interestingly,

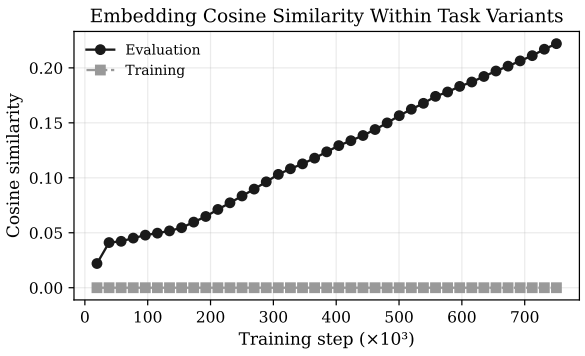

Figure 4: Embedding cosine similarity among augmented variants of the same task during extended pre-training.

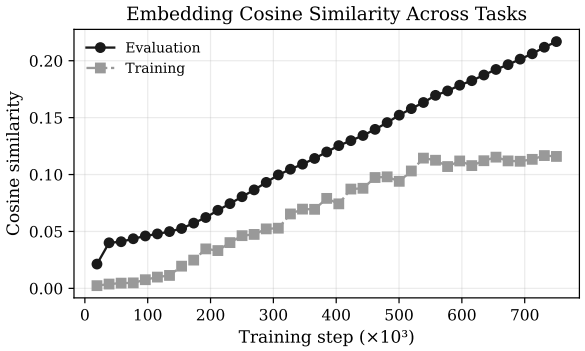

Figure 5: Embedding cosine similarity across base tasks for the extended pre-training runs.

comparing 5 and 4, task id embeddings appear similarly distant within tasks as between tasks, suggesting the relationship between variants of the same task are not clearly expressed - at least within the embeddings alone.

4.5.2 Pre-training a model with explicitly encoded embeddings

Rather than assign an independent task id embedding to each augmented version of a task, one might instead assign the same task id to all variants of a given task, but then encode the augmentation type.

For example, one might simply encode flips and rotatations by enumerating and embedding the 8 dihedral (d4 group) variants. To encode re-colours, one might build a look-up table covering the colour mapping pairs and train embeddings for that look-up table. In principle, this level of entropy can be captured in a single low-dimension embedding, perhaps no larger than 256, or even 128 dimensions.

This approach has the benefit of greatly reducing the number of model parameters by avoiding the need for a 512-dimension embedding for each task. While the TRM paper model has 7M parameters in its trunk, there are 2.5 GB of parameters required to capture 1,000 augmentations of 1,000 tasks (1k tasks x 1k augs/task x 512 dimensions = 512 M parameters). As such, when encoding each task augmentation individually, the TRM is more a 500M+ parameter model than a 7M parameter model. With explicit augmentation encodings, the size of the embeddings drops to 1k tasks x 512 dimensions = 512k parameters.

One might further expect that explicitly encoding augmentations makes it easier for the model to learn, and potentially require less compute to reach convergence. However, this proved not to be the case, and the performance of models trained with explicit encodings for embeddings (detailed in the slim-in and base-in branches of the Trelis fork of TRM [10]) was inferior to augmentation specific task embeddings (see Weights and Biases Project: https://wandb.ai/trelis/Arc2ethard-aug-1000-ACT-torch).

Perhaps the use of augmentation-specific task embeddings forces the model to generalise in a useful manner, but, then why isn't there stronger evidence of this in rising cosine similarity between in-task embeddings during pre-training?

4.6 Post-training Tiny Recursive Models as a Variant of Searching Latent Program Space (SLPS)

In their paper, Searching Latent Program Spaces, McFarlane and Bonnet [3] describe an approach not dissimilar to that involved in post-training a tiny recursive model. They cast the problem of solving ARC tasks as a search for a vector embedding that describes the transformation involved in a an ARC task, that program (or instruction) then being fed to a neural net and executed on a grid input.

All of the steps of post-training a TRM are there in Searching Latent Program Space:

1. There is the pretraining of a neural net on ARC tasks, to establish a landscape for that category of problems.

2. There is the post-training/fine-tuning on train examples from the test tasks, involving the search for the best latent (embedding) to describe the transformation at hand.

