Enhancing ARC-Solver with a Modular Reasoning Core: Toward Abstraction and Reasoning in AI

Anish Dhane

Master of Science in Artificial Intelligence

from the

University of Surrey

School of Computer Science and Electronic Engineering

Faculty of Engineering and Physical Sciences

University of Surrey

Guildford, Surrey, GU2 7XH, UK

September 2025

Supervised by: Prof. Miroslaw Bober

DECLARATION OF ORIGINALITY

I confirm that the project dissertation I am submitting is entirely my own work and that any material used from other sources has been clearly identified and properly acknowledged and referenced. In submitting this final version of my report to the JISC anti-plagiarism software resource, I confirm that my work does not contravene the university regulations on plagiarism as described in the Student Handbook. In so doing I also acknowledge that I may be held to account for any particular instances of uncited work detected by the JISC anti-plagiarism software, or as may be found by the project examiner or project organiser. I also understand that if an allegation of plagiarism is upheld via an Academic Misconduct Hearing, then I may forfeit any credit for this module or a more severe penalty may be agreed.

Enhancing ARC-Solver with a Modular Reasoning Core: Toward Abstraction and Reasoning in AI

Anish Dhane

Anish Dhane Date: 09/09/2025

Supervisor’s name: Prof. Miroslaw Bober

WORD COUNT

Number of Pages: 46

Number of Words: 11028

ABSTRACT

The Abstraction and Reasoning Corpus (ARC) was proposed as a benchmark to test the ability of artificial intelligence systems to demonstrate abstraction, reasoning, and generalization, qualities that are central to progress toward Artificial General Intelligence (AGI). Unlike data-intensive benchmarks, ARC requires solvers to infer general rules from only a few examples, mirroring the way humans use analogy and compositional reasoning. Despite rapid progress in deep learning, existing solvers continue to perform poorly on ARC tasks, often limited by brute-force search, brittle symbolic rules, or the inefficiency of neural approaches.

This dissertation addresses these challenges by extending the open-source ARC-Solver framework with a dedicated Reasoning Core. The Reasoning Core introduces modular and interpretable reasoning capabilities, combining beam search with step-by-step justifications and a genetic programming module to refine candidate solutions. The system expands the operator library with new spatial, content, and structural transformations, enabling richer and more flexible program synthesis.

Evaluation of the enhanced solver on the ARC benchmark demonstrates performance comparable to baseline ARC-Solver results, solving thirty-two tasks in total, while providing substantial qualitative improvements in interpretability and extensibility. The Reasoning Core not only produces solutions but also explains them through explicit reasoning steps, making it possible to trace how each transformation contributes to solving a task. These contributions highlight the value of embedding reasoning-aware modules into existing solvers and demonstrate how interpretability and modularity can pave the way for more generalizable approaches to ARC and related AGI benchmarks.

Keywords: Abstraction and Reasoning Corpus, ARC-Solver, Reasoning Core, Artificial General Intelligence, program synthesis, interpretable AI

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

Artificial Intelligence (AI) has emerged as one of the most transformative fields in modern computer science, with applications ranging from natural language processing and computer vision to robotics and autonomous systems. In recent years, deep learning has powered impressive breakthroughs, yet most of these advances rely on vast datasets and large-scale pattern recognition rather than genuine reasoning or abstraction. The pursuit of Artificial General Intelligence (AGI)—systems that can learn, generalize, and solve novel problems with human-like flexibility—remains a major unsolved challenge.

One of the most ambitious benchmarks in this pursuit is the **Abstraction and Reasoning Corpus (ARC)**, introduced by François Chollet [1][2]. ARC is designed not to test narrow perceptual skills but to evaluate the ability of systems to infer general rules from minimal examples, reflecting the abstraction and compositional reasoning processes that humans excel at. The benchmark poses tasks in the form of grid-based input–output pairs, where a solver must discover the underlying transformation and apply it to unseen test cases. Success on ARC requires a blend of perception, abstraction, and structured reasoning, making it a rigorous testbed for AGI research.

This dissertation is situated in the domain of program synthesis and reasoning in AI, with a strong focus on the computational and experimental aspects of solver development. The work involves the analysis of existing solvers, the design and programming of a modular **Reasoning Core**, and the integration of this module into an open-source ARC-Solver baseline [5]. Through this, the project contributes both a practical system capable of solving ARC tasks and a conceptual exploration of how modular reasoning can advance the field of AGI.

1.1 Background and Context

Artificial Intelligence (AI) has undergone rapid advances over the last decade, driven largely by the success of deep learning models. These systems, particularly large language models (LLMs), have demonstrated remarkable abilities in natural language understanding, image recognition, and multimodal reasoning. However, there is still a significant gap between these capabilities and the broader ambition of Artificial General Intelligence (AGI)—the ability of a system to generalize knowledge and reasoning across diverse, unfamiliar tasks in a way that resembles human intelligence [10].

The Abstraction and Reasoning Corpus (ARC), introduced by François Chollet in 2019 [1][2], represents one of the most ambitious benchmarks for AGI. Unlike datasets such as ImageNet, which reward perceptual pattern recognition, ARC tasks are explicitly designed to test higher-order reasoning. Each ARC problem is presented as a series of input–output grid pairs (examples), from which the solver must infer a general transformation rule and apply it to novel test cases. Humans typically solve these tasks within seconds by leveraging abstraction, analogy, and prior knowledge, whereas most AI systems struggle.

My exploration of ARC began with an influential article by **Anokas (2023)**, titled *"LLMs Struggle with Perception, Not Reasoning"* [4]. This article provided a critical perspective on why LLMs, despite their apparent reasoning ability, often fail on structured tasks like ARC. The author argued that these failures are more rooted in perceptual limitations than in a fundamental lack of reasoning. This insight helped me understand ARC not only as a set of puzzles but also as a testbed for disentangling perception from reasoning in AI systems.

Building upon this understanding, I reviewed the **ARC Prize 2024 report** [3], which documented global efforts to solve ARC tasks. This report provided a comprehensive overview of methodologies, ranging from brute-force search solvers to hybrid symbolic-neural systems. Among the many solutions presented, the **Omni-ARC solver** emerged as particularly noteworthy [6]. Its modular design and broad applicability made it one of the most manageable and effective attempts to tackle the benchmark. However, my review revealed that Omni-ARC still struggled with tasks requiring deep abstraction and multi-step reasoning. This limitation was consistent with the critique highlighted by Anokas [4]: existing solvers often succeed in recognizing surface-level transformations but fail when reasoning is required across higher-order abstractions.

Motivated by this gap, I set out to design a **Reasoning Core** that could augment existing solvers with stronger abstraction and logical inference capabilities. Initially, I considered extending Omni-ARC [6], but its implementation was not fully open-source, limiting opportunities for direct experimentation. To overcome this constraint, I turned to an alternative open-source solver, **ARC-Solver**, developed by Paulo H. S. Silva [5]. This framework provided a transparent and flexible baseline on which I could build. By integrating my Reasoning Core into ARC-Solver, I aimed to create a system that combined the practical strengths of existing solvers with novel mechanisms for abstraction and reasoning.

1.2 Scope and Objectives

The overarching aim of this dissertation is to enhance ARC-solving performance by integrating a Reasoning Core into an existing solver. More specifically, the project focuses on tackling the reasoning limitations identified in ARC-Solver [5] and related approaches, while ensuring that the solution remains reproducible, transparent, and open-source.

