

From Tuning to Teamwork: Leveraging Fine-Tuning and Agentic Collaboration to Enhance Abstract Problem-Solving in Large Language Models

Søren Søndergaard Meiner (SM), 202205445@post.au.dk

School of Communication and Culture, Aarhus University

Jens Chr. Skous Vej 2, 8000, Aarhus

Supervisor: Ross Deans Kristensen-McLachlan

07th January 2025



Abstract

This study explores how Large Language Models (LLMs) can be enhanced through two leading approaches: agentic frameworks and fine-tuning. While agentic frameworks have been proposed to emulate human-like “system 2” reasoning by orchestrating multiple, specialized agents, recent developments reveal that fine-tuning on domain-specific datasets may be equally or more effective for certain tasks. Drawing on the Abstraction and Reasoning Corpus for Artificial General Intelligence (ARC-AGI) - a benchmark specifically designed to evaluate abstract reasoning and skill acquisition - we compare these two strategies in terms of problem-solving efficacy and reasoning capabilities.

This study finds that fine-tuning consistently delivers superior results on ARC tasks, showing significant gains in both solution accuracy and pixel-level precision, particularly when applied to larger GPT-based models. By contrast, the agentic workflow tested here, which integrates structured chains of specialized agents using Domain-Specific Language (DSL), did not outperform baseline prompting. These findings highlight the critical role of fine-tuning in tailoring model weights to domain-specific patterns, particularly for tasks that challenge an AI’s ability to generalize from minimal data.

Despite limited gains from agentic methods, their modular design retains strong conceptual appeal. In principle, collaborative agents can expand an LLM’s effective “context window,” promote tool-based problem-solving, and reduce hallucinations by offloading specialized subtasks. Yet the results indicate that further refinement of the agentic framework is needed in order to see performance gains on ARC tasks. In view of these outcomes, future work may explore hybrid strategies that combine the systematic decomposition benefits of agentic frameworks with the adaptability conferred by robust fine-tuning and ongoing test-time adaptation.

Keywords: Large Language Models, Agentic Frameworks, Fine-Tuning, Abstraction and Reasoning



Table of Contents

Introduction	1
Motivation	1
Large Language Models: Foundations and Advancements	2
Agentic Frameworks: A Team of Specialized Agents	3
Fine-Tuning: Tailoring Models for Specialized Tasks	5
Benchmarking Intelligence: The ARC-AGI Benchmark	6
Hypotheses and Research Objectives	9
Methodology	9
Data Preparation	9
Agentic Framework Implementation	9
Fine-Tuning Procedure	11
Scoring and Evaluation Metrics	12
Results	14
Discussion	15
Key insights and takeaways	15
Findings	15
Comparison with Other Studies	16
Methodological limitations	18
Key improvements for future studies	21
Conclusion	23
Code availability	24
Acknowledgments	24
References	24
Appendices	33
Appendix 1: DSL for agents	33
Appendix 2: Prompts Used in Agentic Workflow	34
Appendix 3: Fine-tuning Prompt	37
Appendix 4: o1 Visual Capabilities	38

Introduction

Motivation

The motivation for this study builds on my previous research on Generative Agent-based Modelling (GABM), an innovative approach introduced by Google Deepmind that employs generative artificial intelligence (GAI) instead of relying on fixed-parameter computations typical of traditional agent-based models (Vezhnevets et al., 2023). This application of GAI introduces agentic frameworks, enabling the orchestration of multiple specialized agents to collaboratively solve complex tasks. An illustrative example of this is the GABM ChatDev by Qian et al. (2024), which organizes agents into distinct roles, such as a programmer, art designer, and chief executive officer, to produce software autonomously. This demonstrates the significant potential of agentic frameworks in addressing complex problem-solving tasks by dividing them into specialized, manageable components.

These frameworks have sparked interest in enhancing the reasoning and problem-solving capabilities of Large Language Models (LLMs), which form the focus of this study. The abstraction reasoning corpus for artificial general intelligence (ARC-AGI) by François Chollet (2019) serves as a benchmark for evaluating skill acquisition, requiring a model to demonstrate novel reasoning and problem-solving abilities to solve tasks. While most of these tasks are simple for humans, they present significant challenges for machine learning and AI systems. Chollet argues that a system can be considered generally intelligent if it effectively acquires novel skills.

The societal impact of creating a generally intelligent system carries enormous potential across numerous domains, including healthcare, education, and scientific research. This potential has driven researchers to explore various methods to enhance the reasoning and problem-solving capabilities of Large Language Models (LLMs). Among these approaches, agentic frameworks and fine-tuning have emerged as promising strategies.

Given the advancements and differing methodologies of agentic frameworks and fine-tuning, this study seeks to address the following research question:

How do agentic frameworks compare to fine-tuning techniques in enhancing the problem-solving and reasoning capabilities of Large Language Models on the ARC-AGI benchmark?

Large Language Models: Foundations and Advancements

In recent years, the field of Natural Language Processing (NLP) has undergone a transformative leap with the introduction of LLMs (Brown et al., 2020). LLMs leverage tokenized vector spaces and transformer neural networks to process and generate language. In tokenized vector spaces, tokens - such as words, subwords, or characters - are represented as dense numerical vectors in a high-dimensional space. The proximity of these vectors has shown to reflect semantic relationships, enabling models to infer linguistic associations (Mikolov et al., 2013). While vector spaces provide a foundational representation of language, they are insufficient for generating complex textual sequences. This limitation is addressed by transformer neural networks, which enable language models to produce coherent and contextually relevant text by capturing intricate patterns in language.

A transformer neural network employs a self-attention mechanism that enables parallel processing of large tokenized vector stores. This design allows the model to attend to contextual information and bind sentences together based on input sequences (Vaswani et al., 2017). Language models are pre-trained on tokenized vector spaces and serve different purposes: for instance, Bidirectional Encoder Representations from Transformers (BERT) specializes in sentence text similarity with a bidirectional encoder-only architecture that attends to all tokens in a sequence simultaneously, while Generative Pre-trained Transformer 1 (GPT-1) predicts the next word in a given context using a unidirectional decoder-only architecture that attends only to previously generated tokens and not future tokens. Both models are typically fine-tuned at later stages for specific tasks (Devlin et al., 2019; Radford & Narasimhan, 2018).

Over time, language models have grown exponentially in size, measured by the number of parameters (i.e., the weights in their neural networks). GPT-1 contained 117 million parameters, while GPT-3 expanded to an astonishing 175 billion parameters (Ali, 2023). These advances redefined "pre-trained models" to "large language models" due to their

massive scale. ChatGPT by OpenAI, a website where users can query (prompt) the GPT models, has quickly grown extremely in popularity due to their ability to generate human-like text (OpenAI, 2024a; Statista, 2023).

The capabilities of LLMs have also rapidly evolved, encompassing reasoning, writing, problem-solving, coding, and more. A lot of these models by different companies have quickly reached the average human level in different benchmark tasks such as the tasks HellaSwag, MMLU, and ARC (not to confuse with ARC-AGI) (Clark et al., 2018; Clementine Fourrier et al., 2024; Hendrycks et al., 2021; Zellers et al., 2019). These benchmarks assess various aspects of a model's functionality. For example, the MMLU benchmark evaluates performance across diverse domains, including elementary mathematics, computer science, and law.

While the priors of traditional LLMs are primarily trained on vast corpora of textual data, the field of multimodal LLMs has started to evolve, which extends the priors of LLMs to include not only language as a modality but also vision and audio. The integration of vision is achieved through the same transformer neural network to encode the data into a high-dimensional vector space and at a later point the data is projected into a shared latent space (Dosovitskiy et al., 2021; Yin et al., 2024). After the data has been encoded into the shared latent space, a common transformer model can align and process the multimodal data by applying cross-attention techniques as well as modality-specific self-attention layers (Su et al., 2020).

Agentic Frameworks: A Team of Specialized Agents

A lot of research on LLMs and their reasoning capabilities draws inspiration from the concept of 'System 2' thinking (Kahneman, 2011; Weston & Sukhbaatar, 2023; Yu et al., 2024). In this dual-process theory, humans are proposed to employ two systems for thinking: System 1, which facilitates fast, intuitive thoughts, and System 2, which supports deliberate, logical reasoning. To approximate System 2 thinking in LLMs, inspiration can be drawn from the problem-solving processes described by Newell & Simon (1972). Their model suggests that humans expand a problem state by generating and exploring various solutions within a defined problem space.

A promising technique to emulate this process in LLMs is the Chain of Thought (CoT) approach, which involves breaking a problem into intermediate steps before attempting to reach the goal state. Implementing CoT has been shown to improve LLMs' reasoning capabilities and enhance their problem-solving skills (Wei et al., 2023). However, a significant challenge with zero-shot prompting (providing a single input prompt to the model) is that LLMs often struggle with long context windows. This can lead to difficulties in attending to the entire context effectively, increasing the risk of information overload and diminishing performance. Additionally, zero-shot prompting increases the risk of "hallucination," where the model generates non-factual or untruthful content (Brown et al., 2020; Ji et al., 2023; N. F. Liu et al., 2023).