Perhaps the performance gap between SLPS and TRM with post-training can be attributed to: 1. The more complete set of augmentations used by TRM (dihedral group variants, recolours and translational augmentations), 2. The fact that SLPS searches only for the best latent (i.e. trains only embeddings in TRM, but not other parameters). As such, SLPS finds the best combination of pre-trained primitives, but cannot add/encode new primitives. 3. The recursive nature of the TRM's neural net, which allows for greatly increased effective depth while maintaining stability during training.

4.7 On the distribution of ARC AGI II tasks

A major challenge in ARC AGI II is that there is little public data that is clearly in the distribution of the eventual ARC AGI II semi-private dataset. It is known that any approach scores remarkably similarly on the ARC AGI II public evaluation set and on the ARC AGI II semi-private (and likely private?) dataset. This means that those datasets are closely in-distribution. By contrast, the ARC AGI II training dataset (and the hard split) appears not to be in distribution as scores do not correlate closely with ARC AGI II evaluation sets.

For the purpose of research, it would have been highly useful to have an ARC AGI II training split of 120 tasks in the same distribution as the public eval and semi-private eval set. This would have allowed for pre-training on such a set, followed by post-training on the public eval set to accurately assess performance.

The closest approximation of this would perhaps have been to pre-train on 60 of the 120 public eval tasks, and post-train on the other 60. However, this has two drawbacks:

- Statistical power, already small at just 120 tasks, is even smaller when one takes a subsplit.
- Model performance is sensitive to the amount of data that must be encoded. For the same model size, one cannot directly compare the performance pre or post training on 120 tasks versus 60 tasks.

5 Other Ablations

Other ablations are reported here, rather informally. Where avaiable, links to Weights and Biases reports are provided.

5.1 On the method of embedding initialisation in post-training

When faced with unseen tasks, the TRM code base by default initialises new embeddings to the mean of pre-trained embeddings.

An ablation was run to instead initialise embeddings to a Gaussian norm with the mean and variance of the pretrained embeddings. The hope was that by adding nonise to the initialisation, the model may overfit less during post-training. Howeever, using a Gaussian norm did not improve post-training performance over initialising all task id embeddings to the mean of those seen in pre-training.

5.2 On the variation of batch size during post training

Another heuristic in machine larning is that reducing batch size can sometimes introduce favourable noise and regularization that improves test performance.

While the pre-training TRM replication was run with the same batch size (768) as the original paper and on 4xH100 SXM, the other two pre-training runs reported were run at at global batch size of 1536 and with the learning rate doubled in order to make full use of 8xH100 compute cores, and cut training time in half. It is possible this had an adverse effect on pass@2 performance.

For post-training, ablations were run at smaller batch sizes of 96 and 32 (comapred to 384) in the hope of improving regularisation. However, performance was not improved, and, post-training takes longer per epoch as CUDA core utilisation is degraded. A selection of post-training results are

visible here: https://wandb.ai/trelis/Arc-eval2-aug-1000-ACT-torch .

5.3 On improvements to majority voting

The TRM code base uses a simple form of majority voting among augmentations for a given task in order to rank top predictions for a task example. TRM does include a halting head designed to indicate whether a task has been solved (which primarily serves to stop recursions early during training). However, for ARC AGI II tasks, although the halting head does learn to detected solved train examples, it does not learn to effectively detect solved evaluation test examples. As such, unlike in Sudoku tasks, the halting head cannot (yet) be effectively used to improve voting among predictions from augmentations.

5.4 On the benefit of joint training on multiiple tasks for compute efficiency

If post training requires fewer optimizer steps than training from scratch – to reach the same performance – then clearly there is meaningful inter-task learning OR at least there are shared concepts involved in training TRM.

The effect can also be understood by pretraining on a single task versus 8 versus 120 tasks on a model of the same size. While it appears possible to achieve similar performance when training on a task individually, or combined with other tasks, it is more efficient – again for a model of fixed size – to jointly train on multiple tasks. Said differently, one can reach the same performance with less compute if one jointly trains on multiple tasks. As such, there appears to be joint concepts to be learned. It is possible of course that much of this joint learning involves primitive concepts relating to grid sizes and colours, as opposed to transformations themselves.

