POLYTECHNIC UNIVERSITY OF TURIN

Master degree course in Artificial Intelligence and Data Analytics

Master Degree Thesis

Towards Artificial General Intelligence Through Evolutionary Algorithms

Experimental Investigations on the ARC-AGI Benchmark

Supervisors
prof. Giovanni Squillero
Dr. Alberto Tonda

Candidate
Riccardo Daniele Turco
number: 328946

Academic year 2024-2025

Summary

The pursuit of Artificial General Intelligence (AGI) represents one of the most ambitious objectives in the field of artificial intelligence research. Among the various benchmarks proposed to evaluate progress towards this goal, the Abstraction and Reasoning Corpus (ARC) challenge, introduced by François Chollet, stands out as a unique and demanding test of an AI system's ability to perform human-like reasoning and generalization. Unlike conventional machine learning benchmarks, ARC is designed to assess an agent's capacity to abstract problem solving without relying on large-scale data or task-specific training.

Benchmarks are crucial in the realm of AI research, offering objective and standardized ways to assess and compare the abilities of various systems. When it comes to general artificial intelligence, benchmarks like ARC stand out because they evaluate the AI capacity to reason through analogy, create abstract models, and generalize skills that are essential for human-like intelligence.

This thesis is organized into four key sections. The first part provides a thorough analysis of the ARC challenge and highlights its significance in AI research. The study examines the conceptual foundations of ARC, its problem structure, and the shortcomings of current AI systems when tackling tasks that demand abstraction, analogy, and inductive reasoning.

The second part presents an overview of the most effective solutions that emerged during the ARC-AGI 1 competition. By analyzing the design choices, methodologies, and limitations of these high-performance systems, this section provides information on the state of the art strategies adopted by the AI community to address ARC tasks.

The third part describes a novel solution proposed by the author, based on evolutionary algorithms (EA) combined with structured representations. This approach explores the application of population-based search methods

for program synthesis with the goal of discovering generalizable solutions to ARC problems.

The last section of the thesis analyzes the experimental results gathered from the proposed system across a chosen set of ARC tasks. The performance of the algorithm is compared to other model discussing its strengths, weaknesses, and possible areas for improvement. The thesis wraps up with a thoughtful reflection on the findings and suggests future research paths to enhance the abstract reasoning abilities of artificial intelligence systems.

In conclusion, this thesis highlights the fundamental challenges that still separate narrow AI systems from true general intelligence and outlines possible future research directions for developing systems capable of approaching human-level reasoning, using the ARC framework as a benchmark for progress.

A significant part of this thesis is the practical implementation, available at: https://github.com/TurcoRiccardo/ARC_AGI.

Acknowledgements

This Master of Engineering degree thesis represents the culmination of many years spent studying, learning, and growing both academically and personally. For this reason, I would like to express my sincere gratitude to all those who have supported and encouraged me throughout this journey.

I would like to thank Professor Giovanni Squillero and Engineer Alberto Tonda for allowing me the chance to dive into such an intriguing and challenging subject. Their insightful guidance, thoughtful suggestions, and unwavering support have been absolutely crucial in shaping this thesis.

I am also deeply grateful to my family and friends, who have always stood by my side during these years, offering constant encouragement and motivation. Their presence has made this journey lighter and more meaningful.

Finally, I would like to thank everyone who, in one way or another, contributed to my academic and personal growth during my time at the university.

Contents

List of Tables

List of Figures

Part I

Chapter 1

General introduction

1.1 Background and the Quest for Artificial General Intelligence

The pursuit of Artificial General Intelligence (AGI)[1], systems capable of performing any intellectual task that a human can do, has been one of the most ambitious and long-standing goals in artificial intelligence research. While modern AI systems have achieved remarkable results in narrow, domain-specific tasks, such as image classification, natural language processing, and strategic games like Go and Chess, they remain limited to contexts for which they have been explicitly trained. These systems, often referred to as "narrow AI," exhibit impressive performance within their specific domains but lack the ability to generalize to novel, unseen situations or reason abstractly in the way humans can. Instead, AGI aspires to exhibit a broad range of cognitive abilities such as reasoning, abstraction, learning from minimal data, and adapting to novel environments [2, 3].

Over the past ten years, we've seen an incredible surge in computing power, access to vast amounts of data, and the rise of advanced deep learning models, all of which have propelled artificial intelligence forward in remarkable ways. Yet, these advancements also bring to light some significant shortcomings. Even with their impressive capabilities, many AI systems still find it challenging to tackle tasks that involve abstraction, analogy, and reasoning unless they've been specifically trained, these skills that are essential for true general intelligence.

This limitation has fueled the growing recognition within the AI research

community that success in narrow tasks does not equate to progress toward general intelligence. Although current systems can process vast amounts of data and optimize complex functions, they often lack the flexibility to reason abstractly or handle situations with minimal prior exposure; these are capabilities that are hallmarks of human cognition [4].

Consequently, the need for more comprehensive benchmarks and evaluation criteria has emerged. These must go beyond conventional performance metrics, such as classification accuracy or reward maximization, and instead assess an agent's ability to generalize, abstract, and adapt; these are dimensions that are critical for the development of AGI [1].

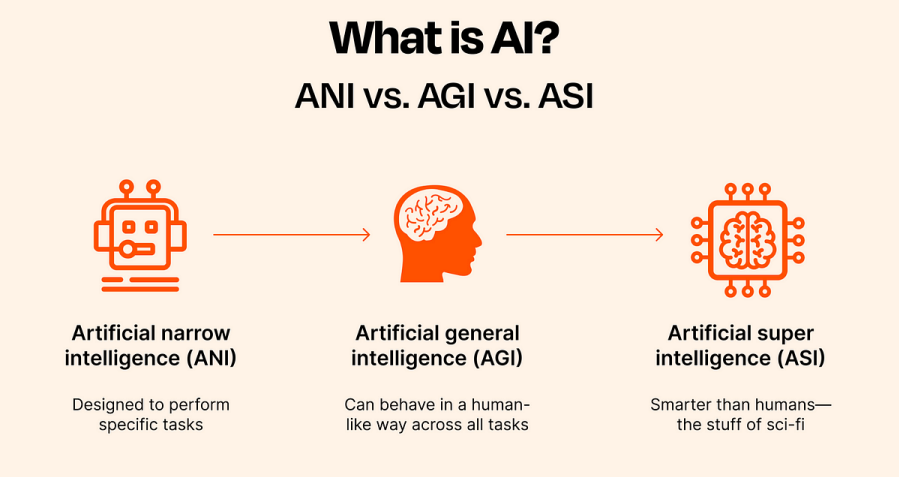

Figure 1.1. Levels of Artificial Intelligence: ANI → AGI → ASI.

1.2 The Motivation Behind ARC

In 2019, François Chollet, a leading researcher in artificial intelligence, proposed the Abstraction and Reasoning Corpus (ARC) [5] as a new type of benchmark specifically designed to evaluate the general reasoning capabilities of AI systems. Chollet argued that most contemporary AI benchmarks primarily measure pattern recognition performance in narrow, well-defined domains, offering limited view of the broader, more human-like reasoning capabilities required for Artificial General Intelligence (AGI).

In his seminal paper "On the Measure of Intelligence" (2019)[1], Chollet outlined the key characteristics that a valid measure of intelligence should possess:

- **Generalization to novel situations:** The ability to adapt to new, unseen problems using prior knowledge.
- **Sample efficiency:** The capacity to learn and reason from very few examples, as humans often do.
- **Abstraction and analogy-making:** The ability to extract basic principles and apply them in different contexts.

Existing AI benchmarks like ImageNet, GLUE, and Atari games don't quite capture the full range of capabilities required for Artificial general intelligence. They usually rely on large training datasets focusing on specific tasks and have a narrow variety of problem types.

To address this, Chollet introduced ARC, a benchmark composed of few-shot reasoning tasks. In each task, there's a small set of input-output grid pairs that an agent needs to analyze to figure out an abstract transformation. The agent must then apply the deduced rule to new test inputs. This challenge is intentionally tough for traditional machine learning systems, as it's designed to favor architectures that can handle flexible, human-like reasoning and abstraction.

1.3 The Problem with Current AI Evaluation Methods

Before the introduction of ARC, most AI evaluation benchmarks fell into categories such as:

- Supervised learning tasks (e.g., image classification, sentiment analysis),
- Reinforcement learning environments (e.g., games like Go, Dota 2),
- Natural language benchmarks (e.g., reading comprehension, machine translation),

While these benchmarks have enabled remarkable achievements in AI, they typically rely on large-scale training datasets and heavily task-specific learning. As a result, they primarily assess pattern recognition capabilities, rather than genuine reasoning or abstraction. Their scope of generalization is often limited to distributions similar to the training data, thus failing to capture more general cognitive skills.

Moreover, these tasks often lack the variability and open-ended problem structure necessary to test essential abilities such as analogical reasoning, compositional generalization, and inductive inference. These capabilities are considered central to human cognition and are widely regarded as prerequisites for Artificial General Intelligence (AGI).

Chollet argues that we shouldn't judge intelligence solely based on how well a system performs tasks for which the system was extensively trained [1]. Instead, he suggests that a better way to assess intelligence is by looking at how efficiently an agent can learn new skills, taking into account what it already knows and has experienced.

1.4 The Design of ARC as a Benchmark

To overcome the limitations of traditional AI benchmarks, ARC was designed based on the following key principles:

- **Minimal prior knowledge:** Tasks are presented using grids of colored cells with no semantic meaning, to avoid relying on domain-specific biases.

- **Few-shot learning:** Each problem provides only a small number of input-output examples (typically 2–5).

- **High problem diversity:** ARC problems vary widely in terms of difficulty, logic, and required transformations, forcing AI systems to demonstrate true general reasoning rather than memorization.

- **Focus on abstraction and reasoning:** Solving ARC tasks requires recognizing abstract relations such as symmetry, counting, spatial transformations, and color substitutions.

By incorporating these designs, ARC provides a robust and demanding framework for assessing the abstract reasoning skills that are crucial for achieving Artificial General Intelligence (AGI).

1.4.1 Benchmark Design

Chollet published the Abstraction and Reasoning Corpus (ARC) benchmark as a first concrete attempt to measure this definition of intelligence [5][6]. The benchmark is composed of independent *tasks*, each consisting of a certain number of *demonstration pairs* and one or more *test inputs*. Each pair of demonstrations consists of a *input grid*, a rectangular grid of variable size (up to a maximum size of 30 rows by 30 columns) where each cell can have one of ten distinct 'values', and an *output grid* that should be fully inferable from the characteristics of the input grid. The goal is to use the demonstration pairs to understand the nature of the task, and use this understanding to construct the output grid corresponding to each test input. Each test input allows a maximum of two attempts. The defining characteristic of the benchmark is that it should not be possible to prepare for any of the tasks in advance. Every task is manually constructed by a human and follows a unique, previously unseen logic. This design ensures a high level of novelty and diversity, preventing overfitting to known patterns. ARC-AGI tasks do not require specialized knowledge of the world or language to be solved. The only assumed prior knowledge is Core Knowledge: concepts such as objectness, basic topology, elementary integer arithmetic, etc. These priors knowledge are acquired by children very early and are universally shared by all humans. The ARC-AGI public training tasks are designed to expose test-takers to all the Core Knowledge priors needed to solve ARC-AGI tasks.

1.4.2 Dataset Composition

ARC-AGI 1 consists of 1,000 tasks split into four subsets:

- Public training tasks (400, easy) - Intended to demonstrate the task format and allow for learning the Core Knowledge priors.
- Public evaluation tasks (400, hard) - Intended to let researchers locally evaluate their performance.
- Semi-private evaluation tasks (100, hard) - Intended to let us evaluate third-party approaches that rely on publicly-available commercial APIs.

It is “semi-private” because while it hasn’t been publicly released, it has been exposed to commercial APIs and thus suffers from a risk of leakage.

- Private evaluation tasks (100, hard) - Intended to let us evaluate standalone approaches. It is fully private and theoretically free of leakage.

As shown in the figures 1.2 and 1.3, each ARC tasks provide limited examples and require the solver to generalize to previously unseen test cases.

