# Basics of Deep Learning: Term Project Report

**Jeongmin Park** [*1,2] **Eun Kho** [*1,2] **Woosung Kim** [*1,2] **Jaewan Park** [*1,2]

## Abstract

ARC-AGI is a challenging benchmark designed to measure AI systems' reasoning and abstraction abilities. In this project, we propose a comprehensive solution based on full fine-tuning of Qwen3-0.6B, selected through various comparative experiments. To enhance robustness and generalization, we implement a multi-stage training pipeline. Also, we introduce several inference techniques such as rank-based task filtering, task-specific test-time training, and confidence-based best-of-N sampling. Experimental results show that our methods significantly improve accuracy.

## 1. Introduction

ARC-AGI (Chollet, 2019) is a benchmark designed to evaluate an AI system's ability to perform general reasoning and abstraction. The dataset consists of independent, puzzle-like tasks, each containing multiple input-output grid pairs.

Diverse approaches have been proposed to solve this challenging benchmark. While early approaches and several leading solutions from the ARC Prize 2024 competition relied on transductive models (Chollet et al., 2025), recent frontier LLMs have demonstrated great performance using long chain-of-thought (CoT) reasoning (Chollet, 2024).

Due to constraints in time and computational resources, we adopted the former approach, training a transductive LLM to solve the tasks. We explored various base models and training methods, as well as several inference strategies to enhance the model's answer accuracy. Our pipeline achieved a 73% score on the leaderboard.

Our contributions can be summarized as follows:

- We present a two-stage training pipeline designed to produce a more generalized and robust model.
- We propose a multi-step inference procedure which ensembles rank-based filtering, test-time training (TTT), confidence-based best-of-N (BoN) sampling.

[*]Equal contribution [1]Team 15 [2]Department of Computer Science and Engineering, Seoul National University.



## 2. Pipeline Overview

We developed a complete pipeline for solving the ARC-AGI benchmark using a transductive LLM, i.e., the model directly predicts the answer to the question. The overall architecture of our pipeline is illustrated in Figure 1.

**Training Pipeline** Our training process consists of two stages. In the first stage, we perform exhaustive full fine-tuning of the base model using the entire provided dataset for three epochs. In the second stage, we resume fine-tuning with an expanded dataset that includes augmented versions of the original data. We found that data augmentation improved the model's generalization and robustness by increasing data diversity. Further analysis of the training procedure is covered in Section 3 and Section 5.

**Inference Pipeline** To maximize the accuracy of the model's outputs, we propose a three-step inference pipeline. First, the model assesses the relative difficulty of a given test task using a heuristic ranking method (Rank-Based Filtering). This way, our model can determine whether further adaptation is needed. Second, for tasks identified as difficult, we apply task-specific fine-tuning by attaching and training a new LoRA adapter to the model (Test-Time Training). This enables the model to better align with the structure and distribution of the specific task. Finally, we generate multiple candidate answers through stochastic decoding and compute a confidence score for each one (Confidence-Based BoN Sampling). The candidate with the highest confidence is selected as the final answer. More details are provided in Section 4 and Section 5.

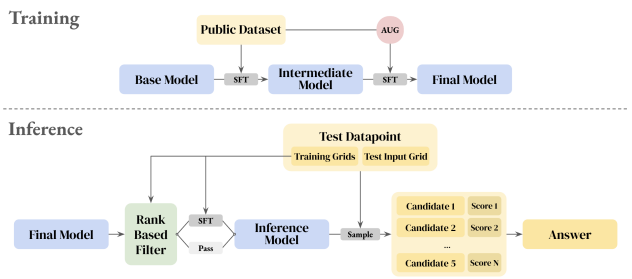


*Figure 1.* Overall Pipeline Architecture

## 3. Training

Given a dataset of 300 tasks with each containing 1000 examples, our training strategy followed a two-stage approach as introduced in Section 2.

**Prompt** Each input prompt consists of a system prompt and a user prompt. The system prompt instructs the model that it will be solving an abstract reasoning task. The user prompt contains a few-shot example format, presenting three input-output pairs followed by a test input for which the model is expected to generate the correct output. The entire prompt template is attached in Appendix A.

