DHBW
Duale Hochschule
Baden-Württemberg

Evaluating Synthetic Chain-of-Thought via RL Fine-Tuning for ARC-AGI Problem Solving

from the course of studies Applied Computer Science

at the Cooperative State University Baden-Württemberg Mannheim

by

Lukas Marschhausen

Marc Schmengler

15.04.2025

Student ID, Course: 1840227, TINF22AI1
1708015, TINF22AI1

Declaration of Authorship

Gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden- Württemberg vom 29.09.2017. Wir versichern hiermit, dass wir unsere Arbeit mit dem Thema:

Evaluating Synthetic Chain-of-Thought via RL Fine-Tuning for ARC-AGI Problem Solving

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben. Wir versichern zudem, dass alle eingereichten Fassungen übereinstimmen.

Eschborn and Berlin 15.04.2025

Lukas Marschhausen

Marc Schmengler

Abstract

TODO

Table of Contents

List of Figures

List of Tables

Code Snippets

[1]This is a simplified representation of the JSON structure for better readability. (training set task 392: f8ff0b80.json)

List of Acronyms

AGI	Artificial General Intelligence
ARC	Abstraction and Reasoning Corpus
LLM	Large Language Model
ML	Machine Learning
RL	Reinforcement Learning

Glossary

Exploit An exploit is a method or piece of code that takes advantage of vulnerabilities in software, applications, networks, operating systems, or hardware, typically for malicious purposes.

Patch A patch is data that is intended to be used to modify an existing software resource such as a program or a file, often to fix bugs and security vulnerabilities.

Vulnerability A Vulnerability is a flaw in a computer system that weakens the overall security of the system.

1 Use of AI in this Paper

As artificial intelligence, particularly Large Language Models (LLMs), continues to advance in capability, this paper acknowledges and strategically incorporates this technology. Computer scientists and society face two distinct options: either disregard AI's existence—potentially sacrificing productivity, code quality, documentation standards, and operational efficiency—or judiciously embrace it to enhance research methodologies where appropriate.
This work does not utilize LLMs for direct content generation. This principled stance remains despite the recognition that—through various obfuscation techniques—artificially generated content has become increasingly difficult to detect or verify. Instead, AI serves an auxiliary function in this paper, primarily for identifying and correcting linguistic errors and suggesting alternative phrasings to enhance readability and comprehension. These suggestions have been selectively implemented to refine our writing, ensuring optimal clarity and coherence throughout the document.
The specific tools employed include: [7], [8], [9], and [10]. While disclosure of such technological assistance may not be formally required in academic publishing standards, we consider it essential to establish transparent practices regarding the utilization of generative tools in scholarly work. Therefore, despite the option to omit this information, we have included this statement to maintain intellectual integrity and transparency regarding the integration of generative AI in this research.

As an example of the extent of changes introduced into this work, the original, unmodified paragraph is provided below:

As artificial intelligence, especially large language models, are getting more capable by the second, this paper will make use of this technology. The reason for that is that we, as computer scientists and even as a society, have two options: either pretend that it does not exist and slowly fall behind in productivity, code quality, documentation quality, and efficiency, or we embrace it and use it where it can

improve our work. We will not use large language models to generate content for this work, even if no one is capable (when applying clever obfuscation) of recognizing or checking if this work contains artificially generated content. We will use artificial intelligence to correct spelling and grammatical mistakes and have large language models suggest different, smoother, and better-to-understand versions of our sentences to improve readability and understandability. It may be a little too early to include this type of information in a project, but we are pretty sure that this needs to become a standard practice of declaring what kind of generative AI was used in the creation of any written work. So even though this paragraph could have been left out completely, we decided to put it in to be transparent and honest about the use of generative tools in this work.

2 Basic Terminology

2.1 Chain of Thought

The chain-of-thought principle describes a strategy in which AI models formulate their thought processes in several comprehensible intermediate steps instead of just presenting the final result directly [11]. This is often done in larger language models in order to deal with complex issues such as mathematical tasks, logical conclusions or the comprehension of long texts [12]. A simple example would be a maths problem in which the model not only states the sum of two numbers, but also explains step by step how this is arrived at: first the numbers are broken down into their place values and added individually until the correct result is finally obtained. Thanks to this step-by-step structuring and presentation of the solution, errors are easier to recognise and can be corrected if necessary. In practice, the method is mainly used where a clear derivation is crucial, for example in tricky puzzles, in the field of automated text analysis or when analysing legal documents. This explicit disclosure of the thought process not only increases accuracy, but also promotes trust in the answers of AI systems, as users can better understand the explanations.

2.2 Scaling Laws

The term “scaling laws” refers to guidelines or observations that describe how the performance of AI models develops depending on their size, the amount of data available and the computing power used. The larger a neural network and the more extensive the training data, the more the accuracy can generally increase - however, diminishing marginal returns often occur from a certain point onwards. This means that while the accuracy is improved rapidly at the beginning by additional growth of the model or more data samples, the gain often only increases slowly later on with equally increasing effort. By analysing such scaling effects, predictions can be made as to how many resources are required to achieve a certain level of

performance, which is of considerable importance for the planning of large-scale training projects. At the same time, scaling laws help to assess where optimisations in the model design or in the data are most effective.

2.3 Training process and inference (Train-Time vs. Test-Time)

Two central phases can be distinguished in the context of modern AI applications: One is the training process and the other is inference. During training, often referred to as 'train time', a model is adapted using large amounts of data and special optimisation methods. This requires extensive computing resources, as each input example is first run through the algorithm in a forward pass before the so-called backpropagation step takes place. This process is used to change the parameters so that the model can make increasingly reliable predictions. Depending on the complexity and size of the network, training can take a long time and is usually associated with high costs for hardware and energy [13].

As soon as the model is sufficiently well trained, it is transferred to the field. This practical application is known as inference or 'test time'. The sole aim here is to retrieve the learnt parameters and apply them to new inputs. In contrast to training, the model is no longer changed during inference, but uses the previously learnt relationships to make predictions or decisions [14]. Although inference is generally much faster and less computationally intensive than training, it can still require resources depending on the size of the model, the environment in which it is used and the number of queries. In productive scenarios, efficient inference is therefore just as important as a well-organised training process, as the AI system used often has to deal with large volumes of queries in a short space of time [15].

2.4 Hyperparameter Tuning

Hyperparameter tuning refers to the systematic adjustment of certain settings in the deep learning model in order to achieve the best possible results [16]. The learning rate plays a central role here: it regulates how quickly the weights change during training. If it is too high, the model can become unstable or not converge at all; if it is too low, it takes a very long time to reach an optimum point [17].

Deep learning is a sub-area of machine learning that focuses on the use of artificial neural networks with many layers (hence the term ‘deep’). These networks are able to automatically learn complex features and patterns in large amounts of data without having to manually define these features in advance [18]. The hierarchical structure enables deep learning models to recognise simple patterns (such as edges in images or simple speech sounds) at a low level and combine this information into increasingly abstract concepts in higher layers. This makes deep learning particularly powerful in areas such as image and speech recognition, natural language processing and many other applications where traditional algorithms often reach their limits.

The batch size must also be chosen carefully: Although large batches speed up calculations on modern hardware, they can cause the model to get ‘caught’ in shallow valleys of the error space and not converge optimally. Small batches, on the other hand, increase the variance in the gradient calculation, but often ensure a more robust, albeit slower, fit.
In machine learning, batches refer to subsets of the training dataset that are processed together in one forward and backward pass during the training process. Rather than using the entire dataset to compute the gradient (as in full-batch gradient descent), the data is divided into smaller groups called batches. This method, known as mini-batch gradient descent, allows the model to update its parameters more frequently, making training more efficient and manageable, especially with large datasets. Moreover, processing batches takes advantage of modern hardware accelerators like GPUs, which are optimized for handling multiple data points in parallel.

In addition, the number of layers influences how deeply the model processes its inputs, with more layers often meaning a higher computational load, but also a greater capacity for abstraction. Finally, dropout plays an important role in regularisation: it determines how many neurons are temporarily ‘switched off’ during training, which should reduce overfitting and improve generalisation. All of these components together significantly determine how well and how quickly a neural network learns.

3 Introduction

Large language models (LLMs) have historically improved through scaling laws, where increases in parameters and training data correlate with enhanced performance. Emergent abilities, such as solving unseen mathematical problems, validated this approach. However, performance gains began to plateau nonlinearly: doubling model size no longer doubled capability, while computational costs grew exponentially. This diminishing return necessitated alternative strategies, leading to the exploration of test-time compute — enhancing reasoning during inference rather than solely relying on larger architectures.

Early LLMs (e.g., GPT-3) demonstrated that scaling parameters unlocked novel capabilities. However, as models grew beyond hundreds of billions of parameters, performance improvements became sublinear relative to resource investment, indicating fundamental limitations to the scaling paradigm.

In 2022, researchers at Google introduced chain-of-thought (CoT) prompting, enabling models to decompose problems into intermediate steps during inference. This method significantly improved performance on arithmetic, commonsense, and symbolic reasoning tasks. [19] Crucially, CoT shifted computational burden to inference time, decoupling performance gains from model size alone and establishing a new direction for enhancing language model capabilities.

On May 31, 2023, OpenAI released PRM800K, a dataset of 800,000 human-curated step-by-step reasoning traces documenting mathematical problem-solving processes. [20] The underlying concept behind these human-generated chains of thought was to train a model to evaluate the accuracy of step-by-step reasoning chains produced by LLMs.

The accompanying paper details how this dataset was used to train a Process Reward Model (PRM), which evaluates intermediate reasoning steps generated by

language models. This approach proved more effective than using an Outcome Reward Model (ORM), which only assessed the final result of a reasoning chain. [21]

A particularly significant observation from this research was articulated in the following statement:

> "We do not discuss any supervision the generator[2] would receive from the reward model if trained with RL. Although finetuning the generator with RL is a natural next step, it is intentionally not the focus of this work."
>
> — [21]

This statement indicates the potential to fine-tune an LLM using this reward model to leverage chain-of-thought reasoning during inference. Approximately a year later, on September 12, 2024, OpenAI released o1 — the first LLM designed to incorporate chain-of-thought reasoning in its inference process by default. It demonstrated superior performance across mathematical and logical reasoning benchmarks at the time of its release. [22]

Despite these advances, o1 still exhibits limitations with certain reasoning tasks. For example:

User

Beth places four whole ice cubes in a frying pan at the start of the first minute, then five at the start of the second minute and some more at the start of the third minute, but none in the fourth minute. If the average number of ice cubes per minute placed in the pan while it was frying a crispy egg was five, how many whole ice cubes can be found in the pan at the end of the third minute? [23]

[2] The LLM that generated a reasoning chain, e.g., GPT-4

> **o1**
>
> [...] Therefore, the number of whole ice cubes in the pan at the end of the third minute is 11[3]

As OpenAI has transitioned from its original open-source mission to a for-profit enterprise, the methodologies behind o1's training remain proprietary. However, in January 2025, DeepSeek — a Chinese AI company — released DeepSeek-R1, an LLM demonstrating performance comparable to OpenAI's o1. DeepSeek made their model weights fully available for download and use, publishing a comprehensive paper detailing how they achieved this performance with minimal training costs. [24]

The fundamental approach described in their paper employs a rule-based outcome reward function that verifies the correctness of final answers for reasoning-based questions in domains like mathematics and coding. This framework allows the model to develop its own approach to solution-finding through reinforcement learning. To encourage structured reasoning, the researchers rewarded the model not only for correct outputs but also for employing a specific output structure:

$$<\text{think}>...</\text{think}><\text{answer}>...</\text{answer}> \tag{1}$$

An intriguing development occurred during training — the model progressively increased its response length and exhibited emergent behaviors without explicit instruction. Notably, it began to self-reflect on previous steps and reevaluate its answers, despite not being exposed to such behaviors in the training data. This spontaneous development of meta-cognitive reasoning strategies was later termed the "aha moment" phenomenon.[4] In one documented instance, the model generated:

[3]The confusion of the model primarily stems from overlooking the contextual information about an egg frying in the pan, focusing instead on the mathematical components of the problem.

[4]Multiple post-training steps, fine-tuning, and even full retraining occurred in later stages, but for the sake of brevity we will not discuss this here

> **deepseek-r1-zero**
>
> [...] Wait, wait. Wait. That's an aha moment I can flag here. Let's reevaluate this step-by-step [...]

Since this approach requires only input-output pairs as training data, we hypothesize that it could be applied to domains where LLMs have previously struggled to produce tangible results. One such domain is spatial generalization on novel tasks, as evaluated by the ARC-AGI-Benchmark. This benchmark consists of example grids that must be transformed into output grids, requiring models to learn from examples and generalize observed patterns to new tasks. Currently, the highest score on this benchmark is held by OpenAI's o3 model at 4%, compared to human performance of 100%. [6]

Our research investigates whether DeepSeek's reinforcement learning methodology can be adapted to enhance pattern generalization capabilities on the ARC-AGI-2 dataset, potentially bridging the substantial gap between current AI systems and human-level performance in abstract reasoning tasks.

4 ARC-AGI Benchmark

4.1 Introduction

In 2019, François Chollet, a researcher at Google, introduced a framework to define and measure the intelligence of computational systems [25]. Chollet differentiated between two distinct categories of intelligence: narrow, skill-based intelligence and generalization-capable intelligence. Narrow intelligence describes systems that excel at singular, predefined tasks but lack flexibility in adapting their knowledge to new or unfamiliar situations.
Conversely, generalization-capable intelligence encompasses systems that can effectively transfer learned patterns and experiences to novel tasks, adapting their understanding dynamically. Chollet observed that most contemporary Reinforcement Learning (RL) and Machine Learning (ML) systems exhibited proficiency in narrow, task-specific scenarios but struggled significantly with generalization. To address this shortcoming, he proposed a comprehensive definition of intelligence.