To address these issues, CoT can be combined with few-shot prompting, where the model is guided to break down a task into intermediate steps before solving it. This method not only enhances reasoning capabilities but also mitigates context window limitations by dividing tasks into smaller, more manageable chunks. However, this approach often requires human intervention, as users must actively guide the LLM through the problem-solving and ensure the appropriate techniques are applied.

A more advanced and autonomous solution is the use of agentic frameworks, where tasks can be divided into steps using CoT, and each step is handled by a specialized LLM agent. These agents work collaboratively by distributing tasks across specialized roles. A critical advantage of this framework is its ability to integrate external tools, such as code execution environments, search engines, or API calls, significantly enhancing the system's capacity to perform domain-specific tasks (Talebirad & Nadiri, 2023). This combination of role specialization and tool integration provides a robust architecture for tackling complex problems that require reasoning and problem-solving.

Agentic frameworks have gained significant traction due to their potential to replicate swarm-like collaborative problem-solving strategies observed in biological systems. In such systems, individual entities, while simple in isolation, can generate highly complex behaviors when organized into networks. Examples include the collective problem-solving seen in ant colonies and the intricate processing of neural networks in the brain (Bonabeau et al., 1999). Some of the most popular frameworks for building these agentic systems are Langgraph,

CrewAI, Autogen and recently OpenAI has released a sample framework called Swarm (CrewAI, 2023; Harrison Chase, 2023; Ilan Bigio et al., 2024; Wu et al., 2023).

Fine-Tuning: Tailoring Models for Specialized Tasks

Fine-tuning is the process of adapting a pre-trained language model to a more specialized domain by performing additional training steps on a smaller task-specific dataset. Instead of training a language model from scratch to a specific domain, which is both time-consuming and resource-intensive, fine-tuning leverages the prior knowledge already acquired in the pre-training phase of the language model. Through exposure to carefully curated examples of the target task, the model's internal representations can be nudged toward a domain-specific task while still maintaining its core linguistic and conceptual foundations (Jeong, 2024).

This approach enhances the model's accuracy, reliability, and adaptability to the desired task. While base models often produce generic outputs and may overlook subtle domain-specific nuances, fine-tuned models can integrate these nuances seamlessly, delivering more accurate and contextually appropriate responses. With increasing accuracy, hallucinations, and irrelevant outputs can be reduced significantly but can also mitigate the need for extensive prompt engineering due to the model's newly gained knowledge of how to perform the specific task. Another significant advantage of fine-tuning is cost efficiency. By tailoring the model to the task, the need for a large context window is reduced, optimizing computational resources (Kang et al., 2024).

When fine-tuning, a set of hyperparameters can be adjusted to specify how the model should be updated such as batch size, learning rates, and number of training epochs, which are the parameters that can be tuned on the OpenAI platform or using the API (OpenAI, 2024b). Since fine-tuning involves updating pre-existing parameters and weights for a model, it is crucial to include the right set of hyperparameters in order not to lose the core foundations of the model or overfit the model, which also becomes specifically significant when dealing with smaller datasets for fine-tuning.

The learning rate controls the size of the step the optimization algorithm takes in the parameter space after each gradient update. Specifying this value is one of the most critical parts of fine-tuning, as it dictates how quickly or slowly the model updates its parameters to

the new data. A smaller learning rate preserves more of the language understanding from the pre-trained data, which is beneficial when retaining well-formed linguistic features is a priority. A larger learning rate can be used if the domain shift is considerable and it is needed for the model to readjust the parameters more aggressively. OpenAI's platform provides a learning rate multiplier, which scales the learning rate relative to the one used during the foundation model's pre-training phase (Miha Cacic, 2023).

Batch size refers to the number of training examples processed by the model before updating its parameters. Training on the entire dataset at once can be computationally expensive, so the data is divided into smaller batches. The choice of batch size often depends on the size of the dataset and the computational resources available. Larger datasets typically benefit from larger batch sizes, whereas smaller datasets may require smaller batches to optimize memory usage and performance.

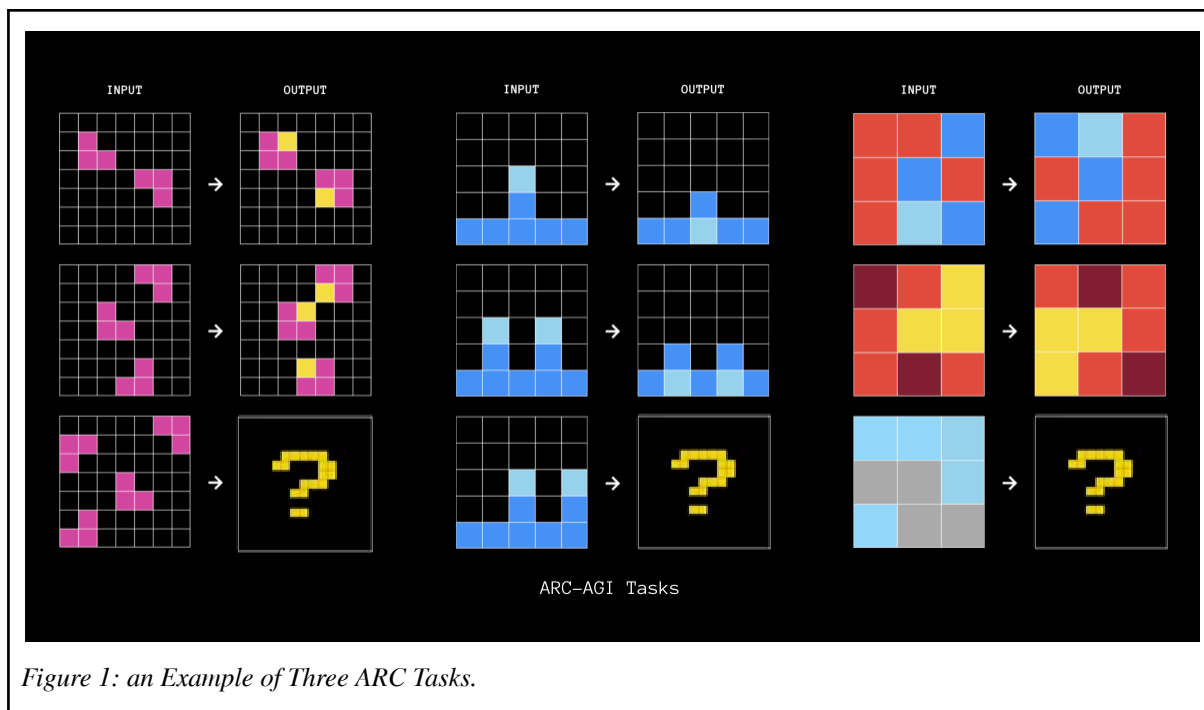
An epoch in machine learning represents one complete pass through the entire training dataset by the algorithm. The number of epochs is crucial for model performance: too few can lead to underfitting, where the model fails to learn adequately, while too many can result in overfitting, where the model becomes overly tailored to the training data and struggles to generalize to new inputs. (Howard & Ruder, 2018).

Benchmarking Intelligence: The ARC-AGI Benchmark

The abstraction reasoning corpus for artificial general intelligence (ARC-AGI), developed by François Chollet (2019), is a benchmark task designed to evaluate whether AI systems can generalize and acquire new skills effectively that reflect aspects of human intelligence. The ARC-AGI benchmark comprises tasks that challenge the solver to identify patterns, gain new skills, and apply abstract reasoning without extensive prior training. The tasks are often quite easy for humans but present significant challenges for AI systems. The ARC-AGI dataset is relatively small, setting it apart from other benchmark tasks that often rely on large datasets that can be used extensively for model training. Instead, ARC-AGI provides only a few examples per task, requiring solvers to generalize from limited information, which humans excel at. The ARC-AGI benchmark is designed to track progress toward achieving AI capable of doing generalization, but it is not intended as a definitive test for AGI.

Chollet has criticized other benchmarks for conflating task-specific proficiency with general intelligence, arguing that this conflation is the reason why LLMs are reaching a plateau on these evaluations. Chollet also highlights the importance of priors and defines them as assumptions and pre-existing knowledge embedded in a system that influences how it interprets and learns from new information. Optimal priors, he argues, should not be overly specific to narrow tasks but sufficiently general to support reasoning and problem-solving across diverse domains. This perspective underscores the importance of designing models that can generalize effectively rather than excelling solely within predefined tasks.

Each ARC task consists of a series of input-output grid pairs, where grids of varying sizes represent cells containing one of ten colors, encoded as numerical values in the dataset. The objective is to infer the transformations required to solve a test task based on patterns observed in the provided examples. Each task necessitates the acquisition of a new skill, emphasizing the importance of generalization, logical reasoning, and abstraction - key attributes of human problem-solving (Anderson, 1993). Figure 1 illustrates three examples of ARC tasks.



For the ARC-AGI benchmark, a public, semi-private, and private dataset has been created to ensure fair evaluation and prevent models from being trained on the private dataset. When

predicting on the public dataset, there is a possibility that the model being used has already been exposed to the data during training. The public dataset consists of 400 tasks and 400 evaluation tasks, while both the semi-private and private dataset only consists of 100 tasks. The private dataset is used for the ARC Prize competition, which offers a one-million-dollar reward for achieving an 85% success rate using a model.