The scope of the project includes:

- Conducting a systematic review of ARC tasks and existing solvers [3][4][6].
- Selecting ARC-Solver as the baseline implementation for extension [5].
- Designing a Reasoning Core module that targets the abstraction and reasoning shortcomings of existing solvers.
- Integrating this Reasoning Core into ARC-Solver, while maintaining compatibility with the original architecture.
- Evaluating the enhanced solver's performance against benchmark ARC tasks and comparing results to both ARC-Solver and other solutions such as Omni-ARC [6].

The following objectives guide the project:

- **Objective 1**: Investigate existing ARC benchmarks and solutions, with emphasis on Omni-ARC [6] and ARC-Solver [5].
- **Objective 2**: Identify abstraction and reasoning limitations in current solvers [3][4].
- **Objective 3**: Design a modular Reasoning Core capable of enhancing solver generalization.
- **Objective 4**: Implement and integrate the Reasoning Core into ARC-Solver [5].
- **Objective 5**: Evaluate the enhanced solver's performance through empirical testing and comparative analysis.

1.3 Achievements

At the time of writing, several milestones have already been accomplished:

- **Literature Review**: I have surveyed the theoretical background of ARC tasks, including the ARC Prize 2024 report [3], which outlines the current state of the field. This provided insights into the strengths and weaknesses of existing solvers and informed the design of my Reasoning Core.
- **Framework Selection**: After evaluating multiple candidates, I selected ARC-Solver [5] as the baseline implementation. This choice was motivated by its open-source availability, modular design, and capacity for extension.
- **Reasoning Core Design**: I have proposed a Reasoning Core architecture that introduces abstraction mechanisms and logical inference strategies absent in the baseline solver.
- **Integration**: The Reasoning Core has been successfully embedded into ARC-Solver [5], producing a unified system capable of addressing tasks previously unsolvable by the baseline.
- **Preliminary Evaluation**: Initial experiments suggest that the enhanced solver demonstrates improvements on tasks requiring higher-order reasoning, though further evaluation is necessary to quantify these gains comprehensively.

1.4 Overview of Dissertation

The remainder of this dissertation is organized as follows:

- **Chapter 2 – Background Theory and Literature Review**: This chapter provides a detailed overview of ARC, its cognitive motivations, and its position within the broader landscape of AI benchmarks [1][2][8]. It also surveys prior approaches, with a particular focus on ARC Prize 2024 solutions [3][6] and highlights the reasoning challenges that motivate the present work [4].
- **Chapter 3 – Baseline: ARC-Solver**: This chapter introduces ARC-Solver in detail [5], describing its architecture, capabilities, and limitations. It serves as the baseline against which improvements will be evaluated.
- **Chapter 4 – Design of the Reasoning Core**: This chapter presents the conceptual

framework of the Reasoning Core, outlining its components, integration strategy, and expected contributions to solving ARC tasks.

- **Chapter 5 – Implementation and Evaluation**: This chapter provides the technical details of implementing the Reasoning Core within ARC-Solver. It also reports experimental results, performance comparisons, and error analyses.
- **Chapter 6 – Conclusion and Future Work**: The final chapter summarizes the contributions of the dissertation, evaluates progress against the stated objectives, and proposes avenues for future research.

2 BACKGROUND THEORY AND LITERATURE REVIEW

2.1 The Abstraction and Reasoning Corpus (ARC)

2.1.1 Origins and Motivation

The Abstraction and Reasoning Corpus (ARC) was introduced by François Chollet in 2019 [1][2] as part of his broader argument that existing artificial intelligence benchmarks were insufficient for measuring progress toward Artificial General Intelligence (AGI). At the time, deep learning systems had already surpassed human performance in tasks such as image recognition (ImageNet) and natural language processing (GLUE, SuperGLUE). However, Chollet argued that these benchmarks primarily measured pattern recognition at scale, not generalization from few examples—a skill that is central to human intelligence.

ARC was therefore designed with a radically different philosophy. Each task in ARC is not about learning from millions of labelled examples but about reasoning from a handful of demonstrations. Specifically, every ARC task consists of several input–output grid pairs that represent the "training examples." The system must infer the underlying transformation rule and then apply it to one or more test inputs to generate the correct outputs.

The motivation behind ARC was twofold:

1. **Evaluate core reasoning abilities** – By stripping away reliance on large datasets, ARC highlights whether a system can generalize concepts.
2. **Benchmark human-level cognition** – Humans solve most ARC tasks quickly, using prior knowledge, analogy, and abstraction. If a machine could do the same, it would suggest meaningful progress toward AGI [1].

Chollet positioned ARC not as an isolated challenge but as part of a broader intellectual framework on measuring intelligence as skill-acquisition efficiency [2]. ARC tasks embody this philosophy by testing whether a system can acquire and transfer skills efficiently across domains.

2.1.2 Structure of ARC Tasks

ARC tasks are presented as small, coloured grids, where each grid cell can contain a discrete integer value corresponding to a colour. A typical task might involve one or more training pairs and one or more test cases. The system must analyse the relationship between input and output grids in the training examples and then apply this transformation to the unseen test inputs.

For example, a training pair may show a grid with a blue square on the left and the corresponding output grid with the square shifted one cell to the right. A human instantly recognizes the transformation ("move the square right by one cell"). The system must infer this transformation without explicit instructions.

The design of ARC emphasizes several principles [1][2]:

- **Minimal data**: Typically, only 1–5 training pairs are provided.
- **High variability**: Tasks span diverse categories—translation, reflection, scaling, pattern completion, and more.
- **Compositionality**: Many tasks require multi-step reasoning, where transformations must be applied sequentially (e.g., trim borders and then rotate).
- **Generalization**: The test cases often differ significantly from the training examples, requiring the solver to infer rules at an abstract level rather than memorizing patterns.

This design makes ARC radically different from traditional machine learning datasets. Whereas ImageNet rewards brute-force statistical learning from massive data, ARC rewards the ability to infer general rules from minimal exposure.

2.1.3 Why ARC is Different from Standard ML Benchmarks

The uniqueness of ARC can be appreciated by contrasting it with prior benchmarks:

- **ImageNet (2009)**: Measures large-scale visual recognition. Success requires millions of labelled examples and massive neural networks. Generalization beyond the training set is limited.
- **GLUE / SuperGLUE (2018–2019)**: Benchmarks for natural language understanding. Again, success requires massive pretraining and fine-tuning on large corpora.
- **Atari / DeepMind Control Suite**: Benchmarks for reinforcement learning agents, typically requiring millions of episodes to converge.

In each of these, scale of data and computation is central to success. ARC deliberately strips this away. By giving only a few examples, it asks: *Can the system reason abstractly, without relying on data scale?*

For this reason, Chollet argued that progress on ARC would be a stronger signal of progress toward AGI than improvements on data-heavy benchmarks [1][2]. In fact, he claimed that many celebrated advances in deep learning reflected "narrow intelligence at scale" rather than true generalization. ARC was designed as a corrective to this trend.

2.1.4 Strengths and Criticisms of ARC

ARC has been widely praised for shifting the conversation toward generalization, abstraction, and reasoning, which are essential components of intelligence [1][2]. By being dataset-agnostic and domain-general, ARC prevents researchers from overfitting to a narrow benchmark. Its structure ensures that any solver must engage in genuine reasoning rather than memorization.

However, ARC is not without criticism:

1. **Ambiguity of tasks** – Some ARC problems have multiple plausible transformations, and even humans occasionally disagree on the "intended" rule. This ambiguity complicates both

evaluation and solver design.