> "The intelligence of a system is a measure of its skill-acquisition efficiency over a scope of tasks, with respect to priors, experience, and generalization difficulty"
>
> — François Chollet [25]

Central to this definition is a system's capability to generalize effectively, leveraging prior knowledge and past experiences to adapt quickly and efficiently to new challenges. This insight led to the creation of the Abstraction and Reasoning Corpus (ARC) Artificial General Intelligence (AGI) Benchmark [26]. The ARC Benchmark assesses a system's proficiency in spatial reasoning and ability to infer novel information from limited prior examples. Through tasks designed to mimic human-like abstract reasoning, ARC provides a robust evaluation framework for generalization-capable intelligence, advancing research in artificial general intelligence.

4.2 Dataset

The ARC Dataset's fundamental concept is that its tasks are not memorizable, requiring LLMs to employ genuine reasoning rather than regurgitating previously encountered information. LLMs typically reproduce statistical patterns observed in their training data—learning which words frequently follow others in specific contexts. This baseline capability is evident in earlier iterations of the GPT series:

User

The DHBW is

gpt-3.5-turbo-instruct[5]

a dual university that combines academic studies with practical training in a company. [...]

However, this pattern recognition capability does not necessarily translate to abstract conceptual understanding. Even after question-answer fine-tuning, models demonstrate limited abstract reasoning capabilities. To investigate this hypothesis systematically, we examined the performance of contemporary LLMs on the ARC dataset, first establishing a clear understanding of the dataset's structure through visualization.

[5]This model is deprecated and no longer available via the OpenAI Platform. This example has been generated in July 2024

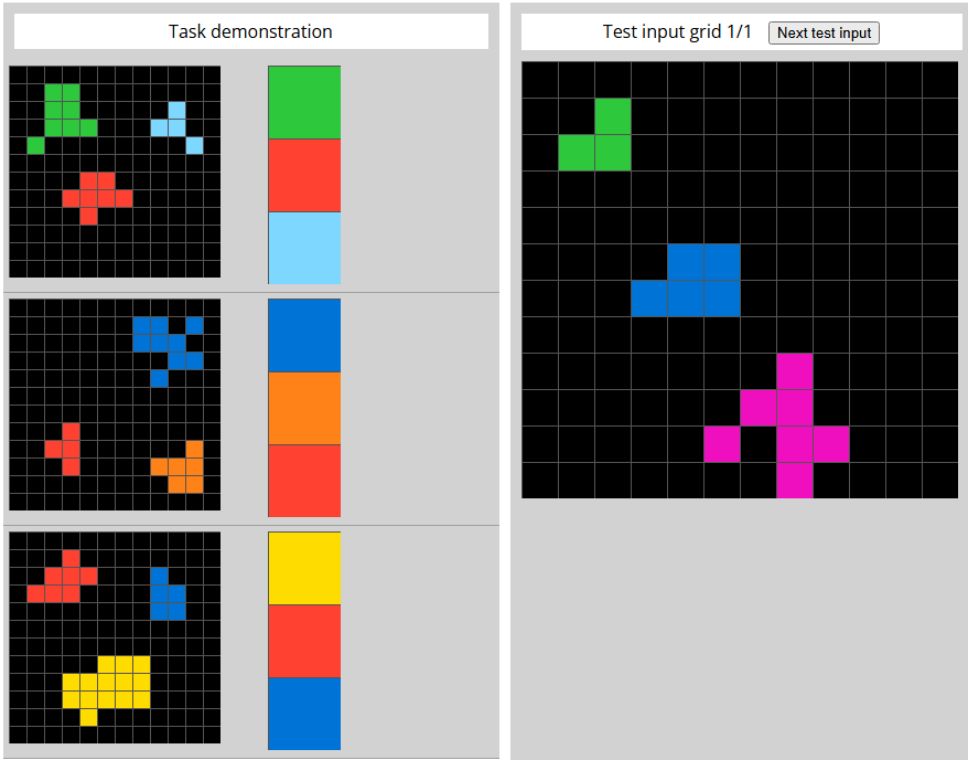

Figure 1 — Examples from the ARC dataset showing three input-output pairs (left) and a test input (right). (training set task 392: f8ff0b80.json) [1]

Humans readily identify the pattern in this example: colors are arranged based on frequency in the input grid, with the most prevalent color appearing first. After observing just a few demonstrations, this abstract rule can be immediately applied to new examples.

Figure 2 — Solution showing the completed transformation, where colors are arranged vertically in descending order of their frequency in the original input grid. (training set task 392: f8ff0b80.json) [1]

For LLMs, however, this transformation is considerably less intuitive. We evaluated GPT-4o's performance on this specific task, providing only the unmodified JSON

input with a temperature setting of 1 to ensure deterministic output. The dataset utilizes numerical values (0-9) rather than colors for grid representation:

```
   "train":
       "input":
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 3, 3, 0, 0, 0, 0, 0, 0, 0, 0],
[...]]
       "output":
[[3], [2], [8]]
       "input": [...]
       "output": [...]
[...]
   "test":
     "input":
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0],
[...]]
     "output": "Searching this. Find out how to create this output."
```

Listing 1 — The actual representation of the task as it is represented in the JSON file[6]

GPT-4o responded with the following analysis:

> gpt-4o
>
> [...]
>
> **Observations**
>
> - The output lists the unique numbers present in the input grid.
> - The numbers are listed in descending order based on their first appearance from top to bottom in the grid.
>
> **Test Input Analysis**

[6] This is a simplified representation of the JSON structure for better readability. (training set task 392: f8ff0b80.json)

For the test input, we have the following numbers in the grid:

- 3 appears first.
- 1 appears next.
- 6 appears last.

Applying the Pattern

Based on the pattern observed, the output for the test input should be:

- [3], [1], [6]

This lists the unique numbers in the order they appear from top to bottom in the grid.

The analysis demonstrates that the LLM failed to identify the correct transformation rule—sorting colors in descending order of their frequency within the original input.

Our initial experiments utilized the ARC-AGI-1 Dataset. However, with the release of the ARC-AGI-2 dataset in March 2024, we transitioned to this updated Benchmark for all subsequent experiments to maintain methodological consistency. The table below presents current performance benchmarks for various models on the ARC-AGI-2 Dataset [6].

Model[7]	**ARC-AGI-2 Score**	ARC-AGI-1 Score
Human	100.0%	98.0%
o3 (low)	4.0%	75.7%
o1 (high)	3.0%	32.0%
o3-mini (medium)	1.7%	29.1%
Gemini 2.0 Flash	1.3%	N/A
Deepseek R1	1.3%	15.8%
Gemini-2.5-Pro-Exp-03-25	1.3%	12.5%
Claude 3.7 (8K)	0.9%	21.2%
GPT-4.5	0.8%	10.3%
o1 (low)	0.8%	25.0%
GPT-4o	0.0%	4.5%
GPT-4o-mini	0.0%	N/A

Table 1 — The current ARC AGI Leaderboard. [6]

4.3 Investigation of Prompt Engineering Efficacy

To evaluate prompt engineering's potential for improving abstract reasoning capabilities, we implemented a systematic evaluation framework using two distinct prompting strategies across all 120 ARC-AGI-2 evaluation tasks.

Our initial experimental condition utilized GPT-4o with the following prompt architecture:

[7] Multiple other models and approaches were omitted to reduce the table length, yet no models with higher scores were omitted.

```
SYSTEM_PROMPT_TEMPLATE = (
     "You are an ARC puzzle solver. Analyze the train examples (input/output
pairs). "
    "Apply the deduced rule to the test input grid. "
   "Output your reasoning and then a JSON array in a codeblock for the predicted
test output grid."
)

USER_PROMPT_TEMPLATE = (
    "**TRAIN EXAMPLES:**\n"
    "{train_examples_string}\n\n"
    "**TEST INPUT GRID:**\n"
    "{test_input_string}\n\n"
    "**PREDICTED OUTPUT GRID:**"
)
```

Listing 2 — The Used Prompt Structure

The experimental results revealed complete inefficacy, with zero correct predictions across all 120 evaluation tasks (0% success rate). Our experimental design utilized a temperature setting of 1 for GPT-4o across the comprehensive evaluation suite comprising 172 individual tests (as some tasks contain multiple test conditions). This methodology resulted in approximately 1 million tokens processed during evaluation, with associated computational costs of approximately €5 per complete benchmark evaluation.

Given these resource constraints, we prioritized experimental breadth over replicated trials, though we acknowledge that averaging across multiple runs would enhance statistical robustness. Our findings align with official ARC-AGI-Benchmark metrics, which report 0% performance for GPT-4o [6]. [27].

Our computational efficiency analysis identified tokenization overhead as a significant contributor to processing costs, with raw JSON string representations resulting in inefficient token utilization—nearly every character requiring individual tokenization.

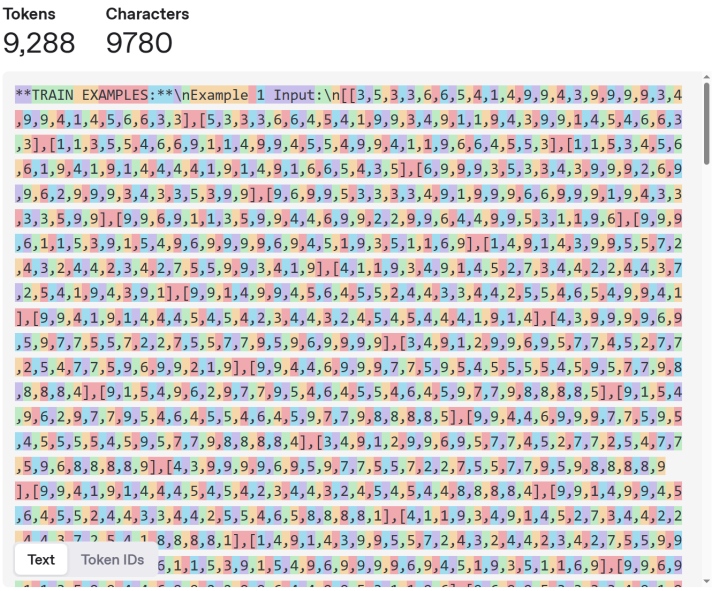

Figure 3 — Tokenization of the Input to the LLM for task 0934a4d8 [2]

To address these limitations, we implemented an alternative representation strategy with dual objectives: First, enhancing semantic interpretability by transforming JSON structures into human-readable grid formats, and second, optimizing computational efficiency through reduced token consumption. Interestingly, our analysis revealed that even with grid-formatted input, the tokenization pattern remained highly granular:

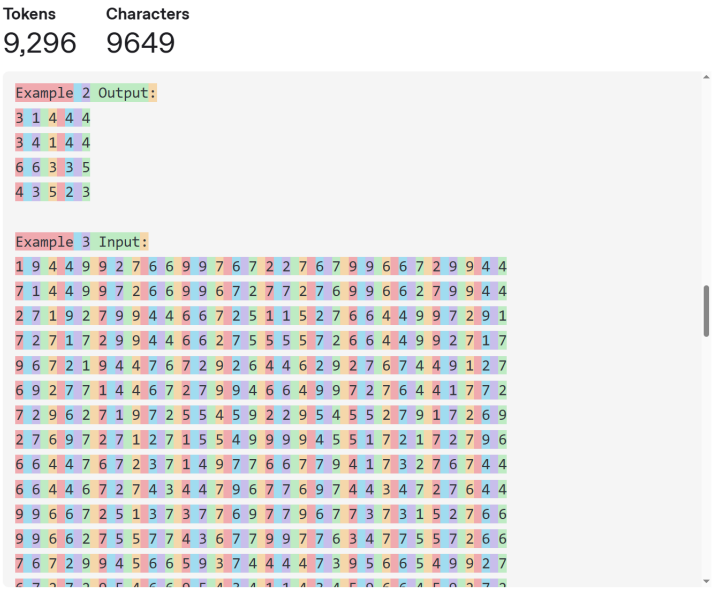

Figure 4 — Tokenization of the Input to the LLM for task 0934a4d8 [2]

This tokenization behavior can be attributed to architectural attention mechanisms that benefit from precise information representation. Comparative analysis between GPT-3 and GPT-4 tokenization strategies reveals significant evolutionary improvements in numerical data processing—GPT-4 implements consistent tokenization where each number is processed as a discrete token regardless of contextual whitespace, whereas earlier models treated “3” and “3 “ (with trailing space) as entirely distinct tokens.

This tokenization refinement directly enhances mathematical reasoning capabilities by maintaining referential integrity across computational contexts. Empirical research demonstrates that consistent tokenization significantly impacts arithmetic performance. Studies further indicate that models trained with consistently tokenized instances achieve enhanced cross-domain performance, accelerated convergence, and reduced hallucination. [28], [29], [30].

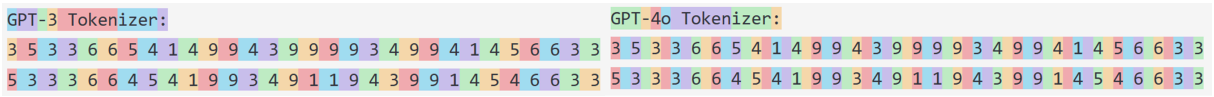

Figure 5 — Tokenization Comparison between GPT-4o and GPT-3 [2]

After evaluating multiple representation strategies, we determined that our approach optimized tokenization efficiency within text-based constraints without leveraging GPT-4o’s visual processing capabilities. While theoretical alternatives exist—such as Unicode color block representation (🟥, 🟦, 🟧, 🟨, 🟩, 🟪, 🟫, ⬛, ⬜, 🔳, 🔲)—our analysis suggested minimal potential performance improvements from such adaptations.