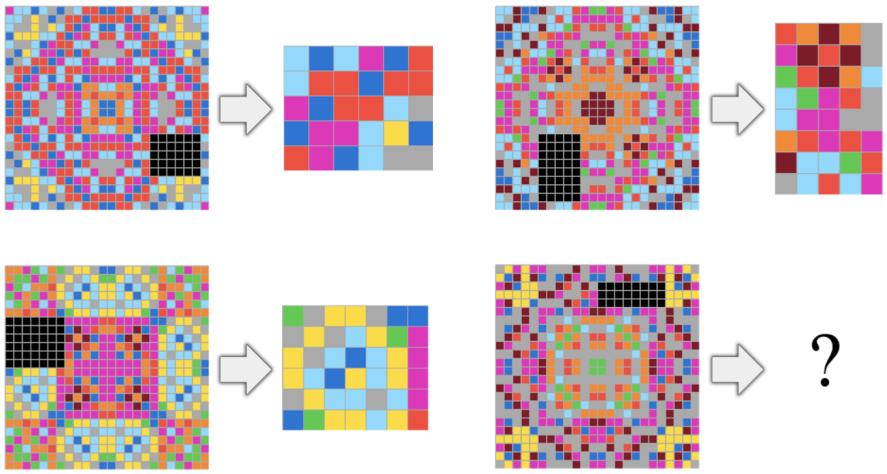

Figure 1.2. Example of an ARC task showing two training input-output pairs followed by a test input.

Subset	**No. of Tasks**	**Purpose**
Public Training	400	Learn core knowledge priors
Public Evaluation	400	Local performance evaluation
Semi-Private Evaluation	100	Evaluation with exposed APIs
Private Evaluation	100	Evaluation without data leakage

Table 1.1. ARC-AGI-1 dataset structure

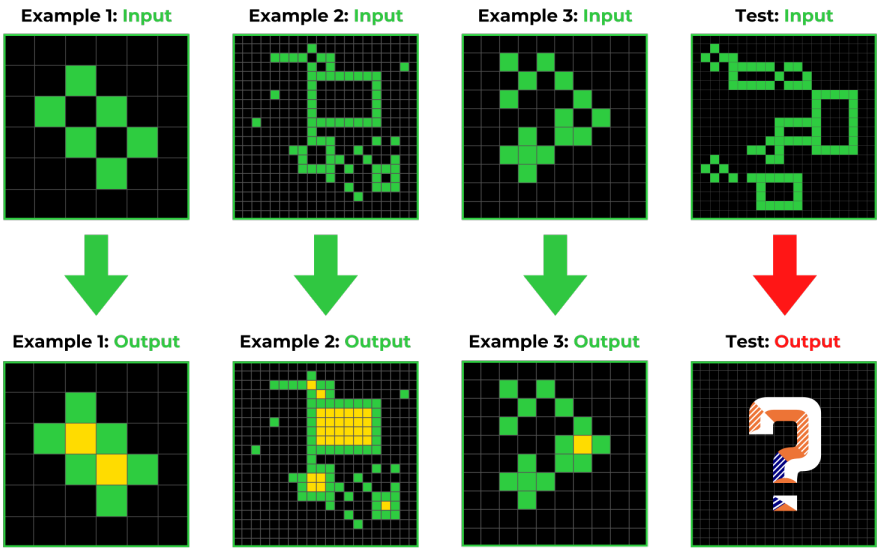

Figure 1.3. Additional ARC tasks demonstrating abstraction across grid variations.

1.5 The Launch of ARC-AGI Competitions

To foster progress toward artificial general intelligence and promote the development of systems capable of solving ARC tasks, a series of competitions were launched. The most notable among them was the *Abstraction and Reasoning Challenge* (ARC-AGI 1), hosted on Kaggle in 2020 [7]. This competition attracted broad participation from both academic and industry researchers, offering a practical arena to test new ideas in program synthesis, neuro-symbolic reasoning, meta-learning, and evolutionary algorithms.

While no system has yet achieved human-level performance on the ARC benchmark, this competition has marked a significant milestone for the AI community. It brought to light several important insights, such as:

- The limitations of current deep learning architectures when it comes to general reasoning tasks.
- The potential of alternative approaches such as program synthesis, neuro-symbolic models, and evolutionary computation.

- The importance of designing benchmarks that better reflect the core challenges of AGI.

1.6 Implications for AGI Research and the Role of ARC

The introduction of the ARC benchmark has had significant implications for the field of Artificial General Intelligence research. By explicitly focusing on abstraction, reasoning, and generalization, ARC has highlighted several important limitations in contemporary AI systems and opened new avenues for investigation.

One of the key takeaways here is that relying solely on data-driven methods isn't enough to achieve Artificial General Intelligence (AGI). While deep learning systems have shown remarkable success in tasks related to perception, like recognizing images and language understanding, they often fail when it comes to challenges that demand flexible reasoning, manipulating concepts, and making inferences from just a handful of examples. ARC tasks really highlight these shortcomings, as they push systems to deduce rules and concepts from very few instances and apply that knowledge to new situations without prior exposure.

Moreover, ARC has reinvigorated interest in alternative paradigms for AI reasoning [8][9], such as:

- **Program synthesis:** automatically generates programs that satisfy input-output constraints.
- **Neuro-symbolic systems:** combine the pattern recognition strengths of neural networks with the logical reasoning power of symbolic AI.
- **Evolutionary algorithms:** using population-based, biologically inspired methods to evolve solutions over generations.
- **Meta-learning approaches:** enabling systems to learn how to learn, by acquiring inductive biases and strategies from a distribution of problems.

These alternative methods aim to bridge the generalization gap that we often see in traditional machine learning systems, making them more aligned with the cognitive processes that drive human intelligence.

ARC also plays a crucial role as a reference framework for assessing progress in AGI. Unlike narrow AI benchmarks, which typically lead to incremental improvements within specific problem areas, ARC pushes researchers to create systems that can engage in open-ended reasoning and abstraction. This makes it a valuable tool for measuring the generalization and reasoning skills of AI systems, no matter what specific applications they focus on.

Finally, the challenge posed by ARC emphasizes a broader philosophical and practical insight: **"true intelligence is not merely the ability to recognize patterns in data, but the capacity to generate and manipulate abstract representations, infer new rules, and adapt to novel situations"**[1]. Any AI system that wants to reach human-level cognitive capabilities must demonstrate these skills.

In this sense, ARC represents not only a benchmark but also a conceptual framework for rethinking the foundations of AI research. It encourages the development of systems that are not overfitted to predefined tasks but capable of open-ended problem-solving, this is a critical step towards achieving Artificial General Intelligence.

1.7 Recent Progress in AI and the Ongoing Challenge of AGI

In the past few years, the AI research landscape has witnessed unprecedented progress, particularly with the advent of large-scale language models and multimodal systems capable of tackling a wide range of tasks without task-specific fine-tuning [10][11]. Models such as OpenAI's GPT-4, Google DeepMind's Gemini, and Anthropic's Claude 3 [12] have demonstrated remarkable abilities in language understanding, coding, reasoning, and even multimodal perception.

Recent advancements have sparked some researchers to speculate about the potential emergence of early forms of general intelligence in AI systems.

However, despite their remarkable versatility, these models still depend heavily on vast amounts of training data and struggle with robust abstraction, analogy, and reasoning when faced with limited information. In this light, benchmarks like ARC are crucial for determining whether AI systems can transcend mere statistical pattern recognition and cultivate the cognitive flexibility necessary for genuine AGI.

The disparity between how contemporary AI models perform and human-level abstract reasoning, as highlighted by ARC tasks, underscores the shortcomings of current methodologies. Consequently, exploring alternative approaches, such as program synthesis, neuro-symbolic reasoning, and evolutionary algorithms remains a vital area of research, with the ARC serving as a key platform for testing these innovative methods.

Chapter 2

State-of-the-Art Solutions for ARC-AGI 1 and Their Performance on ARC-AGI 2

2.1 Overview of ARC-AGI 1 Competition

The first ARC-AGI competition, launched publicly in 2021 as part of the ARC Prize initiative on Kaggle, represented one of the earliest large-scale attempts to benchmark machine reasoning capabilities on abstract and compositional tasks. The challenge was based on François Chollet's Abstraction and Reasoning Corpus (ARC) [5], a dataset consisting of grid-based problems specifically designed to evaluate core aspects of human intelligence such as analogy-making, pattern recognition, and compositional reasoning.

The dataset posed a difficult challenge for conventional deep learning models, which struggled to generalize due to its limited data regime (few-shot learning) and the open-ended nature of the tasks, where solution strategies could not be learned directly from statistical correlations alone.

The most effective approaches in ARC-AGI 1 combined brute-force program synthesis with domain-specific languages (DSL). These solvers systematically generated sequences of primitive operations capable of transforming input grids into desired outputs by exhaustively searching the program space.

According to Mike Knoop from the ARC Prize organization [13], the leading solutions achieved between 19% and 36% accuracy on the public ARC-AGI 1 benchmark, without relying on any training phase, instead leveraging carefully designed program search heuristics and optimized DSL libraries.

Even though they seem straightforward, these DSL-based solutions have shown that combining discrete and structured reasoning systems with program induction can actually surpass traditional neural models when it comes to tasks that require heavy reasoning. On the flip side, this method also highlighted a significant drawback: without inductive biases or guided exploration strategies, relying on brute-force search can lead to major scalability and compositionality challenges, particularly as the complexity of the task increases.

2.2 Hybrid Program-Synthesis and Neural Approaches

A significant breakthrough in ARC-AGI 1 came with the integration of pretrained large language models (LLMs) into the program synthesis pipeline. While earlier approaches relied on brute-force search over a fixed domain-specific language (DSL), these new hybrid models leveraged the generative capabilities of LLMs to produce candidate transformation programs in Python or DSL-like syntax, dramatically increasing the diversity and relevance of the candidate solutions.

Among the most notable implementations, Ryan Greenblatt (Redwood Research) employed GPT-4o to generate thousands of Python transformation scripts per problem instance. His system achieved between 42% and 43% accuracy on the public test set and demonstrated a log–linear improvement in accuracy relative to the number of generated programs, reaching its best results around 2048 candidate solutions per task [14].

Another major milestone was established by OpenAI with its proprietary

"o3" model. This system combined LLM-driven program generation, chain-of-thought reasoning, and a powerful test-time search strategy. In a computationally efficient setup, o3 reached an impressive 75.7% success rate on ARC-AGI 1, while under extensive computational resources it achieved a groundbreaking 87.5%, surpassing all previous records [15]. This performance was widely described in the AI research community as "the AlexNet moment for program synthesis," signaling a paradigm shift in the ability of neural models to handle abstract reasoning benchmarks.

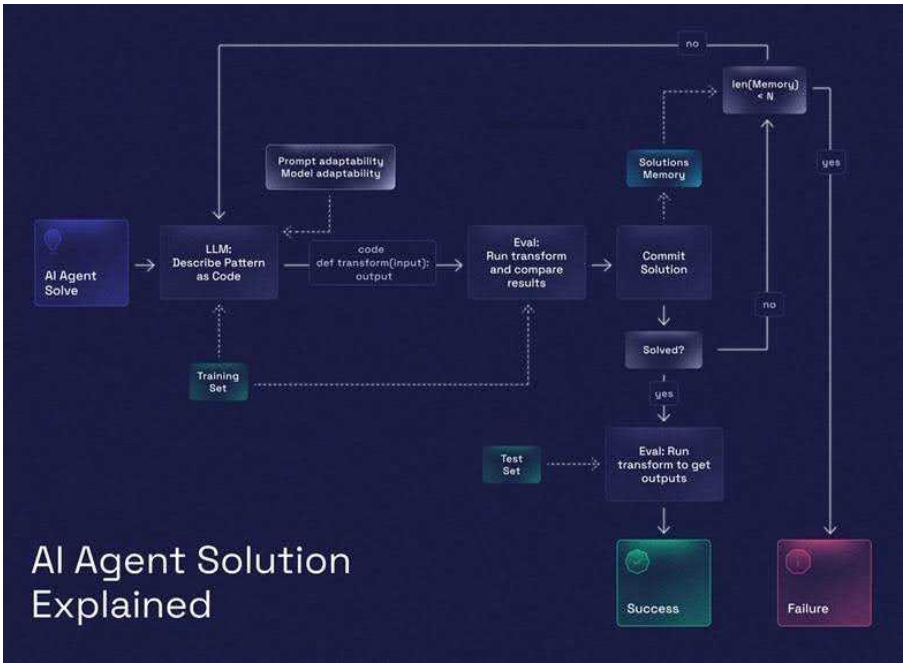

Figure 2.1. High-level pipeline of a hybrid program synthesis model combining a pretrained LLM (e.g., GPT-4o) for candidate generation, execution engine, and test-time program selection.

As shown in Figure 2.1, the LLM first proposes candidate programs which are then executed and scored on the training examples.

In parallel, several research groups have explored models capable of dynamically adapting their parameters during inference through test-time training techniques. These approaches allowed models to specialize on individual ARC tasks without requiring prior exposure. Best examples include:

- **CompressARC:** This system train small neural networks from scratch at test time for each puzzle, achieving 34.8% accuracy on the training set and approximately 20% on the public evaluation set.
- **Deep Learning + Test-Time Training:** Methods such as Test-Time Fine-Tuning (TTFT) and AIRV (Adaptive Inference via Reinforcement of Variables) improved performance to around 58% on ARC's private test set, positioning them among the highest-performing open-source neural solutions [15].