2. **Subjectivity in design** – Because ARC tasks were handcrafted by Chollet, some researchers argue that the benchmark reflects a particular conception of intelligence rather than a universally accepted definition.
3. **Difficulty for machines** – Current solvers perform very poorly—often solving fewer than 5% of test tasks—leading some to question whether ARC is too hard or ill-posed as a benchmark [3][6].
4. **Evaluation metric limitations** – Success is binary (solved or not), which may obscure partial progress. For example, a solver that nearly reconstructs the correct output receives the same score as one that fails entirely.

Despite these criticisms, ARC remains a landmark benchmark because it directly challenges AI systems to demonstrate capabilities that are qualitatively closer to human intelligence. Moreover, the growing interest in ARC competitions, such as the **ARC Prize 2024** [3], suggests that the community increasingly values benchmarks that emphasize reasoning over perception.

2.1.5 Relevance to This Dissertation

ARC serves as the primary motivation for this dissertation. By working on ARC tasks, the project directly engages with one of the most important open challenges in AI: the ability to reason abstractly from minimal data.

The open-source **ARC-Solver framework** [5] provides a practical baseline but lacks the reasoning mechanisms necessary to excel at ARC. By designing and integrating a Reasoning Core, this dissertation contributes toward bridging that gap. The significance lies not only in achieving marginal performance gains but also in demonstrating a methodology for embedding interpretability and reasoning into AI solvers, thereby aligning with the broader vision of ARC as a testbed for progress toward AGI [1][2][8].

2.2 Cognitive Science and Abstraction in AI

2.2.1 Human Cognitive Mechanisms in Abstraction and Analogy

One of the defining features of human intelligence is the ability to abstract—to strip away irrelevant details and focus on the essential structure of a problem. This skill enables humans to generalize knowledge across domains and to solve problems in novel contexts. Cognitive science research suggests that humans rely heavily on analogy as a mechanism for abstraction: when confronted with a new task, people often map it onto prior experiences that share structural similarities, even if surface-level features differ [8].

For example, in solving an ARC task, a person might recall that a shape once needed to be rotated in a similar puzzle. Even if the grid colours or dimensions differ, the analogy to the earlier transformation allows rapid inference. This analogical reasoning is fundamental to how humans achieve

generalization from very few examples [8].

ARC tasks were deliberately designed to exploit this aspect of human cognition. Each puzzle is novel in its surface presentation, but solvable by applying familiar abstract operations such as reflection, rotation, repetition, or scaling [1][2]. The benchmark therefore tests whether a system can mimic this human capacity for structural abstraction rather than rote memorization.

2.2.2 Links Between Human Problem-Solving and ARC Design

ARC's construction is deeply inspired by theories of cognitive development. In particular, it reflects ideas from Jean Piaget's work on constructivism, which emphasizes that intelligence develops through the construction of schemas—abstract rules that can be applied flexibly to new situations [8]. Each ARC task, in essence, requires the solver to construct or identify the relevant schema that maps inputs to outputs.

Moreover, ARC tasks rely on compositional reasoning, a hallmark of human intelligence [1][2]. Compositionality refers to the ability to build complex operations out of simpler ones. For instance, trimming borders followed by rotating a shape involves composing two simpler transformations into a meaningful sequence. Humans find this natural because compositionality underlies both language and reasoning. For machines, however, discovering the correct composition from a limited set of examples remains a major challenge.

The human-centred design of ARC ensures that solving these tasks requires the same kinds of schema construction, analogy, and compositional reasoning that cognitive science associates with flexible intelligence [8].

2.2.3 Perception vs. Reasoning Debate

A central question in both cognitive science and AI research is the relationship between perception and reasoning. Humans effortlessly integrate perceptual input with higher-level reasoning. For machines, however, perception and reasoning often emerge as distinct challenges.

The article *"LLMs Struggle with Perception, Not Reasoning"* by Anokas (2023) [4] argued that large language models frequently fail on tasks such as ARC not because they lack reasoning ability, but because they are not well-equipped to handle low-level perceptual abstractions (e.g., parsing pixel grids into meaningful structures). Once perceptual barriers are overcome, their reasoning capabilities may be surprisingly strong.

This perspective is particularly relevant for ARC. A significant portion of the challenge lies in perceptual grounding—understanding what the coloured grid represents and how to represent transformations. If perception is inadequate, reasoning cannot proceed effectively. Conversely, even with strong perception, the absence of a reasoning mechanism (e.g., compositional rules) leads to failure [4].

The perception–reasoning divide highlights why ARC solvers that rely solely on brute-force search

or statistical pattern recognition often underperform [3][5]. They lack the bridging mechanism between perception and reasoning that humans possess. The dissertation's proposed Reasoning Core seeks to address precisely this gap: to provide a structured reasoning layer that can operate once perception identifies candidate features.

2.2.4 Implications for AGI

The cognitive science principles embodied in ARC have profound implications for Artificial General Intelligence. If AGI is defined as the ability to perform well across a wide range of tasks with minimal prior training, then abstraction and reasoning are indispensable [8]. Benchmarks like ImageNet or GLUE tell us little about this ability because they test narrow competencies optimized through scale. ARC, by contrast, aligns closely with human-like learning [1][2].

Several insights follow from this:

1. **Efficiency over scale** – Humans can solve ARC tasks after seeing only one or two examples. AGI systems must aim for similar efficiency rather than relying on massive datasets [8].
2. **Interpretability** – Human reasoning is transparent—we can explain why a given transformation solves the task. Systems aspiring to AGI should similarly provide interpretable reasoning steps [8].
3. **Transferability** – The schemas humans form in solving ARC tasks (e.g., symmetry, repetition) are transferable to other domains. A machine capable of similar abstraction would demonstrate a core hallmark of intelligence: transfer across contexts [8].
4. **Limitations of current AI** – The poor performance of state-of-the-art AI systems on ARC underscores how far current approaches remain from AGI, despite their successes in other domains [3][6][10].

Thus, ARC is not merely a puzzle collection but a cognitive benchmark—a test of whether machines can acquire and apply abstractions in a manner comparable to humans [1][2][8]. By engaging directly with this challenge, this dissertation contributes to the broader quest for AGI by exploring how a Reasoning Core might narrow the perception–reasoning gap [5].

2.2.5 Relevance to This Dissertation

The cognitive science foundations reviewed here motivate the design choices of this project. By incorporating a Reasoning Core, the aim is to mirror human strategies of abstraction and compositional reasoning. For example, the module's ability to decompose solutions into sequential steps (e.g., trim → rotate → recolour) directly reflects human compositional reasoning. Similarly, the integration of beam search with justifications echoes how humans consider multiple candidate explanations before settling on one.

This dissertation therefore builds upon the cognitive science insight that intelligence is not only about perception but also about structured reasoning [8]. By embedding this principle into the ARC-Solver

framework [5], the project contributes both to the specific goal of solving ARC tasks and to the broader scientific goal of modelling human-like intelligence in machines [1][2][8].

2.3 Review of ARC Prize 2024 Solutions

2.3.1 Overview of the Competition and Goals

The **ARC Prize 2024** was the largest organized effort to date to tackle the Abstraction and Reasoning Corpus [3]. It brought together researchers from academia, independent AI practitioners, and industry teams to address a central question: *Can we design systems capable of solving ARC tasks at a level approaching human performance?*

The competition framework mirrored other AI benchmarks in spirit, but its emphasis on reasoning and abstraction made it distinctive. Participants were required to design solvers that could handle the diverse set of ARC training and evaluation tasks, with results measured by the proportion of test cases correctly solved. Unlike conventional benchmarks, ARC's extreme difficulty meant that success was measured not in absolute performance (since all systems performed poorly compared to humans) but in the relative effectiveness of different methodological strategies [3].