**Model Choices** Before executing the full pipeline, we conducted preliminary experiments to identify the most suitable training configuration. This involved selecting the base model family, model size, and fine-tuning strategy. Based on the results, we selected Qwen3-0.6B (Qwen Team, 2025) with full fine-tuning, which updates all model parameters during training. Further analysis of these design choices is provided in Section 5.

**Training** We performed full fine-tuning on all 300 tasks for 5 epochs. For each task, the 1000 examples were divided into 250 sets of four examples each. From each set of four examples, we constructed four training instances by rotating the role of the test input—each example in the set was used once as the test input, while the remaining three served as example input-output pairs. This resulted in 1000 training samples per task. The first three epochs were conducted exclusively on these data, while the final two epochs were done together with augmented data. The complete set of hyperparameters used is provided in Appendix B.

**Data Augmentation** To increase data diversity during full-finetuning, we applied data augmentation to random subset of tasks. For selected tasks, random transformations were introduced to both the input and output grids. The following types of augmentations were used:

- **Flip and rotation**: Inputs and outputs were randomly rotated by 90°, 180°, or 270° clockwise, or flipped horizontally or vertically.
- **Shuffle numbers**: Digits from 0 to 9 were randomly permuted, altering their symbolic representation while preserving the underlying task logic.

Examples of these augmentations are shown in Figure 2.

**Unsuccessful Attempts** We also explored some ideas that was unsuccessful but somewhat interesting. One was adding dedicated tokens to the tokenizer for each pixel color. The motivation was that the default tokens 0–9 may carry implicit numerical meaning, which could interfere with the model's understanding of color as a purely symbolic feature. By introducing new tokens specifically representing tile colors, we aimed to eliminate this ambiguity.

*Figure 2.* Data augmentation by flip and rotation / shuffle numbers

Another attempt was problem augmentation. The intuition behind this approach was to expose the model to tasks derived from the original dataset but framed in a different format. For example, we designed inputs that presented four input-output pairs and asked the model to identify the one that originated from a different task.

Both modifications did not lead to any measurable performance improvement. However, we still find these approaches promising and believe it may yield benefits when applied at larger scales.

## 4. Inference

At inference time, few-shot examples wrapped with the same prompt format used in training were given to the model. However, rather than greedily accepting the model's output, we utilized Test-Time Training (TTT) and Best-of-N (BoN) sampling, guided with few filtering methods, to increase the model's generation accuracy.

**Why Inference-Time Train?** ARC tasks exhibit two key characteristics:

- **Distribution shift**: Each task follows a unique rule unseen during training.
- **Few-shot scarcity**: Only a small number of examples are provided per task.

Due to these properties, many models adopt inference-time training to perform localized, gradient-based adaptation during inference, enabling effective task-specific reasoning (Franzen et al., 2024; Bency et al., 2024; Akyürek et al., 2024).

**Rank-Based Filtering** When running ARC-AGI solvers, it is often observed that some tasks are solved with very high accuracy, while others are consistently unsolvable. In other words, within each task, the accuracy does not vary significantly across examples, suggesting that the notion of *answerability* can be discretely defined at the task level.

The major problem of TTT was that when applied to answerable tasks, in many cases the model rather lost its ability

to solve that task. Therefore, we first applied a prefilter to identify answerable tasks. The goal of the filter is to detect answerable tasks at inference time and exclude them from TTT. Since incorrectly identifying an *unanswerable* task as *answerable* is less harmful than the reverse, we aimed to define a strict and conservative decision boundary.

In ARC-AGI, all pixels of the output grid must be correct for a solution to be accepted. This implies that every token must have high likelihood under the model's prediction. In turn, this means that the correct token should consistently rank near the top of the model's probability distribution at each decoding step. If all tokens have ranks near 1, it indicates that the model has strong confidence in its predictions. To formalize this intuition, we introduce a rank-based criterion that reliably separates answerable tasks from unanswerable ones.