For our second experimental condition, we implemented an enhanced prompt incorporating Chain of Thought reasoning and step-by-step verification methodologies—techniques empirically demonstrated to significantly improve model reasoning capabilities [19], [21]:

```
SYSTEM_PROMPT_TEMPLATE = (
"""
You will be provided with example inputs and outputs. Analyze the train examples.
These tasks follow the style of ARC (Abstraction and Reasoning Corpus) problems,
where the objective is to deduce transformation rules from visual or structural
patterns.

Your goal is to find common rules that are applied to the input to be transformed
into the output.
To achieve this, do the following:
1. Find possible rules that could be applied in combination to achieve the
transformation from the input to the output. Be really precise in the rule
definition. What transformations have to be applied exactly? What are they
based upon?
2. Test those rules by applying them to all the available train examples and
seeing if they reproduce the desired output. You have to verify that the deduced
ruleset actually works with the train examples before proceeding to the test.
If the desired output is achieved in all present examples, then apply those
found rules to the given test input.
If the ruleset you deduced fails at any of the train examples, begin again from
step one and modify the rules you deduce.
Then test again for all train examples before proceeding to the test. (Output
your final solution as a JSON array in a code block)
"""
)

# Adjusted template slightly to ensure good spacing with multi-line grids
USER_PROMPT_TEMPLATE = (
    "**TRAIN EXAMPLES:**\n\n"
    "{train_examples_string}\n\n\n\n"
    "**TEST INPUT GRID:**\n\n"
    "{test_input_string}\n\n"
)
```

Listing 3 — The New Prompt Structure

Despite these methodological enhancements, experimental results maintained a 0% solve rate. Our analysis indicates that current LLMs fundamentally lack the necessary abstraction capabilities to understand the transformation logic under-

pinning ARC tasks—particularly those requiring world knowledge concepts such as gravity, suction, rotation, and mirroring. This experimental condition processed 865,000 input tokens across 174 API requests (120 tasks comprising 172 tests, plus 2 repeated requests due to API errors), with approximate computational costs of €5.

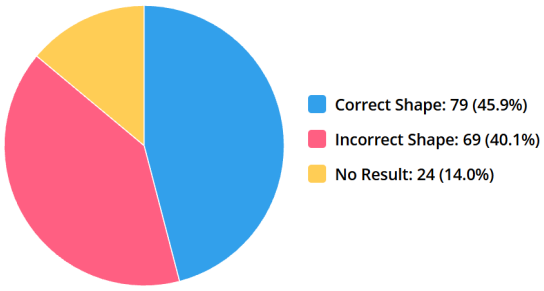

Figure 6 — The Type of Failure of the Second Run. For full run details, see [3]

Based on these experimental findings, we conclude that prompt engineering alone — even employing methodologies such as Chain of Thought reasoning and step-by-step verification — cannot overcome the fundamental abstraction limitations preventing LLMs from solving ARC-AGI-tasks.

5 Previous Methodologies and Approaches

This chapter presents an examination of leading methodological frameworks that have demonstrated significant performance on the ARC-AGI Benchmark. While OpenAI's proprietary models currently occupy prominent positions on the leaderboard, our analysis focuses on two independent research teams—"ARChitects" and "Icecuber"—whose open methodologies have achieved substantial results across both the ARC-AGI-1 and ARC-AGI-2 Benchmarks, as illustrated in Table 2.

Model	**ARC-AGI-2 Score**	ARC-AGI-1 Score[8]	**Cost/Task**
Human Panel	100.0%	98.0%	$17.00
o3 (low)*	4.0%	75.7%	$200.00
o1 (high)	3.0%	32.0%	$4.45
ARChitects	2.5%	56.0%	$0.200
o3-mini (medium)	1.7%	29.1%	$0.280
Icecuber	1.6%	17.0%	$0.130

Table 2 — The current top 5 of the ARC AGI Leaderboard. [6]

5.1 The ARChitects: A Perspective-Based Approach to ARC-AGI

The methodology developed by "The ARChitects" in "The LLM ARChitect: Solving ARC-AGI Is A Matter of Perspective" [31] presents significant advancements in addressing the challenges posed by the Abstraction and Reasoning Corpus (ARC) benchmark. Their approach implements multiple methodological innovations that collectively enhance model performance on spatial reasoning tasks.

5.1.1 Model Selection and Dataset Augmentation

The research team employed the Mistral-NeMo-Minitron-8B-Base model—a distilled variant of Mistral optimized by NVIDIA for inference efficiency while maintaining high performance characteristics. This foundation model underwent fine-tuning on a comprehensive dataset comprising [32]:

[8]Provided for reference, yet not used in this paper.

Dataset	Tasks Used in Training
Re-ARC [33]	Up to 257,600
ARC-AGI Eval (75% used)	Up to 51,200
Concept-ARC [34]	Up to 22,528
ARC-Heavy [35]	Up to 200,000

Table 3 — Overview of datasets and the number of training tasks used.

A central innovation in their methodology was the systematic application of data transformations to enhance pattern recognition capability. These transformations included:

- Spatial transformations (rotations and reflections forming D8 symmetry operations)
- Color permutations (rearrangements of the 10 possible color values)
- Example order permutations (varying the sequence of training examples)

These augmentation strategies expanded the effective training dataset to 531,318 examples, enhancing the model's ability to recognize pattern invariants across different representations.

5.1.2 Inference Optimization Framework

The ARChitects' approach implements a three-stage inference optimization framework:

1. **Multi-perspective task presentation**: The system applies the same transformations used during training to generate 8-16 alternative perspectives of each input problem, enabling the model to approach problems from angles where the underlying pattern might be more apparent.
2. **Depth-First Search (DFS) candidate generation**: Traditional token selection methods were replaced with a custom DFS algorithm that explores solution paths with cumulative probability exceeding a specified threshold, efficiently identifying solutions with highest overall confidence.
3. **Cross-perspective candidate evaluation**: The selection strategy aggregates model confidence scores across multiple transformed perspectives of the same

task, using the product of probabilities to identify consistently confident solutions.

This selection stage improved their score by approximately 25% over baseline approaches, demonstrating the effectiveness of multi-perspective evaluation.

—

5.2 The Icecuber: A Search-Based Approach

The submission by Johan Wind (known as "Icecuber") represents an wildly different approach to the ARC benchmark that diverges from conventional machine learning methods. [36] Instead of training a neural network, Wind developed a system that searches for sequences of image transformations to solve each task.

5.2.1 Core Approach

Wind's approach consisted of three main components:

1. **Transformation Library**: A collection of 142 different image operations derived from 42 core concepts. These operations included basic functions like rotating images, isolating specific colors, finding the largest shape, and combining image components.

2. **Search Process**: A systematic exploration of possible combinations of these transformations (up to 4 operations in sequence) until finding a sequence that correctly transformed all training examples for a given task.

3. **Efficient Implementation**: The entire system was built in C++ with careful optimization for speed and memory usage, allowing the search process to explore more possibilities within the competition's time constraints.

The DSL implementation encapsulated fundamental visual reasoning primitives including component segmentation, color manipulation, geometric transformation, and compositional operations. This transformation library was empirically derived through manual analysis of approximately 200 ARC tasks, ensuring com-

prehensive coverage of recurring visual patterns. The search process implemented state deduplication mechanisms through efficient hashing techniques, enabling the exploration of substantially larger solution spaces than would otherwise be feasible within the competition's computational constraints. For tasks where no single transformation sequence solved all training examples, the system employed a "greedy stacking" approach that combined multiple partial solutions by selecting the most effective transformation for each specific example.

5.2.2 Solution Strategy

The search process worked by:

1. Starting with an input images
2. Applying each possible transformation to create many intermediate images
3. Continuing this process up to 4 steps deep
4. Selecting the transformation sequence that correctly solved all training examples

5.2.3 Performance Enhancement Techniques

Wind employed several techniques to improve results:

1. **Multiple Perspectives**: Running separate searches on transformed versions of the tasks (particularly diagonal flips), which helped solve problems that were easier to recognize from different orientations.

2. **Color Normalization**: Preprocessing the images to standardize colors, helping the system focus on patterns rather than specific color values.

3. **Ensemble Approach**: Running multiple configurations with different parameters and selecting the best result based on training accuracy.

Interestingly, the perspective-based approach pioneered by Wind in 2020 was later adopted and expanded upon by "The ARChitects" team in their work "The LLM ARChitect: Solving ARC-AGI Is A Matter of Perspective" . The ARChitects similarly

utilized multiple perspectives through spatial transformations and achieved a reported 25% performance improvement using this technique. [31]

6 Our Approach

Current open-source approaches predominantly rely on direct inferencing without implementing chain-of-thought or intermediate reasoning steps. Notably, solutions such as ARChitect and Icecuber, as well as other open-source methodologies represented on the leaderboard, lack mechanisms that explicitly extract reasoning processes and logical thinking required to solve ARC tasks. However, as evidenced by the ARC benchmark results (see [6]), reasoning-oriented models like OpenAI's o1 and o3 achieve superior performance without additional optimization compared to non-reasoning models.

This performance differential stems from their implementation of test-time compute capabilities. These models are structured to incorporate an intermediate step between input processing and output generation, utilizing chain-of-thought mechanisms to reason through potential solutions before producing answers.

DeepSeek pioneered the commoditization of this approach by releasing their model with open weights and publishing their methodological framework. Subsequently, Jian Pan successfully replicated DeepSeek's methodology in his project TinyZero [37], which provides the foundation for our research. We aim to adapt this framework to develop a reasoning-oriented model specifically tailored for ARC tasks.

Our primary research objective is to investigate whether emergent reasoning behaviors can be cultivated when training on complex spatial datasets. Unlike previous approaches that focus primarily on pattern matching or transformation libraries, our methodology emphasizes the development of intermediate reasoning processes that more closely approximate human problem-solving strategies when addressing abstract reasoning challenges.

7 Data Augmentation

Effective training of Large Language Models (LLMs) to develop abstract reasoning capabilities for Abstract Reasoning Corpus (ARC) tasks requires addressing the inherent limitations of the original dataset. The standard ARC benchmark provides only 1,000 tasks, which presents a significant challenge: models trained on such a limited dataset are prone to memorization rather than developing generalizable reasoning abilities. Without proper augmentation, a model might simply learn to retrieve the appropriate output for a given task based on specific input patterns rather than understanding the underlying logical principles.

7.1 Analysis of Original Dataset Characteristics

To establish a foundation for our augmentation strategy, we first examined the statistical properties of the original ARC training dataset. Figure 7 illustrates the distribution of training and test examples across the task corpus.

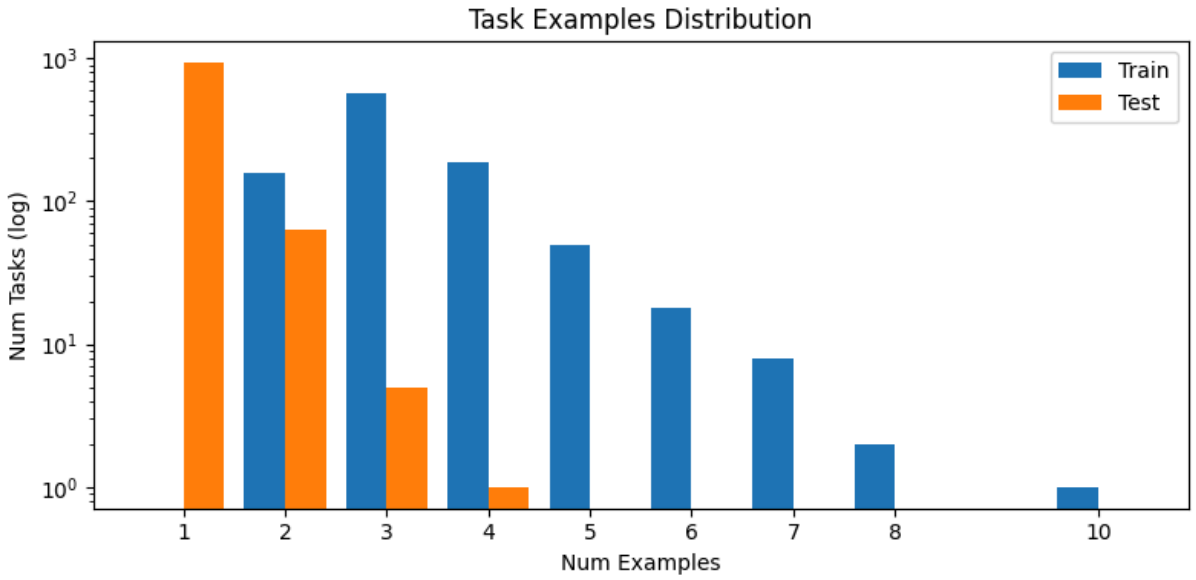

Figure 7 — Distribution of ARC tasks based on the number of train and test examples per task. The Y-axis uses a logarithmic scale. [4]

The distribution analysis reveals several statistical outliers that deviate from the normal distribution pattern. Figure 8 presents two representative examples of these outliers.

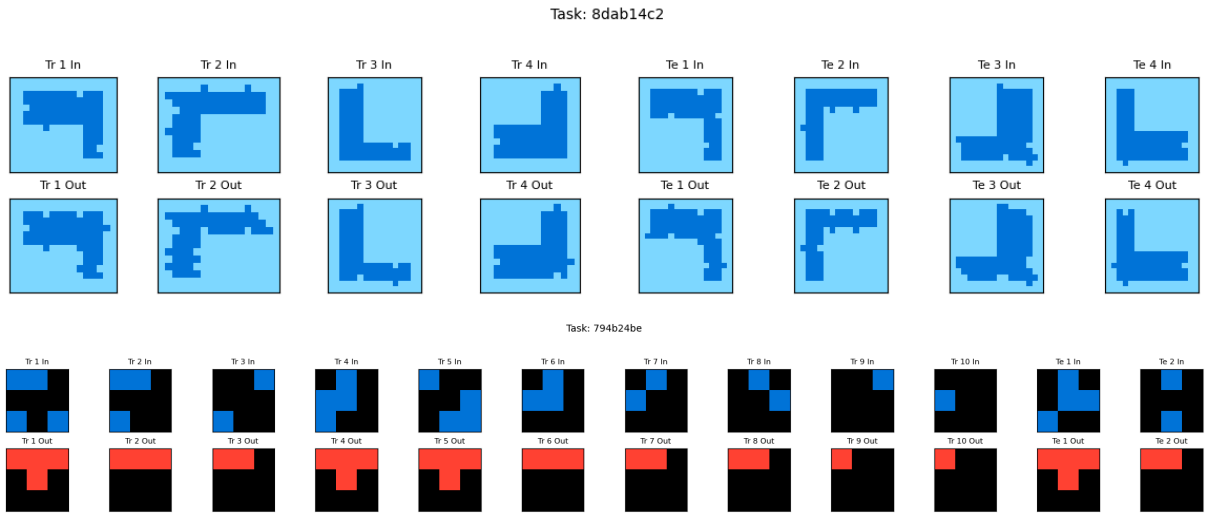

Figure 8 — Task 8dab14c2 with 4 test inputs and 794b24be with 10 train inputs

7.2 Geometric Transformation Techniques

Building upon methodologies established in prior research [31], we implemented systematic geometric transformations to preserve task structure while expanding the dataset. Our approach leverages the fact that only eight unique operations can be performed without introducing redundancy:

1. Identity (original orientation)
2. 90° rotation
3. 180° rotation
4. 270° rotation
5. Horizontal mirroring
6. Horizontal mirroring + 90° rotation
7. Horizontal mirroring + 180° rotation
8. Horizontal mirroring + 270° rotation

Notably, vertical mirroring was excluded as it produces outcomes identical to the combination of horizontal mirroring and 180° rotation, which would create duplicates in the augmented dataset.