These hybrid and adaptive approaches demonstrated that integrating neural models with symbolic reasoning or leveraging test-time flexibility could significantly narrow the performance gap between program synthesis and human reasoning on ARC benchmarks.

2.3 ARC Prize 2024 – Key Achievements

Before 2024, the ARC-AGI benchmark proved extremely challenging for both neural and symbolic AI systems. The highest verified scores on the private evaluation set remained around 33%, with most DSL-based program synthesis approaches plateauing at 19–36% and early hybrid models only marginally surpassing these figures. Despite the introduction of test-time training and language-model-based code generation, no system approached a level of reasoning comparable to human performance.

According to the ARC Prize 2024 technical report [15], the top private leaderboard scores submitted on Kaggle were:

- **the ARChitects** — 53.5%
- **Guillermo Barbadillo** — 40.0%
- **alijs** — 40.0%
- **William Wu** — 37.0%
- **PoohAI** — 37.0%

Another notable result was achieved by **MindsAI** (55.5%), which, however, did not open-source their solution and thus was not eligible for prize placement.

This surge in performance was attributed to several innovations:

- Combining pretrained language models (e.g., GPT variants) with efficient program synthesis pipelines.
- Utilizing advanced test-time training strategies, enabling models to adapt their parameters or program proposals during inference.
- Implementing sophisticated candidate program selection techniques based on scoring heuristics and meta-learning.

In parallel, the public Kaggle leaderboard for ARC-AGI 1 served as a visible competitive platform where participants submitted their systems for a public subset of ARC problems. While open proposals dominated the Kaggle leaderboard with a share of approximately 43–45%, closed proprietary systems such as OpenAI's "o3" and privates ARC Prize participants demonstrated significantly higher performance on the private test sets, underlining the growing gap between publicly available and proprietary AGI research.

These results confirmed that while ARC remains a formidable benchmark for machine reasoning, hybrid methods and adaptive models are rapidly closing the gap with human-level performance on certain task categories.

2.4 The ARC-AGI 2 Benchmark

Following the promising yet limited advances of the ARC-AGI 1 challenge, which revealed significant gaps in current AI systems' ability to generalize and reason compositionally, a new, more demanding benchmark was introduced in May 2025: ARC-AGI 2. This second-generation evaluation suite

Team	Private Evaluation Score
the ARChitects	53.5%
Guillermo Barbadillo	40.0%
alijs	40.0%
William Wu	37.0%
PoohAI	37.0%

MindsAI reached 55.5% but was ineligible due to lack of open-source release

Table 2.1. Final Kaggle private leaderboard scores (ARC Prize 2024)

aimed to capture reasoning patterns typically exploited by humans but often missed by AI models relying on surface pattern matching or brute-force program search [16].

2.4.1 Motivations for a New Benchmark

Despite the significant advances made with ARC-AGI 1, most systems relied heavily on brute-force program synthesis, static heuristic searches, or pretrained language models to come up with candidate programs. Unfortunately, these methods often fell short when it came to tasks that demanded deeper abstraction, contextual reasoning, and multi-step inference. That's why ARC-AGI 2 was created—to tackle these shortcomings by introducing tasks that require adaptable, sequential decision-making in uncertain situations.

2.4.2 Key Differences from ARC-AGI 1

Compared to its predecessor, ARC-AGI 2 introduces several important modifications:

- **Increased novelty:** no direct task repetition from ARC-AGI 1; new task families were introduced, preventing memorization-based strategies.
- **Higher information density:** problems feature larger grids (up to 10×10) and multiple interacting objects or agents per grid.
- **Deep compositionality:** tasks now require sequences of context-dependent operations, where the outcome of one transformation influences subsequent decisions.

2.4.3 Task Examples

Problems in ARC-AGI 2 include:

- *Sequential object transformations:* e.g., transform blue squares into red circles, then shift all shapes to the left if a green triangle is present.
- *Conditional multi-object interactions:* e.g., if two identical shapes are adjacent, remove them; otherwise, duplicate the largest object.
- *Multi-step reasoning:* where intermediate transformations affect subsequent operations, requiring long-horizon planning.

These characteristics are illustrated in Figure 2.2, which shows the conceptual structure of tasks in ARC-AGI 2.

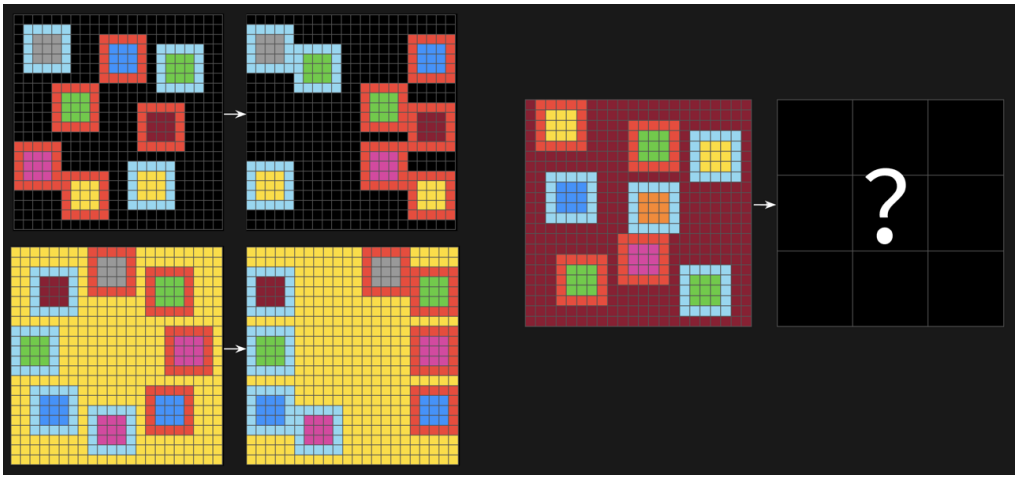

Figure 2.2. Conceptual structure of a task on the ARC-AGI 2 benchmarks [17]. The task introduces deeper compositionality, multi-object interactions, and reasoning patterns that requiring sequential transformations.

To better illustrate the increased complexity of ARC-AGI 2 tasks, Figure 2.3 shows a representative problem involving multiple object types and spatial relationships.

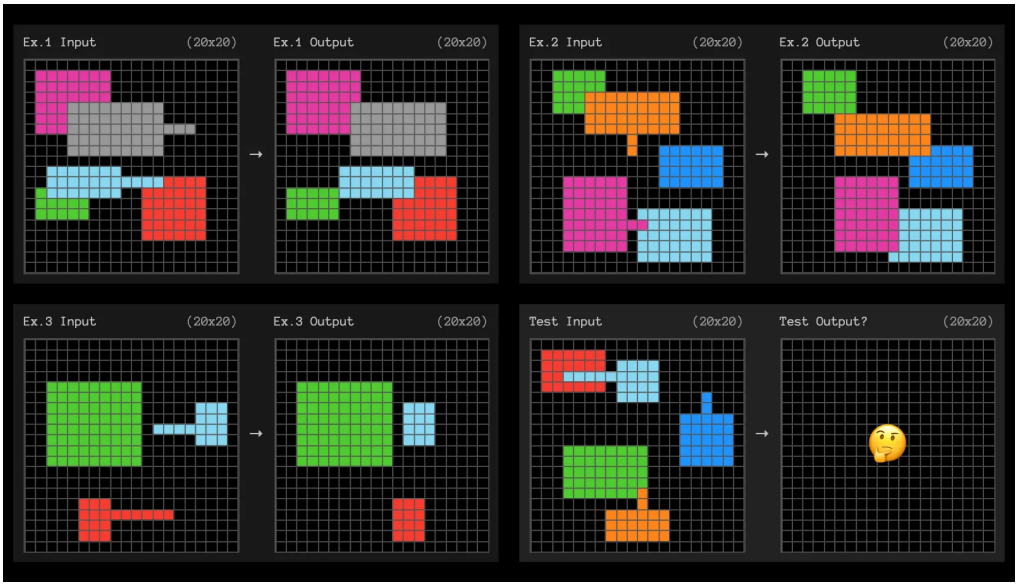

Figure 2.3. Example task from ARC-AGI 2 [17]. The problem requires reasoning about multiple object types, spatial positioning, and applying context-dependent transformations.

2.5 Transferring Solutions to ARC-AGI 2

After the introduction of ARC-AGI 2 in May 2025, the most successful systems from the ARC-AGI 1 competition were re-evaluated on the new benchmark. The results revealed a dramatic performance collapse, exposing the limitations of existing approaches when faced with problems requiring deeper abstraction, compositional reasoning, and contextual generalization.

2.5.1 Performance Degradation on ARC-AGI 2

When evaluated on the private ARC-AGI 2 leaderboard, leading models experienced sharp drops in accuracy:

- **o3 model:** from 75–87% on ARC-AGI 1 to just 4%.
- **The ARChitects team:** from 53.5% to 3%.
- **Frontier models:** generally achieving between 0–20% under the competition's cost constraints.

This notable drop in performance really highlighted how effective ARC-AGI 2 is at uncovering structural flaws in architectures focused on program synthesis and test-time training methods. This is especially true when it comes to handling multi-object, sequential, and context-dependent reasoning.

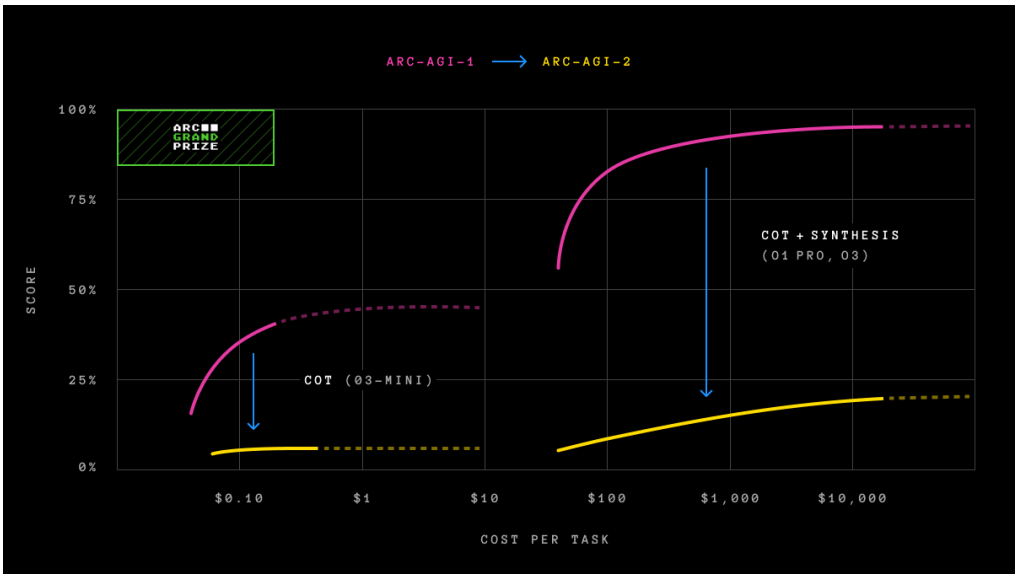

Figure 2.4. Performance comparison between ARC-AGI 1 (magenta) and ARC-AGI 2 (yellow) across different computational costs per task and model configurations [17]. The chart highlights the significant performance degradation when moving from ARC-AGI 1 to ARC-AGI 2.

2.5.2 Limitations of previous solvers on ARC-AGI 2

Several key factors contributed to this collapse in performance:

- **Overfitting to shallow patterns:** Many ARC-AGI 1 systems relied on superficial pattern matching or brute-force program enumeration without generalizable reasoning.
- **Lack of context-dependent reasoning:** Most models lacked mechanisms for multi-step reasoning where the result of one operation affects subsequent decisions.
- **Inability to handle deep compositionality:** ARC-AGI 2 has brought in tasks that demand long-term reasoning and sequential transformations, which go beyond what fixed program templates or simple greedy heuristics can handle.
- **Poor generalization to unseen task types:** The increased novelty and information density of ARC-AGI 2 rendered memorization-based and DSL-specific approaches ineffective.

These results confirmed that while ARC-AGI 1 solvers achieved competitive performance through scalable program search and prompt-based code generation, they lacked the kind of adaptive, context-sensitive reasoning that characterizes human problem-solving abilities.

2.6 Summary and Insights

The ARC-AGI 1 competition revealed that program synthesis combined with test-time adaptation could substantially outperform purely neural approaches on symbolic reasoning tasks. However, these solutions remained limited in their ability to perform human-like, abstract reasoning and often relied on superficial pattern-matching or brute-force enumeration.

The introduction of ARC-AGI 2 exposed these weaknesses: models that had previously achieved state-of-the-art performance struggled with problems requiring flexible, multi-step, compositional reasoning over abstract symbolic spaces. As a result, even highly optimized systems like OpenAI's o3 experienced severe performance drops.