If few-shot inputs $\mathcal{I}^{(i)}$ and outputs $\mathcal{O}^{(i)}$ $(i = 1, 2, 3)$ were given, then the task was classified as *answerable* if

$$\frac{1}{3}\sum_{i=1}^{3}\frac{1}{|\mathcal{O}^{(i)}|}\sum_{t=1}^{|\mathcal{O}^{(i)}|} r_t^{(i)} < 1.00 + \alpha,$$

where

$$r_t^{(i)} := \mathrm{rank}\left(\mathcal{O}_t^{(i)} \,\middle|\, \underbrace{\mathcal{I}^{(a)}, \mathcal{O}^{(a)}, \mathcal{I}^{(b)}, \mathcal{O}^{(b)}}_{\text{samples other than } i}, \mathcal{I}^{(i)}, \mathcal{O}_{<t}^{(i)}\right).$$

Here, $\mathrm{rank}(t \mid \mathbf{s})$ refers to the generation rank of the token $t$ if the model were given a sequence $\mathbf{s}$ as input.

Through empirical evaluation, we found that setting $\alpha = 0.02$ led to reliable identification of answerable tasks. Compared to log probabilities or logits, this rank-based approach significantly reduced false positives—cases where unanswerable tasks were misclassified as answerable—thus producing more reliable results in practice. A quantitative comparison of alternative criteria is provided in Section 5.

**Test-Time Training (TTT)** If a task was not classified as *answerable*, TTT was applied. A *leave-one-out* dataset of size 3 was constructed from the few-shot samples, where for each sample, that sample became the test sample and the other two samples became the train samples.

We attached a LoRA (Hu et al., 2022) adapter with rank 128 and trained for 15 epochs. Due to the limit of execution time, only few training steps were allowed. Hence, we adopted a high learning rate (1e-4) and set the effective batch size 1 to encourage the model to learn the underlying rule fast.

**Confidence-Based BoN Sampling** The final model (whether test-time trained or not) was prompted 5 times to generate diverse predictions. Then, the prediction with the highest average log probability of the generation tokens was chosen as the answer.

*Table 1.* Performance across fine-tuning strategies

| Base Model | Training Method | Train Accuracy | Test Accuracy |
|---|---|---|---|
| Qwen3-0.6B | Full Fine-tune | **52.3** | 10.5 |
| Qwen3-4B | QLoRA ($r$ = 64) | <u>26.8</u> | <u>10.9</u> |
| Qwen3-4B | QLoRA ($r$ = 128) | 25.5 | **11.8** |

## 5. Discussion

### 5.1. Training Procedure Design

**Base Model Selection** Although any model officially released from Meta, Google or Qwen was allowed to use, we chose to use the Qwen3 model family for the following reasons:

(1) Llama 3 employs the `tiktoken` tokenizer (OpenAI, 2022; Llama Team, 2024), which tokenizes numbers by three digits. We expected that this could lead to misleading results when pixels were represented as single digit numbers. (Even if they were manually split.)

(2) Gemma models exhibit poor fine-tuning behavior.[1]

(3) Qwen also has the math model series, but these models are optimized to use chain-of-thought (CoT) reasoning or tool-integrated reasoning (TIR) to solve problems (Yang et al., 2024). This was not suitable for our use case, as we had execution time limits and could not accommodate lengthy responses.

**Fine-Tuning Strategy** With the base model family selected, the next decision concerned model size and fine-tuning strategy. Specifically, we compared two approaches: *full fine-tuning a small model* versus *parameter-efficient fine-tuning (PEFT) to a larger model*. We conducted experiments with models of sizes 0.6B and 4B, evaluating both full fine-tuning and PEFT. For PEFT, we applied QLoRA (Dettmers et al., 2023) with ranks 64 and 128, using fixed hyperparameters: $\alpha$ = 16, no bias, and no dropout. The training dataset consisted of 220 tasks with 250 questions generated per task. We trained for 1 epoch, and all other training hyperparameters were kept consistent with those used in Section 3. Experiment results are presented in Table 1.

While applying QLoRA to the larger model slightly improved generalization, it led to a large performance drop on in-distribution tasks. Since our evaluation included both

[1]While not conclusively verified, multiple community reports have criticized the difficulty of fine-tuning Gemma models, including comments from Nathan Lambert on X and Daniel Han on r/LocalLLaMA. We observed similar issues in our experiments.