The initial rotation transformations expanded the dataset from 1,000 to 4,000 tasks. Subsequent application of horizontal mirroring further increased the corpus to 8,000 tasks.

7.3 Advanced Augmentation Strategies

To further diversify the dataset while maintaining task integrity, we implemented additional transformations designed to challenge the model's pattern recognition capabilities without altering the fundamental task logic.

7.3.1 Boundary Padding

We introduced a stochastic padding mechanism that adds uniform boundary elements around task grids. This transformation was applied independently to inputs and outputs with a 50% probability for each, resulting in four possible outcomes for any given task (no padding, input padding only, output padding only, or both input and output padding). This approach generated approximately 6,000 additional tasks, as the combined probability of applying padding to at least one component is 75% per task.

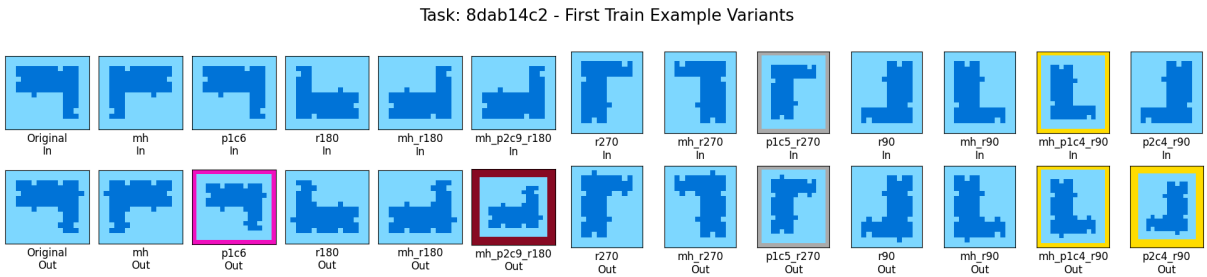

Figure 9 — Visualization of augmentations applied to the first training example (input/output pair) from ARC task 8dab14c2. Each column represents a different transformation (Original, Horizontal Mirror (mh), Padding (pXcZ), Rotation (rX), or a combination), applied to both the input grid (top row) and the output grid (bottom row). [4]

This phase of augmentation yielded approximately 14,000 tasks. While recent research suggests that even modest datasets can effectively guide LLM adaptation to novel behaviors [38]—with OpenAI reporting successful fine-tuning with as few as 10 tasks [39]—we implemented several additional augmentation techniques to enhance dataset diversity without introducing statistical biases.

7.4 Structural Modifications and Randomization

7.4.1 Test Pair Isolation

To facilitate single-task inference, we restructured tasks containing multiple test pairs. For each task with n test pairs, we generated n separate tasks, each preserving the original training examples but containing only a single test example. This restructuring ensures compatibility with the LLM's inference paradigm, which processes one test case at a time.

7.4.2 Task Duplication and Color Permutation

We duplicated the entire task corpus and applied distinct color transformations to each copy. This process involved creating a random color-to-color mapping for each task and applying it uniformly across all elements within that task. Unlike previous approaches, we deliberately included background colors in our mapping strategy to encourage the model to develop a more abstract concept of background rather than consistently associating it with a specific color value (typically 0).

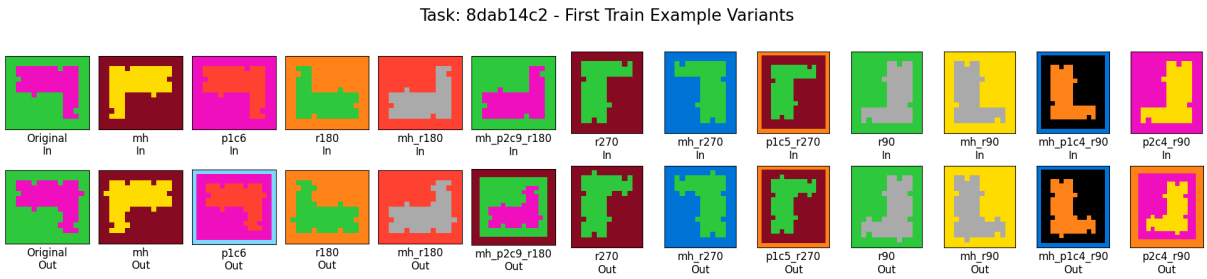

Figure 10 — Visualization of augmentations and color shuffle applied to the first training example (input/output pair) from ARC task 8dab14c2. [4]

7.4.3 Training Example Permutation

As a final augmentation step, we randomly shuffled the order of training examples within each task. This transformation does not alter the logical structure of the tasks, as the model processes all training data during inference regardless of sequence. However, it introduces additional variation that discourages memorization by creating superficially different presentations of identical logical problems.

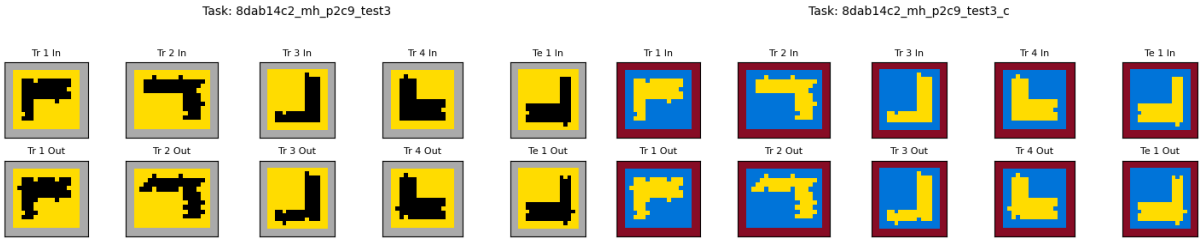

Figure 11 — Task 8dab14c2: Two copies of the same task with shuffled colors

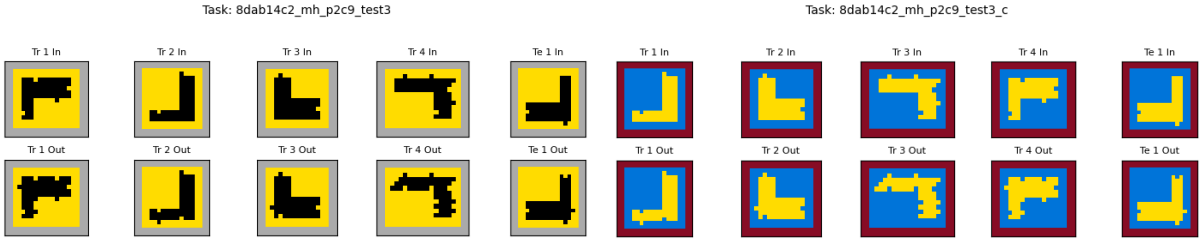

Figure 12 — Task 8dab14c2: Two copies of the same task with shuffled train order and colors

7.5 Summary of Augmentation Process

Our comprehensive augmentation methodology expanded the original 1,000-task dataset to approximately 28,000 tasks through a systematic application of geometric transformations (rotations and reflections), boundary modifications (padding), color permutations, and structural reorganizations (test pair isolation and training example reordering). The augmented dataset was formatted as JSONL for subsequent model training.

8 Prompt Structure Optimization

8.1 Prompt Engineering Context and Challenges

The optimization of prompt structures for abstract reasoning tasks necessitates addressing multiple interdependent variables that significantly impact model performance. Our systematic analysis of current literature on few-shot prompting revealed factors affecting reasoning capabilities within the Abstract Reasoning Corpus (ARC) task domain.

Empirical research quantified the substantial impact of example sequencing, demonstrating performance variance from 54% to 93% on sentiment analysis benchmarks based solely on permutations of identical training examples. [40] This finding carries significant implications for abstract reasoning tasks where pattern recognition is highly context-dependent. Further investigations have documented a recency bias phenomenon in large language models (LLMs), whereby models assign disproportionate weight to examples appearing later in the sequence, potentially compromising generalization capabilities. [41]

Contrary to conventional assumptions regarding few-shot learning efficacy, recent findings from experiments with test-time compute models indicate performance degradation. Researchers by OpenAI and Microsoft [42] observed statistically significant decreases in task performance when applying few-shot prompting to the o1 model architecture. These results align with independent observations by DeepSeek regarding their test-time compute model DeepSeek-R1, suggesting a fundamental limitation in current few-shot learning paradigms for certain model architectures. [24]

8.2 Prompt Structure Development

Our approach to prompt optimization established a framework with three primary objectives:

1. Implementation of explicit demarcation between examples using consistent syntactic indicators (e.g., “#Example 1”)
2. Standardization of delimiter systems to enhance input parsing reliability
3. Transformation of complex data structures into formats optimized for model processing

These structural imperatives were derived from theoretical considerations regarding token-level processing in transformer-based architectures and subsequently validated through empirical testing.

8.3 Tokenization Analysis for Qwen2.5-3B Model

A critical component of our research involved tokenization analysis of the Qwen2.5-3B model when processing numerical grid representations. Given the central importance of grid-based pattern recognition in ARC tasks, this investigation was essential for establishing an empirical foundation for subsequent optimization strategies.

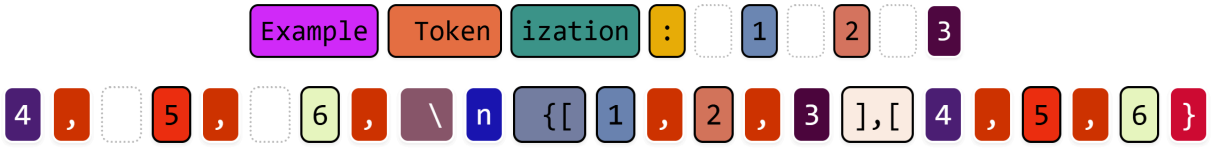

Figure 13 — Token Visualisation of Different Strings [5]

Our analysis revealed a distinctive tokenization pattern wherein Qwen2.5-3B encodes individual numerals as discrete tokens, contrasting with the encoding mechanisms employed in GPT-3 model architectures. This tokenization characteristic aligns with research documenting enhanced mathematical processing capabilities through appropriate numerical tokenization strategies. [28], [29], [30] For ARC tasks specifically, this property facilitates precise numeric pattern recognition — a capability essential for abstract reasoning functions.

To establish tokenization consistency across varying syntactic contexts, we conducted experiments examining numeral tokenization in the absence of delimiting characters:

Figure 14 — Token Visualisation of Different Strings [5]

The experimental results confirmed consistent preservation of individual token status for numerals regardless of delimiter presence, suggesting potential optimization opportunities for grid representation efficiency.

8.4 Array Representation Optimization

Building upon our tokenization findings, we hypothesized that delimiter-free representations could substantially reduce computational overhead while maintaining structural integrity. We designed and implemented a compression that transformed standard JSON array notation into a more efficient string format:

```
[                                  
[1, 2, 3],                  123\n
[4, 5, 6],        ⟶        456\n
[7, 8, 9]                   789
]
```

Listing 4 — Array representation (left) converted to space-efficient string format (right) for optimal tokenization in Qwen2.5-3B model. This transformation reduces token count while preserving grid structure for ARC tasks.

Comparative tokenization analysis between these representational formats provided quantitative validation of the efficiency improvements:

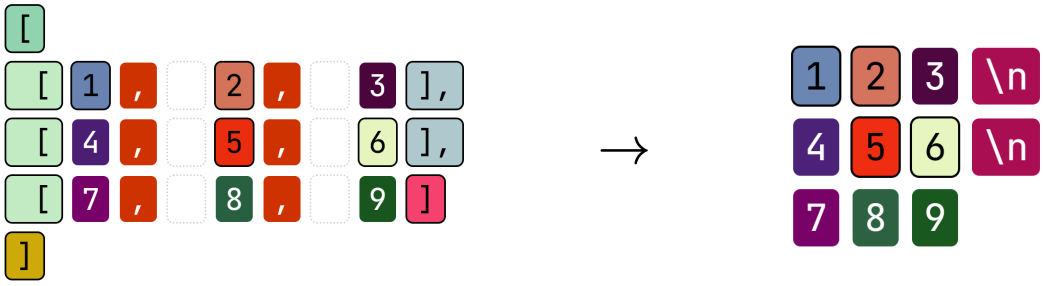

Figure 15 — Tokenization visualization comparing nested array representation (left) with compressed string format (right). Individual tokens are color-coded, demonstrating how the string format reduces token count while preserving grid structure for ARC tasks. [5]

This optimization approach yielded a 62% reduction in token count (from 29 to 11 tokens) in our experimental implementation while preserving all structural information necessary for pattern recognition—a significant efficiency enhancement with potential implications for computational resource utilization in the subsequent training runs.

8.5 Model Output Format Preference Analysis

To assess the practical viability of implementing delimiter-free representations, we evaluated the model's innate format preferences through systematic generative testing. We analyzed the Qwen2.5-3B model's representation tendencies by prompting it to generate matrix structures.