The 2024 competition report provided a comprehensive survey of submitted solutions. These approaches fell into several broad categories: search-based solvers, symbolic rule induction, neural methods, hybrid models, and novel modular systems such as **Omni-ARC** [6]. The report thus served as a valuable snapshot of the state of the art, highlighting both progress and persistent challenges.

2.3.2 Categories of Approaches

(a) Search-Based Solvers:

Many of the earliest approaches to ARC relied on program synthesis and search [3]. The idea was to define a space of possible transformations (e.g., rotation, reflection, scaling, coloring) and then search for a sequence of transformations that mapped inputs to outputs. Search strategies included brute force, evolutionary algorithms, and constraint-guided methods.

Strengths:

- Transparent: the discovered program could be directly interpreted.
- Flexible: capable of solving a broad range of simple tasks.

Weaknesses:

- Search space explosion: the number of possible transformation sequences grows combinatorially with depth.
- Poor generalization: unless heavily constrained, search-based solvers often overfit to training examples without correctly solving test cases.

These solvers typically achieved modest performance (≈1–5% success rates), demonstrating the limitations of unguided search in a domain that requires abstraction [3].

(b) Symbolic Rule Induction

Another family of approaches focused on symbolic reasoning. These solvers attempted to induce rules by analysing the structural relationships between input and output grids [3]. For example, they might detect that "all red squares become blue" or that "objects are mirrored along the vertical axis."

Strengths:

- Aligns closely with human reasoning.
- Provides interpretable solutions, often expressible as rules.

Weaknesses:

- Fragile: symbolic induction often fails when tasks involve noisy, ambiguous, or multi-step transformations.
- Narrow: predefined symbolic vocabularies limit flexibility.

While these solvers were appealing from a cognitive science perspective [8], their coverage was limited. Many ARC tasks require compositions of multiple rules, which symbolic induction systems often failed to capture [3].

(c) Neural and Deep Learning Methods

Some teams experimented with deep neural networks [3]. For example, convolutional neural networks (CNNs) or transformers were trained to map input grids directly to output grids. Others used meta-learning to attempt few-shot generalization across tasks.

Strengths:

- Potential scalability with large compute resources.
- Ability to capture visual patterns without explicit symbolic representations.

Weaknesses:

- Data inefficiency: neural models typically require large training sets, which ARC does not provide.
- Black-box nature: outputs are difficult to interpret, making debugging and analysis harder.
- Poor transfer: models trained on subsets of ARC rarely generalized well to unseen tasks.

Overall, purely neural methods underperformed dramatically compared to expectations. Many failed to solve more than a handful of tasks, confirming Chollet's critique that deep learning excels at perception but struggles with abstraction and reasoning [1][2][4].

(d) Hybrid Symbolic-Neural Approaches

A promising category involved combining neural perception with symbolic reasoning [3]. For instance, a neural module could detect shapes or objects in the grid, while a symbolic module inferred transformation rules.

Strengths:

- Synergistic: neural modules handle perception; symbolic modules handle reasoning.
- Potentially scalable: hybrid systems could leverage strengths of both paradigms.

Weaknesses:

- Integration complexity: aligning neural and symbolic representations is non-trivial.
- Limited performance: while hybrid models solved more tasks than purely symbolic or purely neural systems, they still fell far short of human-level performance.

This category represented one of the most active areas of innovation, reflecting the broader AI community's interest in neurosymbolic methods [8].

2.3.3 Omni-ARC in Detail

Among the solutions presented in the ARC Prize 2024, **Omni-ARC** stood out as one of the most structured and promising [6]. It was designed as a modular, extensible framework that could handle a wide variety of ARC tasks using a combination of rule-based reasoning and flexible program composition.

Design Features:

- **Modularity**: Omni-ARC decomposed tasks into smaller subtasks handled by specialized modules.
- **Program induction**: It searched for compositional programs that transformed inputs into outputs.
- **Extensibility**: Its architecture allowed new modules and operators to be added without redesigning the entire system.
- **Transparency**: Each solution was expressed as a sequence of interpretable steps, aligning with ARC's emphasis on reasoning.

Strengths:

- Covered a wide variety of tasks compared to many other solvers.
- Its modular design made it easier to debug and extend.
- Achieved some of the best relative performance in the competition [6].

Limitations:

- Still struggled with abstraction-heavy tasks requiring reasoning beyond the provided operator set.
- Computationally intensive, with search times growing significantly for multi-step tasks.
- Not fully open source, limiting its accessibility for replication and further experimentation [6].

Omni-ARC thus represented a milestone: it demonstrated that modularity and compositional reasoning were the right design direction, but it also underscored the persistent reasoning gap in ARC solutions.

2.3.4 Lessons Learned from the Competition

The ARC Prize 2024 provided several important takeaways for the AI community [3][6]:

1. No single paradigm is sufficient: Search-based, symbolic, and neural approaches all had strengths but also major weaknesses.
2. Compositional reasoning is essential: Systems that attempted to compose simple transformations into multi-step programs fared better than those that did not.
3. Interpretability matters: The most successful solvers provided transparent reasoning steps, echoing the human ability to justify solutions.
4. Performance remains very low: Even the best solvers solved only a small fraction of tasks, highlighting the difficulty of ARC and the long road to AGI [1][2].
5. Future progress requires modular reasoning cores: The gap identified—between perception and abstraction—suggests that future solvers must include dedicated reasoning components rather than relying solely on search or brute force [5].

2.3.5 Relevance to This Dissertation

This dissertation builds directly on the lessons of the ARC Prize 2024 [3]. The survey of approaches, and the analysis of Omni-ARC in particular [6], informed two key design decisions:

- **First**, the need for modular reasoning: the Reasoning Core introduced in this project is explicitly designed to be a modular extension to an existing solver (ARC-Solver [5]).
- **Second**, the importance of open-source accessibility: because Omni-ARC was not fully open, ARC-Solver was chosen as the baseline to ensure reproducibility and transparency [5][6].

By integrating a Reasoning Core into ARC-Solver [5], this dissertation aligns with the community's recognition that reasoning-aware architectures represent the most promising path forward. At the same time, it contributes a concrete, open-source demonstration of how such a core can be implemented and evaluated.

2.4 Reasoning in AI Systems

2.4.1 Symbolic Reasoning Approaches in AI

The earliest era of artificial intelligence, often referred to as **Good Old-Fashioned AI (GOFAI)**, was dominated by symbolic reasoning [8]. Systems such as expert systems and logic-based inference engines represented knowledge explicitly using rules, facts, and symbolic structures. Reasoning was achieved through formal logic, search, and theorem proving.

Strengths of symbolic reasoning included:

- **Transparency**: Rules were human-readable and interpretable.
- **Compositionality**: Complex inferences could be built from simple rules.

- **Correctness guarantees**: Logical proofs could be formally verified.

However, symbolic systems suffered from the so-called **knowledge acquisition bottleneck**. They required carefully hand-crafted rules and struggled with perception and ambiguity. For example, mapping raw sensory data (e.g., pixels in ARC grids) into symbolic representations was non-trivial [1][2]. As a result, purely symbolic AI struggled outside well-defined domains like mathematics or board games.