These findings highlight the shortcomings of the current architectures focused on program synthesis and encourage us to look into different reasoning frameworks. Representation-driven evolutionary algorithms present an exciting avenue, merging structured program induction with adaptive search strategies. In Chapter 3, this thesis introduces such an approach, utilizing evolutionary search over static geometric representations to improve generalization in symbolic reasoning benchmarks such as ARC-AGI.

As shown in Table 2.2, none of the leading ARC-AGI 1 solutions sustained competitive performance on ARC-AGI 2. This reinforces the importance of developing new reasoning paradigms capable of dynamic, context-sensitive compositional problem solving.

Table 2.2. Comparison of top-performing approaches on ARC-AGI 1 and ARC-AGI 2 benchmarks.

Approach	**ARC-AGI 1 Score**	**ARC-AGI 2 Score**	**Characteristics**	**Limitations**
Brute-force DSL Search	$\sim 36\%$	$\approx 0\%$	Simple, no training required	Non-scalable, poor generalization
Grammar-Guided EA + DSL	$\sim 7\%$	$\approx 0\%$	Evolves programs using grammar rules	Ineffective on complex compositional tasks
GPT-4o + LLM-driven synthesis	42–43%	$\approx 0\%$	Strong program priors combined with search	Costly, brittle generalization
OpenAI o3	75–87%	$\sim 4\%$	Program synthesis, chain-of-thought, test-time search	Fails on deep multi-step reasoning
Test-time Training Neural Models	$\sim 58\%$	$\sim$ 0–20%	Adaptive inference, pattern learning	Limited symbolic reasoning

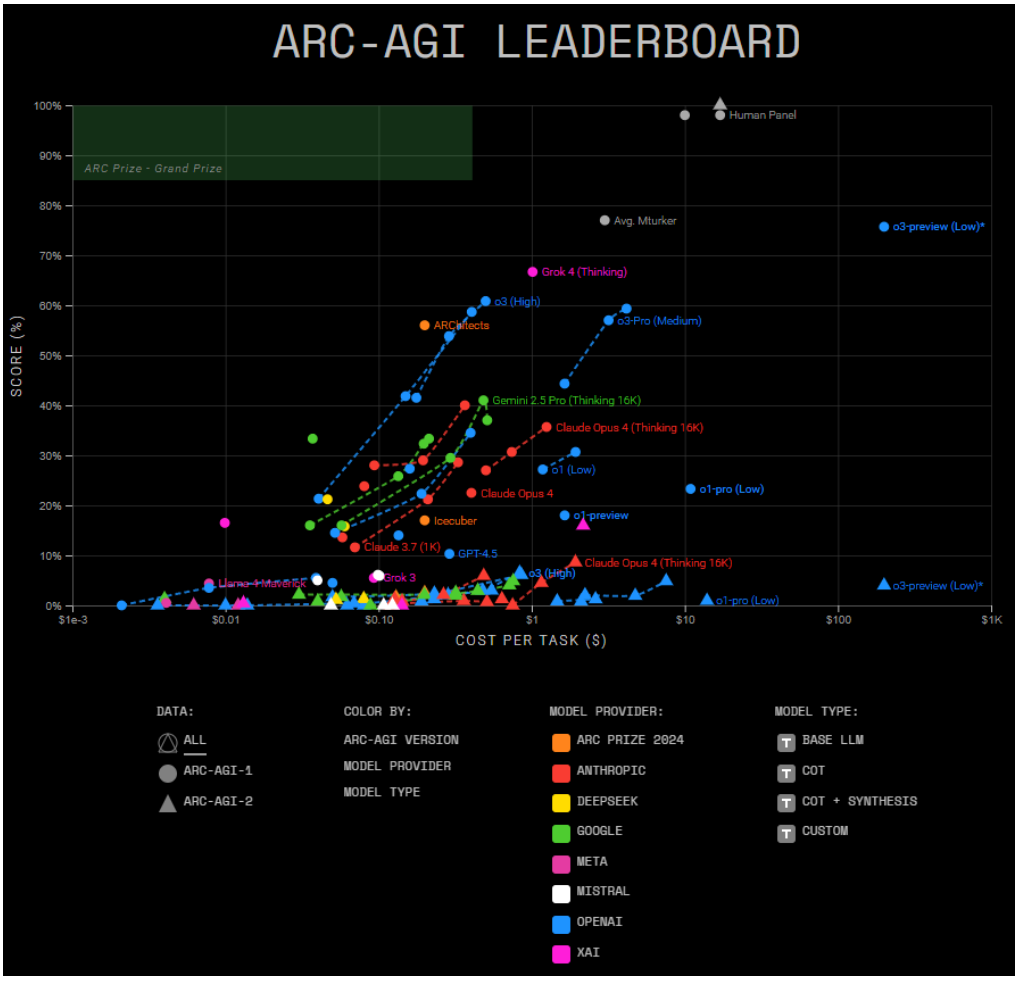

Figure 2.5. ARC-AGI public leaderboard as of July 2025 [18]. Performance comparison of various models on both ARC-AGI 1 (circles) and ARC-AGI 2 (triangles). The chart highlights the persistent difficulty of achieving high scores on ARC-AGI 2, with only a few models approaching the threshold for the Grand Prize.

2.6.1 Future Directions Suggested by ARC-AGI 2 Results

The results of ARC-AGI 2 revealed structural limitations in existing program synthesis and neural-based approaches, particularly in handling multi-step, compositional reasoning tasks. As highlighted in the ARC Prize 2024 technical report [16], even high-performing models like OpenAI's *o3* exhibited performance drops when facing tasks involving deeply nested transformations, variable-length sequences, or context-dependent rule applications.

These findings suggest several promising directions for future research:

- **Neuro-symbolic architectures:** combining symbolic program induction with neural modules capable of learning abstract representations and relational reasoning.
- **Adaptive program synthesis:** blend test-time search with meta-learning techniques to flexibly modify search strategies according to the task's structure.
- **Programmatic reinforcement learning:** evolving reasoning policies in discrete program spaces guided by performance feedback.
- **Evolutionary algorithms for program induction:** leveraging evolutionary search over compositional program representations to improve generalization and flexibility, as explored in this thesis.

These directions highlight the importance of hybrid and representation-based methods for reduce the disparity between narrow AI systems and human-level general reasoning abilities.

Part II

Chapter 3

Evolving different Geometric Representations

3.1 Overview of the Proposed Approach

The ARC-AGI challenge has highlighted how conventional approaches to abstract reasoning tasks such as brute-force program synthesis, large language models (LLMs) and test-time training still struggle to perform flexible, multi-step reasoning over symbolic domains. While recent breakthroughs like OpenAI's o3 model demonstrated the potential of combining program synthesis with language priors and chain-of-thought reasoning, these systems remain limited in scalability, interpretability, and compositional generalization, especially when applied to unseen tasks in the more demanding ARC-AGI 2 benchmark.

The method proposed in this thesis aims to explore a complementary research direction based on evolutionary algorithms (EAs) and geometric representations of ARC grids. The key idea is to convert the raw grid-based input into multiple abstract representations, each capturing different structural and geometrical properties of the problem. For instance, an ARC grid can be represented as:

- A list of individual pixels with their absolute positions and colors.

- A set of segments connecting neighboring pixels.
- A list of higher-level objects like figures, square; each characterized by properties such as shape, color, size, position, and relative relationships.

Each representation is associated with a dedicated set of primitive geometric actions, such as moving objects, changing their color, rotating them, or combining adjacent shapes. The aim of the system is to automatically synthesize a sequence of actions that transforms the input grid into the desired output configuration.

Crucially, every action is paired with a selector, a parameterized configuration that defines Which elements to act upon (e.g., all red pixels, a specific object, or on group of object), How the action should be applied (e.g., move up by one cell, recolor to blue, rotate by 90°) and possible grouping or ordering constraints, such as selecting elements based on position, color, or order. This mechanism of selectors associated with actions introduces an additional abstraction layer, enabling the system to compose flexible, context-aware transformations.

The list of available actions for each representation is defined through an initial analysis of the ARC task. In some cases, this preliminary inspection allows excluding certain actions that are not relevant for the specific task, effectively reducing the search space for the evolutionary algorithm and improving its convergence speed. An evolutionary algorithm is then used to evolve sequences of these action-selector pairs. The EA maintains a population of individuals, each representing a candidate sequence of transformations, and iteratively refines this population through variation operators and fitness-based selection. Two variants of the algorithm were designed:

- A **single-example approach**, where one evolutionary run is performed for each example within a task, and the best-performing solution is selected individually.
- A **multi-example (multi-objective) approach**, where a single evolutionary run attempts to find a program that performs well across all examples simultaneously, it can employ strategies like NSGA-II [19] and lexicase selection [20] for balancing trade-offs between multiple objectives.

The search process integrates various evolutionary operators. Parent selection is performed using strategies such as tournament selection and lexicase selection. The system applies mutation operators to the action sequences, including add, tweak, and swap mutations. Finally, survival selection is carried out by either retaining the top-N individuals based on fitness or, in the second variant of the system, by employing multi-objective sorting to balance different optimization criteria. This approach aims to combine symbolic program induction over structured representations with adaptive evolutionary search mechanisms, offering a novel alternative for tackling the complex reasoning problems posed by the ARC-AGI framework.

All the code used to implement the proposed system is available at the following GitHub repository: https://github.com/TurcoRiccardo/ARC_AGI.

3.2 Problem Representation

One of the standout features of the proposed system is its use of multiple structured geometric representations for each ARC grid. The concept here is to move beyond the raw grid and visualize it from different angles, with each perspective shining a light on particular structural or spatial characteristics. This approach enables us to tackle the problem at various levels of abstraction and detail.

Some representations, such as the Color Layer representation and Border representation, were initially considered but later discarded due to redundancy with other structures or limited effectiveness in problem-solving. For completeness, their associated selectors are still reported. The following representations have been designed and implemented:

- **Pixel Representation:** The simplest form of a representation, where the grid is seen as a list of individual pixels, each defined by its position (x, y) and color.
- **Row Representation:** The grid is interpreted as a sequence of horizontal rows. Each row is treated as a unit, enabling actions that operate

at the row level, such as shifting, recoloring, or modifying individual pixels in a row.

- **Column Representation:** Similar to Row Representation, but the grid is decomposed into vertical columns, facilitating column-wise operations.

- **Color Layer Representation:** Each color in the grid is isolated into a separate binary grid (layer), where pixels of the target color are marked and others are ignored. (Note: this idea was explored in early prototypes but later discarded due to limited performance improvements.)

- **Rectangle Representation:** The grid is parsed into rectangles, contiguous blocks of pixels of the same color. Each rectangle is characterized by its color, width, height, and absolute position relative to the top-left corner of the grid.

- **Figure Representation:** Connected groups of pixels (regardless of shape) are clustered into figures. Each figure is associated with a color, a shape pattern, and an absolute position. This abstraction allows reasoning about multi-pixel objects as single entities.

- **Color Figure Representation:** An extension of Figure Representation where figures can be composed of multiple colors and are further split into inner regions and borders. This enables actions that selectively manipulate the borders or the central part of a figure.

- **Border Representation:** The grid is analyzed to detect and represent open and closed borders formed by contiguous pixels. (Note: this concept was initially considered but was not further developed in the final implementation.)

- **First Diagonal Representation:** The grid is decomposed into diagonals running from the top-left to the bottom-right. Each diagonal can be manipulated as a sequence.

- **Second Diagonal Representation:** Similar to the First Diagonal Representation, but the diagonals run from the top-right to the bottom-left.

3.2.1 Available Actions per Representation

Each problem representation is associated with a set of basic actions, designed to manipulate the elements of the representation according to geometric and structural transformations. These actions constitute the atomic operators used by the evolutionary algorithm to generate candidate solutions. Below is the list of available actions for each representation:

Pixel Representation

- `movePixel`: moves the selected pixels by one unit in a specified direction.
- `changeColorPixel`: changes the color of selected pixels.
- `removePixel`: removes selected pixels.
- `duplicatePixel`: duplicates selected pixels.
- `expandGrid`: expands the grid size.
- `reduceGrid`: reduces the size of the grid.

Row Representation

- `moveRow`: moves a selected row by one unit.
- `changeColorRow`: changes the color of pixels in the selected row.
- `changeColorRowPixel`: changes the color of selected pixels within a row.
- `modifyRowAdd`: add pixels in a row.
- `modifyRowDel`: delete pixels in a row.
- `modifyRowMove`: swap pixels in a row.
- `expandGrid`: expands the grid size.
- `reduceGrid`: reduces the size of the grid.

