# Test-time Adaptation of Tiny Recursive Models

Ronan McGovern
Trelis LTD
Trelis.com

## 1 Introduction

### 1.1 compute requirements

The ARC Prize 2025 competition allows for the use of four L4 accelerators for twelve hours. As a reference point, 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, such pre-training 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.

The premise of Test-time adaptation is that, by conducting pre-training prior to competition submissions, one can achieve better performance with fine-tuning than one could conducting pre-training from scratch.

## 2 Methods

A recursive transformer model was pre-trained on ARC AGI II training tasks in close accordance to the Tiny Recursive Model paper [**?** ]. 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 (reference here) 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 [**?** ]. 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 training 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 [**?** ] 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 and then full-finetuning

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.