The competition is hosted on Kaggle and prohibits internet access, requiring participants to use locally hosted models, either on Kaggle or their own systems. For each task, participants are allowed two prediction attempts per test output grid. A score of 1 is awarded if either attempt is correct. For tasks with two test outputs, a partial score of 0.5 is given if only one of the outputs is correct (*ARC Prize*, 2024).

The tasks in the public training dataset can be used to train models, whereas the evaluation set is reserved exclusively for testing model performance. The evaluation tasks are often more challenging than those in the training set. On average, human performance on the public training set is 76%, while the average drops to 64% on the public evaluation set (LeGris et al., 2024).

Recent evaluations of newer models made by Anthropic and OpenAI have shown improved performance on the tasks, though still significantly below average human levels. Both OpenAI's model o1-preview and Anthropic's Claude Sonnet 3.5 scored 21% on the public evaluation tasks which is quite an improvement from the previously best-performing model GPT-4o which scored 9% (Anthropic, 2024; Knoop, 2024; OpenAI, 2024a).

One of the previous best attempts at solving the ARC tasks was made by Ryan Greenblatt (2024) which included generating 8000 samples with predictions on the example tasks using GPT-4o. This approach leveraged few-shot prompting, where the model was instructed to do CoT reasoning on the training examples before being queried to generate code (P. Liu et al., 2023). Greenblatt later identified the model that performed best on the example tasks and used it to predict the test output. This method achieved a 50% score on the public evaluation set, and was a stepping stone in solving the ARC task, showing that with novel engineering, a base model was capable of great performance.

Hypotheses and Research Objectives

This study aims to evaluate and compare the effectiveness of agentic frameworks and fine-tuning techniques in enhancing the problem-solving and reasoning capabilities of LLMs on the ARC-AGI benchmark. Based on the foundational concepts of fine-tuning and agentic frameworks the following null hypotheses are proposed:

H₀₁: Agentic frameworks do not significantly improve performance on public ARC tasks compared to baseline models.

H₀₂: Fine-tuning techniques do not significantly improve performance on public ARC tasks compared to baseline models.

Methodology

Data Preparation

All experiments were conducted using the ARC-AGI 2024 public dataset developed by François Chollet et al. (2024). The dataset comprises 800 tasks - 400 for training and 400 for evaluation - stored in JSON format and loaded using standard Python 3 libraries using Visual Studio Code (Microsoft, 2015; Van Rossum & Drake, 2009). Training and evaluation datasets were processed separately, with their respective challenges and solutions organized into distinct task sets.

Further preprocessing of the tasks was done to reduce costs by creating a subset of the evaluation tasks. This subset included only tasks with grid sizes smaller than 15x15, resulting in a reduced set of 28 evaluation tasks. This preprocessing step not only lowered expenses but also potentially reduced the complexity of the problems, a possibility that will be discussed in more detail later.

Agentic Framework Implementation

The LLMs used in this experiment were accessed through LangChain, a framework designed to simplify the process of chaining interactions between various LLMs. LangChain supports

integration with local models as well as those provided by external companies like OpenAI, enabling flexibility in model selection (Harrison Chase, 2022).

The models in this framework were accessed by loading API keys from an environment file, after which clients were initialized with the desired model configurations. A maximum token usage limit of 3000 was set to prevent excessively long responses and reduce costs. For experimental purposes, less expensive models such as GPT-4o-mini by OpenAI were used. For evaluating performance, more capable models like OpenAI's GPT-4o were employed (OpenAI, 2024b).

In contrast to a simple prompt-and-answer format, a multi-step agentic framework was utilized, guiding the model through predefined agents before reaching a prediction. To do this, the framework Langgraph was used which allows for creating a directed acyclic graph that systematically partitions the solution process into distinct phases (Harrison Chase, 2023). Before prompting the first LLM in the graph, the tasks were converted into a string rather than its original JSON format since LLMs are better at handling strings and perform poorly at exporting strict JSON formats (Lee et al., 2024).

The first agent was tasked with identifying the logic of the problem. It was provided input-output pairs for the task and instructed to verbalize the underlying transformations, effectively reasoning about which skill needed to be learned using CoT. Additionally, the agent was prompted with the predicted output grid size for the task, generated by a mathematical algorithm developed by Kai Hammond (2024) which achieved 86.4% accuracy on the training data set.

After articulating the identified patterns, the model advanced to a pattern recognition agent, tasked with determining which pre-defined helper functions were required to solve the task. These helper functions (also referred to as tools) effectively constituted a domain-specific language (DSL) tailored for grid-based transformations. They were designed to perform operations such as rotating grids, copying grids, changing colors, and filling empty cells which can be seen in Appendix 1.

Following pattern recognition, the model proceeded to a code generation agent. This agent was asked to produce a Python function called `'solve_task(input_grid)'` and use the helper functions recognized to apply the same patterns as observed from the input and output grids.

The agent was given instructions to use the helper functions provided and not import additional libraries or add extraneous explanations.

Once the function was generated, it was passed to a code revision agent tasked with reviewing and refining the code. This agent checked for potential improvements and ensured correctness. To clean the code and remove non-code elements, such as statements like “Here’s the corrected code,” regex algorithms were applied (Aho, 1991).

Finally, the code was parsed to a function capable of executing the code including the global variables such as the tools and Python libraries needed.

Running the agentic workflow on the 28 tasks with GPT-4o cost approximately 1.5\$ using the API. All prompts used for the agents can be seen in Appendix 2.

If the code failed to execute, it was retried up to three times to ensure a successful prediction was made. Furthermore, logging was implemented by using Python’s logging module, which allowed for documentation and inspection of run-time information, error messages, and intermediate reasoning steps generated by the model.

The final dataset of predictions was exported to JSON for scoring, which is also a requirement for the ARC competition. For each task, two attempts were allowed and the agentic framework was run twice ensuring that the model produced independent attempts and did not continue from previously generated logic.

Fine-Tuning Procedure

In preparation for generating data suitable for fine-tuning the model, the challenges and solutions of the training dataset were loaded using standard Python libraries (Van Rossum & Drake, 2009). A prompt was then generated for each challenge as seen Appendix 3, which included the input-output pairs and instructed the LLM to learn the underlying patterns, predict the output grid, and return only the output grid.

For the fine-tuning dataset, the correct solution for each challenge was appended as the expected response from the LLM. The resulting dataset was saved in JSONL format, compatible with OpenAI’s fine-tuning tool (OpenAI, 2024b). This process produced a dataset designed to explicitly model the desired behavior of the LLM when presented with new, unseen test inputs.

After the JSONL file was generated, it was uploaded to OpenAI's platform for fine-tuning. Two different models were trained on the data; the GPT-4o mini and the standard GPT4o model. It was preferable to test both a smaller and a larger model to compare the performance of models with different parameter sizes and determine whether updating parameters alone is sufficient or if the model's base size significantly impacts its performance (Howarth, 2024).

Both models were fine-tuned for three epochs with a total token size of approximately 4.13 million tokens. The batch-size hyperparameter was set to one, since the dataset was relatively modest in size and since a new skill is acquired for each ARC task, it was preferred that the model would learn as much as possible from these examples (Keskar et al., 2017). The learning rate multiplier was set to two, allowing for the model to update its parameters with a multiplier of two compared to the learning rate of its foundational model (Miha Cacic, 2023).

Once fine-tuning was complete, Langchain was used as the framework for calling the new models, and the maximum usage of tokens was set to 3000 (Harrison Chase, 2022). The fine-tuning models were run through the same pipeline as the agentic model with 3 retries and 2 attempts for each try. A single agent was run with a prompt similar to the one used for fine-tuning. However, since this agent was trained to output humanly readable grids, a JSON output parser from Langchain was used for conversion. The final predictions were exported to a JSON file for scoring.

Scoring and Evaluation Metrics

All submissions were evaluated using the scoring methodology from the ARC Prize competition, supplemented with a pixel accuracy metric to provide additional insight into performance (François Chollet & Mike Knoop, 2024).

To enhance the interpretability of results, each task was visualized with its solution input grid, the predicted output grid, and the actual output grid. This visualization was implemented using the Matplotlib and Seaborn libraries (Hunter, 2007; Waskom, 2021). Functions were developed to calculate the percentage of correctly solved tasks and to measure the proportion of correctly predicted pixels in the output grids. All of the results were summarized in a single HTML report to facilitate inspections of solutions by the individual models.

The limited sample size of 28 tasks per model made it challenging to conduct robust statistical comparisons based solely on the percentage of correctly solved tasks. To address this limitation, the total number of correctly predicted pixels was used as an alternative metric to provide a more nuanced evaluation of the models' performance. Visual inspection of the results suggested that models closer to solving a task tended to produce outputs with a higher proportion of accurate pixels, making this a reasonable metric for solution proximity. If the analysis had been conducted on the full evaluation set of 400 tasks, a statistical comparison of the binary correct/incorrect outcomes would have been more appropriate, as it provides a clearer measure of task-level success.

For statistical comparison of pixel-level correctness between models, pairwise tests were conducted. Given that the data did not meet normality assumptions, the non-parametric Wilcoxon signed-rank test was employed using the SciPy library (Virtanen et al., 2020).