Column Representation

- `moveColumn`: moves a selected column by one unit.
- `changeColorColumn`: changes the color of pixels in the selected column.
- `changeColorColumnPixel`: changes the color of selected pixels within a column.
- `modifyColumnAdd`: add pixels in a column.
- `modifyColumnDel`: delete pixels in a column.
- `modifyColumnMove`: swap pixels in a column.
- `expandGrid`: expands the grid size.
- `reduceGrid`: reduces the size of the grid.

Color Layer Representation (discarded)

- `moveLayer`: moves a selected layer by one unit.
- `moveLayerPixel`: moves a selected pixel in a layer by one unit.
- `layerUnion`: performs a union operation between 2 layers
- `delPixelLayer`: delete pixels in a layer.
- `addPixelLayer`: add pixels in a layer.
- `expandGrid`: expands the grid size.
- `reduceGrid`: reduces the size of the grid.

Rectangle Representation

- `moveRectangle`: moves a selected rectangle by one unit.
- `changeColorRectangle`: slightly changes the color of the selected rectangle.
- `removeRectangle`: remove the selected rectangle from the rectangle list.
- `duplicateRectangle`: duplicate the selected rectangle.

- `changeOrder`: change the display order of the rectangles.
- `scaleUpRectangle`: scale up the selected rectangle.
- `scaleDownRectangle`: scale down the selected rectangle.
- `expandGrid`: expands the grid size.
- `reduceGrid`: reduces the size of the grid.

Figure Representation

- `moveFigure`: moves a selected figure by one unit.
- `changeColorFigure`: slightly changes the color of the selected figure.
- `addElementFigure_row`: add a element in the figure in the selected row.
- `addElementFigure_column`: add a element in the figure in the selected column.
- `moveElementFigure_row`: move the selected row in the figure.
- `moveElementFigure_column`: move the selected column in the figure.
- `removeElementFigure_row`: remove the element in the selected figure row.
- `removeElementFigure_column`: remove the element in the selected figure column.
- `duplicateFigure`: duplicate the selected figure.
- `removeFigure`: remove a figure from the figure list.
- `rotateFigure`: rotate the selected figure.
- `mergeFigure`: merge two figures that are next to each other.
- `divideFigure_row`: divide the selected figure based on the selected row.
- `divideFigure_column`: divide the selected figure based on the selected column.
- `changeOrder`: change the display order of the figures.
- `expandGrid`: expands the grid size.
- `reduceGrid`: reduces the size of the grid.

Color Figure Representation

- `moveFigure`: moves a selected figure by one unit.
- `changeColorFigureBorder`: changes the color of the border of the figure index based on color.
- `changeColorFigureCenter`: changes the color of the center of the figure index according to the color.
- `changeColorFigure_row_column`: changes the color of a pixel in the figure index based on color.
- `fillFigureCenter`: fill the center of the figure index based on color.
- `expandFigure`: expand the figure in the direction.
- `reduceFigure`: reduce the figure in the direction.
- `addElementFigure_row_column`: add a element in the figure in the selected row and column.
- `moveElementFigure_row_column`: move a pixel in the figure based on the direction.
- `removeElementFigure_row_column`: remove the element in the figure in the selected row and column.
- `duplicateFigure`: duplicate the selected figure from the figure list.
- `removeFigure`: remove a figure from the figure list based on the index.
- `rotateFigure`: rotate the selected figure from the figure list to the right or to the left.
- `mergeFigure`: merge two figures that are next to each other of the same color.
- `divideFigure_row`: divide the selected figure based on the selected row.
- `divideFigure_column`: divide the selected figure based on the selected column.
- `changeOrder`: change the display order of the figures.
- `expandGrid`: expands the grid size.
- `reduceGrid`: reduces the size of the grid.

Border Representation (discarded)

- `moveBorder`: moves a selected border by one unit.
- `changeColorBorder`: slightly changes the color of the selected border.
- `changeColorCenter2`: slightly changes the color of the central colored pixel of the selected border.
- `changeColorCenter3`: slightly changes the color of the central area of the selected border.
- `modifyBorderFigure`: slightly changes the dimension of the selected border.
- `expandGrid`: expands the grid size.
- `reduceGrid`: reduces the size of the grid.

First Diagonal Representation

- `moveDiagonal`: moves a selected diagonal by one unit.
- `changeColorDiagonal`: slightly changes the color of the pixel in the selected diagonal.
- `changeColorDiagonalPixel`: changes the color of the selected pixel in the diagonal index based on color.
- `modifyDiagonalAdd`: add a new colored pixel in the selected diagonal.
- `modifyDiagonalDel`: delete a colored pixel in the selected diagonal.
- `modifyDiagonalMove`: modify the selected diagonal by swapping two pixels.
- `expandGrid`: expands the grid size.
- `reduceGrid`: reduces the size of the grid.

Second Diagonal Representation

- `moveDiagonal`: moves a selected diagonal by one unit.
- `changeColorDiagonal`: slightly changes the color of the pixel in the selected diagonal.
- `changeColorDiagonalPixel`: changes the color of the selected pixel in the diagonal index based on color.
- `modifyDiagonalAdd`: add a new colored pixel in the selected diagonal.
- `modifyDiagonalDel`: delete a colored pixel in the selected diagonal.
- `modifyDiagonalMove`: modify the selected diagonal by swapping two pixels.
- `expandGrid`: expands the grid size.
- `reduceGrid`: reduces the size of the grid.

3.2.2 Selectors

In addition to the actions associated with each representation, the proposed system employs a set of *selectors*. Selectors are responsible for providing the parameters required by the actions and for specifying selection or grouping criteria over the elements of a given representation. Essentially, a selector determines:

- Which objects within a representation should be selected for manipulation.
- How the selected objects should be grouped, ordered, or prioritized.
- The specific parameters to be used by an action (such as a target color, a movement direction, or a shape property).

The meaning and usage of selector parameters are representation-dependent. Each representation interprets selectors according to the structural characteristics of its elements and the type of operations available. Some representations, such as the *Color Layer* and *Border*, were initially considered but later discarded due to redundancy with other structures or limited effectiveness in problem-solving. For completeness, their associated selectors are still reported here.

Pixel Representation
We can select or group pixels in various ways:

- based on the order of the pixel list
- based on the reverse order of the pixel list
- based on the central pixel in the pixel list
- based on the color of the pixel
- selecting all the pixels

Row Representation
We can select or group rows in various ways:

- based on the order of the row list
- based on the reverse order of the row list
- based on the central row in the row list
- based on whether the row contains a certain color
- selecting all the rows

We can also select or group the components within a row:

- based on the order of the components
- based on the reverse order of the components
- based on the central component in the row
- based on the color of the components in the row
- selecting all the components in the row

Column Representation
We can select or group columns in various ways:

- based on the order of the column list
- based on the reverse order of the column list
- based on the central column in the column list

- based on whether the column contains a certain color
- selecting all the columns

We can also select or group the components within a column:

- based on the order of the components
- based on the reverse order of the components
- based on the central component in the column
- based on the color of the components in the column
- selecting all the components in the column

Color Layer Representation (discarded)
We can select or group layers in various ways:

- based on the order of the color associated with each layer
- selecting all the layers

And select or group the components of a layer:

- based on the order of the pixels in the layer
- based on the reverse order of the pixels in the layer
- based on the central pixel in the layer
- selecting all the pixels in the layer

Rectangle Representation
We can select or group rectangles in various ways:

- based on the order of the rectangle list
- based on the reverse order of the rectangle list
- based on the central rectangle in the rectangle list
- based on the color of the rectangle
- selecting all the rectangles

Figure Representation
We can select or group figures in various ways:

- based on the order of the figure list
- based on the reverse order of the figure list
- based on the central figure in the figure list
- based on the color of the figure
- selecting all the figures

We can also select or group the components of a figure:

- based on the order, reverse order, or central row within the figure
- selecting all the rows
- based on the order, reverse order, or central column within the figure
- selecting all the columns

Color Figure Representation
We can select or group figures in various ways:

- based on the order of the figure list
- based on the reverse order of the figure list
- based on the central figure in the figure list
- based on whether a figure contains a certain color
- selecting all the figures

We can also select or group the components of a figure:

- based on the order, reverse order, or central row within the figure
- selecting all the rows
- based on the order, reverse order, or central column within the figure
- selecting all the columns

Border Representation (discarded)
We can select or group borders in various ways:

- based on the order of the border list
- based on the reverse order of the border list
- based on the central border in the border list
- based on the color of the center area
- selecting all the borders

We can also select or group the components within a border:

- based on the order, reverse order, or central pixel within the border
- selecting all the pixels in the border

First and Second Diagonal Representation
We can select or group diagonals in various ways:

- based on the order of the diagonal list
- based on the reverse order of the diagonal list
- based on the central diagonal in the list
- based on whether the diagonal contains a certain color
- selecting all the diagonals

We can also select or group the components of a diagonal:

- based on the order, reverse order, or central component within the diagonal
- based on the color of the components
- selecting all the components in the diagonal

Overall, selectors play a crucial role in determining how actions are applied within each representation. By providing a flexible and configurable mechanism for selecting and grouping elements, they allow the system to adapt to a wide variety of transformation tasks in ARC.

Table 3.1. Summary of how objects can be grouped by the selector based on representation

Representation	Order	Reverse	Center	By Color	All
Pixel	X	X	X	X	X
Row	X	X	X	X	X
Column	X	X	X	X	X
Rectangle	X	X	X	X	X
Figure	X	X	X	X	X
Color Figure	X	X	X	X	X
First/Second Diagonal	X	X	X	X	X
Color Layer (discarded)	X	–	–	–	X
Border (discarded)	X	X	X	X	X

Table 3.2. Summary of how object component can be grouped by selector based on representation

Representation	Order	Reverse	Center	By Color	All
Pixel	–	–	–	–	–
Row	X	X	X	X	X
Column	X	X	X	X	X
Rectangle	–	–	–	–	–
Figure	X	X	X	–	X
Color Figure	X	X	X	–	X
First/Second Diagonal	X	X	X	X	X
Color Layer (discarded)	X	X	X	–	X
Border (discarded)	X	X	X	–	X

3.3 Evolutionary Algorithm

Once the problem has been represented through one or more geometric abstractions, and a set of available actions and selectors has been defined for each, the core of the proposed system consists of an evolutionary algorithm designed to automatically synthesize a sequence of action-selector pairs capable of transforming an input grid into its corresponding output.

Two different evolutionary strategies have been developed that differ in how they manage the training examples within a task. Specifically, a first variant performs a separate evolutionary run for each training example and

representation, while a second variant operates across all training examples simultaneously for each representation.

3.3.1 Single-Example Per Representation Evolution

In the first approach, for each representation and each training example, an independent evolutionary run is performed. This procedure begins by generating an initial population of candidate solutions using a simplified version of the evolutionary algorithm, which only applies the add_mutation operator. This ensures that the initial population contains not empty individuals with at least one action-selector pair, and also to promote diversity, a small number of empty individuals are also added.

Once the initial population is prepared, the evolutionary process iteratively applies selection, mutation, and survival steps:

1. **Tournament selection** is used to choose individuals for mutation.

2. Selected individuals undergo mutation operations (described in the next section) according to predefined probabilities.

3. A fitness function tailored for each specific representation to assesses how good each individual is, usually by looking at the spatial or structural differences between what's produced and the desired target.

4. **Survival selection** retains the top-performing individuals in the population.

After the evolutionary run has ended, the best individual is applied to all other training examples within the task. A score is then computed for this individual based on the average error rate (ratio of incorrect to correct pixels) across all other training examples. In addition, a penalty is added if the program does not produce a solution equal to the training output result, also another penalty proportional to the number of actions in the individual is added, favoring simpler solutions.

This procedure is repeated for all available representations, and the action-selector sequence with the highest score is ultimately applied to the test input grid to generate the final output.

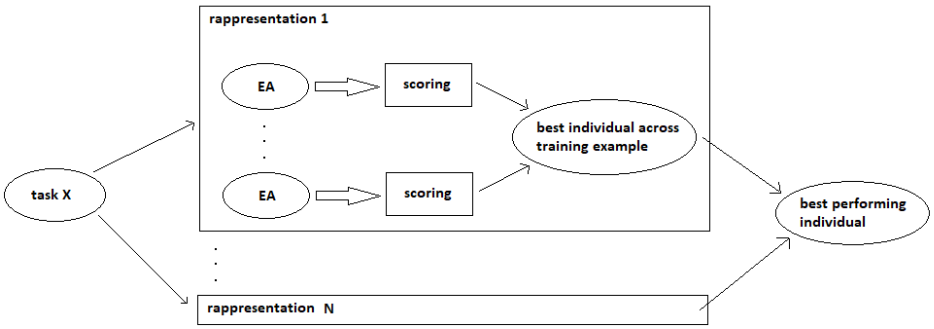

Figure 3.1. Single-Example Evolution: for each representation and training example, a dedicated evolutionary algorithm instance is executed, and the best individual is selected based on scoring across training examples.