When tasked with generating a "matrix structure consisting of 3×3 numbers," the model demonstrated a statistically significant preference (70% of outputs across 30 independent trials) for traditional nested array formats:

```
[
[1, 2, 3],
[4, 5, 6],
```

```
[7, 8, 9]
]
```

When the experimental prompt was modified to request a “grid structure consisting of 6×6 numbers,” we observed a shift in representational preference toward space-delimited formats (72% of outputs):

```
1 2 3 4 5 6
7 8 9 10 11 12
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 30
31 32 33 34 35 36
```

A particularly notable finding from our experimental series was the complete absence of delimiter-free representations across 60 generation attempts. This empirical observation suggests that despite the computational advantages of such representations, their practical implementation would necessitate increased training to establish reliable processing capabilities. This could potentially divert computational resources from the primary abstract reasoning objectives. Furthermore, the model clearly showed a preference for delimiter-separated outputs.

8.6 Grid Structure Comprehension Analysis

Our experimental investigation further examined the impact of different grid representations on structural interpretation capabilities. Through controlled testing of space-delimited grid presentations without explicit structural indicators, we identified a significant limitation in the model’s ability to recognize spatial relationships:

User

Train Example 1:
Input:
3 1
1 4
Output:
[...]

Qwen2.5-3B

[...] **Train Example 1 Analysis:**
The input consists of two numbers, 3 and 1.
For each row in the output, we repeat one number twice (forming the pattern: <number><number>). [...]

Train Example 2 Analysis:
The input consists of two numbers, 9 and 0. [...]

This pattern of structural misinterpretation demonstrated remarkable consistency across multiple experimental iterations (15/15 outputs), revealing a fundamental constraint in the model's capacity to infer unified grid structures from space-delimited representations. Instead of recognizing the spatial relationships that define grid structures, the model interpreted each line as containing discrete, unrelated values—essentially decomposing the two-dimensional representation into a one-dimensional sequence and thereby losing the critical spatial context necessary for abstract pattern recognition.

8.7 Optimizing the Prompt Structure

Our investigations revealed a fundamental methodological tension between tokenization efficiency and structural comprehension integrity. While delimiter-free representations offered substantial computational advantages through token

reduction, they simultaneously introduced significant structural interpretation challenges that would require dedicated training to overcome.

Through iterative testing and systematic evaluation of this performance trade-off, we established that structural comprehension reliability must precede tokenization efficiency for effective abstract reasoning task performance. Therefore, our final prompt design employed nested arrays with explicit structural indicators while eliminating superfluous whitespace characters:

```
### Train Example 1:
Input:
[
[3,1],
[1,4],
]

Output:
[
[3,1,3,1,3,1],
[1,4,1,4,1,4],
[1,3,1,3,1,3],
[4,1,4,1,4,1],
[3,1,3,1,3,1],
[1,4,1,4,1,4],
]
```

```
 ### Test Input:
[
[6,5],
[9,3],
]
```

Listing 5 — Prompt structure left: {train}, right: {test}

This optimized format systematically addresses multiple performance constraints:

1. Explicit structural demarcation through standardized array notation, enhancing grid relationship recognition
2. Computational efficiency through whitespace elimination
3. Alignment with empirically validated model representation preferences, reducing cognitive dissonance during pattern identification and generation

Note

Despite identifying the nested array format with delimiters as optimal for model comprehension, computational limitations forced a compromise. The semantically optimal variant generated 22,000-token prompts, creating unacceptable overhead during training on our dual H200 GPU setup and significantly slowing backpropagation.

We implemented a hybrid format that preserved structural integrity while reducing computational demands by maintaining brackets for hierarchical structure but eliminating internal delimiters between numerical elements (e.g., `[[123],[456]]` instead of `[[1,2,3],[4,5,6]]`). This optimization substantially reduced token count while preserving essential spatial relationship processing capabilities.

8.8 System Prompt Implementation

Our investigation into prompt structure optimization extends to the fundamental mechanisms through which language models process sequential input during training procedures. While the structural elements of prompts establish semantic frameworks, their implementation within neural language model architectures requires token-level control mechanisms that directly interface with the model's generative processes. Language models operate fundamentally as conditional probability distribution functions that predict subsequent tokens based on the preceding context. When transitioning from API-mediated interactions to direct model training, we must engage with the underlying token-level architecture through control sequences:

```
<|im_start|>system\nYou will be provided with example inputs and outputs. Analyze the train examples. Your Goal is to find common Trasnformation pattern among those and apply the found patterns to the test input to create the Test Output.<|im_end|>\n<|im_start|>user\n {train} \n\n\n {test}\n\n Figure out how to create the Test Output. Use <think> </think> tags to reason about the problem. Return the final answer in <answer> </answer> tags as a nested list.<|im_end|>\n<|im_start|>assistant\nLet me solve this step by step.\n<think>
```

These control tokens (<|im_start|> and <|im_end|>) serve as attentional anchors within the model's representational space, establishing contextual boundaries that modulate next-token prediction dynamics. Unlike conventional interface abstractions, these tokens directly influence the attention mechanisms and hidden state transformations that govern the model's generative behavior. Each role designation (system, user, assistant) activates distinct parameter configurations encoded during the model's pre-training phase, effectively constraining the probability distribution toward role-appropriate output patterns.

The full JSONL dataset of train, test, and train_answer pairs employing this prompt structure can be found on Hugging Face [43].

9 Defining a Reward Function

Paramount to a successful training in RL is to have a reward function that actually reflects what we want to achieve in the RL training run. It's important to first of all achieve the foundational goals before heading to more advanced goals. In our case, one of the most foundational goals of the LLM is to produce algorithmically parsable output so we can even start to compare it to the actually desired result.

9.1 Insights from MORLAIF

The paper 'Multi-Objective Reinforcement Learning from AI Feedback' [44] (MORLAIF) provides valuable insights into the optimisation of AI evaluation systems by separating different evaluation aspects. These findings can be applied to the development of a two-stage evaluation structure that evaluates syntax and content agreement separately.

MORLAIF demonstrates that decomposing complex evaluation tasks into more specific subtasks leads to better overall results. Rather than using a single preference algorithm that covers all aspects of evaluation simultaneously, the paper shows clear advantages of developing separate models for different principles such as factuality, toxicity and comprehensibility.

This realisation can be applied directly to the evaluation of AI expenditure. Instead of using a single evaluation metric that covers all aspects, the separation into syntactic and content evaluation dimensions is logical and effective.

9.2 Application to Grid-based Output

If you apply this to our goal, you get two evaluation aspects: firstly, compliance with the required list format and the correct grid dimensions, and secondly, compliance of the content result with the requirement.

The advantages of such a separate evaluation of syntax and content are confirmed by the MORLAIF paper: 'By breaking down preference modelling into specific principles, the feedback collection and preference model training becomes a more straightforward and well-defined task, which we hypothesize will lead to improved preference model performance.' This hypothesis was confirmed in the experiments, with the specific models achieving significantly higher accuracy than individual models.

9.3 Scalarisation for Combined Evaluation

We also use scalarisation functions to combine the separate ratings. These functions offer flexible methods for combining the two evaluation dimensions (syntax and content) into an overall evaluation. By merging them, we can then utilise and evaluate the reward score more efficiently.

9.4 Dual-Faceted Score Management

The score management in our evaluation system is designed around the insight that separating evaluation into distinct facets — syntax and content — can lead to a more nuanced and effective reward mechanism for our RL training. Building on the MORLAIF-inspired architecture, we decompose the problem into two specific evaluations.

9.5 Evaluation Framework for Grid and Content Similarity

9.5.1 Syntax (Structural) Evaluation

The first component focuses on syntax evaluation, ensuring that LLM output adheres to the required grid format. The **evaluate_grid_similarity** function handles this assessment through several key steps:

1. **Input Parsing** : Both the expected answer and solution are parsed into Python objects using a safe literal evaluation method. If parsing fails or the structure doesn't match the expected format (a list of lists), a minimal baseline score (0.1) is assigned.

2. **Structural Comparison**: The function measures similarity by first comparing the number of rows (calculating a row similarity ratio), then iterating through common rows to compute column-level similarity. For each row, it calculates the ratio between the lengths of corresponding rows.

3. **Score Mapping**: An exponential transformation is applied to the structural score:

$$\text{score} = 0.1 + 0.8 \cdot \frac{e^{k \cdot \text{ structural\_score}} - 1}{e^k - 1} \tag{2}$$

Where structural_score is a combined metric reflecting both row and column similarity of the grids, and k=4 was selected after extensive testing.
This k-value choice was deliberate - testing various values revealed that k=4 effectively penalizes outputs that fail to generate proper list/grid formats while providing appropriate rewards once the model produces correct list structures. This value also emphasizes the importance of achieving the right matrix dimensions.

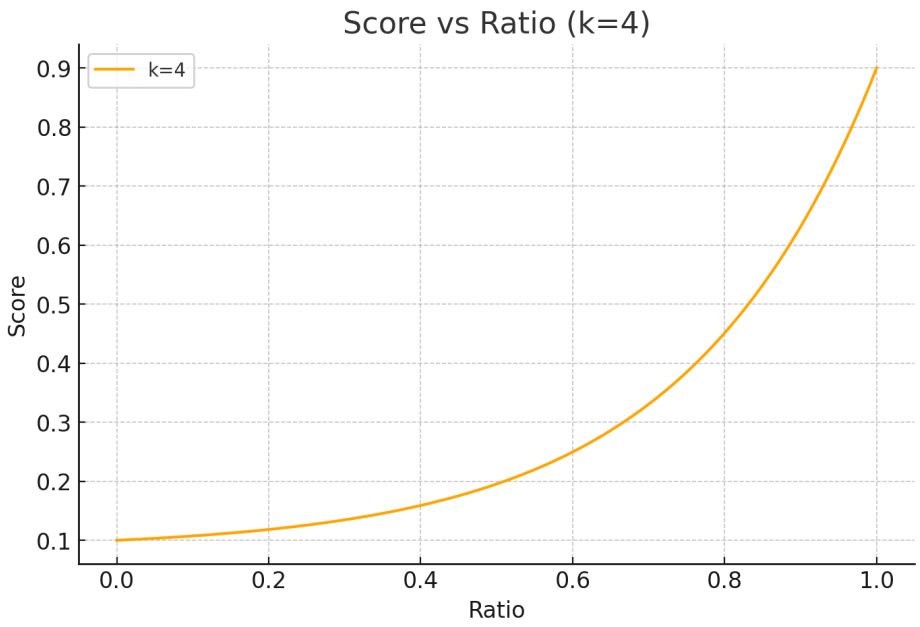

Figure 16 — Exponential Score Mapping for Grid Structure Evaluation (k = 4)

9.5.2 Content (Semantic) Evaluation

The second component evaluates content quality through the compare_answers function:

1. **Direct and Fallback Parsing:** The function attempts to parse both expected and actual answers. With successful parsing, it performs direct comparison.

2. **Flattening Technique:** When exact matches aren't achieved, the function "flattens" nested lists to create a simplified comparison baseline and uses SequenceMatcher to calculate content alignment.

3. **Regex Fallback:** For solutions that resist direct parsing due to formatting issues, a regex-based mechanism extracts numbers and computes a similarity ratio between expected and provided values.

4. **Similarity Scoring:** The content similarity is transformed using:

$$\text{score} = 0.1 + 0.8 \cdot \frac{e^{k \cdot \text{ ratio}} - 1}{e^{-k} - 1} \tag{3}$$

   Where ratio measures the similarity between expected and actual answers, and k=7 controls the sensitivity of the exponential scaling.
   This higher k-value (k=7) was selected after testing various parameters. It prioritizes achieving those final percentage points of accuracy, as testing revealed the model already produces answers with high correlation to correct responses based solely on the examples provided.

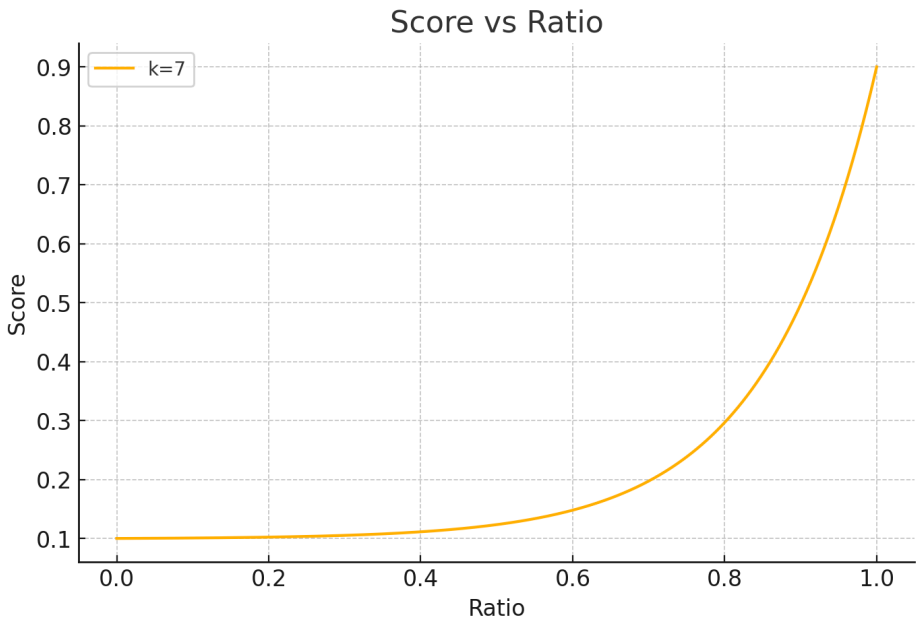

Figure 17 — Exponential Score Mapping for Content Similarity Evaluation (k = 7)

This dual evaluation approach — assessing both structure and content — provides a comprehensive framework for measuring how closely LLM-generated grid outputs match expected answers, with graduated scoring that rewards incremental improvements while maintaining high standards for complete accuracy.