6 On the use of smaller or larger model parameter counts

In the hope of reducing pre-training requirements, ablations were run where the characteristic dimension of the model, and of task id embeddings, was reduced from 512 down to 256 - resulting in a model of roughly one quarter of the original parameter count. The code is available in the 'slim' branch of the Trelis TRM fork [10], and results are shown in https://wandb.ai/trelis/Arc2ethard-aug-1000-ACT-torch . Smaller models were capable of scoring in the low single digits on ARC AGI II evaluation tasks, when comapred on an iso-compute basis. However, there was not enough clear advantage to justify pursuing the direction further.

One ablation with a larger (1024 characteristic dimension) model was also run and is reported here: https://wandb.ai/trelis/Arc2concept-aug-1000-ACT-torch . It appeared to trend similarly in pass@2 evaluation score to the base model on an iso-compute basis, but a longer and more thorough run is required to make firm conclusions.

The limited ablations run leave unanswered the question of what the optimal model dimension should be, and what the optimal ratio of task embedding dimension relative to model trunk dimenstion (default ratio of 1) should be.

Intutively, the size of the task embeddings relative to the model trunk dimension perhaps should be reflective of the relative entropy within task transformations versus that required to "execute" such tasks in the model trunk. Certainly the TRM design is very heavily weighted towards storing information in the task embeddings because there is a 512 dimension embedding not just for every task but also for every augmentation. For 1k tasks one has 500M+ parameters for embeddings and just 7M for the trunk.

7 Conclusion and future work

Full fine-tuning allowed a pre-trained tiny recursive transformer model to be efficiently adapted in the compute-limited environment of the ARC AGI II competition. Effective adaptation appears to require updating both task id embeddings AND the trunk of the model at competition time.

Currently, the use of tiny recursive models has been limited to single or low double digit scores on ARC AGI II tasks. While post-training of a pre-trained TRM is much more compute efficient than pre-training from scratch, it has not been shown that post-training can exceed or reach the performance achieved in pre-training. That pass@1000 metrics

reach well above pass@2 metrics, often above 20% on ARC AGI II difficulty tasks, suggests that performance improvements are possible. It remains an open question how far performance could be pushed through model size or hyper parameter improvements with or without further data additions or augmentations.

Future work may consider:

1. Closer visual inspection of the tasks solved versus not solved by TRM-type approaches. Is it conceivable that those tasks are solved owing to the types of augmentations (rotate, flip, translate, re-colour) that are applied? If so, are there other augmentations (e.g. shearing, repatterning) that might assist in solving more tasks?

2. What size model (specifically, size of hidden dimension) is most compute efficient for pretraining on a fixed number of tasks? Chinchilla laws for LLMs suggest that larger models may be more compute efficient (on a training compute basis) than smaller models. Such ablations were tried but, given the compute budget, it was not possible to reach a firm conclusion on the best model size.

3. For a constant size model hidden dimension, does increasing or decreasing the embedding dimension improve performance. Currently, it appears that the embedding dimension is not capturing similarities between task variants. Does this suggest the task id embedding is too large or too small?

4. Improved optimisation of hyperparameters, especially the number of higher and lower loops that control effective model depth. Does increasing the level of recursion increase or decrease compute efficiency when holding the effective depth constant?

5. Currently, at test time, the halting signal is ineffective, i.e. while the model learns to halt at early iterations on training data, it does not effectively halt on evaluation data. Are there ways to improve the halting behaviour OR are the tasks fundamentally too hard, perhaps suggesting that an even larger effective depth is required for training?

8 Acknowledgements

This work was supported by Runpod, which provided $15k of compute, and by Lambda Labs, which provided $1k of compute. Thanks to Lewis Hemens for months of collaboration on ARC Prize research and support for compute costs. Thanks to Jack Boylan for assistance in running the pre-training replication.