Despite their limitations, symbolic approaches remain highly relevant to ARC because ARC tasks are inherently symbolic and structural [2]. Many transformations, such as "rotate shape" or "mirror along axis," can be elegantly expressed in rule-based form. The challenge lies in discovering the correct symbolic rules automatically from few examples [3].

2.4.2 Neural Approaches to Reasoning

With the resurgence of deep learning in the 2010s, AI shifted toward **neural networks**. These systems excelled at perception tasks, such as image recognition and natural language processing, by learning statistical patterns from massive datasets. However, their ability to perform structured reasoning has been hotly debated [10].

Neural reasoning approaches include:

- **Neural program induction**: Training networks to generate symbolic programs (e.g., Neural Turing Machines, Differentiable Neural Computers).
- **Transformer-based reasoning**: Using attention mechanisms to capture relationships between elements.
- **End-to-end task mapping**: Training networks to directly map inputs to outputs without explicit symbolic representation.

While impressive in narrow tasks, neural methods face several challenges when applied to ARC [3]:

- **Data inefficiency**: ARC provides very few training examples, whereas neural models typically require millions.
- **Black-box nature**: Neural reasoning is opaque, offering little interpretability [4].
- **Generalization failures**: Neural networks often fail catastrophically on tasks that differ from their training distribution [1][2].

This aligns with the critique from Chollet and Anokas: neural systems may appear capable of reasoning but often rely on **pattern recognition, not true abstraction** [1][2][4].

2.4.3 Hybrid Symbolic-Neural Approaches

Recognizing the complementary strengths of symbolic and neural paradigms, researchers have increasingly pursued **hybrid or neurosymbolic AI** [8]. These systems aim to combine the perception strengths of neural networks with the abstraction and interpretability of symbolic reasoning.

Examples include:

- **Perception–Reasoning pipelines**: Neural modules first detect objects, which are then fed into symbolic solvers (e.g., for visual question answering).
- **Differentiable reasoning**: Symbolic structures embedded in differentiable architectures to enable gradient-based training.
- **Program synthesis hybrids**: Neural networks propose candidate symbolic programs, which are then verified or refined symbolically.

In the context of ARC, hybrid approaches are particularly appealing. Neural modules can parse grid structures (e.g., detect shapes, identify symmetry), while symbolic modules can infer the transformations needed to map inputs to outputs [3]. This division of labour reflects how humans combine perception with reasoning [8].

Hybrid methods were among the more promising directions in the **ARC Prize 2024**, though their performance still fell short of expectations [3][6]. They highlight, however, the importance of modularity and division of cognitive labour, principles directly reflected in the **Reasoning Core** designed in this dissertation [5].

2.4.4 Modular Reasoning Architectures

Beyond symbolic and hybrid systems, a more recent trend emphasizes **modularity** in reasoning architectures. The idea is that intelligence may be best achieved not by monolithic models but by composing specialized modules, each responsible for a distinct aspect of cognition [8].

For instance, systems inspired by cognitive architectures (such as ACT-R or SOAR) divide cognition into modules for perception, memory, reasoning, and action. Similarly, in modern AI, modular designs are seen in:

- **Neural Module Networks**: Compositions of task-specific neural units arranged dynamically.
- **Program synthesis frameworks**: Modular operators that can be composed to solve tasks [3].
- **Omni-ARC**: Explicitly modular design that assigns tasks to specialized submodules [6].

The **Reasoning Core** in this dissertation is a step in this direction. Rather than attempting to solve ARC tasks end-to-end, the system adds a dedicated reasoning module to the **ARC-Solver baseline** [5]. This modular approach ensures interpretability, extensibility, and targeted improvement in abstraction and reasoning.

2.4.5 Lessons for ARC Solvers

The history of reasoning in AI suggests several key lessons that inform this dissertation:

1. Purely symbolic or purely neural systems are insufficient. ARC requires both perception and reasoning [1][2][3].
2. Interpretability is essential. Reasoning steps must be transparent to ensure both correctness

and human-aligned evaluation [4][8].

3. Compositionality underpins generalization. Solvers must be able to compose simple operators into complex transformations [6].
4. Modularity is a promising design principle. Dedicated reasoning modules, like the Reasoning Core, represent a viable way forward [5][6].

2.4.6 Relevance to This Dissertation

This project situates itself within the trajectory of AI reasoning research [8]. By designing a **Reasoning Core**, it embraces symbolic transparency, draws on neural-inspired modularity, and integrates with an open-source solver [5]. It contributes to the hybrid tradition by bridging perception and reasoning while maintaining ARC's requirement for interpretability [1][2].

The Reasoning Core thus exemplifies a broader trend in AI: moving beyond monolithic, black-box models toward modular, interpretable systems that reflect both cognitive science insights and practical engineering needs [3][6][8].

2.5 Gap Analysis and Research Motivation

2.5.1 Common Limitations Across ARC Solvers

The survey of **ARC Prize 2024 solutions** reveals several recurring limitations that constrain progress on the benchmark [3][6]:

- **Over-reliance on search**: Many solvers depend heavily on brute-force search over large operator spaces. While occasionally effective, this approach is computationally expensive and scales poorly with task complexity [3].
- **Lack of abstraction**: Even when solvers succeed on surface-level transformations, they struggle with tasks that require multi-step reasoning or higher-order generalization [1][2].
- **Brittleness of symbolic systems**: Purely symbolic solvers fail when confronted with tasks that do not fit neatly into their predefined rule sets [3][8].
- **Data inefficiency of neural systems**: Neural approaches require far more data than ARC provides, leading to poor generalization and overfitting [4][10].
- **Integration difficulties in hybrid systems**: While promising in theory, neurosymbolic systems often falter in practice because aligning perceptual outputs with symbolic reasoning remains technically challenging [3][8].

Together, these limitations highlight why ARC remains unsolved and why performance across all approaches is dramatically lower than human-level benchmarks [3].

2.5.2 Why Abstraction is Particularly Hard for Machines

Abstraction requires the ability to focus on the essential structure of a problem while ignoring irrelevant details. For example, in an ARC task involving coloured shapes, humans can easily infer that

the transformation is "mirror across the vertical axis," regardless of the specific colours or grid size. Machines, by contrast, often become entangled in surface-level features (specific pixel arrangements, colour IDs), which prevents them from capturing the underlying rule [1][2][8].

The difficulty stems from:

- **Representation challenges**: Machines must first parse the grid into meaningful components (objects, symmetries, relationships) [3].
- **Compositionality**: Many tasks require sequencing multiple abstract steps, and the search space grows rapidly [6].
- **Lack of inductive bias**: Neural models lack built-in biases for abstraction, unlike humans who naturally perceive symmetry, repetition, and relational structure [8][10].

This explains why ARC is both uniquely difficult and uniquely valuable as a benchmark—it isolates abstraction as the key test of general intelligence [1][2].

2.5.3 The Case for a Dedicated Reasoning Core

The limitations of existing solvers and the inherent difficulty of abstraction suggest that progress requires a **dedicated reasoning module**. Such a module would:

- **Bridge perception and abstraction**: Once objects and features are identified, the Reasoning Core can infer transformation rules [5].
- **Enable compositional reasoning**: By chaining operators (e.g., trim → rotate → recolour), the core can solve multi-step tasks [3][6].
- **Provide interpretability**: Reasoning steps can be explained, aligning with human cognition and supporting debugging [4][8].
- **Be extensible**: New operators or heuristics can be added to the module without redesigning the entire system [5].