9.6 Reward Range Design Principles

The decision to confine our reward score within the range of 0.1 to 0.9 for partial matches — with 0.1 representing the worst-case outcome and 0.9 representing nearly optimal performance, and a perfect score of 1.0 reserved only for an exact match — is driven by two key concepts: normalization/scaling and reward clipping. [45]

9.7 Normalization and Scaling:

In reinforcement learning, especially within complex multi-objective frameworks like those discussed in MORLAIF, it is crucial to ensure that the reward signal remains within a manageable and meaningful range. By normalizing our partial reward scores to lie between 0.1 and 0.9, we ensure that the signal is neither too weak nor excessively large. This scaling prevents issues such as reward saturation,

where excessively high rewards may lead the model to overestimate the value of its predictions or cause instability during training. Inspired by approaches in modern RL systems — as also observed in studies on deep reinforcement learning for congestion control — the normalized range serves as a consistent baseline that fosters smoother gradient updates and more stable policy learning. Essentially, even when the model produces an output that is only partially correct, it still receives a non-zero reward (at least 0.1), which guarantees that the learning signal persists throughout the training process.

9.8 Reward Clipping:

Reward clipping is another important mechanism that helps control the variance and stability of the training process. By capping partial rewards at 0.9, we deliberately prevent the model from receiving a near-perfect reward for outputs that are still not completely accurate. This technique mirrors the clipping practices observed in advanced RL algorithms like those implemented in PPO and as demonstrated in the ablation studies for DRL-based congestion control systems. In those studies, omitting clipping often led to erratic policy updates and convergence issues. Clipping ensures that while partial correctness is rewarded, it never reaches the level of an exact match. This careful capping of the reward avoids excessive optimism in the policy updates, ensuring that only the completely correct outputs garner the full reward of 1.0.

9.9 Putting It All Together:

By setting the reward range from 0.1 to 0.9 for any output that is not an exact match, we integrate both normalization/scaling and clipping methods into our reward design. The scaling ensures that the model's learning dynamics remain stable and that the gradients are suitably informative even when outputs are only partially correct. Meanwhile, clipping keeps the reward signal bounded, which helps to prevent overshooting during policy updates and maintains a clear distinction between near-perfect performance (0.9) and absolute correctness (1.0).

This approach, inspired by the insights from MORLAIF and corroborated by empirical findings in related reinforcement learning ablation studies, ultimately leads to a more robust and efficient training process — one where the reward function accurately reflects progress toward both the foundational goal of producing algorithmically parsable output and the advanced goal of content accuracy.

9.10 Final Score Calculation

Finally, the main function evaluate_score combines both dimensions using scalarisation. It applies a scalarisation function that immediately returns a perfect score (1.0) if the solution exactly matches the expected answer. Otherwise, it computes the final score as a weighted sum of the syntax score and the content score. By default, both aspects are weighted equally (0.5 each), but these weights can be adjusted to better reflect their relative importance in different contexts.

9.11 Benefits of the Bifurcated Approach

By managing the score with this bifurcated approach, our method encourages foundational correctness by prioritizing the creation of algorithmically parsable output, establishing a solid structural foundation before delving into more sophisticated content verification. It also improves accuracy by isolating the evaluation into more specific tasks, helping in pinpointing and rewarding improvements in both structure and semantic quality, akin to the benefits observed in the MORLAIF study. Additionally, it provides flexibility through the use of scalarisation functions, allowing us to finely tune the overall reward, so that improvements in either syntax or content result in a corresponding improvement in the final score, leading to more efficient and targeted model training. This careful separation and subsequent recombination of evaluation aspects not only mirrors the empirical findings of MORLAIF but also ensures that our reward function accurately reflects the foundational and advanced goals necessary for successful reinforcement learning training.

10 Experimental Setup

For our project, we initially attempted to set up the development environment on our Windows 10 machines, following the steps outlined in Project TinyZero [37]. We created a Conda environment using Python 3.9, installed CUDA 11.8, and configured the appropriate environment variables (including CUDA_HOME, CUDA_PATH, and adding CUDA's include and bin directories to the system PATH).

To install PyTorch with CUDA support, we specified the exact version using the command: pip install torch==2.0.0 –index-url https://download.pytorch.org/whl/cu118 and subsequently installed matching torchvision and other dependencies.

We also installed NVIDIA Nsight Visual Studio Edition and set up additional components (including Visual Studio Build Tools, C++ compilers, and various CUDA profiling tools). Despite carefully following these steps, we encountered persistent errors during the installation of the flash-attention package. The issues primarily stemmed from compatibility problems between NVIDIA's nvcc compiler and our installed Visual Studio version. We made several attempts to resolve these issues —including reinstalling CUDA, adjusting environment variables, trying different Visual Studio installations, and even using nvcc flags to allow unsupported compilers —but the build errors continued.

Due to these ongoing challenges with the Windows setup, we ultimately switched to an on-demand cloud instance running Linux, where the setup process proved significantly smoother.
On the cloud instance we automated the deployment with a script to simplify future setups—this proved especially valuable since the entire environment is deleted when the instance is decommissioned.

After successfully running the project on Linux, we encountered challenges related to insufficient GPU memory (VRAM) and optimization issues during training. Memory errors initially emerged due to high RAM requirements from long model

sequences and large batch sizes. To address these issues, we conducted systematic hyperparameter tuning and hardware evaluations. Below is a detailed summary of our optimization strategy:

10.1 Hyperparameter Tuning Overview

10.1.1 Sequence Length Adjustments

- **Parameter Changed:** data.max_prompt_length
- **From**: 22000
- **To**: 11300
- **Purpose**: Reducing the maximum token length for model responses significantly decreased the memory required for storing activations and gradients during both forward and backward passes, substantially lowering the total memory footprint.

10.1.2 GPU Memory Utilization Reduction

- **Parameter Changed:** actor_rollout_ref.rollout.gpu_memory_utilization
- **From**: 0.4
- **To**: 0.05
- **Purpose**: This parameter controls the fraction of GPU memory reserved for vLLM's key-value (KV) cache. A large reservation can starve other training operations. Reducing it freed up necessary VRAM for the model's activations and parameters.

10.1.3 Batch Size Minimization

- **Parameter Changed:** ppo_micro_batch_size (for both Actor and Critic models)
- From: 8
- To: 1
- Purpose: Decreasing the micro-batch size reduced the number of samples processed per forward/backward pass, thereby lowering peak memory con-

sumption for gradients and activations. Although this increased the number of iterations, it was essential for avoiding out-of-memory (OOM) errors.

10.1.4 Enabling Gradient Checkpointing

- **Parameters Changed:**
  - actor_rollout_ref.model.enable_gradient_checkpointing (Actor)
  - critic.model.enable_gradient_checkpointing (Critic)
- **From**: Disabled (False)
- **To**: Enabled (True)
- **Purpose**: Gradient checkpointing trades increased computation time for reduced memory usage by discarding intermediate activations during the forward pass and recomputing them during the backward pass. This optimization was key to lowering VRAM requirements during training.

10.1.5 FSDP (Fully Sharded Data Parallel) Offloading

- **Parameters Changed:**
  - actor_rollout_ref.actor.fsdp_config.grad_offload
  - critic.model.fsdp_config.grad_offload
  - actor_rollout_ref.actor.fsdp_config.optimizer_offload
  - critic.model.fsdp_config.optimizer_offload
- **From:** Disabled (False)
- **To:** Enabled (True)
- **Purpose:** Enabling these settings offloads gradients and optimizer states (like momentum and variance) from GPU to CPU memory. This redistribution relieved GPU memory pressure, although it introduced additional overhead from CPU–GPU data transfers.

10.2 Hardware Evaluations and Final Deployment

In parallel with our hyperparameter optimizations, we evaluated various GPU models to address hardware constraints. We tested several GPUs including the NVIDIA GeForce RTX 4090, A6000, and H100 models. Ultimately, by leveraging two H200 GPUs alongside our optimized hyperparameters, we achieved a stable training process.

10.3 Conclusion

By transitioning from Windows to a Linux-based on-demand cloud instance, automating our deployment process, and methodically tuning hyperparameters, we successfully resolved multiple memory and optimization challenges. The key adjustments—reducing sequence length and batch size, enabling gradient checkpointing, and implementing FSDP offloading—allowed us to work within our VRAM constraints and successfully run the TinyZero project. The final tuning configurations, combined with more powerful H200 GPUs, provided the necessary stability to complete the training process. This experience helped us realize that our Windows PC lacked the hardware capabilities required to run these training workloads effectively.

11 Training the Model

After preparing the training data, establishing the reward function, and tuning the hyperparameters, we proceeded with model training using dual H200SMX5 GPUs with a combined 282GB VRAM capacity. Our initial experiment employed a 3-billion-parameter model. However, we did not observe the desired emergent reasoning behavior. Instead, the model primarily optimized for the critic score by exploiting weaknesses in our reward function. With our initial configuration allocating 30% for structural accuracy and 70% for content quality, the model discovered it could easily satisfy the structural requirements by generating correctly formatted output using the prescribed brackets (<output></output>) and nested array structure. This optimization strategy resulted in a fixed, ineffective thinking pattern—essentially producing formulaic, non-informative responses as illustrated in Figure 18.

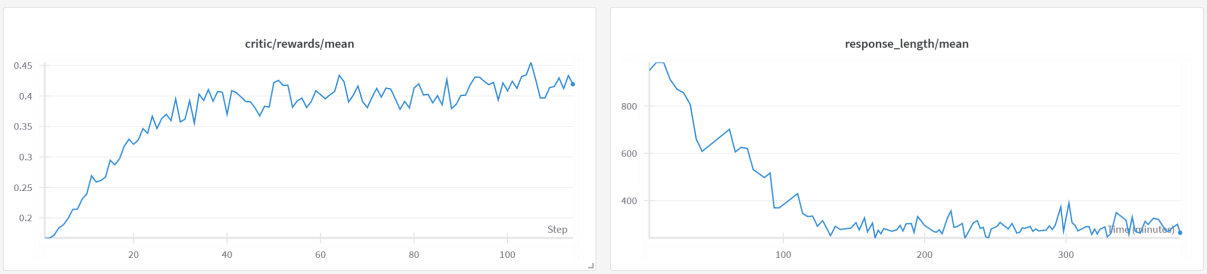

Figure 18 — Mean Critic Rewards Left and Models Response Length right for the first training run with a 3B model https://wandb.ai/lukhausen-dhbw/TinyZero/runs/vps13688?nw=nwuserlukhausen

```
<think>
Let me solve this step by step. 1. I'll compare the input and output for each example. 2. I'll look for common patterns in the number changes. 3. I'll try to find the transformation pattern. 4. I'll apply that pattern to the test input.
</think>
```

Listing 6 — Example of Local Minimum thinking pattern. This pattern was present in all outputs of the model

After terminating this initial training run, we considered two potential explanations: either our reward function lacked proper balance, or the model's capacity (3B parameters) was insufficient to develop the complex reasoning capabilities required for ARC tasks. This limitation of smaller models to develop sophisticated reasoning capabilities aligns with observations documented by Jian Pan in the TinyZero project [37], [46]. In our case, the reward structure imposed a minimum score of 0.1 for structural compliance, with a maximum potential structural reward of 0.3. Combined with the minimum content reward of 0.1, this created a performance ceiling of approximately 0.4, which is evident in the critic reward plateau shown in Figure 18. The model failed to discover strategies for improving content quality beyond this threshold.

To test these hypotheses, we extracted a checkpoint from the initial model and modified the reward distribution to 0.1 for structure and 0.9 for content, thereby significantly increasing the incentive for content improvement. After running this modified configuration for approximately three hours (80 steps), we observed no significant performance improvements—the critic reward remained stagnant, and response length stayed consistently flat.

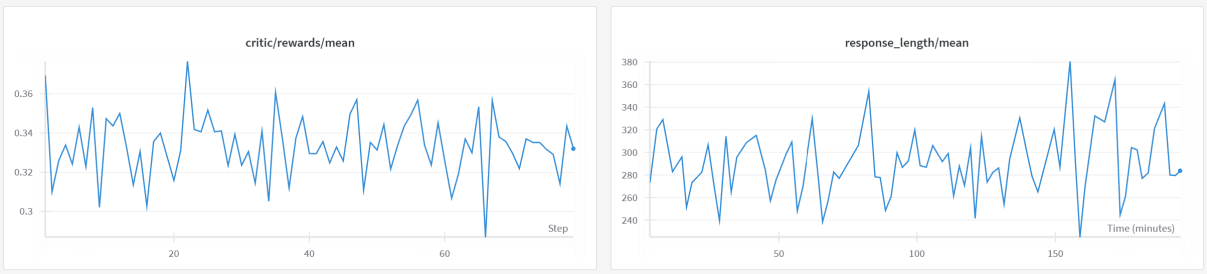

Figure 19 — Stagnant Critic response and stagnant Response length https://wandb.ai/lukhausen-dhbw/TinyZero/runs/tbo3orw4?nw=nwuserlukhausen

Based on these results, we concluded that the 3-billion-parameter model lacked sufficient capacity to develop the reasoning capabilities required for ARC tasks. We subsequently scaled up to a 7-billion-parameter model and repeated the experimental process.

This larger model was trained for approximately 450 minutes (7.5 hours). Despite prior research by Jian Pan suggesting that models sometimes experience delayed emergence of reasoning capabilities, we observed no improvements in response quality or reward metrics throughout this extended training period, as illustrated in Figure 20.