The results were further analyzed and visualized using R using the libraries Tidyverse and gt for data manipulation and presentation (Iannone et al., 2024; R Core Team, 2023; Wickham et al., 2019). Visualizations included a comprehensive summary table including the amount of correctly solved tasks, pixel correctness scores and results from the Wilcoxon signed-rank test. A visualization of the entire workflow can be seen in Figure 2.

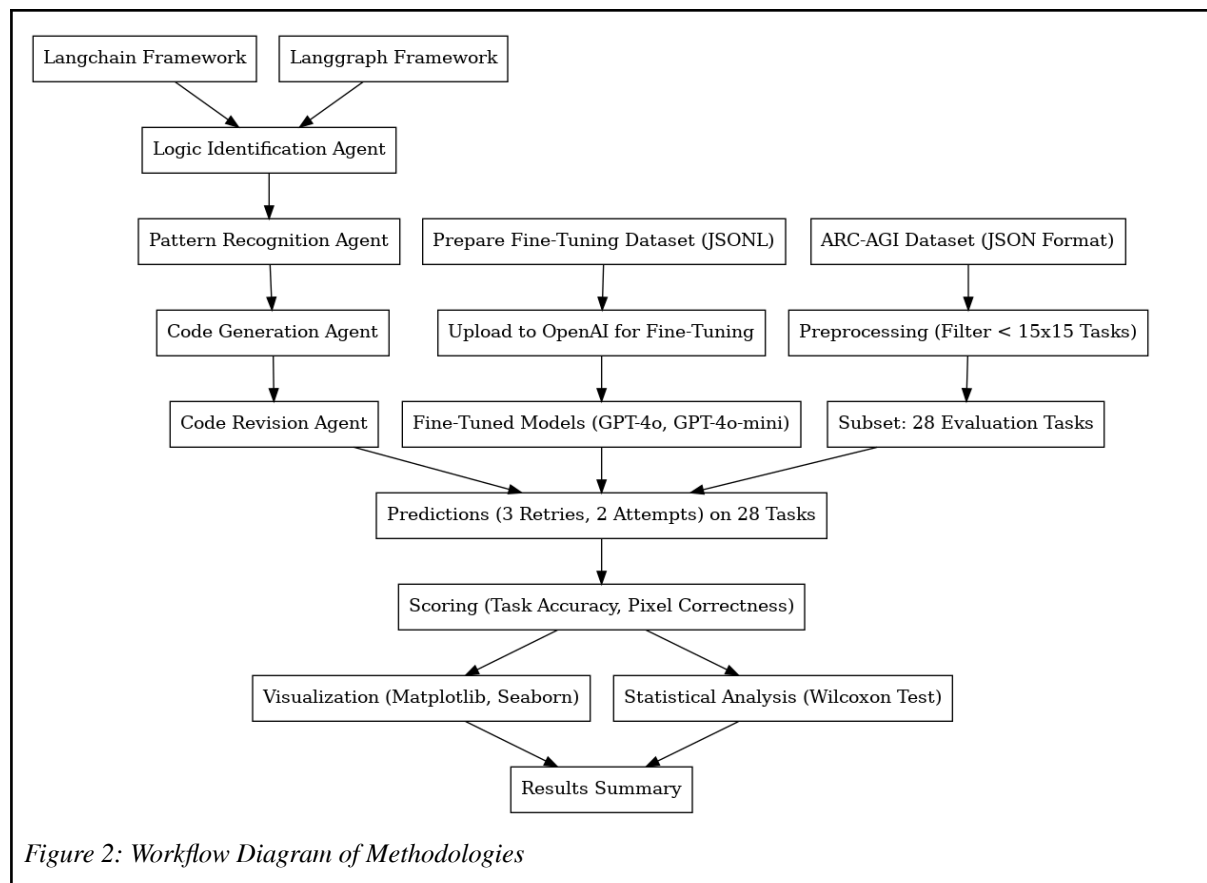


Figure 2: Workflow Diagram of Methodologies

Results

The performance of various submissions was evaluated using a subset of 28 tasks from the ARC-AGI evaluation dataset, comprising a total of 33 tests (as some tasks included paired tests), filtered to include only grid sizes smaller than 15x15.

Out of 28 tasks, GPT-4o-mini successfully solved 3.5 tasks, while the agentic framework achieved 7.5. GPT-4o solved 8.5 tasks, the fine-tuned GPT-4o-mini solved 10.5, and the fine-tuned GPT-4o achieved the highest score with 16.0 tasks solved.

All statistical analyses of pixel accuracy were carried out using the non-parametric Wilcoxon signed-rank test since the data showed deviations from normality. GPT-4o was set as the reference model for comparison of the Wilcoxon signed-rank test.

Results for GPT-4o-mini indicated no significant difference in pixel correctness compared to GPT-4o ($W = 146.000$, $p = .1220$), and similarly, the agentic framework with tool integration did not significantly outperform GPT-4o ($W = 130.500$, $p = .3890$).

By contrast, the fine-tuned GPT-4o model demonstrated a significant improvement ($W = 32.000$, $p = .0064$). Fine-tuned GPT-4o-mini exhibited no statistically reliable advantage over GPT-4o ($W = 152.500$, $p = .7878$).

The results of the models can be seen in Table 1 which summarizes key metrics of the submissions such as tasks solved, correct percentage, and p-values from the Wilcoxon signed-rank test performed on the pixel accuracy.

ARC-AGI Evaluation Summary						
Model Name	Tasks Evaluated	Tasks Solved	Percentage Correct (%)	Mean Pixel Accuracy	Standard Deviation (Pixel)	p-value ¹
Fine-tuned GPT-4o	28	16.00	54.55	86.88	16.80	0.0064**
Fine-tuned GPT-4o-mini	28	10.50	36.36	73.62	28.80	0.7878
GPT-4o	28	8.50	30.30	71.51	32.62	Reference
Agents + Tools	28	7.50	27.27	63.62	32.69	0.3890
GPT-4o-mini	28	3.50	12.12	61.47	30.92	0.1220
¹ p-values computed via Wilcoxon test on pixel correctness.						
Significance codes: ***p < 0.001, **p < 0.01, *p < 0.05						

Table 1: Evaluation Summary of Predictions on the ARC Tasks

Discussion

Key insights and takeaways

Findings

In light of these findings, several key observations emerge regarding the comparative effectiveness of agentic frameworks and fine-tuning techniques for enhancing Large Language Models on the ARC-AGI benchmark. First, and most notably, fine-tuning of the GPT-4o model produced the strongest performance gains, demonstrating statistically significant improvements in pixel accuracy and effectively doubling the number of solved tasks relative to the baseline of GPT-4o. These results support the rejection of H_{02} , which posits that fine-tuning does not significantly improve performance on ARC tasks, confirming that fine-tuning has a substantial impact on solving these tasks.

On the other hand, while the fine-tuning of the GPT-4o-mini model did not result in statistically significant improvements relative to the GPT-4o reference, it still demonstrated a notable enhancement over its base model. The fine-tuned GPT-4o-mini solved an average of 10.5 tasks, a substantial increase from the base model's performance of 3.5 tasks. This suggests that fine-tuning successfully adjusted the model's parameters and weights, steering them toward domain-specific features critical for solving ARC tasks.

By contrast, the agentic framework developed for this study did not outperform the baseline model. A considerable amount of effort went into designing the agentic pipeline in which specialized agents tackled different stages of reasoning with the usability of tools. Despite the conceptual appeal and growing interest in swarm-like agentic solutions, this approach did not yield measurable advantages over simpler prompting (Bonabeau et al., 1999; Kahneman, 2011; Talebirad & Nadiri, 2023). Consequently, the null hypothesis H_{01} , which posits that agentic frameworks do not significantly improve performance on ARC tasks, cannot be rejected in this context. One possible explanation is that successful deployment of agentic strategies may require more elaborate domain-specific heuristics or stronger synergy between the agents and the tasks' underlying patterns. Various other applications of agentic structures

could also be explored, which could have a positive impact on solving the ARC-AGI, which will be further discussed.

The results observed in this study suggest that the subset of 28 tasks selected from the ARC-AGI dataset may represent a set of relatively easier tasks, especially when comparing performances to the overall public evaluation dataset. While GPT-4o is reported to solve only 9% of the public evaluation dataset tasks, even the smallest model in this study, GPT-4o-mini, achieved a correct solution rate of 12.12% on the 28-task subset. Moreover, GPT-4o solved 30.30% of the tasks in the subset (François Chollet et al., 2024). Smaller grids may limit the number of possible transformations and permutations required to solve a task, simplifying the reasoning process. Consequently, this subset might inadvertently favor models that perform better on less computationally intensive tasks or tasks with fewer abstract reasoning requirements. Furthermore, since the initial release of GPT-4o's results, the models may have been exposed to the ARC dataset or similar tasks, which may have contributed to their enhanced performance.

Comparison with Other Studies

In the research by Tan & Motani (2023), they demonstrated that deploying their agentic framework solved 50 out of 111 tasks in the public ARC-AGI training dataset, which had a very strong appeal. However, they did not report how their model performed on the evaluation dataset. Their approach seemed promising, incorporating a DSL for agents to use and a feedback loop to handle compile errors and validate generated code against training examples. The agentic workflow used in this study adopted a similar feedback loop with retries for generated code that failed to execute correctly. However, the architecture of this current study did not include a feedback loop for testing whether the generated code was successful at solving the training examples. Such an implementation could be viewed as a more novel approach compared to the approach by Ryan Greenblatt (2024), where he drew 8000 samples of generated code. Ideally, the agentic workflow with that implementation would enable agents to iteratively improve their performance through feedback loops, rather than relying on the generation of thousands of random code samples.