3.3.2 Multi-Example Per Representation Evolution

The second evolutionary strategy operates differently: for each representation, a single evolutionary run is performed using the entire set of training examples simultaneously. This variant is designed as a multi-objective evolutionary algorithm, seeking programs capable of generalizing across multiple examples.

As in the previous approach, the initial population is generated using a simplified evolutionary algorithm that use only the add_mutation operator, with a small number of empty individuals included to increase population diversity. During evolution:

1. Two parent selection methods are available:
   - **Tournament Selection**, as in the first variant.
   - **Lexicase Selection** [20], which tends to perform better in multi-objective contexts by preserving individuals that excel in different specific cases.
2. The mutation operators described in the next section are applied according to predefined probabilities.
3. Each individual is evaluated by applying it to all training examples, using the same fitness criteria as in the first variant.
4. Fitness values are then aggregated using one of two strategies:

- **Aggregated Fitness**, computed as the sum of fitness scores across all examples.
- **NSGA-II** [19], an elitist Pareto-based sorting method that maintains a diverse set of non-dominated solutions, promoting trade-offs between competing objectives.

5. **Survival selection** preserves the best individuals according to the selected multi-objective strategy.

At the end of the evolutionary run, the best individual for each representation is selected based on the aggregated fitness, and the final program to be applied to the test input grid is chosen by comparing these best individuals.

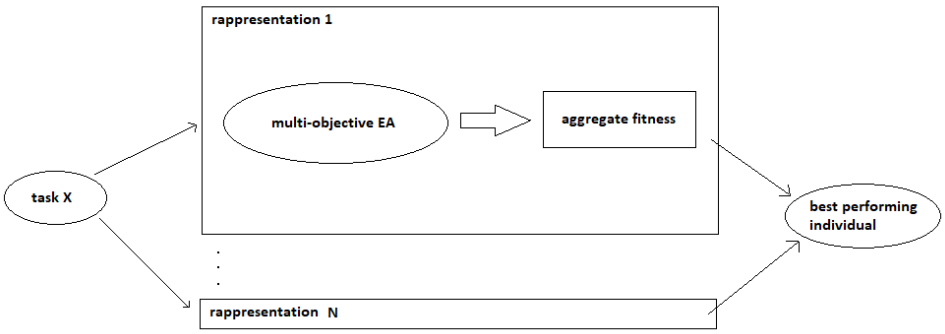

Figure 3.2. Multi-Example Evolution: for each representation, a multi-objective evolutionary algorithm is executed across all training examples, combining fitness values into an aggregate fitness to identify the best performing individual.

3.3.3 Mutation Operators

Both evolutionary strategies described above rely on the same set of mutation operators to introduce variations in the candidate solutions. These operators modify individuals by altering the sequence of action-selector pairs in different ways:

- `add_mutation`: Adds a new action-selector pair to the individual's program, expanding its behavior.

- `tweak_mutation`: Modifies the selector associated with a specific action, typically by altering its parameters or replacing it with a slightly different one.

- `swap_mutation`: Swaps the positions of two pairs of action-selectors in the program, which can change the order of operations and ultimately affect the final outcome.

These mutation operators are applied based on specific probabilities to strike a balance between exploring new possibilities and exploiting known ones during the evolutionary process.

3.4 Single-Example vs Multi-Example Evolution

The two evolutionary strategies described in the previous sections present complementary advantages and limitations, which primarily depend on how the training examples are processed and how the evolutionary algorithm (EA) is guided toward a solution.

The **Single-Example Evolution** approach makes things a bit easier for the EA by focusing on just one training example at a time. By narrowing its goal also the complexity decrease allowing the evolutionary process to focus exclusively on the patterns and transformations present within a single specific example, making it generally easier for the algorithm to discover a valid solution. Plus, this method usually needs fewer generations to reach a conclusion since the search space is more limited.

On the flip side, a major drawback of this method is the inability to capture rules or transformations that are distributed across multiple training examples. For example, if one training example demonstrates a rule that changes blue pixels into yellow, while another example has a different rule that turns orange pixels to red, the Single-Example Evolution strategy cannot produce a single solution capable of applying both transformations unless those rules can be clearly identified within the same training grid. Consequently, this approach tends to overfit to individual examples, limiting its capacity to generalize to unseen grids.

On the other hand, the **Multi-Example Evolution** approach addresses this limitation by evolving programs across all training examples of a task simultaneously. This design enables the algorithm to synthesize programs that aggregate transformations from different grids, making it possible to capture more complex and general rules that are distributed across the training grids. In principle, this allows for greater generalization capability and a more faithful modeling of the underlying task logic.

Evolving programs in a multi-objective setting comes with its fair share of challenges. One of the biggest hurdles is figuring out how to combine fitness values from various training examples. Often, an action that improves the fitness on some grids may simultaneously worsen it on others. This conflict makes the search process tricky, often preventing the EA from effectively converging toward a globally satisfactory solution. While there are sophisticated methods like *Lexicase Selection* and *NSGA-II* that aim to mitigate this issue, the difficulty in balancing competing objectives remains a significant limitation of this approach.

In summary, the choice between Single-Example and Multi-Example Evolution involves a trade-off between ease of optimization and the ability to generalize across different training examples. The former makes the search simpler but limits expressiveness, while the latter enhances modeling capabilities but complicates convergence due to those conflicting fitness goals.

3.5 Summary of the Proposed Methodology

This section summarizes the proposed approach for solving the ARC tasks using evolutionary algorithms applied to multiple geometric representations on the grids. The methodology can be divided into the following main steps:

1. **Geometric Representations:** Each ARC grid is transformed into one or more alternative geometric representations. These representations capture the different spatial and structural properties of objects, the correlations between them and their relationships within the grid.

2. **Initial Analysis:** An initial analysis of the ARC task is also performed

to identify the most relevant actions for each representation. This preliminary inspection allows excluding unnecessary actions that are unlikely to be useful for the specific task, reducing the search space for the evolutionary algorithm and potentially improving its efficiency and convergence.

3. **Initial Population Generation:** For each representation, the program generate an initial population of individuals using a simplified evolutionary algorithm. This preliminary step, based solely on the *add_mutation* operation, guarantees the creation of non-empty individuals, then to increase diversity we add some empty individuals to the population.

4. **Evolutionary Algorithm Execution:** Two alternative strategies are applied:

   - **Single-Example Evolution:** An evolutionary algorithm is executed independently for each training example and representation. The best individual is selected and evaluated across the other training examples, scoring its performance based on error rate and solution quality. The best-performing individual among all representations is then applied to the test grid.
   - **Multi-Example Evolution:** A single evolutionary algorithm is executed for each representation across all training examples simultaneously. The fitness function aggregates the performance on multiple examples and the representation whose best individual achieves the highest aggregated fitness is applied to the test grid.

5. **Mutation Operators:** The evolutionary process uses on three mutation operators:

   - *add_mutation*: adds a new action-selector pair to an individual.
   - *tweak_mutation*: modifies the selector of a specific action in the individual.
   - *swap_mutation*: swaps the positions of two action-selector pairs within the individual.

6. **Parent Selection** Individuals are selected for mutation using tournament selection (or lexicase selection in the multi-example case).

7. **Survival Selection** Survival selection retains the top-performing individuals based on either individual fitness values or aggregated multi-objective performance.

8. **Solution Generation:** The best individual from the selected representation is applied to the test grid to generate the final predicted solution.

This methodology allows the system to dynamically exploit different representations of the grid while adapting the evolutionary search process to either single-example or multi-example optimization scenarios.

Chapter 4

Analysis of results

4.1 Introduction

This chapter presents a detailed analysis of the results obtained by applying the two evolutionary approaches described in Chapter 3 to a set of selected ARC-AGI tasks. The goal of this analysis is to evaluate the effectiveness, robustness, and limitations of both the *Single-Example Evolution* and the *Multi-Example Evolution* strategies, highlighting the impact of different selection methods, fitness aggregation techniques, and grid representations on the overall performance.

In particular, we compare the two methods in terms of training and test performance, number of actions used in the generated solutions, and convergence speed. Additionally, we investigate how different representations of the grids influence the evolutionary process and assess the effect of various selection and survival strategies in multi-objective scenarios.

4.2 Experimental Setup

The experiments were conducted on a standard hardware setup, and all algorithms were implemented in Python. A set of 10 tasks from the public ARC-AGI training set was selected for evaluation, covering different types of grid transformations and varying numbers of training examples. Table 4.1 summarizes the selected tasks:

Table 4.1. Overview of the selected ARC-AGI tasks

Task ID	Description	Training Examples
007bbfb7	Pattern repetition	5
00d62c1b	Fill the central part of a random figure	5
017c7c7b	Pattern repetition	3
025d127b	Figure transformation based on translation	2
045e512c	Figure duplication and pattern repetition	3
0520fde7	Color pixels on condition	3
05269061	Pattern repetition	3
05f2a901	Unify two random figures	3
06df4c85	Connect dots within a noisy grid	3
08ed6ac7	Color columns based on height	2

A task is considered correctly solved when the predicted output grid matches the expected output for all testing examples.

For both evolutionary approaches, the following parameters were kept constant:

- **Population size: 50** individuals
- **Mutation probabilities:** `add_mutation` = **0.2**, `tweak_mutation` = **0.4**, `swap_mutation` = **0.4**
- **Initial population:** generated by a simplified EA using only `add_mutation` and some empty individuals.
- **Termination criteria:** Each run continues until the maximum number of generations per task is reached. Even if a valid solution was found before these limits, the algorithm continued evolving the population in an attempt to discover more optimized solutions with a reduced number of actions.

Specific parameters for each approach:

- **Single-Example Evolution:**
  - Maximum number of generations: 2500
  - Selection method: Tournament Selection with a tournament size of **2**

  - Survival strategy: Top-N individuals, with $N = \mathbf{50}$
  - Fitness function: representation-specific fitness based on distances between objects and color values

- **Multi-Example Evolution:**
  - Maximum number of generations: depending on the problem size (from **1000** to **8000**)
  - Selection method: **Lexicase Selection**
  - Survival strategy: either based on **Aggregate Fitness** or using **NSGA-II**
  - Fitness aggregation: sum of fitness values computed across all training examples

To account for the stochastic nature of the evolutionary algorithms, each experiment was repeated **10** times, and the final results were averaged.

4.3 Results and Discussion

The performance of the proposed evolutionary approaches was evaluated on the 10 ARC-AGI tasks shown in the experimental setup. Each configuration was run 10 times to account for the stochastic nature of the algorithms, and the average number of correctly solved tasks per run was recorded.

The results can be summarized as follows:

- The **Single-Example Evolution** approach successfully solved up to **3 out of 10** tasks in the best run, with an average of **2 out of 10** tasks correctly solved across the 10 iterations.
- The **Multi-Example Evolution** approach achieved an average of only **1 out of 10** tasks correctly solved per run and can successfully solved up to **3 out of 10** tasks in the best run.

The results showed a moderate variance across different runs, with the single-example evolution showing relatively stable performance and the multi-example

evolution occasionally producing higher-performing solutions but with less consistency.

Although the Multi-Example algorithm is conceptually more powerful, being capable of capturing rules shared across different training examples, its search space is significantly larger, which negatively affects its convergence speed and overall performance within the limited computational resources. The combination of *Lexicase Selection* and *NSGA-II* was employed to mitigate this issue by improving selective pressure and maintaining diversity, but this was not sufficient to fully overcome the difficulty introduced by the expanded search space.

In contrast, the Single-Example approach simplifies the problem for the evolutionary algorithm by focusing on one training example at a time. This reduces the complexity of the search space and allows the algorithm to converge more easily to a valid solution. However, this method cannot capture rules that are expressed across multiple grids, limiting its generalization ability, tasks where different rules appear in separate training examples can't be solved.

These results highlight the tradeoff between search space complexity and rule generalization capability. The Multi-Example strategy is definitely an exciting avenue to explore, thanks to its theoretical benefits. However, we still need to make some strides in refining evolutionary search operators, fitness aggregation methods, and selection mechanisms to effectively tackle the growing complexity of multi-objective optimization in this area.

Table 4.2. Summary of average and best-case performance for the two evolutionary approaches over 10 ARC-AGI tasks.

Approach	**Best Case (tasks solved)**	**Average (tasks solved)**
Single-Example	3/10	2/10
Multi-Example	3/10	1/10

In conclusion, while the Multi-Example strategy remains promising, further work is required to improve evolutionary search techniques and aggregation mechanisms in order to fully exploit its capabilities.

4.4 Qualitative Analysis of Individual Tasks

This section presents a qualitative, task-by-task analysis of how the proposed algorithms perform on the test cases introduced in the experimental setup. Rather than focusing solely on aggregate performance metrics, we examine the specific behaviors exhibited by the system for each task, highlighting both effective strategies and recurring failure modes.