*Table 2.* Performance across data diversity variations

| Questions Per Task | Training Epochs | Train Accuracy | Test Accuracy |
|---|---|---|---|
| 50 | 5 | 37.3 | 6.5 |
| 80 | 3 | 44.5 | **12.5** |
| 250 | 1 | **52.3** | 10.5 |

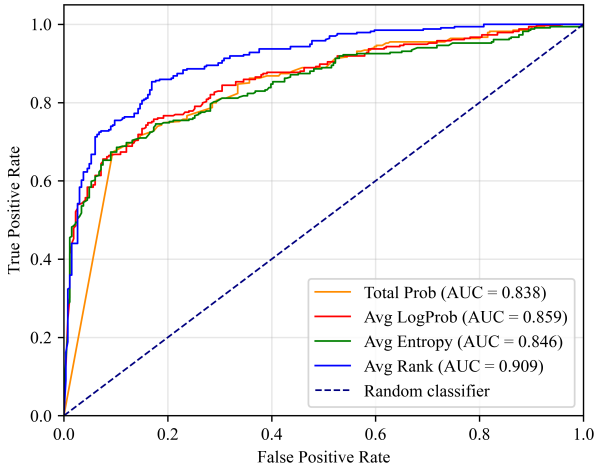


*Figure 3.* ROC curves for question filtering methods

in- and out-of-distribution tasks, this trade-off was critical. Larger models also incurred slower inference speed, so we chose the 0.6B model and rather applied test-time scaling.

**Data Diversity** Another hypothesis we explored was that, under a fixed training time budget, reducing data (not task) diversity and increasing training epochs could lead to better performance. To test this, we conducted experiments varying the number of questions per task and the number of training epochs accordingly. Other settings were kept equal to the 0.6B full fine-tuning setup from above. The results are presented in Table 2.

We observed that training over the same set of questions multiple times improved test accuracy at a certain level, suggesting that repeated exposure may help the model internalize general problem-solving patterns. However, this approach also resulted in poor in-distribution task performance, so we chose to expose the model to the full dataset and then increase the number of training epochs.

### 5.2. Ablation Studies on Inference-Time Approaches

**Alternatives to the Rank-Based Filter** Other than the rank-based classifier, we also tested using the (1) entire probability of the answer (2) average log probability at each token, (3) average entropy at each token. ROC curves for all methods are shown in Figure 3. The rank-based filter performed the best, so we chose it with threshold 1.02.

**Generation Strategy** With the final model checkpoint, we evaluated several decoding strategies, including our main inference method, greedy decoding, and beam search. The results are shown in Appendix C. Our inference strategy consistently outperformed the alternatives.

## Author Contributions

**Jeongmin Park** contributed to the method through data augmentation, analyzed Gemma, Qwen3-0.6B (LoRA), and the ARC-AGI dataset, developed the team workspace (Notion), implemented code for test result rendering and data augmentation, and wrote Section 4.

**Eun Kho** contributed to analysis (Qwen3-1.7B, Qwen3-0.6B with new tokens), coding (test result printing), and wrote section Section 3.

**Woosung Kim** conducted experiments on Llama 3 and Qwen3-4B with QLoRA, implemented and analyzed test time training method in inference time pipeline, refactored code, and wrote Sections 1 and 2.

**Jaewan Park** conducted training on Qwen3-0.6B with full fine-tuning, experimented with adding new tokens to the tokenizer and applying problem augmentation, analyzed TTT pre-filtering strategies and generation strategies, implemented code for comparing training configurations, implemented code for the final training and inference pipelines, and wrote Section 5.

## References

Akyürek, E., Wang, Z., Geng, X., Raileanu, R., Zoph, B., Vasudevan, V., Ghosh, R., Soroush, E., Dai, Z., Chen, X., and Le, Q. V. The Surprising Effectiveness of Test-Time Training for Few-Shot Learning. *arXiv preprint arXiv:2411.07279*, 2024.

Bency, M., Agrawal, S., Shah, R., and Kiela, D. ARC Prize: A competition to advance general intelligence through benchmarking and open science. *arXiv preprint arXiv:2412.04604*, 2024.

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

Chollet, F. OpenAI o3 Breakthrough High Score on ARC-AGI-Pub, Dec 2024. URL https://arcprize.org/blog/oai-o3-pub-breakthrough.