While writing this paper, new approaches for solving the tasks were released (December 2024), which rely more heavily on fine-tuning the models compared to earlier methods. The

winning team, "The ARChitects," fine-tuned their model on four specialized datasets: the original ARC dataset, RE-ARC (an augmented version of ARC), ConceptARC (a precursor dataset), and ARC-heavy (a dataset mimicking ARC tasks generated by LLMs) (Franzen et al., 2024; Hodel, 2024; Li et al., 2024; Moskvichev et al., 2023). After fine-tuning the model with these datasets, The ARChitects implemented a test-time fine-tuning strategy. This strategy involves further fine-tuning of the model on the specific task and generating additional task-specific data by applying various transformations, such as rotations, transpositions, and color permutations. The second-best best solution by Barbadillo (2024) similarly demonstrated the effectiveness of test-time fine-tuning with augmented datasets, further validating this approach. The results from the competition highlight the promise of fine-tuning and test-time adaptation for solving ARC tasks, aligning with the findings of this study, which demonstrated the impact of fine-tuning on GPT-4o's performance (François Chollet et al., 2024).

Following the conclusion of the 2024 ARC-AGI competition, OpenAI and the ARC Prize team made a striking announcement on December 20th, 2024 regarding the performance of a new, unreleased model: o3. This model scored 87.5% on the semi-private ARC-AGI dataset when running at a high-compute setting, breaking the 85% barrier which was set to mark significant progress towards achieving AGI. However, the high-compute model did not adhere to the budget rules of the competition (costs of a submission should be below \$10,000). OpenAI also tested a low-compute model, which adhered to the rules, scoring 75.7%, with a total retail cost of \$2,012. Reportedly by OpenAI, the models had been tuned on 75% of the training dataset, but they made no reports of how the base model scores on the task (François Chollet, 2024). Furthermore, Altman, (2025), CEO of OpenAI, released a blog post on the 6th of January 2025, that OpenAI is confident that they now know how to build AGI by their definitions, and predicting that AI might be joining "the workforce" in 2025.

OpenAI has announced that o3 utilizes CoT reasoning techniques during testing, employing an architecture similar to those used in recent approaches for solving ARC tasks (Guan et al., 2024). Recent advancements in enhancing the reasoning capabilities of LLMs indicate that simply scaling the models is not the only factor that increases their effectiveness but the architecture of the models also plays a crucial role, which can be seen in the effectiveness in

solving the ARC tasks compared to previous models. While OpenAI has not released the specific details of the architecture deployed, the paper by Zeng et al. (2024) approximates the architecture laying the foundation for o3, potentially leading to the creation of open-source models capable of reasoning. They suggest that the model is built on four base components; policy initialization, search, learning, and a reward design.

Using o3 is a lot more expensive compared to the solution deployed in this study, and even at a low compute setting, it costs approximately \$20 per task, whereas solving a single task using the fine-tuned or the agentic workflow costs approximately \$0.05-\$0.20 per task.

Methodological limitations

A significant limitation in solving ARC tasks with LLMs is their inadequate visual capabilities. Although some models, such as OpenAI's GPT-4o series, include multimodal capabilities for image recognition and generation, they perform poorly in tasks requiring fine-grained visual detail, such as object separation or scene parsing (Jiao et al., 2024). For example, as shown in Appendix 4, o1, OpenAI's most advanced reasoning model, failed to correctly identify the number of light blue squares in a simple ARC task. Consequently, the visual capabilities of these models are insufficient for solving ARC tasks, necessitating the conversion of visual information into textual strings for processing.

Both Ryan Greenblatt (2024) and Tan & Motani (2023) underscore this challenge and the lack of these innate priors, noting that if humans had similarly poor visual abilities, solving even basic ARC tasks would be far more time-consuming. Moreover, requiring humans to solve these tasks using only textual representations, followed by generating Python code, would likely result in failure for most individuals. For primates, nearly half of the neocortex is dedicated to vision, underscoring its central role in spatial reasoning and general cognitive function. It has also been shown that for humans, the way a problem is presented can have a huge impact on the ability to solve the problem, and different ways of presenting the puzzles to LLMs could also be explored (Felleman & Van Essen, 1991; Solaz-Portolés & Sanjosé López, 2007).

Another notable limitation of using LLMs is the concern that models might generate correct answers simply because the data was included in their training material. One way to mitigate

this issue could have been to reformat the evaluation data, for example, by encoding the strings with letters or textual representations of colors instead of their original format.

The ARC Prize competition addresses this limitation by keeping the test data private, ensuring that models are not trained on it. Additionally, ARC-AGI includes a semi-private evaluation dataset for its public competitions. Interestingly, submissions tend to perform better on the public evaluation set than on the semi-private evaluation set. For instance, GPT-4o scores 9% on the public evaluation set but only 5% on the semi-private evaluation set. Assuming the difficulty of both datasets is similar, this discrepancy may indicate that the models have been partially trained on the public evaluation data (François Chollet et al., 2024).

LLMs are prone to generate plausible-sounding, but incorrect information known as the phenomenon of hallucinations. This issue lies in the nature of the LLMs as they are models that based on a set of tokens predict the next token in a sequence, without a grounded understanding of factual truths. In a task like ARC-AGI hallucinations can become critical if one step of the reasoning is untrue, the logic will be incorrect and thereby the answer will be incorrect. For this study, multiple instances of hallucinations were observed such as agents trying to import packages that don't exist for coding, generating logic that doesn't match the task, and inserting numbers out of the range told. Recent studies propose that hallucinations for LLMs are inevitable and are an innate limitation of LLMs due to the way they are trained. The issue can be mitigated with either fine-tuning or using agents to correct for the hallucinations, but are inevitable to omit completely (Xu et al., 2024; Zhang et al., 2023).

Slight changes in prompt phrasing or context length often resulted in varying outputs, which posed significant challenges in the agentic workflow. Since the agents depended on each other to build a coherent chain of reasoning, any inconsistency in one agent's output could propagate errors and lead to incorrect final results. One potential solution to mitigate this issue would have been to adjust the temperature settings of the agents. Lower temperatures typically produce more deterministic outputs, while higher temperatures generate more diverse responses by introducing randomness, ranging from 0 to 1 (Peeperkorn et al., 2024).

The effect of temperature adjustments was not explored in this study, but it was assumed that solving tasks requiring new skills might benefit from higher temperatures, which promote innovation. OpenAI's API sets the temperature hyperparameter to 1 by default (OpenAI,

2024b). Higher temperatures may aid the "Identify logic" agent in discerning patterns, while lower temperatures likely suit the "Generate code" agent, as its task involves producing code based on prior answers rather than fostering creativity.

Cost was a significant factor and limitation in this study, particularly during evaluation. It was challenging to determine which steps had a positive impact on the model's performance, as testing was conducted using cheaper models. To evaluate the model's ability to generalize, it needed to run on multiple tasks due to the diverse nature of the data. Additionally, each agent was limited to a maximum of 3000 tokens, which restricted the length of logic or code generated. Further experimentation of varying context lengths for different agents could be beneficial.

As also noted by Ryan Greenblatt (2024), GPT-4o is also quite poor at using long contexts compared to other models. A model like Claude Sonnet 3.5 could have been used, as it also performs better at coding and scores higher on solving ARC tasks. Experimentation using a mixture of GPT-4o and Claude Sonnet 3.5 was tested but ultimately ruled out, as fairness in evaluating the results would require also evaluating Claude Sonnet 3.5 on the subset, which would exceed budgetary limits (Anthropic, 2024).

A more comprehensive analysis could also have been carried out on the entire evaluation set, where a statistical analysis of the binary outcomes would have been more fitting. However, this approach was not feasible given the cost constraints of evaluating the full set of 400 tasks.

If cost was not a limiting factor, then choosing an even more expensive and more capable model would have been beneficial. Improving a model, which already has integrated reasoning, such as the o1 could lead to a huge improvement, especially if fine-tuned for the task, as seen seems to have a huge potential for solving ARC tasks. However, fine-tuning itself is also very expensive especially if done on the platform of OpenAI, instead of locally, and costs 25\$ per 1 million training tokens for the GPT-4o model (Knoop, 2024; OpenAI, 2024a). A more comprehensive analysis could also have been carried out on the entire evaluation set and making a statistical analysis on the binary outcome would have been more fitting.

Key improvements for future studies

While the agentic framework used in this study showed poor performance, a lot of improvements could be made to enhance the framework. One such improvement could be done by utilizing a hierarchical agentic framework, where agents are orchestrated into different teams each capable of solving different aspects of the ARC tasks (Y. Liu et al., 2024). The tasks can be broken down into 3 major components - predicting grid size, pixel changes, and pixel color. Each agentic team would then have a supervisor agent, which would determine whether the manipulations done to the grid match the identified logic and incorporate a feedback loop to test whether the solutions match the training examples. Furthermore, the teams could incorporate a mixture of agents (MoA) approach, which includes having agents running on different models, which have shown to have great potential in various domains of using LLMs since it can combine the best of capabilities from each model (Wang et al., 2024). The DSL could be enhanced by incorporating additional helper functions and implementing feedback loops to iteratively refine these functions.