For every task, we break down the problem, explore potential solutions, and highlight the types of errors observed. This detailed analysis not only sheds light on the strengths and weaknesses of our approach across various contexts but also offers valuable insights that can steer future enhancements.

Task 007bbfb7

This task involves identifying a 3×3 pattern of colored blocks and reproduce the same structure multiple times within a larger 9×9 grid based on the position of a reference colored pixel.

The current algorithms fail to solve this task, as they lack a representation capable of treating a sub-grid as a compositional unit and capturing spatial correlations between such units.

Figure 4.1. ARC task `007bbfb7`. The goal is to identify a 3×3 pattern and replicate it in a larger grid.

Task 00d62c1b

In this task, the system must detect green polygonal structures in the input grid and determine whether they are fully enclosed. If a closed shape is detected, its interior must be filled with yellow; otherwise, no modification is made.

Both the single-example and multi-example algorithms are able to solve this task, the geometric representations used are effective at distinguishing between interior and exterior regions.

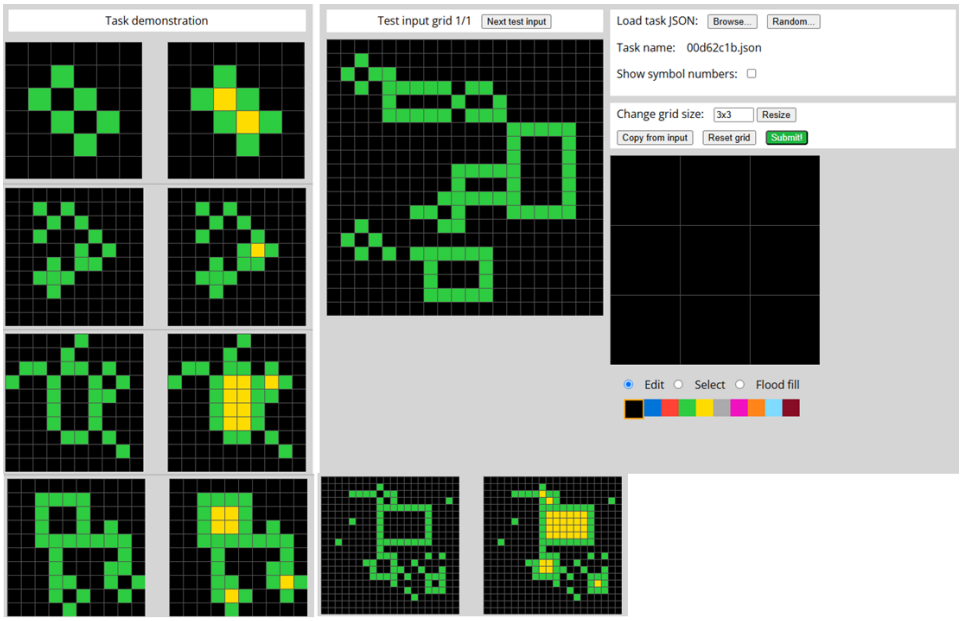

Figure 4.2. ARC task `00d62c1b`. The objective is to detect closed green shapes and fill them with yellow.

Task 017c7c7b

This task involves swapping out all the blue cells for red ones while keeping the original layout intact and expanding the pattern into a larger dimensional grid.

The current algorithms are unable to solve this task. A better representation is needed—one that can model repetition and continuation of patterns in space.

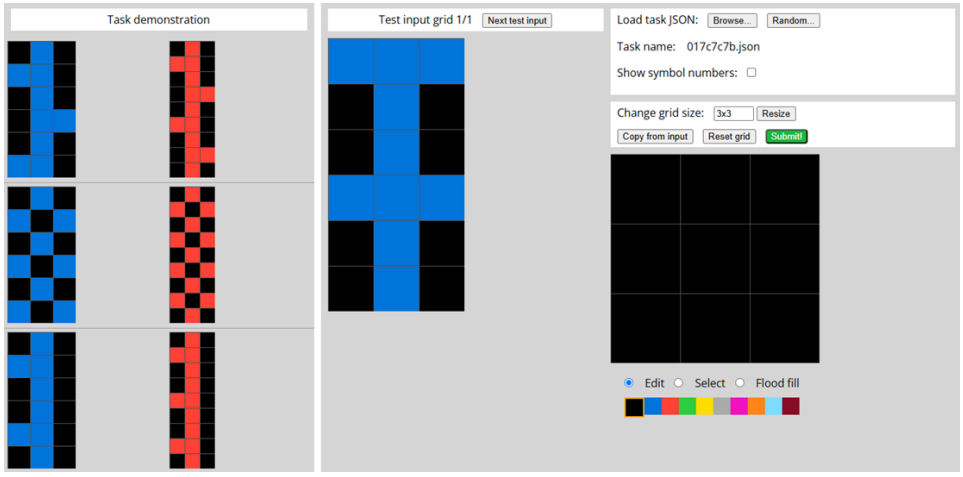

Figure 4.3. ARC task `017c7c7b`. The transformation involves extending a repeated pattern using a new color.

Task 025d127b

This task requires shifting the upper part of a figure to the right while keeping the base fixed. The color of the figures and dimensions of the grid remain unchanged.

Both algorithms can solve this task, but it's not without its challenges. The real trick is figuring out how to isolate and transform just a part of the figure while keeping the rest unchanged for the subsequent translation.

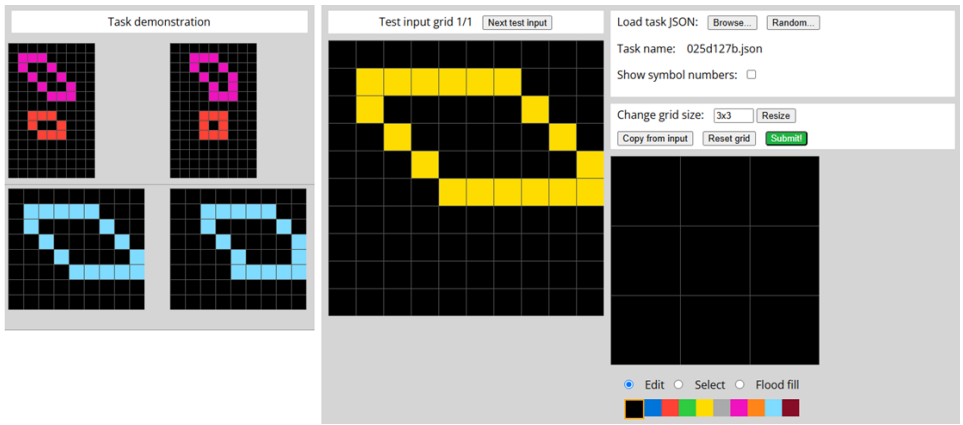

Figure 4.4. ARC task `025d127b`. The transformation consists of shifting the top part of a figure to the right.

Task 045e512c

This task requires duplicating a figure in a specified direction and color, while keeping the overall grid size unchanged.

The current algorithms are unable to solve this task. Despite having some mechanisms for duplication and translation, they fail to generalize the repetition of patterns in spatial directions.

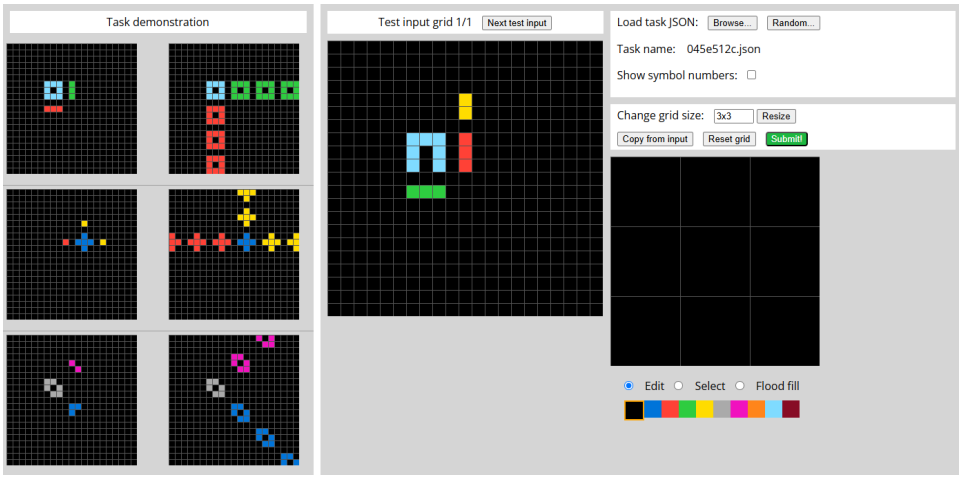

Figure 4.5. ARC task `045e512c`. The transformation involves duplicating a figure in the indicated direction using a specified color.

Task 0520fde7

In this task, the input grid contains two 3×3 patterns separated by a gray line. The goal is to color in red the pixels that are present in both patterns.

The current algorithms are not able to solve this task. A more advanced representation is required, capable of identifying distinct subregions and evaluating logical conditions between them.

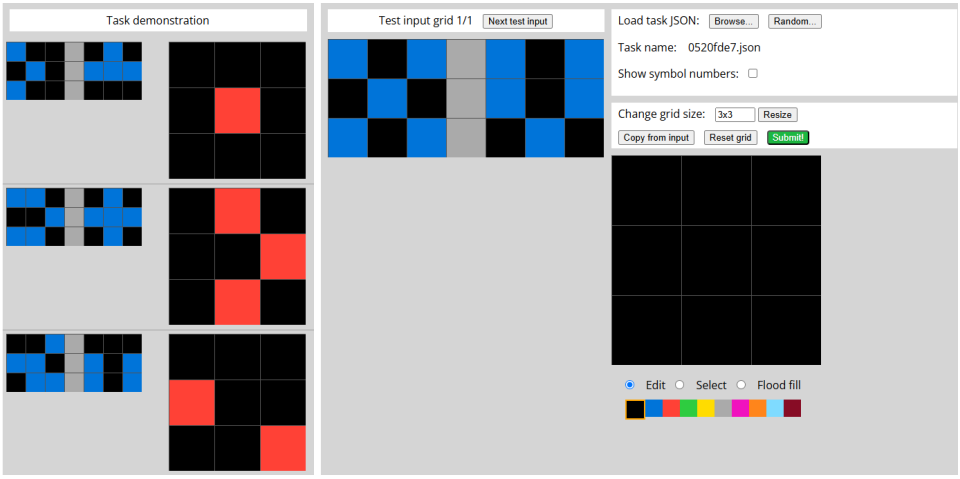

Figure 4.6. ARC task `0520fde7`. The transformation involves coloring in red the pixels that appear in both 3×3 patterns.

Task 05269061

This task is all about spreading diagonals of various colors throughout the entire grid, starting from a given diagonal setup. The size of the grid stays the same.

The current algorithms are unable to solve this task due to the lack of pattern generalization, recognition and repetition mechanisms in oblique directions.

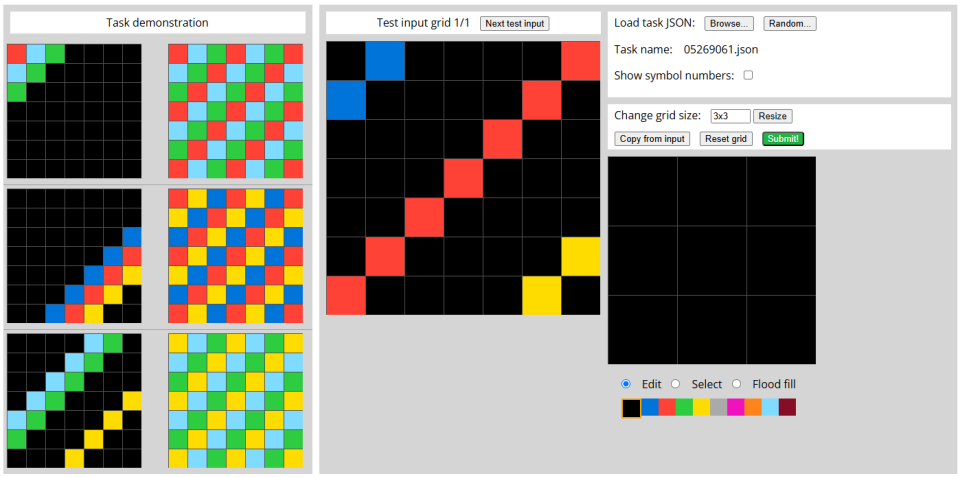

Figure 4.7. ARC task `05269061`. The transformation consists of repeating colored diagonals across the grid.

Task 05f2a901

The objective of this task is to move the red figure toward the blue one until they make contact. The size of the grid remains unchanged.