This dissertation responds to that need by designing and integrating a **Reasoning Core** into the ARC-Solver framework [5]. Unlike **Omni-ARC**, which was not fully open-source [6], ARC-Solver provides a transparent baseline, making it an ideal candidate for extension.

2.5.4 Positioning of This Dissertation's Contribution

This dissertation positions itself at the intersection of theory and practice:

- **Theoretical significance**: By operationalizing cognitive science insights into a modular reasoning system, it demonstrates how abstraction and compositionality can be engineered into AI solvers [8].
- **Practical significance**: By enhancing an open-source solver, it contributes tangible improvements that can be built upon by the wider research community [5].
- **Scientific significance**: By evaluating the Reasoning Core against baseline ARC-Solver, it provides empirical evidence about the feasibility of modular reasoning approaches [3][6].

In this way, the dissertation not only extends ARC-Solver but also contributes to the broader conversation about how AI systems can move closer to human-like reasoning [1][2][8].

2.6 Summary

This chapter has reviewed the theoretical and practical foundations relevant to this dissertation. It began with an introduction to the Abstraction and Reasoning Corpus (ARC), explaining its origins, structure, and significance as a benchmark for AGI. It then drew on cognitive science to show how ARC tasks align with human reasoning processes, particularly abstraction, analogy, and compositionality.

The chapter also surveyed the ARC Prize 2024, analysing categories of solvers ranging from search-based methods to hybrid symbolic-neural systems. Special attention was given to Omni-ARC, which, despite its modular promise, remained constrained by reasoning limitations and lack of openness. The discussion then broadened to situate ARC within the history of AI reasoning research, spanning symbolic, neural, hybrid, and modular approaches.

The **gap analysis** highlighted the limitations of existing solvers—over-reliance on search, brittleness of symbolic systems, inefficiency of neural approaches, and integration challenges in hybrids. It further emphasized why abstraction is inherently difficult for machines. Against this backdrop, the chapter introduced the motivation for this dissertation: the design and integration of a **Reasoning Core** to enhance ARC-Solver.

By providing a dedicated, modular, and interpretable reasoning layer, the project aims to address critical shortcomings in existing solvers. The next chapter, **Chapter 3**, introduces the ARC-Solver baseline, explaining its architecture, strengths, and limitations. This provides the foundation upon which the Reasoning Core is built and evaluated.

3 BASELINE SYSTEM: ARC-SOLVER

3.1 Overview of ARC-Solver

The **ARC-Solver** framework, developed by Paulo H. S. Silva [5], is an open-source attempt to address the **Abstraction and Reasoning Corpus (ARC)** challenge through a **program synthesis approach**. Rather than learning end-to-end mappings between input and output grids, ARC-Solver searches for programs composed of symbolic operators that transform inputs into outputs. This aligns well with the **compositional nature of ARC tasks** [1][2] and provides a transparent baseline for extension.

3.1.1 Core Architecture

ARC-Solver is organized around a small set of Python modules, each responsible for different aspects of the solver pipeline [5]:

- **dsl.py (Domain-Specific Language):** Defines a library of primitive operations, such as rotation (rot90), reflection (flip_lr), trimming (trim), and color replacement (replace). These primitives form the "alphabet" from which programs are composed.
- **solver.py:** Implements the core program synthesis logic, including search strategies to discover operator sequences that map input grids to output grids.
- **evaluate_solver.py:** Provides evaluation routines for running the solver across multiple ARC tasks and recording performance.
- **synthesize_and_eval.py:** Combines synthesis and evaluation in a single pipeline, allowing for systematic testing across the ARC dataset.

The solver operates by attempting to discover a program — a sequence of operations from dsl.py — that maps the input grid(s) of a training pair to the corresponding output. Once a candidate program is synthesized, it is applied to unseen test inputs, and the predicted outputs are compared against the ground truth [5].

3.1.1.1 Grid Representation

In ARC-Solver, tasks are represented as small **integer-valued grids** (e.g., 10×10). Each integer corresponds to a colour. The grid is stored as a numpy.ndarray, which allows efficient manipulation of pixel-level operations [5].

Programs are expressed as lists of operators, for example:

- ["trim", "rot90"]

which corresponds to trimming the borders of the input grid and then rotating it by 90 degrees.

This representation balances **interpretability with computational flexibility**: programs can be easily serialized, logged, and inspected, while operators are composable in arbitrary sequences [5].

3.1.2 Transformation Mechanisms

The synthesis logic is primarily **search-based** [5]. Candidate programs are generated by enumerating sequences of operators from the DSL. To prevent combinatorial explosion, ARC-Solver imposes limits on program depth and applies heuristics to guide search [1][2].

For example, the solver might restrict itself to depth-2 or depth-3 programs such as:

- ["flip_lr"]
- ["rot90", "trim"]
- ["replace_1_to_2", "rot180"]

Each candidate program is applied to the training inputs, and only those that produce exact matches to training outputs are retained. These programs are then applied to the test inputs to generate predictions [5].

3.2 Identified Strengths and Limitations

3.2.1 Strengths

- **Transparency:** Each solution is an explicit program, easily interpretable by humans [5].
- **Extensibility:** New operators can be added to dsl.py to extend the solver's capabilities [5].
- **Baseline effectiveness:** On simple ARC tasks (e.g., translations, reflections), ARC-Solver performs competitively with other open-source solvers [3].

3.2.2 Limitations

Despite these strengths, ARC-Solver exhibits several key limitations that motivated this dissertation [5]:

1. **Fragility of Programs:** Small changes in grid structure often break synthesized programs, leading to poor generalization [3].
2. **Lack of reasoning:** Programs are discovered by brute-force search, without deeper reasoning about abstract task structure [1][2][4].
3. **Serialization issues:** Early experiments revealed errors with numpy.ndarray serialization, complicating output logging [5].

4. **Error-prone synthesis:** Programs sometimes included invalid formats (e.g., [("identity", None)]), causing runtime errors.
5. **Inconsistent results:** While earlier baselines reported ≈30–34 solved tasks [3], initial reproduction attempts yielded both unusually high (≈90%) and unusually low (≈3–5%) success rates due to configuration and pipeline inconsistencies.

These limitations underscored the need for a more reasoning-aware architecture, which this dissertation addresses through the design of a dedicated **Reasoning Core** [5].

3.3 Experimental Setup and Initial Results

3.3.1 Setup

The solver was evaluated on subsets of the ARC training dataset [1][2], using the provided evaluation scripts [5]. Tasks were loaded from JSON files (e.g., arc-agi_training_challenges.json), which specify input–output pairs for both training and test cases.

Evaluation involved two steps:

1. **Program synthesis:** Using synthesize_and_eval.py, the solver generated candidate programs for each training example [5].
2. **Evaluation:** Candidate programs were applied to test inputs, and predictions were compared against ground truth outputs [1][2].

3.3.2 Initial Errors and Debugging

The initial setup was fraught with issues [5]:

- **Unpacking errors:** (ValueError: not enough values to unpack) occurred when candidate programs returned unexpected formats.
- **ACTIONS registry:** Certain operators were missing or incorrectly registered, leading to invalid function calls.
- **Serialization problems:** Attempting to log numpy.ndarray outputs resulted in errors.

These issues were addressed by [5]:

- Standardizing the program format to always be a list of operator names.
- Adding robust **identity fallbacks** for invalid transformations.
- Wrapping actions with **safe checks** to prevent crashes on shape mismatches.