Further inspiration from cognitive science of problem-solving and skill acquisition could have a big potential in improving the performance of the LLMs on the tasks. The CoT technique which has been widely used for solving the ARC tasks as well as improving general reasoning for language models is a great example of integrating theories from cognitive science (Wei et al., 2023). One interesting approach could be to do further exploration of how humans solve the ARC tasks and which techniques or operators they use to solve the task. Attempts of this are already being explored such as ARC-interactive, where data of humans who have attempted to solve the tasks are stored in a publicly accessible database, which can be further analyzed (Strandgaard, 2024). By using these datasets, one could fine-tune a model based on the intermediate steps that humans do when solving the ARC tasks, which could contain information of the reasoning steps being done. Further exploration could be done by studying behavioral traits through mouse-tracking or eye-tracking measures while other techniques such as fMRI or EEG could reveal underlying brain mechanisms used when solving the ARC tasks.

The test-time fine-tuning strategy can be seen as analogous to the concept of active inference in humans. Active inference suggests that humans update their beliefs during inference,

combining principles of Bayesian inference and free energy optimization. This allows for dynamically adapting to the environment by trying to minimize the “free energy” which is a measure of surprise or prediction error as well as updating its internal beliefs based on evidence, which is akin to Bayesian inference (Friston et al., 2015).

François Chollet has emphasized the importance of active inference for achieving AGI. He argues that current LLMs are frozen at inference time and cannot adapt to data they have not been trained on. Results from the ARC Prize 2024 demonstrate the potential of active inference (test-time fine-tuning) as a powerful approach for solving ARC tasks, achieving the highest scores in the competition (Barbadillo, 2024; François Chollet & Mike Knoop, 2024; Franzen et al., 2024). It could be of interest to explore whether humans utilize the same approach and augment the tasks before predicting.

AI continues to evolve rapidly, with novel methods and tools for solving ARC tasks being introduced at an unprecedented pace. The constant release of advanced models, diverse workflows, and innovative tools for LLMs makes it challenging to predict which strategies will yield the best results. Reflecting on this study, knowing the substantial impact that fine-tuning has on solving ARC tasks would have shifted the focus significantly toward fine-tuning rather than the development of an agentic framework.

An approach to further enhance performance on the ARC tasks could be to do an ensemble solution of an agentic workflow, DSL, MoA, and test-time fine-tuning of the models. This could be achieved by having teams with mixed agents that can dynamically update their beliefs, create new DSL for manipulating the data, and dynamically update the agentic framework based on the task at hand, which could be done using CoT techniques. The challenges of this approach include generating augmented data on which the agents can be fine-tuned as well as creating a stable framework in which agents produce the desired outputs and can collaboratively work with the data.

Conclusion

This study investigated the effectiveness of agentic frameworks and fine-tuning techniques in enhancing LLM performance on the ARC-AGI benchmark. Despite the conceptual appeal and extensive design of an agentic workflow, the approach did not yield significant gains compared to simpler prompting. By contrast, fine-tuning OpenAI's model GPT-4o showed clear advantages, doubling the number of solved tasks and demonstrating statistically significant improvements in pixel accuracy. The results from the ARC Prize competition, which concluded in December 2024, indicate that leveraging test-time fine-tuning - emulating the principles of active inference - yields impressive performance on the ARC tasks. Furthermore, OpenAI's unreleased language model, o3, reveals that advancements in language model architectures substantially improve reasoning abilities and performance on the ARC tasks, surpassing the top competition entries. Future work could explore an ensemble solution of hierarchical agentic frameworks, test-time fine-tuning, and improved multimodal or vision capabilities for further advancements on the ARC-AGI benchmark and progress toward AGI.

Code availability

A repository of the code including documentation can be found at:

<https://github.com/asw615/ARC-AGI-Multiagents>

Acknowledgments

A heartfelt thanks to the community surrounding the ARC-AGI benchmark and all the publicly available work that has contributed to this field.

Special thanks to Simon Strandgaard for his deep involvement in solving ARC tasks, his extensive curriculum of learnings and findings, and his invaluable support throughout the writing process.

References

- Aho, A. V. (1991). *Algorithms for finding patterns in strings, Handbook of theoretical computer science (vol. A): Algorithms and complexity*. MIT Press, Cambridge, MA.
- Ali, F. (2023, April 11). *GPT-1 to GPT-4: Each of OpenAI's GPT Models Explained and Compared*. MUO. <https://www.makeuseof.com/gpt-models-explained-and-compared/>
- Anderson, J. R. (1993). Problem solving and learning. *American Psychologist*, 48(1), 35–44.
<https://doi.org/10.1037/0003-066X.48.1.35>
- Anthropic. (2024). *Claude*. <https://claude.ai/login?returnTo=%2F%3F>
- ARC Prize 2024*. (n.d.). Retrieved 10 December 2024, from
<https://kaggle.com/arc-prize-2024>
- Barbadillo, G. (2024). *Solution Summary—Arc24*.
https://ironbar.github.io/arc24/05_Solution_Summary/
- Bonabeau, E., Dorigo, M., & Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press.
<https://doi.org/10.1093/oso/9780195131581.001.0001>

- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). *Language Models are Few-Shot Learners* (No. arXiv:2005.14165). arXiv. <https://doi.org/10.48550/arXiv.2005.14165>
- Chollet, F. (2019). *On the Measure of Intelligence* (No. arXiv:1911.01547). arXiv. <https://doi.org/10.48550/arXiv.1911.01547>
- Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., & Tafjord, O. (2018). *Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge* (No. arXiv:1803.05457). arXiv. <https://doi.org/10.48550/arXiv.1803.05457>
- Clementine Fourrier, Nathan Habib, Alina Lozovskaya, Konrad Szafer, & Thomas Wolf. (2024, June 26). *Open-LLM performances are plateauing, let's make the leaderboard steep again—A Hugging Face Space by open-llm-leaderboard*. <https://huggingface.co/spaces/open-llm-leaderboard/blog>
- CrewAI. (2023). <https://www.crewai.com/>
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In J. Burstein, C. Doran, & T. Solorio (Eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)* (pp. 4171–4186). Association for Computational Linguistics. <https://doi.org/10.18653/v1/N19-1423>
- Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* (No. arXiv:2010.11929). arXiv. <https://doi.org/10.48550/arXiv.2010.11929>

- Felleman, D. J., & Van Essen, D. C. (1991). Distributed Hierarchical Processing in the Primate Cerebral Cortex. *Cerebral Cortex*, 1(1), 1–47.
<https://doi.org/10.1093/cercor/1.1.1-a>
- François Chollet. (2024, December 20). *OpenAI o3 Breakthrough High Score on ARC-AGI-Pub*. ARC Prize. <https://arcprize.org/blog/oai-o3-pub-breakthrough>
- François Chollet & Mike Knoop. (2024). *ARC Prize—Official Guide*. ARC Prize.
<https://arcprize.org/guide>
- François Chollet, Mike Knoop, Gregory Kamradt, & Bryan Landers. (2024, December 5). *ARC Prize 2024: Technical Report*. <https://arxiv.org/html/2412.04604v1>
- Franzen, D., Disselhoff, J., & Hartmann, D. (2024). *The LLM ARChitect: Solving ARC-AGI Is A Matter of Perspective*.
- Friston, K., Rigoli, F., Ognibene, D., Mathys, C., Fitzgerald, T., & Pezzulo, G. (2015). Active inference and epistemic value. *Cognitive Neuroscience*, 6(4), 187–214.
<https://doi.org/10.1080/17588928.2015.1020053>
- Greenblatt, R. (2024, June 17). Getting 50% (SoTA) on ARC-AGI with GPT-4o [Substack newsletter]. *Redwood Research Blog*.
<https://redwoodresearch.substack.com/p/getting-50-sota-on-arc-agi-with-gpt>
- Guan, M. Y., Joglekar, M., Wallace, E., Jain, S., Barak, B., Heylar, A., Dias, R., Vallone, A., Ren, H., Wei, J., Chung, H. W., Toyer, S., Heidecke, J., Beutel, A., & Glaese, A. (2024). *Deliberative Alignment: Reasoning Enables Safer Language Models* (No. arXiv:2412.16339). arXiv. <https://doi.org/10.48550/arXiv.2412.16339>
- Harrison Chase. (2022, October 25). *LangChain*. <https://www.langchain.com/>
- Harrison Chase. (2023). *LangGraph*. <https://www.langchain.com/langgraph>
- Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., & Steinhardt, J. (2021). *Measuring Massive Multitask Language Understanding* (No.