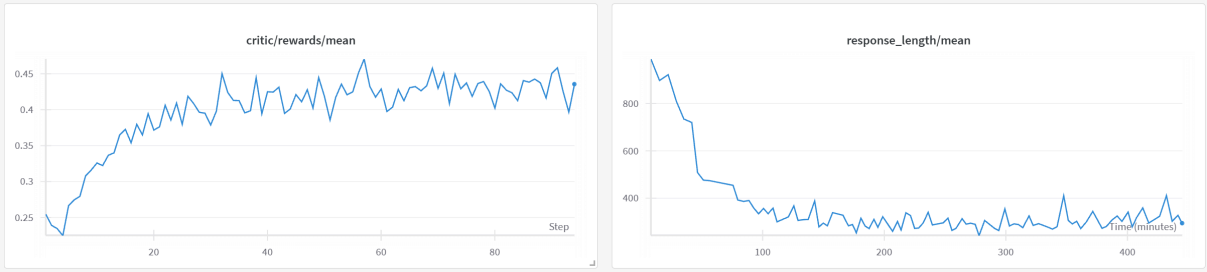

Figure 20 — No significant Changes in the behavior of the model https://wandb.ai/lukhausen-dhbw/TinyZero/runs/tbo3orw4?nw=nwuserlukhausen

The 7B model exhibited the same optimization pattern as the 3B variant—focusing exclusively on structural compliance while failing to develop meaningful reasoning capabilities. The outputs continued to display identical, formulaic thinking patterns, suggesting no substantive improvement in reasoning. We hypothesized this failure could stem from either an inherent limitation in the model's capacity to learn the complex ARC tasks or suboptimal reward function design that encour-

aged reward hacking rather than genuine reasoning. In previous runs, we had allocated 30% of the reward to structural compliance and 70% to content quality.

We subsequently adjusted the reward distribution to 10% for structure and 90% for content, further emphasizing content quality. This configuration ran for approximately 220 minutes (3.5 hours) and completed 43 training steps. However, the reward curve maintained its logarithmic shape without improvement, and response length continued to decrease.

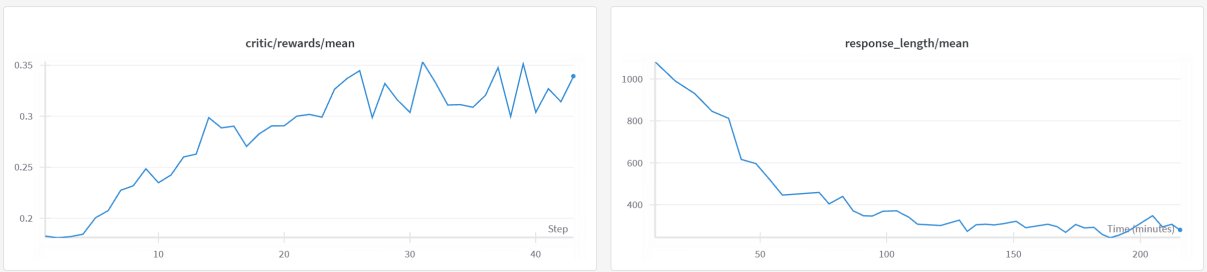

Figure 21 — Logarithmic curve even after adjusting the reward score https://wandb.ai/lukhausen-dhbw/TinyZero/runs/acmyhkji?nw=nwuserlukhausen

To address this persistent local minimum, we implemented a novel incentive structure designed to encourage more extensive reasoning. We modified the reward function to explicitly reward the length of content within the thinking tags, with the goal of promoting more elaborate reasoning sequences. Using the checkpoint from the previous 7B model, we implemented a balanced reward distribution: 10% for structural correctness, 40% for thinking output length, and 50% for content quality.

This approach aimed to first establish longer reasoning chains before gradually shifting focus toward reasoning quality and content accuracy. We deployed this length-optimized reward function starting from step 30 of the previous model checkpoint, which had already learned the correct output structure but struggled with coherent reasoning. Within just 10 steps, the model demonstrated notable improvements in reasoning length while maintaining correct output formatting.

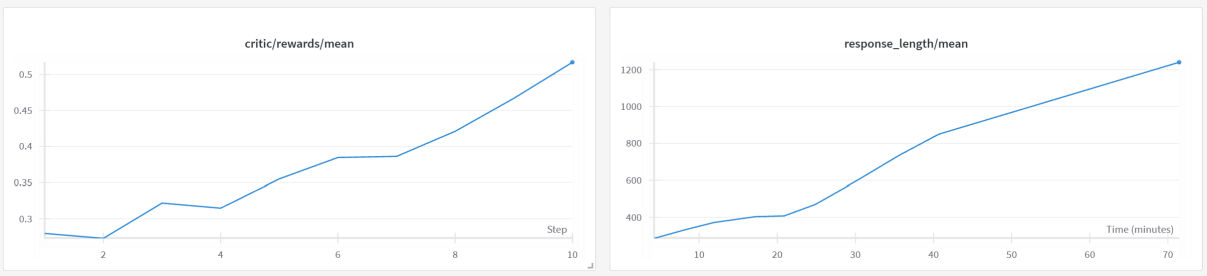

Figure 22 — Ten steps of boosting the thinking length of the model. https://wandb.ai/lukhausen-dhbw/TinyZero/runs/oq6kjqvf?nw=nwuserlukhausen

After just 10 steps with the modified reward function, we observed significant improvements in the reasoning patterns produced by the model:

```
<think>
Let me solve this step by step. 1. I observe that in each train example, the input is a 2-dimensional list with varying number of rows and two elements in each row. 2. The output pattern is that each row of the input is duplicated one or more times to create the corresponding row of the output. 3. The number of times a row is duplicated in the output seems to be related to the number of digits in the elements of the row - the more digits, the more times it seems to be duplicated. 4. However, the row at the top of the input ("3999" in this example) is never duplicated. Applying these observations to the test input, I see that:
- The top row ("3999") should not be duplicated in the output.
- The second row ("9355") has four digits, so it should be duplicated four times in the output.
- The third row ("9569") has four digits, so it should be duplicated four times in the output.
- The fourth row ("9964") has four digits, so it should be duplicated four times in the output.
- The bottom row ("9994") has four digits, so it should be duplicated four times in the output.
</think>
```

Listing 7 — Example of increased reasoning length through modified reward function

After successfully increasing the reasoning elaboration, we transitioned to a purely content-based reward function to encourage meaningful reasoning rather than merely verbose output.

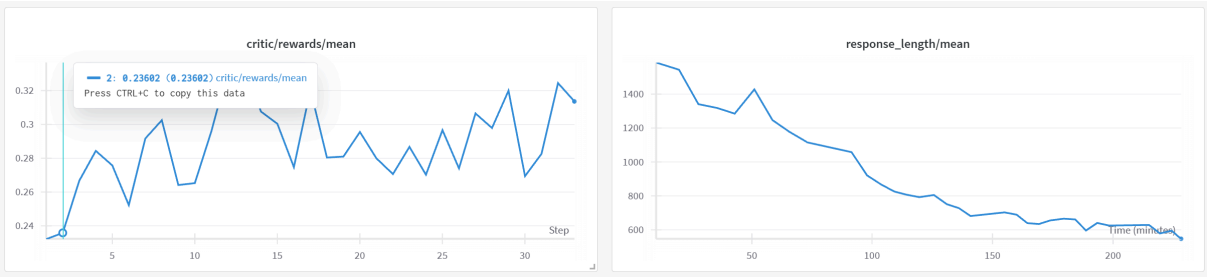

Figure 23 — Running the purely content based reward. https://wandb.ai/lukhausen-dhbw/TinyZero/runs/vbfszi8j?nw=nwuserlukhausen

Despite this intervention, we observed that output length decreased again, and even after more than 30 additional training steps, the model failed to overcome its fundamental limitations in reasoning capabilities.

11.1 Training on an Easier Dataset

We hypothesized that the 7B parameter scale might be insufficient for developing the sophisticated reasoning capabilities required for complex ARC tasks. Due to budget constraints, we opted not to scale to a 14B model, as the required resources (minimum 4× H200 GPUs) were prohibitively expensive for on-demand cloud GPU instances. Instead, we strategically reduced task difficulty.

Having focused exclusively on the ARC-AGI-2 dataset, we created a new dataset incorporating easier variants of similar tasks. This dataset combined the ARC-AGI-1 training set with the Concept Arc Dataset, which includes various simplified tasks [34]. Our strategy was to first determine if the model could develop emergent reasoning capabilities on simpler problems before gradually increasing task complexity. This approach is supported by research demonstrating that training on simpler examples can significantly enhance reasoning and generalization capabilities [47].

After creating this dataset[9] we initiated training with the 7B model.

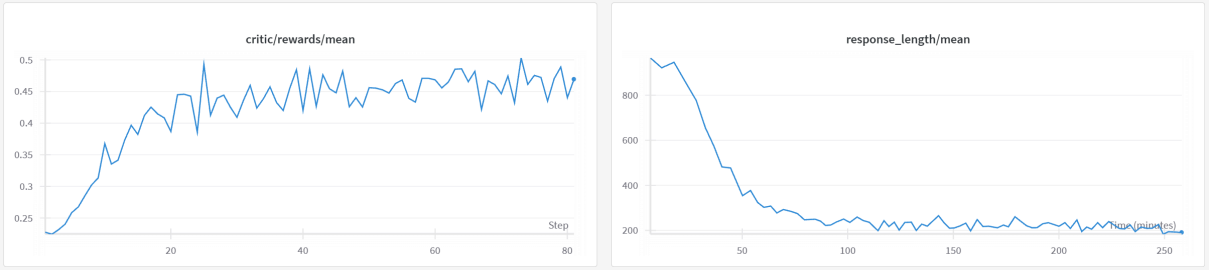

Figure 24 — Reward and response length for the easy dataset. https://wandb.ai/lukhausen-dhbw/TinyZero/runs/vps13688?nw=nwuserlukhausen

Rather than limiting training to 40 steps as in previous runs, we allowed the model to converge fully, selecting the checkpoint at step 70 which achieved the highest reward score. The reward distribution for this run maintained our 10% structure and 90% content allocation.

Despite the simplified dataset, the model still failed to develop emergent reasoning capabilities. We attempted to kickstart reasoning development using the same approach that previously succeeded with the 7B model on the more complex dataset. The reasoning boost run implemented a reward distribution of 30% for output length, 60% for content correctness, and 10% for structural compliance.

The model quickly adapted to this reward function and began producing longer outputs. However, unlike our previous experience where extended outputs demonstrated meaningful task-specific reasoning, this time the model exploited the reward mechanism by generating verbose but uninformative content that bore little relevance to the tasks at hand:

[9]https://huggingface.co/datasets/Lukhausen/arc-agi-lepus-v1-easy

```
<think>
Let me solve this step by step.
1. I will carefully analyze the given train examples, focusing on the pattern
of transformation from the train input to the train output. I will look for
similarities in the transformation patterns among the examples, such as the
repetitive sequences and the specific elements that remain unchanged during the
transformation process.
2. I will pay attention to the recurring

[...]

40. I will reflect on the problem-solving process and identify potential areas for
improvement in understanding the significance and relevance of the transformation
patterns in relation to the overall problem. By considering the broader context
and implications of the transformation process, I can gain a more holistic
understanding of its impact and contribute to a more effective and meaningful
solution.
</think>
```

Listing 8 — Long, yet non-informational reasoning chain.

The model developed a meta-cognitive pattern of thinking about how to think, rather than applying reasoning to the specific task. Instead of engaging with the problem content, it generated increasingly lengthy pseudo-reasoning chains devoid of task-relevant information or insights.

Upon reverting to our standard reward configuration (10% for structure and 90% for content), the model persistently maintained its non-substantive reasoning patterns without demonstrating performance improvements. We attribute this phenomenon to premature convergence during initial training phases, which appears to have constrained the model's optimization trajectory to a suboptimal local minimum in the parameter space. This early fixation potentially impeded the model's capacity to explore more promising regions of the solution landscape, effectively crystallizing ineffective reasoning strategies that proved resistant to subsequent refinement efforts. This observation underscores a fundamental challenge in reinforcement learning for complex reasoning tasks: maintaining sufficient

exploration capabilities throughout the optimization process to avoid entrapment in suboptimal solution manifolds.

12 Benchmarking of the Trained Models

In this chapter, we outline the process and methodology for benchmarking our models, which have been refined using reinforcement learning techniques. The primary focus during the benchmarking phase is to evaluate the content of the generated answers—assessing their correctness and relevance—without the distraction of syntax evaluation, which was primarily handled during training.

12.1 Overview of the Benchmarking Process

After completing the training phase, where the reward function evaluated both output syntax and content, our benchmarking process isolates the content evaluation to ensure that the models generate accurate and meaningful responses. While syntax was necessary during training to guide the models toward well-structured output, the ultimate performance metric rests on the correctness and appropriateness of the content itself.

To achieve this, we developed a dedicated Python script that builds on the content evaluation mechanism used in our reward function. This script calculates two distinct performance scores:

1. **Content Reward Score:** This metric leverages the same evaluation function defined in the “Defining Reward Function” chapter. It specifically assesses the content quality of the output without considering syntax. The only difference is that we don’t use a linear function as in the actual reward function. Instead, we output the ratio value directly without incorporating it into an e-function. By using this score, we can directly compare the models and see more indeph learning improvments.
2. **Correctness Percentage:** Independently, the script checks each response to determine whether it is factually correct or incorrect. By tallying the number of correct responses and dividing by the total number of tests, we obtain a percentage score that reflects the overall accuracy of the model.

12.2 Methodology and Data Set

For benchmarking, we employ a separate data set that was specifically curated and adapted for our purposes. Our source for this test data is the ARC-AGI-2 [48] test set, which was provided as an initial collection of examples. Recognizing that direct application might not align perfectly with our unique requirements, we made several modifications. These adjustments ensured that the test set effectively challenges the models in areas crucial to our application domain.
Our methodology follows these steps:

1. **Test Set Adaptation:** The downloaded ARC-AGI-2 data was reformatted to facilitate easier processing within our evaluation pipeline. This adaptation process involved only structural changes to the data format, without altering the content or examples themselves. The reformatting allowed us to feed the test data directly to our models without requiring additional processing steps during the evaluation.
2. **Content Evaluation:** Using the Python script, each output generated by the models is examined against predetermined content criteria. The reward score derived from this evaluation provides insight into how well the model understands and correctly conveys the intended information.
3. **Correctness Verification:** In parallel to content scoring, our evaluation mechanism categorizes each response as either correct or incorrect. This binary classification is then used to compute a percentage of correct answers, offering a straightforward metric of accuracy.
4. **Score Aggregation:** The final output of the benchmarking process includes both the content reward score and the correctness percentage as seen in Listing 9 These combined metrics offer a comprehensive view of the model's performance, highlighting strengths in content quality and areas that may need further refinement.

```
arc-agi-lepus-v1-evaluation
Correct: 0
Wrong: 120
Score: 0.0
RewardScore: 18.30003843640773
```

Listing 9 — Example benchmark output for the arc-agi-lepus-v1-evaluation model

12.3 Conclusion

The benchmarking phase serves as a critical step in validating the effectiveness of our reinforcement learning approach. By isolating the evaluation of content from syntax and concentrating on the factual correctness of the output, we ensure that the models are not only well-trained in generating fluent language but also in delivering reliable and relevant information.