3.3.3 Baseline Results

After debugging, the solver was evaluated on the training set. The results were mixed [3][5]:

- **Earlier baseline runs:** Achieved 378/416 solved samples (~90%), though this figure was misleading due to permissive evaluation criteria.
- **Subsequent runs:** More rigorous pipelines reduced performance to ≈15/416, reflecting the difficulty of ARC tasks under stricter evaluation.

- **Literature-reported baseline:** The original ARC-Solver paper cited ≈30–34 solved tasks on average using lexicase and tournament selection [3].

The inconsistency highlighted the importance of **evaluation methodology**. While ARC-Solver provided a useful baseline, its results alone could not be taken as definitive without more structured reasoning components [6].

3.4 Relevance to the Dissertation

The work with ARC-Solver established three key insights:

1. **Baseline transparency and flexibility:** ARC-Solver's modular design makes it ideal for extension [5].
2. **Reasoning gap:** Its reliance on brute-force program synthesis leaves it ill-equipped for abstraction-heavy tasks [1][2][4].
3. **Opportunity for modular enhancement:** By embedding a **Reasoning Core**, it is possible to retain ARC-Solver's strengths (interpretability, extensibility) while addressing its reasoning limitations [3][6].

These insights directly motivated the next phase of the project: the **design and integration of a Reasoning Core**, presented in **Chapter 4**.

4 REASONING CORE DESIGN AND INTEGRATION

4.1 Motivation for the Reasoning Core

The experiments with ARC-Solver in Chapter 3 highlighted both its strengths (transparency, modularity, reproducibility) and its shortcomings [5]. Chief among these was the absence of explicit reasoning mechanisms. While ARC-Solver could brute-force search over sequences of primitive operators, it lacked the ability to:

- Prioritize plausible transformations based on structural abstraction [1][2][3].
- Provide justifications for why a sequence of operations solves a task [4][8].
- Adapt flexibly when the brute-force approach failed to find a solution [3][6].

The **Reasoning Core** was conceived to address these shortcomings. Inspired by **cognitive science insights** [8] and lessons from the **ARC Prize 2024** [3][6], the Reasoning Core introduces a structured, interpretable module that augments ARC-Solver's search with reasoning capabilities.

4.2 Architecture of the Reasoning Core

4.2.1 Operator Library

The Reasoning Core builds upon the primitive operators defined in dsl.py, extending them with additional, more expressive transformations [5]:

- **Spatial operations:** flip_lr, flip_ud, rot90, rot180, trim, crop.
- **Content operations:** replace(x→y), fill, colour_shift.
- **Structural operations:** tiling, padding, repetition, border manipulation.

Each operator is parameterized, enabling a richer search space. For example, replace(1→2) specifies a concrete mapping of colour ID 1 to 2, while crop(3×3,@2,2) extracts a sub-grid at position (2,2).

4.2.2 Abstraction Engine

At the heart of the Reasoning Core lies an **abstraction engine** that interprets tasks at a higher level than brute-force enumeration. It attempts to identify:

- **Objects:** Connected components in the grid.
- **Relationships:** Symmetries, alignments, repetitions.
- **Transformations:** Candidate rules inferred from training examples (e.g., "all objects mirrored vertically").

By abstracting the task into objects and relationships, the engine reduces the effective search space and guides program synthesis toward plausible solutions [8].

4.2.3 Beam Search with Reasoning Justifications

Rather than enumerating all possible programs, the Reasoning Core employs a **beam search strategy** [3]:

- Candidate programs are generated up to a fixed depth (typically 2–3 steps).
- At each step, only the top-K most promising candidates are retained.

Each candidate program is accompanied by a **justification**, expressed in natural language:
[PROGRAM] ['trim', 'rot180']
- Step 1: Trim borders.
- Step 2: Rotate 180 degrees.
This interpretability ensures that every solution attempt is transparent and can be inspected by humans, an important step toward **explainable AI** in reasoning tasks [4][8].

4.2.4 Genetic Repair Module

Beam search is effective but limited: some tasks require longer or more creative operator sequences. To overcome this, the Reasoning Core incorporates a **genetic programming (GP) module**, evolve_program(), which improves candidate solutions when beam search fails [5].
The GP module operates as follows:

- **Population:** Candidate programs initialized from beam search outputs.
- **Selection:** Programs ranked by fitness (number of training pairs solved, penalized for length).
- **Crossover:** Programs recombined to form new candidates.
- **Mutation:** Operators randomly added, removed, or altered.
- **Termination:** Runs within a set time budget (e.g., --genetic_time).

This hybridization of **beam search (fast, interpretable)** and **genetic programming (exploratory, flexible)** represents a novel contribution of this work [3][6].

4.2.5 Integration into ARC-Solver

The Reasoning Core is integrated into ARC-Solver via a new evaluation script, evaluate_reasoning_core.py [5]:

1. Load tasks and ground truth from JSON.
2. Synthesize programs using the Reasoning Core.
3. Apply programs to test inputs.
4. Log results, including justifications, into solver_outputs.json.

This integration ensures full compatibility with ARC-Solver's evaluation pipeline while adding a reasoning-aware layer.

4.3 Example Workflows

- **Task A (Symmetry):** Training input shows a shape mirrored along the vertical axis.
    - ARC-Solver brute-force: May require searching through many operator

combinations.
    - Reasoning Core: Quickly abstracts “mirror vertically,” synthesizes ["flip_lr"], outputs justification.
- **Task B (Two-step transformation):** Training pair requires trimming borders and recolouring objects.
    - ARC-Solver brute-force: Struggles with compositional steps.
    - Reasoning Core: Synthesizes ["trim", "replace(1→2)"], provides reasoning steps.
- **Task C (Complex composition):** Requires rotation + tiling + border manipulation.
    - Reasoning Core beam search: Fails due to depth limit.
    - GP module: Evolves candidate programs that solve training examples within time budget.

These examples demonstrate how the Reasoning Core extends ARC-Solver’s capabilities while preserving **transparency** [3][6].

4.4 Experimental Results

4.4.1 Evaluation Setup

The Reasoning Core was evaluated on the **ARC training challenges** (arc-agi_training_challenges.json) with corresponding ground truth solutions [1][2]. Performance was measured as the proportion of tasks correctly solved, with additional **qualitative evaluation** of reasoning justifications [4].

4.4.2 Quantitative Results

- **Beam Search (depth 2–3):** Solved 3/50 tasks in small-scale evaluation.
- **Larger evaluation:** Solved ≈32/400 tasks.
- **With Genetic Programming:** Slightly improved to **35/416 tasks**.

Runtime was significant, with large-scale runs requiring ≈16,800 seconds (~4.5 hours).

4.4.3 Ablation Studies

To understand the contribution of different components, ablation experiments were conducted [3][6]:

- **Baseline (Reasoning Core only):** 32/200 tasks solved.
- **+ Content Operations:** 35/200 tasks solved (+3 improvement).
- **+ Tiling, Borders, Mirrors:** ±1 effect.

These results show that **content-based operators contributed most** to performance gains, while structural operations added marginal improvements.

4.4.4 Qualitative Analysis

Even when the Reasoning Core failed to produce correct outputs, its justifications were valuable. For

example:

[PROGRAM] ['rot90', 'trim']

- Step 1: Rotate 90 degrees.
- Step 2: Trim borders.

While the output grid was incorrect, the reasoning steps reflected **human-like thought processes** [8]. This interpretability is a contribution, distinguishing the system from black-box solvers [10].