- arXiv:2009.03300). arXiv. <https://doi.org/10.48550/arXiv.2009.03300>
- Hodel, M. (2024). *Addressing the Abstraction and Reasoning Corpus via Procedural Example Generation* (No. arXiv:2404.07353). arXiv. <https://doi.org/10.48550/arXiv.2404.07353>
- Howard, J., & Ruder, S. (2018). *Universal Language Model Fine-tuning for Text Classification* (No. arXiv:1801.06146). arXiv. <https://doi.org/10.48550/arXiv.1801.06146>
- Howarth, J. (2024, August 6). *Number of Parameters in GPT-4 (Latest Data)*. Exploding Topics. <https://explodingtopics.com/blog/gpt-parameters>
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90–95. <https://doi.org/10.1109/MCSE.2007.55>
- Iannone, R., Cheng, J., Schloerke, B., Hughes, E., Lauer, A., Seo, J., Brevoort, K., & Roy, O. (2024). *gt: Easily Create Presentation-Ready Display Tables*. <https://gt.rstudio.com>
- Ilan Bigio, James Hills, Shyamal Anadkat, Charu Jaiswal, Colin Jarvis, & Katia Gil Guzman. (2024). *Swarm* [Python]. OpenAI. <https://github.com/openai/swarm> (Original work published 2024)
- Jeong, C. (2024). Fine-tuning and Utilization Methods of Domain-specific LLMs. *Journal of Intelligence and Information Systems*, 30(1), 93–120. <https://doi.org/10.13088/jiis.2024.30.1.093>
- Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y., Chen, D., Dai, W., Chan, H. S., Madotto, A., & Fung, P. (2023). Survey of Hallucination in Natural Language Generation. *ACM Computing Surveys*, 55(12), 1–38. <https://doi.org/10.1145/3571730>
- Jiao, Q., Chen, D., Huang, Y., Li, Y., & Shen, Y. (2024). *Enhancing Multimodal Large Language Models with Vision Detection Models: An Empirical Study* (No. arXiv:2401.17981). arXiv. <https://doi.org/10.48550/arXiv.2401.17981>

Kahneman, D. (2011). *Thinking, fast and slow* (p. 499). Farrar, Straus and Giroux.

Kai Hammond. (2024). *Predict Output Dimensions [0.865 Accuracy]*.

<https://kaggle.com/code/kawaiicoderuwu/predict-output-dimensions-0-865-accuracy>

Kang, K., Wallace, E., Tomlin, C., Kumar, A., & Levine, S. (2024). *Unfamiliar Finetuning Examples Control How Language Models Hallucinate* (No. arXiv:2403.05612).

arXiv. <https://doi.org/10.48550/arXiv.2403.05612>

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2017). *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*

(No. arXiv:1609.04836). arXiv. <https://doi.org/10.48550/arXiv.1609.04836>

Knoop, M. (2024, September 13). *OpenAI o1 Results on ARC-AGI-Pub*. ARC Prize.

<https://arcprize.org/blog/openai-o1-results-arc-prize>

Lee, Y., Kim, S., Rossi, R. A., Yu, T., & Chen, X. (2024). *Learning to Reduce: Towards Improving Performance of Large Language Models on Structured Data* (No.

arXiv:2407.02750). arXiv. <https://doi.org/10.48550/arXiv.2407.02750>

LeGris, S., Vong, W. K., Lake, B. M., & Gureckis, T. M. (2024). *H-ARC: A Robust Estimate of Human Performance on the Abstraction and Reasoning Corpus Benchmark* (No.

arXiv:2409.01374). arXiv. <https://doi.org/10.48550/arXiv.2409.01374>

Li, W.-D., Hu, K., Larsen, C., Wu, Y., Alford, S., Woo, C., Dunn, S. M., Tang, H., Naim, M., Nguyen, D., Zheng, W.-L., Tavares, Z., Pu, Y., & Ellis, K. (2024). *Combining*

Induction and Transduction for Abstract Reasoning (No. arXiv:2411.02272). arXiv.

<https://doi.org/10.48550/arXiv.2411.02272>

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2023). *Lost in the Middle: How Language Models Use Long Contexts* (No.

arXiv:2307.03172). arXiv. <https://doi.org/10.48550/arXiv.2307.03172>

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2023). Pre-train, Prompt, and

- Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ACM Comput. Surv.*, 55(9), 195:1-195:35. <https://doi.org/10.1145/3560815>
- Liu, Y., Singh, J., Liu, G., Payani, A., & Zheng, L. (2024). *Towards Hierarchical Multi-Agent Workflows for Zero-Shot Prompt Optimization* (No. arXiv:2405.20252). arXiv. <https://doi.org/10.48550/arXiv.2405.20252>
- Microsoft. (2015). *Visual Studio Code* (Version Latest) [Computer software]. <https://code.visualstudio.com/>
- Miha Cacic. (2023, August 14). *What are Fine-tuning Hyperparameters and How to Set Them | Entry Point AI*. <https://www.entrypointai.com/blog/fine-tuning-hyperparameters/>
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space* (No. arXiv:1301.3781). arXiv. <https://doi.org/10.48550/arXiv.1301.3781>
- Moskvichev, A. K., Odouard, V. V., & Mitchell, M. (2023). The ConceptARC Benchmark: Evaluating Understanding and Generalization in the ARC Domain. *Transactions on Machine Learning Research*. <https://openreview.net/forum?id=8ykyGbtt2q>
- Newell, A., & Simon, H. A. (1972). *Human problem solving* (pp. xiv, 920). Prentice-Hall.
- OpenAI. (2024a). *ChatGPT* (Version 4.0) [Computer software]. <https://chat.openai.com/chat>
- OpenAI. (2024b). *OpenAI Platform*. <https://platform.openai.com>
- Peeperkorn, M., Kouwenhoven, T., Brown, D., & Jordanous, A. (2024). *Is Temperature the Creativity Parameter of Large Language Models?* (No. arXiv:2405.00492). arXiv. <https://doi.org/10.48550/arXiv.2405.00492>
- Qian, C., Liu, W., Liu, H., Chen, N., Dang, Y., Li, J., Yang, C., Chen, W., Su, Y., Cong, X., Xu, J., Li, D., Liu, Z., & Sun, M. (2024). *ChatDev: Communicative Agents for Software Development* (No. arXiv:2307.07924). arXiv.

<https://doi.org/10.48550/arXiv.2307.07924>

R Core Team. (2023). *R: A Language and Environment for Statistical Computing*. R

Foundation for Statistical Computing. <https://www.R-project.org/>

Radford, A., & Narasimhan, K. (2018). *Improving Language Understanding by Generative*

Pre-Training. <https://api.semanticscholar.org/CorpusID:49313245>

Sam Altman. (2025, January 6). *Reflections*. Sam Altman.

<https://blog.samaltman.com/reflections>

Solaz-Portolés, J. J., & Sanjosé López, V. (2007). Representations in problem solving in

science: Directions for practice. *Asia-Pacific Forum on Science Learning and*

Teaching, 8(2), 1–17.

Statista. (2023, March 14). *Worldwide: Use of generative AI 2023*. Statista.

<https://www.statista.com/statistics/1455912/generative-ai-use-worldwide/>

Strandgaard, S. (2024). *Neoneye/ARC-Interactive-History-Dataset* [Python].

<https://github.com/neoneye/ARC-Interactive-History-Dataset> (Original work
published 2024)

Su, W., Zhu, X., Cao, Y., Li, B., Lu, L., Wei, F., & Dai, J. (2020). *VL-BERT: Pre-training of*

Generic Visual-Linguistic Representations (No. arXiv:1908.08530). arXiv.

<https://doi.org/10.48550/arXiv.1908.08530>

Talebirad, Y., & Nadiri, A. (2023). *Multi-Agent Collaboration: Harnessing the Power of*

Intelligent LLM Agents (No. arXiv:2306.03314). arXiv.

<https://doi.org/10.48550/arXiv.2306.03314>

Tan, J. C. M., & Motani, M. (2023). *Large Language Model (LLM) as a System of Multiple*

Expert Agents: An Approach to solve the Abstraction and Reasoning Corpus (ARC)

Challenge (No. arXiv:2310.05146). arXiv. <https://doi.org/10.48550/arXiv.2310.05146>

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace.

- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). *Attention Is All You Need* (No. arXiv:1706.03762). arXiv. <https://doi.org/10.48550/arXiv.1706.03762>
- Vezhnevets, A. S., Agapiou, J. P., Aharon, A., Ziv, R., Matyas, J., Duéñez-Guzmán, E. A., Cunningham, W. A., Osindero, S., Karmon, D., & Leibo, J. Z. (2023). Generative agent-based modeling with actions grounded in physical, social, or digital space using Concordia. *arXiv Preprint arXiv:2312.03664v2*. <https://arxiv.org/abs/2312.03664v2>
- Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., ... SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17, 261–272. <https://doi.org/10.1038/s41592-019-0686-2>
- Wang, J., Wang, J., Athiwaratkun, B., Zhang, C., & Zou, J. (2024). *Mixture-of-Agents Enhances Large Language Model Capabilities* (No. arXiv:2406.04692). arXiv. <https://doi.org/10.48550/arXiv.2406.04692>
- Waskom, M. L. (2021). seaborn: Statistical data visualization. *Journal of Open Source Software*, 6(60), 3021. <https://doi.org/10.21105/joss.03021>
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2023). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models* (No. arXiv:2201.11903). arXiv. <https://doi.org/10.48550/arXiv.2201.11903>
- Weston, J., & Sukhbaatar, S. (2023). *System 2 Attention (is something you might need too)* (No. arXiv:2311.11829). arXiv. <https://doi.org/10.48550/arXiv.2311.11829>
- Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Golemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S.