The dual metrics—a detailed content reward score and a clear percentage of correct responses—provide a robust framework for continuous evaluation and future improvements. This structured approach to benchmarking not only confirms the current state of our models but also lays the groundwork for ongoing optimizations in response to real-world challenges.

13 Limitations and Future Research Directions

Our investigation into emergent reasoning capabilities in language models for abstract reasoning tasks revealed several constraints and opportunities for enhancement. This chapter systematically examines methodological refinements that could potentially improve performance on the ARC benchmark and advance our understanding of emergent reasoning in LLMs.

13.1 Model Parameter Scaling

The most evident limitation of our current approach involves parameter scale constraints. Our experiments utilized models with parameter counts of 3 billion and 7 billion, both of which demonstrated limited capacity to develop the sophisticated reasoning patterns required for ARC tasks. Scaling to larger architectures with 14+ billion parameters could potentially determine whether the inherent reasoning capabilities required for spatial abstraction tasks emerge at specific parameter thresholds. Previous research suggests that certain cognitive capabilities in language models manifest only after reaching critical parameter densities, making this a promising direction for future investigation.

13.2 Advanced Inference Optimization Strategies

While our research focused primarily on model training methodology rather than inference optimization, implementing multi-perspective inference would likely enhance performance substantially. Previous studies have demonstrated that transformation-based inference techniques—particularly those involving geometric manipulations such as rotation and mirroring—can significantly improve performance on ARC tasks. Our training data incorporated these transformations, but we did not leverage them during inference.

A particularly promising approach would involve:

1. Generating multiple task perspectives through systematic geometric transformations
2. Processing each perspective independently through the model
3. Implementing statistical consensus mechanisms to aggregate outputs across perspectives
4. Deriving final predictions through probability-weighted pixel-level voting

This methodology would leverage the model's accumulated knowledge across different spatial orientations, potentially overcoming orientation-specific pattern recognition limitations.

13.3 Foundation Model Selection Optimization

Our research utilized base Qwen models as foundation architectures. However, initializing from models already fine-tuned for reasoning tasks could provide substantial performance advantages. Specifically, models such as NVIDIA's NeMo-Minitron series built on the LLaMA architecture have demonstrated enhanced reasoning capabilities that could serve as a more effective starting point for reinforcement learning.

The principal advantage of such pre-optimized foundation models lies in their established reasoning pathways, which our reinforcement learning approach could potentially enhance rather than develop from rudimentary capabilities. This hypothesis is supported by our observation that the model could independently develop reasoning strategies for ARC tasks without explicit instruction, particularly when incentivized through our thinking-reward mechanism.

13.4 Tool Integration and Computational Augmentation

A methodological enhancement with significant potential involves integrating programmatic tools within the model's reasoning framework. Implementing a pipeline that enables the model to generate and execute code during inference could substantially enhance analytical capabilities. Such a system would allow the model

to leverage mathematical libraries (e.g., NumPy, Pandas) to identify statistical patterns and correlations across examples.

This computational augmentation approach would shift the model's operation from pure reasoning to a hybrid system that combines language model capabilities with structured analytical tools. While this enhancement would likely require substantial additional computational resources, it presents a promising direction for overcoming the inherent limitations of pure-LLM approaches to abstract reasoning tasks.

13.5 Conclusions on Emergent Capabilities

Our experimental findings provide insights regarding emergent reasoning capabilities in language models. The results indicate that while reasoning models can be developed with relatively modest computational resources, the emergence of sophisticated reasoning patterns appears contingent upon foundational model intelligence. In scenarios where the base model lacks sufficient cognitive capacity, reasoning patterns do not spontaneously emerge through reinforcement learning alone.

However, our most significant observation came from the training run incorporating explicit rewards for reasoning length, where we observed the emergence of more sophisticated analytical patterns. This suggests that while complete reasoning capabilities may not emerge spontaneously, they can be methodically cultivated through targeted incentive mechanisms that guide the model toward more structured analytical approaches.

14 Conclusion

14.1 Summary of contributions

14.2 Recommendations for research extensions

14.3 Broader impact considerations

15 Chapter Authorship

References

[1] F. Chollet, “Abstraction and Reasoning Corpus for Artificial General Intelligence (ARC-AGI).” Accessed: Mar. 24, 2025. [Online]. Available: https://github.com/fchollet/ARC-AGI

[2] OpenAI, “Tokenizer.” Accessed: Apr. 08, 2025. [Online]. Available: https://platform.openai.com/tokenizer

[3] L. Marschhausen, “Lepus: run_benchmark_v1.ipynb.” Accessed: Apr. 08, 2025. [Online]. Available: https://github.com/Lukhausen/Lepus/blob/main/experimental/lukas/benchmarking/run_benchmark_v1.ipynb

[4] L. Marschhausen, “data_auguentation.ipynb.” Accessed: Apr. 11, 2025. [Online]. Available: https://github.com/Lukhausen/Lepus/blob/main/experimental/lukas/preprocessing/data_auguentation.ipynb

[5] L. Marschhausen, “token_visualisation.py.” Accessed: Apr. 12, 2025. [Online]. Available: https://github.com/Lukhausen/Lepus/blob/main/experimental/lukas/preprocessing/token_visualisation.py

[6] A. P. Foundation, “ARC-AGI Leaderboard.” Accessed: Apr. 08, 2025. [Online]. Available: https://arcprize.org/leaderboard

[7] Google, “Google AI Studio – New Chat.” Accessed: Apr. 13, 2025. [Online]. Available: https://aistudio.google.com/prompts/new_chat

[8] OpenAI, “OpenAI API Platform.” Accessed: Apr. 13, 2025. [Online]. Available: https://platform.openai.com/

[9] Anthropic, “Claude AI.” Accessed: Apr. 13, 2025. [Online]. Available: https://claude.ai/

[10] I. Grammarly, “Grammarly Editor.” Accessed: Apr. 13, 2025. [Online]. Available: https://app.grammarly.com/

[11] “Was ist eine Gedankenkette (Chain of Thought, CoT)?.” Accessed: Mar. 13, 2025. [Online]. Available: https://www.ibm.com/de-de/topics/chain-of-thoughts

[12] M. Larsen and D. Weßels, “Deeper Insights in KI-Sprachmodelle – mit Chain of Thought Prompting als Erfolgsfaktor?.” Accessed: Mar. 13, 2025. [Online]. Available: https://the-decoder.de/deeper-insights-fuer-ki-sprachmodelle-mit-chain-of-thought-prompting-als-erfolgsfaktor/

[13] K. Isenberg, “The Magic of Prolonged Thinking: Test-Time Compute | Part 2.” Accessed: Mar. 13, 2025. [Online]. Available: https://www.forwardfuture.ai/p/the-magic-of-prolonged-thinking-test-time-compute-part-2

[14] Y. Sun and others, “Test-Time Training (TTT): A New Approach to Sequence Modeling.” Accessed: Mar. 13, 2025. [Online]. Available: https://medium.com/thedeephub/test-time-training-ttt-a-new-approach-to-sequence-modeling-8baf1ea79ed7

[15] “What is Test Time Training.” Accessed: Mar. 13, 2025. [Online]. Available: https://nanonets.com/blog/what-is-test-time-training/

[16] “Was ist Hyperparameter-Einstellung?.” Accessed: Mar. 13, 2025. [Online]. Available: https://aws.amazon.com/de/what-is/hyperparameter-tuning/

[17] “What are Fine-tuning Hyperparameters and How to Set Them.” Accessed: Mar. 13, 2025. [Online]. Available: https://www.entrypointai.com/blog/fine-tuning-hyperparameters/

[18] J. Holdsworth and M. Scapicchio, “Was ist Deep Learning?.” Accessed: Mar. 13, 2025. [Online]. Available: https://www.ibm.com/de-de/topics/deep-learning

[19] J. Wei *et al.*, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.” [Online]. Available: https://arxiv.org/abs/2201.11903

[20] OpenAI, “PRM800K: A Process Supervision Dataset.”

[21] H. Lightman *et al.*, “Let's Verify Step by Step.” [Online]. Available: https://arxiv.org/abs/2305.20050

[22] OpenAI, “OpenAI o1 System Card.” Sep. 2024. Accessed: Apr. 10, 2025. [Online]. Available: https://openai.com/index/openai-o1-system-card/

[23] S. Team, “Try Yourself - SimpleBench.” Accessed: Apr. 10, 2025. [Online]. Available: https://simple-bench.com/try-yourself

[24] DeepSeek-AI *et al.*, “DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning.” [Online]. Available: https://arxiv.org/abs/2501.12948

[25] F. Chollet, “On the Measure of Intelligence.” [Online]. Available: https://arxiv.org/abs/1911.01547

[26] ARC Prize, “What is ARC-AGI?.” Accessed: Mar. 13, 2025. [Online]. Available: https://arcprize.org/arc

[27] L. Marschhausen, “Lepus: run_benchmark_v0.ipynb.” Accessed: Apr. 08, 2025. [Online]. Available: https://github.com/Lukhausen/Lepus/blob/main/experimental/lukas/benchmarking/run_benchmark_v0.ipynb

[28] K. Sun, P. Qi, Y. Zhang, L. Liu, W. Y. Wang, and Z. Huang, “Tokenization Consistency Matters for Generative Models on Extractive NLP Tasks.” [Online]. Available: https://arxiv.org/abs/2212.09912

[29] K. Bostrom and G. Durrett, “Byte Pair Encoding is Suboptimal for Language Model Pretraining.” [Online]. Available: https://arxiv.org/abs/2004.03720

[30] A. K. Singh and D. Strouse, “Tokenization counts: the impact of tokenization on arithmetic in frontier LLMs.” [Online]. Available: https://arxiv.org/abs/2402.14903

[31] D. Franzen, J. Disselhoff, and D. Hartmann, “The LLM ARChitect: Solving ARC-AGI Is A Matter of Perspective.” Accessed: Apr. 09, 2025. [Online]. Available: https://da-fr.github.io/arc-prize-2024/the_architects.pdf

[32] S. T. Sreenivas *et al.*, “LLM Pruning and Distillation in Practice: The Minitron Approach.” [Online]. Available: https://arxiv.org/abs/2408.11796

[33] M. Hodel, “Addressing the Abstraction and Reasoning Corpus via Procedural Example Generation.” [Online]. Available: https://arxiv.org/abs/2404.07353

[34] A. K. Moskvichev, V. V. Odouard, and M. Mitchell, “The ConceptARC Benchmark: Evaluating Understanding and Generalization in the ARC Domain,” *Transactions on Machine Learning Research*, 2023, [Online]. Available: https://openreview.net/forum?id=8ykyGbtt2q

[35] W.-D. Li *et al.*, “Combining Induction and Transduction for Abstract Reasoning.” [Online]. Available: https://arxiv.org/abs/2411.02272

[36] top-quarks, “ARC-solution: Code for 1st place solution to Kaggle's Abstraction and Reasoning Challenge.” Accessed: Apr. 09, 2025. [Online]. Available: https://github.com/top-quarks/ARC-solution/

[37] J. Pan, J. Zhang, X. Wang, L. Yuan, H. Peng, and A. Suhr, “TinyZero.” [Online]. Available: https://github.com/Jiayi-Pan/TinyZero

[38] D. Wu, S. Tan, Y. Meng, D. Stap, and C. Monz, “How Far Can 100 Samples Go? Unlocking Overall Zero-Shot Multilingual Translation via Tiny Multi-Parallel Data.” [Online]. Available: https://arxiv.org/abs/2401.12413

[39] OpenAI, “Fine-tuning.” Accessed: Apr. 11, 2025. [Online]. Available: https://platform.openai.com/docs/guides/fine-tuning

[40] T. Z. Zhao, E. Wallace, S. Feng, D. Klein, and S. Singh, “Calibrate Before Use: Improving Few-Shot Performance of Language Models.” [Online]. Available: https://arxiv.org/abs/2102.09690

[41] D. Cleary, “The Few Shot Prompting Guide.” Accessed: Apr. 12, 2025. [Online]. Available: https://www.prompthub.us/blog/the-few-shot-prompting-guide

[42] H. Nori *et al.*, “From Medprompt to o1: Exploration of Run-Time Strategies for Medical Challenge Problems and Beyond.” [Online]. Available: https://arxiv.org/abs/2411.03590

[43] L. Marschhausen, *arc-agi-lepus-v1*. (2025). Accessed: Apr. 12, 2025. [Online]. Available: https://huggingface.co/datasets/Lukhausen/arc-agi-lepus-v1

[44] M. Williams, “Multi-objective Reinforcement learning from AI Feedback.” [Online]. Available: https://arxiv.org/abs/2406.07295

[45] H. Naqvi and B. Anggorojati, “Ablation Study of Deep Reinforcement Learning Congestion Control in Cellular Network Settings,” in *2022 25th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, 2022, pp. 80–85. doi: 10.1109/WPMC55625.2022.10014846.

[46] J. Pan, “TinyZero Project on Weights & Biases.” Accessed: Apr. 15, 2025. [Online]. Available: https://wandb.ai/jiayipan/TinyZero?nw=nwuserjiayipan

[47] P. Hase, M. Bansal, P. Clark, and S. Wiegreffe, “The Unreasonable Effectiveness of Easy Training Data for Hard Tasks.” [Online]. Available: https://arxiv.org/abs/2401.06751

[48] ARC Prize Initiative, “ARC-AGI-2 Evaluation Dataset.” Accessed: Apr. 15, 2025. [Online]. Available: https://github.com/arcprize/ARC-AGI-2/tree/main/data/evaluation