Chollet, F., Knoop, M., Kamradt, G., and Landers, B. Arc prize 2024: Technical report. *arXiv preprint arXiv:2412.04604*, 2025.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. QLoRA: Efficient Finetuning of Quantized LLMs. In *Advances in Neural Information Processing Systems*, volume 36, pp. 10088–10115. Curran Associates, Inc., 2023.

Franzen, D., Disselhoff, J., and Hartmann, D. The LLM ARChitect: Solving ARC-AGI Is A Matter of Perspective, 2024. URL https://github.com/da-fr/arc-prize-2024/blob/main/the_architects.pdf.

Hu, E. J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. LoRA: Low-Rank Adaptation of Large Language Models. In *The Tenth International Conference on Learning Representations*, 2022.

Llama Team. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024.

OpenAI. tiktoken, 2022. URL https://github.com/openai/tiktoken.

Qwen Team. Qwen3 Technical Report. *arXiv preprint arXiv:2505.09388*, 2025.

Yang, A., Zhang, B., Hui, B., Gao, B., Yu, B., Li, C., Liu, D., Tu, J., Zhou, J., Lin, J., Lu, K., Xue, M., Lin, R., Liu, T., Ren, X., and Zhang, Z. Qwen2.5-Math Technical Report: Toward Mathematical Expert Model via Self-Improvement. *arXiv preprint arXiv:2409.12122*, 2024.

## A. Prompt Template

### System

You are a puzzle solving wizard. Your job is to solve tasks from the abstraction and reasoning corpus developed by Francois Chollet.

### User

These are some input–output grid examples that define the task.

input:
[ Train Input Grid 1 ]
output:
[ Train Output Grid 1 ]

input:
[ Train Input Grid 2 ]
output:
[ Train Output Grid 2 ]

input:
[ Train Input Grid 3 ]
output:
[ Train Output Grid 3 ]

Now, solve the following puzzle based on its input grid by applying the rules you have learned from the training data:

input:
[ Test Input Grid ]
output:

### Assistant

[ Test Output Grid ] : *Expected Output*

Each grid is formatted in a simple manner: rows are separated by `\n`, and pixels within each row are concatenated without any spaces.

## B. Hyperparameters for Training

*Table 3.* Hyperparameter choices for the two-stage training process. Text in bold are the ones that differ across stages.

| | **First Stage (w/o Data Augmentation)** | **Second Stage (w/ Data Augmentation)** |
|---|---|---|
| Epochs | **3** | **1** |
| Batch Size | 1 | 1 |
| Gradient Accumulation Steps | 32 | 32 |
| Gradient Checkpointing | Activated | Activated |
| Learning Rate | **5e-5** | **1e-6** |
| Learning Rate Scheduler | Cosine Annealing | Cosine Annealing |
| Warmup Ratio | 0.03 | 0.03 |
| Weight Decay | 0.01 | 0.01 |
| Optimizer | AdamW | AdamW |

## C. Experiment Results on Various Generation Strategies

*Table 4.* Performance across generation strategies, without test-time training. All evaluation were made on the final checkpoint. Since the model was trained on the full dataset, **train accuracy** is reported over arbitrary questions sampled from the entire dataset, while **test accuracy** is measured on *task-augmented data*, where augmentation is applied to either the input or output alone to create a new task. Blank leaderboard scores indicate that no submission was made.

| Generation Strategy | Configuration | Train Accuracy | Test Accuracy | Leaderboard Score |
|---|---|---|---|---|
| Greedy Decoding | – | 79.0 | 14.0 | – |
| Beam Search | Beam Size = 5 | 81.0 | 18.0 | 63.0 |
| | Beam Size = 10 | 81.0 | 17.0 | 63.0 |
| Confidence-Based BoN Sampling | Temperature = 0.7, Top-k = 5 | <u>84.0</u> | **20.0** | <u>69.0</u> |
| | Temperature = 0.7, Top-k = 10 | 83.0 | <u>19.0</u> | – |
| | Temperature = 1.5, Top-k = 5 | <u>84.0</u> | 18.0 | – |
| | Temperature = 1.5, Top-k = 10 | **86.0** | **20.0** | **70.0** |