References

[1] Xin Gao. Tiny recursive models – arc agi 1. https://huggingface.co/Sanjin2024/TinyRecursiveModels-ARC-AGI-1, 2025. Accessed: 2025-11-03.

[2] Alex Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. 2025. doi: 10.48550/arXiv.2510.04871. URL https://doi.org/10.48550/arXiv.2510.04871.

[3] Matthew V. Macfarlane and Clément Bonnet. Searching latent program spaces. 2024. doi: 10.48550/arXiv.2411.08706. URL https://arxiv.org/abs/2411.08706. Accepted as Spotlight at NeurIPS 2025.

[4] Simon Strandgaard. Arc dataset collection: Tama split. https://github.com/neoneye/arc-dataset-collection/tree/main/dataset/arc-dataset-tama, 2024. Accessed: 2025-11-03.

[5] Trelis. Arc prize post-training sweeps (arc-eval2-aug-1000-act-torch). https://wandb.ai/trelis/Arc-eval2-aug-1000-ACT-torch, 2025. Accessed: 2025-11-03.

[6] Trelis. Arc prize post-training hold-out diagnostics (arc-evaluation2test-aug-1000-act-torch). https://wandb.ai/trelis/

Arc-evaluation2test-aug-1000-ACT-torch, 2025. Accessed: 2025-11-03.

[7] Trelis. Tiny recursive models – arc agi ii. https://huggingface.co/Trelis/TRM-ARC-AGI-II, 2025. Accessed: 2025-11-03.

[8] Trelis. Tiny recursive models – arc agi ii (all-200k). https://huggingface.co/Trelis/TRM-ARC-AGI-II-all-200k, 2025. Accessed: 2025-11-03.

[9] Trelis. Tiny recursive models – arc agi ii (hard-1m). https://huggingface.co/Trelis/TRM-ARC-AGI-II-hard-1M, 2025. Accessed: 2025-11-03.

[10] Trelis Research. Tinyrecursivemodels (trelis fork). https://github.com/TrelisResearch/TinyRecursiveModels, 2025. Accessed: 2025-11-03.

[11] Guanzhi Wang, Jianing Li, Yuxiao Sun, Xi Chen, Chuhan Liu, Yuan Wu, Mingyu Lu, Shuran Song, and Yasin Abbasi-Yadkori. Hierarchical reasoning model. 2025. doi: 10.48550/arXiv.2506.21734. URL https://doi.org/10.48550/arXiv.2506.21734.

Table 2: Raw dataset splits used across experiments.

Challenges file	Puzzles	Avg. train inputs	Avg. test inputs
arc-agi_concept_challenges.json	160	2.67	3.00
arc-agi_training2_challenges.json	1000	3.23	1.08
arc-agi_evaluation2_challenges.json	120	2.98	1.43
arc-agi_tama_challenges.json	50	3.18	1.52
arc-agi_test_challenges.json	240	3.20	1.08

Table 3: Derived splits constructed for extended pre-training and evaluation.

Challenges file	Puzzles	Avg. train inputs	Avg. test inputs
arc-agi_evaluation2train_challenges.json	100	2.96	1.44
arc-agi_evaluation2eval_challenges.json	10	2.90	1.60
arc-agi_evaluation2test_challenges.json	10	3.30	1.20
arc-agi_traininghard_challenges.json	120	2.98	1.09
arc-agi_evaluation2clean_challenges.json	114	2.97	1.46

A ARC Task Example Data Splits

Notes on derived splits. The evaluation2train, evaluation2eval, and evaluation2test files are all sampled from the ARC AGI II evaluation split. These subsets supply the pre-training and post-training tasks for the second model configuration.

Side notes.

- Six tasks in the ARC AGI II evaluation split also appear in ARC AGI I. When adapting models pre-trained on ARC AGI I, the arc-agi_evaluation2clean_challenges.json split filters these duplicates to avoid contamination.
- The placeholder arc-agi_test_challenges.json split contains fewer test examples than the evaluation set. Inference on this set is roughly 33% faster than the final competition rerun and can under-estimate runtime, risking notebook timeouts during submission.