Although the algorithms can recognize and move shapes, they are not equipped with the concept of movement driven by goal satisfaction or termination upon contact.

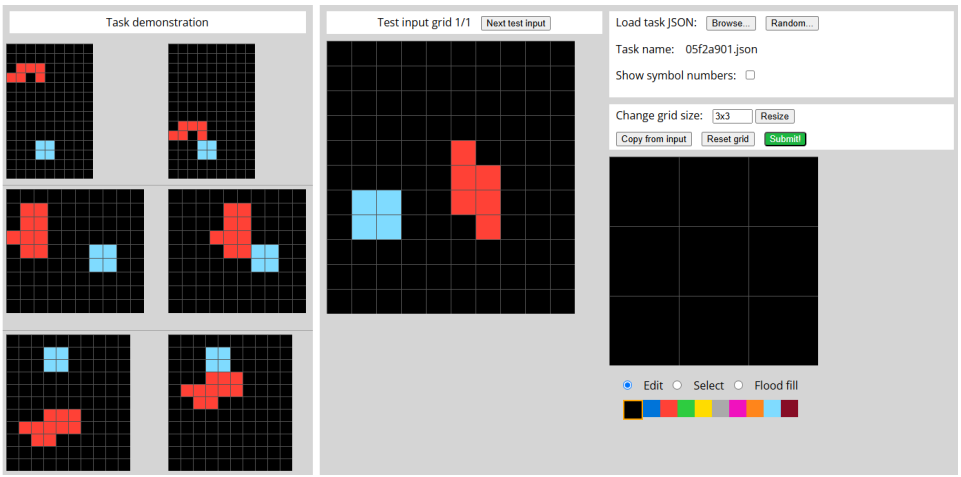

Figure 4.8. ARC task `05f2a901`. The red figure must be moved toward the blue one until they touch.

Task 06df4c85

This task requires connecting pairs of points with lines of the same color but with a virtual grid overlays the canvas that defines the connection points.

The current algorithms cannot solve this task due to the absence of a mechanism for connecting points via lines. Additionally, the task is computationally expensive because of the large grid size and the number of objects created by certain representations.

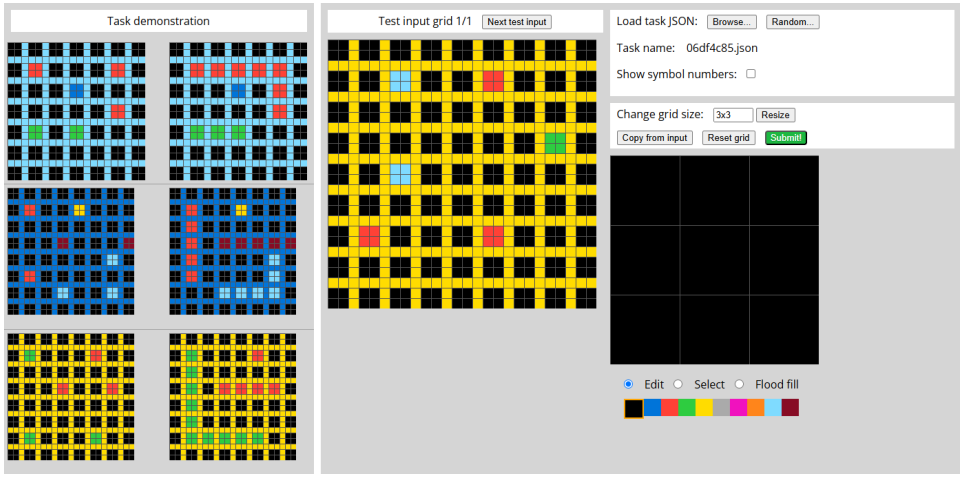

Figure 4.9. ARC task `06df4c85`. The transformation consists of connecting colored points with lines.

Task 08ed6ac7

In this task the represented gray columns must be colored based on their height, from tallest to shortest, using the colors blue, red, green, and yellow.

Both algorithms are able to solve this task. The system successfully correlates color assignments with column height to generalize the transformation rule.

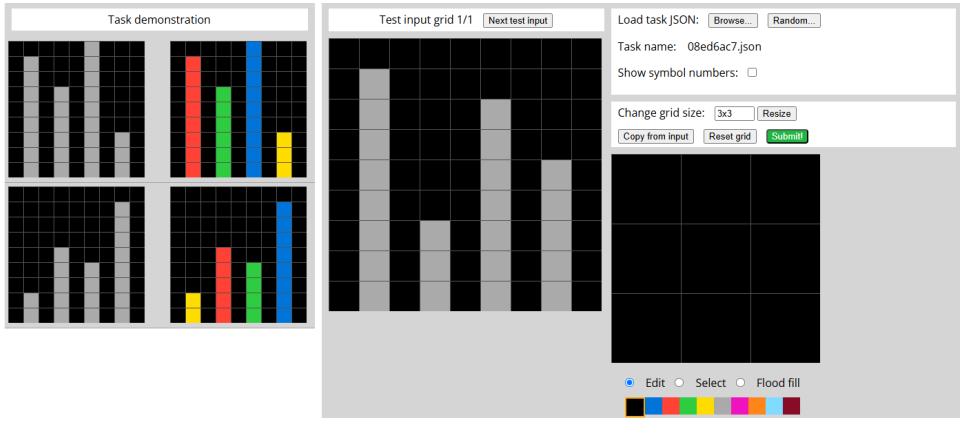

Figure 4.10. ARC task `08ed6ac7`. The transformation consists of coloring columns based on their height.

Chapter 5

Conclusion

This thesis presented an evolutionary framework for solving tasks from the ARC-AGI challenge, leveraging multiple geometric representations of the input grids and evolutionary algorithms tailored for both single-example and multi-example optimization scenarios.

The proposed methodology leverages the flexibility of viewing grid-based problems from various geometric angles, paired with an evolutionary process that can uncover sequences of actions to transform input grids into desired outputs. A crucial part of this framework is the initial task analysis phase, which identifies the best set of actions to use for each representation. This step helps narrow down the search space and boosts the efficiency of the evolutionary search.

Two different evolutionary strategies were compared:

- **Single-Example Evolution:** where an independent evolutionary algorithm is executed for each training example and representation, simplifying the search objective but limiting the ability to generalize rules across multiple grids.

- **Multi-Example Evolution:** where a single evolutionary process operates on all training examples simultaneously for each representation, enabling the discovery of more generalizable transformations but increasing the complexity of the search space.

The experimental evaluation, conducted on a set of 10 ARC-AGI tasks, showed that the single-example evolution strategy achieved better average

and best-case performance compared to the multi-example strategy. Specifically, the single-example approach successfully solved up to 3 out of 10 tasks in the best run, with an average of 2 tasks per run, while the multi-example approach achieved a lower average performance, solving approximately 1 task per run. Despite its conceptual advantages, the multi-example algorithm struggled with the large and complex search space, and current selection and survival strategies such as Lexicase Selection and NSGA-II, while helpful, were not sufficient to overcome this limitation.

5.1 Limitations

Despite the initial promising results and the flexibility of the proposed framework, several limitations emerged during the experimental evaluation:

- **Limited Task Set:** The evaluation was conducted on a relatively small subset of 10 ARC-AGI training examples. Although these tasks were chosen to cover different types of transformations, a larger and more varied evaluation would provide a more reliable assessment of the framework's generalization capabilities.

- **Fixed Algorithm Parameters:** The parameters of the evolutionary algorithm, like mutation probabilities and the number of generations, were kept the same for all tasks and representations. While this approach simplifies the experimental setup, it might not be the best fit for every example and could have affected performance on certain tasks.

- **Search Space Complexity:** The search space remains very large, especially in the multi-example setting, where the algorithm must simultaneously discover solutions compatible with multiple training examples. Although selection strategies such as Lexicase Selection and NSGA-II were employed to address this issue, they were not sufficient to fully overcome the difficulties posed by the high dimensionality and complexity of the search space.

- **Termination Criteria:** The evolutionary algorithms were designed to continue evolving until the maximum number of generations was reached or the time limit expired. Even when a correct solution was found, the search continued in order to potentially discover more optimized solutions with a lower number of actions. While this allows for solution

refinement, it can also result in unnecessary computational effort once a satisfactory solution is already available.

- **Dependence on Predefined Representations:** The performance of the framework depends on how good and expressive the chosen geometric representations are. If there are tasks that need reasoning methods that the current representations just can't handle, those might end up being left unsolved or only poorly approximated.

These limitations suggest several possibilities for future work, including automatic parameter tuning, dynamic representation selection, and the development of advanced evolutionary operators in line with the specific structure of the ARC-AGI problems.

5.2 Future Work

Several directions for future improvements and extensions of this work can be identified:

- Enhancing the initial task analysis phase to automatically infer suitable action sets and reduce the search space even further.
- Develop new actions that cover basic geometry and topology concept not covered by current actions in each representation.
- Develop new representations that cover logical and geometric concepts not covered by current representations.
- Improving fitness functions to try to increase convergence speed.
- Investigating more advanced evolutionary techniques, such as adaptive mutation rates or hybrid methods combining evolutionary algorithms with program synthesis approaches and neural networks, aiming to improve generalization capabilities and convergence speed.
- Expanding the evaluation to a larger and more diverse subset of ARC-AGI tasks to assess the generality and robustness of the proposed framework.

Overall, this work demonstrates the potential of evolutionary algorithms combined with geometric representations for solving complex grid-based reasoning tasks, while highlighting the challenges associated with multi-example optimization in such domains.

References

[1] Chollet, F. (2019). On the measure of intelligence. *arXiv preprint* arXiv:1911.01547.

[2] Goertzel, B. (2014). Artificial general intelligence: Concept, state of the art, and future prospects. Journal of Artificial General Intelligence, 5(1), 1.

[3] Legg, S., & Hutter, M. (2007). Universal intelligence: A definition of machine intelligence. Minds and machines, 17(4), 391-444.

[4] Lake, B. M., Ullman, T. D., Tenenbaum, J. B., & Gershman, S. J. (2017). Building machines that learn and think like people. Behavioral and brain sciences, 40, e253.

[5] Chollet, F. (2019). The abstraction and reasoning corpus (ARC). Unpublished manuscript.

[6] Chollet, F., Knoop, M., Kamradt, G., & Landers, B. (2024). Arc prize 2024: Technical report. arXiv preprint arXiv:2412.04604.

[7] Chollet, F., Tong, K., Reade, W., & Elliott, J. (2020). Abstraction and reasoning challenge. Retrieved from https://kaggle.com/competitions/abstraction-and-reasoning-challenge

[8] Besold, T. R., Bader, S., Bowman, H., Domingos, P., Hitzler, P., Kühnberger, K. U., ... & Zaverucha, G. (2021). Neural-symbolic learning and reasoning: A survey and interpretation. In *Neuro-symbolic artificial intelligence: The state of the art* (pp. 1–51). IOS Press.

[9] Kalyan, A., Mohta, A., Polozov, O., Batra, D., Jain, P., & Gulwani, S. (2018). Neural-guided deductive search for real-time program synthesis from examples. *arXiv preprint* arXiv:1804.01186.

[10] Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., ... & Anadkat, S. (2023). GPT-4 technical report. *arXiv preprint* arXiv:2303.08774.

[11] Anil, R., Borgeaud, S., Alayrac, J. B., Yu, J., Soricut, R., ... & Blanco, L. (2023). Gemini: A family of highly capable multimodal models. *arXiv preprint* arXiv:2312.11805.

[12] Introducing Claude 3, Anthropic 2024 https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf
[13] ARC Prize (2024). Abstraction and reasoning challenge: Blog. Retrieved from https://arcprize.org/blog/beat-arc-agi-deep-learning-and-program-synthesis
[14] Greenblatt, R. (2024, July). Getting 50% (SoTA) on ARC-AGI with GPT-4o. Blog post.
[15] Chollet, F., Knoop, M., Kamradt, G., & Landers, B. (2024). ARC Prize 2024: Technical report. *arXiv preprint* arXiv:2412.04604.
[16] Chollet, F., Knoop, M., Kamradt, G., Landers, B., & Pinkard, H. (2025). ARC-AGI-2: A new challenge for frontier AI reasoning systems. *arXiv preprint* arXiv:2505.11831.
[17] ARC Prize (2024). Abstraction and reasoning challenge. Retrieved from https://arcprize.org/
[18] ARC Prize (2024). Leaderboard. Retrieved from https://arcprize.org/leaderboard
[19] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2), 182–197.
[20] La Cava, W., Helmuth, T., Spector, L., & Moore, J. H. (2019). A probabilistic and multi-objective analysis of lexicase selection and E-lexicase selection. *Evolutionary Computation*, 27(3), 377–402.

The implementation supporting this research is available online: https://github.com/TurcoRiccardo/ARC_AGI.