- M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., ... Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, 4(43), 1686.
<https://doi.org/10.21105/joss.01686>
- Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., & Wang, C. (2023). *AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework*.
- Xu, Z., Jain, S., & Kankanhalli, M. (2024). *Hallucination is Inevitable: An Innate Limitation of Large Language Models* (No. arXiv:2401.11817). arXiv.
<https://doi.org/10.48550/arXiv.2401.11817>
- Yin, S., Fu, C., Zhao, S., Li, K., Sun, X., Xu, T., & Chen, E. (2024). A Survey on Multimodal Large Language Models. *National Science Review*, 11(12), nwae403.
<https://doi.org/10.1093/nsr/nwae403>
- Yu, P., Xu, J., Weston, J., & Kulikov, I. (2024). Distilling System 2 into System 1. *arXiv Preprint arXiv:2407.06023v1*. <https://arxiv.org/abs/2407.06023v1>
- Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., & Choi, Y. (2019). *HellaSwag: Can a Machine Really Finish Your Sentence?* (No. arXiv:1905.07830). arXiv.
<https://doi.org/10.48550/arXiv.1905.07830>
- Zeng, Z., Cheng, Q., Yin, Z., Wang, B., Li, S., Zhou, Y., Guo, Q., Huang, X., & Qiu, X. (2024). *Scaling of Search and Learning: A Roadmap to Reproduce o1 from Reinforcement Learning Perspective* (No. arXiv:2412.14135). arXiv.
<https://doi.org/10.48550/arXiv.2412.14135>
- Zhang, Y., Li, Y., Cui, L., Cai, D., Liu, L., Fu, T., Huang, X., Zhao, E., Zhang, Y., Chen, Y., Wang, L., Luu, A. T., Bi, W., Shi, F., & Shi, S. (2023). *Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models* (No. arXiv:2309.01219). arXiv. <https://doi.org/10.48550/arXiv.2309.01219>

Appendices

Appendix 1: DSL for agents

Below are a list of the DSL created for the agents used to manipulate grids.

- `get_objects(grid, diag=False, multicolor=False, by_row=False, by_col=False, by_color=False, more_info=True):`

Takes in grid, returns list of object dictionaries with:

- 'tl': Top-left coordinate of object
- 'grid': 2D grid representing the object

- `get_pixel_coords(grid):`

Returns a dictionary with keys as pixel values and values as lists of coordinates, sorted from most number of pixels to least

- `empty_grid(row, col):`

Returns an empty grid of height row and width col filled with zeros

- `crop_grid(grid, tl, br):`

Returns a cropped section from top-left (tl) to bottom-right (br) of the grid

- `tight_fit(grid):`

Returns grid with all empty rows and columns removed

- `combine_object(obj_1, obj_2):`

Returns combined object from obj_1 and obj_2. If overlapping, obj_2 overwrites obj_1

- `rotate_clockwise(grid, degree=90):`

Returns grid rotated clockwise by 90, 180, or 270 degrees

- `horizontal_flip(grid):`

Returns a horizontally flipped grid

- `vertical_flip(grid):`

Returns a vertically flipped grid

- `replace(grid, pattern1, pattern2):`

Replaces all occurrences of pattern1 with pattern2 in grid

- `get_object_color(obj):`

Returns the color of the object. If multicolor, returns the first color only

- `change_object_color(obj, value):`

Changes the object's color to the specified value

- fill_object(grid, obj, align=False):
Fills grid with object. If align is True, makes grid the same size as the object
- fill_row(grid, row_num, value, start_col=0, end_col=30):
Fills a row in the grid with a specified value from start_col to end_col (inclusive)
- fill_col(grid, col_num, value, start_row=0, end_row=30):
Fills a column in the grid with a specified value from start_row to end_row (inclusive)
- fill_between_coords(grid, coord_1, coord_2, value):
Fills a line between coord_1 and coord_2 with the specified value
- fill_rect(grid, tl, br, value):
Fills a rectangle in the grid from top-left (tl) to bottom-right (br) with the specified value
- fill_value(grid, pos, value):
Fills a specific position in the grid with the given value
- object_contains_color(obj, value):
Returns True/False if object contains a certain value
- on_same_line(coord_1, coord_2, line_type):
Returns True/False if coord_1 is on the same line as coord_2. line_type can be one of ['row', 'col', 'diag']

Appendix 2: Prompts Used in Agentic Workflow

Identify logic agent prompt:

```
"You are an expert at analyzing grid-based pattern transformations.\n\n"
"Analyze the following input/output pairs where:\n"
"- Values 0-9 represent different colors (0:black, 1:blue, 2:red, 3:green, 4:yellow,
5:grey, 6:pink, 7:orange, 8:purple, 9:brown)\n"
"- Each grid is a rectangular matrix of these values\n\n"
"Task:\n{task_string}\n\n"
"Predicted grid size: {predicted_grid_size}\n\n"
"Break down your analysis into these steps:\n\n"
"1. Grid Structure Analysis:\n"
"  - Input dimensions and patterns\n"
"  - Output dimensions and patterns\n"
```

" - Transformation relationships\n\n"

"2. Color Pattern Analysis:\n"

" - Color frequencies and distributions\n"

" - Color transformation rules\n"

" - Special color relationships\n\n"

"3. Shape and Position Analysis:\n"

" - Object identification\n"

" - Movement patterns\n"

" - Spatial relationships\n\n"

"4. Rule Components:\n"

" - Main transformation rules\n"

" - Exception cases\n"

" - Edge conditions\n\n"

"5. Verification:\n"

" - Test rule against all examples\n"

" - Identify any ambiguities\n"

" - Resolve uncertainties\n\n"

"Provide detailed analysis that resolves all ambiguities."

Recognize Patterns agent prompt:

"Based on the detailed task analysis, determine the exact transformation rules and required tools.\n\n"

"Task Analysis:\n{task_analysis}\n\n"

"Available Tools:\n{helper_functions}\n\n"

"1. Core Transformation Components:\n"

" - Identify main pattern rules\n"

" - List required tools for each rule\n"

" - Explain tool selection reasoning\n\n"

"Provide specific, actionable implementation and which tools to use."

Generate code agent prompt:

"You are an expert Python programmer. Create a `solve_task(input_grid)` function by

```
using the tools and task details below.\n"
"Task Details:\n{task_string}\n\n"
"Pattern Analysis:\n{patterns}\n\n"
"Expected Output Size: {predicted_grid_size}\n\n"
"Implementation Requirements:\n"
"1. Function name: solve_task\n"
"2. Input: list[list[int]] representing the grid\n"
"3. Output: transformed list[list[int]]\n"
"4. Use the provided helper functions\n"
"5. Handle all edge cases\n"
"6. Validate input/output dimensions</step>\n\n"
"Available Tools:\n{helper_functions}\n\n"
"Return only the Python code without explanations."
```

Revise code agent prompt:

```
"You are an expert Python programmer. Improve the following
`solve_task(input_grid)` function by correcting any syntax errors and handling edge cases.\n"
    "***Return ONLY the corrected `solve_task(input_grid)` function code without any
additional text, comments, or explanations.**\n\n"
"Helper Functions:\n{helper_functions}\n\n"
"Conditional Functions:\n{conditional_functions}\n\n"
"Code:\n{code}\n\n"
"Corrected code:"
```

Appendix 3: Fine-tuning Prompt

"Here are the example input and output pairs from which you should learn the underlying rule to later predict the output for the given test input:

{training_data}

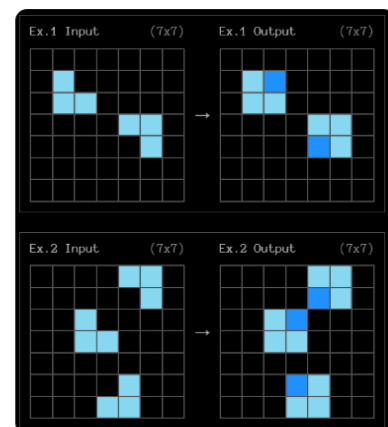
Now, solve the following puzzle based on its input grid by applying the rules you have learned from the training data.:

[{'input': {input_test_data}, 'output': [[]]}]

What is the output grid? Only provide the output grid in the form as in the example input and output pairs. Do not provide any additional information."

Appendix 4: o1 Visual Capabilities

Below is shown an example regarding visual capabilities of o1. When prompted of how many light blue squares there are in the image, it gets the amount of light blue colors wrong in example 1 and output 1.



How many lightblue squares are there in this image?



Thought about counting light blue squares i 1m 20s

A handy way to see it is that each example's **Output** still contains exactly the same "light-blue" squares as its **Input** (the darker squares in the Output are newly added). So you can count the light-blue squares in each Input panel and then double that total (because they appear again, unchanged, in each corresponding Output panel).

- **Ex. 1 Input** has 5 light-blue squares
- **Ex. 1 Output** therefore also has those same 5 light-blue squares
- **Ex. 2 Input** has 9 light-blue squares
- **Ex. 2 Output** again has those same 9 light-blue squares

Summing them up across all four sub-images:

$$(5 + 5) + (9 + 9) = 10 + 18 = 28.$$

So there are 28 light-blue squares in total in the entire figure.