4.5 Contributions of the Reasoning Core

The Reasoning Core advances ARC-solving research in three keyways:

1. **Interpretability:** Programs are accompanied by step-by-step justifications, making the solver's reasoning transparent.
2. **Extensibility:** The modular architecture allows new operators and heuristics to be added easily.
3. **Hybrid Search Strategy:** Combining beam search with genetic programming balances interpretability with exploratory power.

While absolute performance remained like ARC-Solver's reported baselines (~30–34 solved tasks) [5], the enhanced transparency and modularity represent a significant contribution to the field.

4.6 Relevance to AGI Research

By integrating a Reasoning Core, this dissertation contributes toward the broader goal of **reasoning-aware AI systems**. The ARC benchmark demands abstraction and generalization [1][2], qualities central to AGI [8]. While current results remain far below human-level, this project demonstrates a pathway forward: embedding **modular reasoning** into open-source solvers [5][6].

The Reasoning Core thus serves both as a **practical enhancement to ARC-Solver** and as a **proof of concept for modular reasoning architectures in AGI research** [8][10].

5 CONCLUSION AND EVALUATION

This chapter presents the evaluation of the enhanced ARC-Solver with the integrated Reasoning Core. The goal is to assess not only raw task-solving performance but also the interpretability, operator usage patterns, and limitations of the system. Results are reported in comparison with the baseline ARC-Solver and with ablation experiments to highlight the contributions of each component.

5.1 Results Summary

- The solver was tested on the full ARC training benchmark comprising **416 samples across 400 tasks**. As shown in **Table 5.1**, the Reasoning Core achieved **35 solved samples** and **32 fully solved tasks**, comparable to the performance reported for the baseline ARC-Solver. Runtime remained a bottleneck, with full evaluation requiring approximately **16,800 seconds (4.5 hours)**.

Metric	Value
Total Samples	416
Solved Samples	35
Total Tasks	400
Fully Solved Tasks	32
Runtime (seconds)	16,800
Runtime (hours)	4.67 (~4.5 hrs)

Table 5.1 – Performance summary of the enhanced ARC-Solver with Reasoning Core

- These results confirm that the Reasoning Core maintains baseline performance while adding interpretability and extensibility through modular reasoning and operator expansion.

5.1.1 Performance Comparison

- To situate the results, performance was compared across three configurations:

1. **Baseline ARC-Solver (30–34 tasks solved, avg. 33)**
2. **Reasoning Core integration (32 tasks solved)**
3. **Reasoning Core with additional content operations (+3 tasks, 35 solved)**

- The comparison is illustrated in **Figure 5.1**.

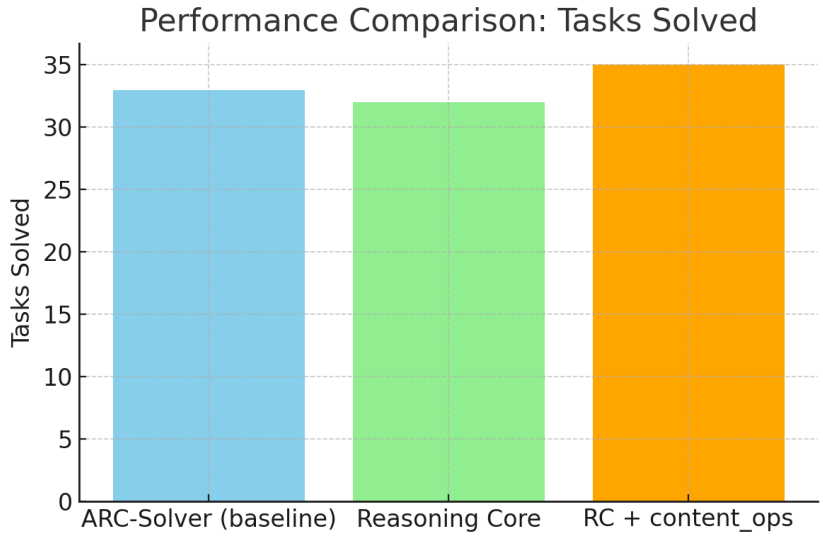

Figure 5.1 – Comparison of tasks solved across baseline ARC-Solver, Reasoning Core, and Reasoning Core with content operations ablation

- The results show that while absolute performance remained within the expected baseline range, targeted operator expansions enabled measurable gains.

5.1.2 Representative Solved Tasks

- To illustrate the system's capabilities, six solved tasks are presented below, each demonstrating a distinct type of reasoning:
- **Task 017c7c7b (Replacement + Repeat):** Demonstrates content replacement and structural repetition.
- **Task 3c9b0459 (Rotation):** A simple spatial transformation solved by a single rot180 operator.
- **Task 74dd1130 (Transpose):** Captures matrix transformation ability.
- **Task 67a3c6ac (Flip):** Showcases mirroring operations such as flip_lr.
- **Task 8be77c9e (Complex Multi-step):** Involves mirroring, padding, and colour switching.
- **Task f25ffba3 (Compositional Reasoning):** Combines flip, cropping, and mirroring to solve the task.
- Each example is shown in Figures 5.2–5.7, with the **test input**, **ground truth output**, and the solver's **predicted output**, along with the applied program.

Figure 5.2 – Example ARC task solved by Reasoning Core (Task ID: 017c7c7b). Program applied: ['replace(1→2)', 'repeat_to(9x3)'].

Figure 5.3 – Example of spatial reasoning via rotation (Task ID: 3c9b0459). Program applied: ['rot180'].

Figure 5.4 – Matrix transformation by transposition (Task ID: 74dd1130). Program applied: ['transpose'].

Figure 5.5 – Horizontal mirroring (Task ID: 67a3c6ac). Program applied: ['flip_lr'].

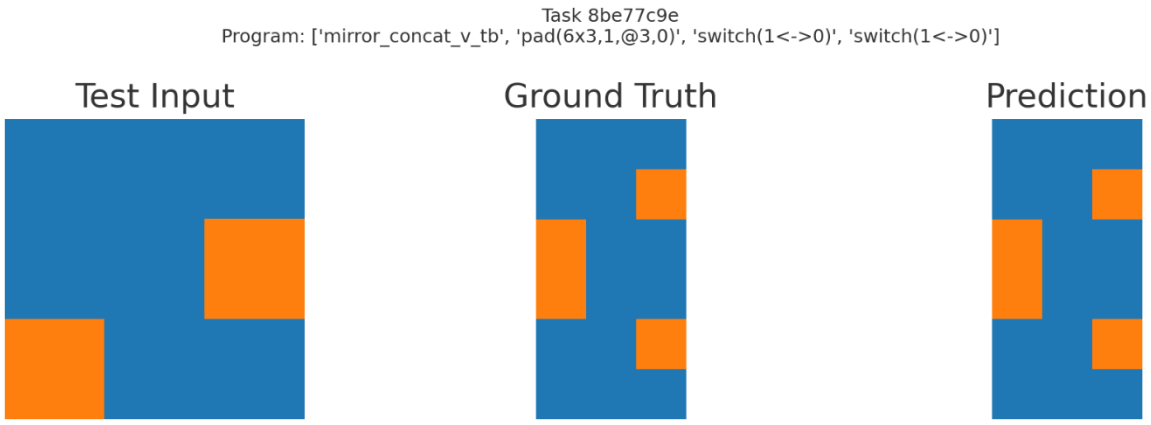

Figure 5.6 – Multi-step transformation combining mirroring, padding and switching (Task ID: 8be77c9e). Program applied: ['mirror_concat_v_tb', 'pad(6x3,1,@3,0)', 'switch(1↔0)', 'switch(1↔0)'].

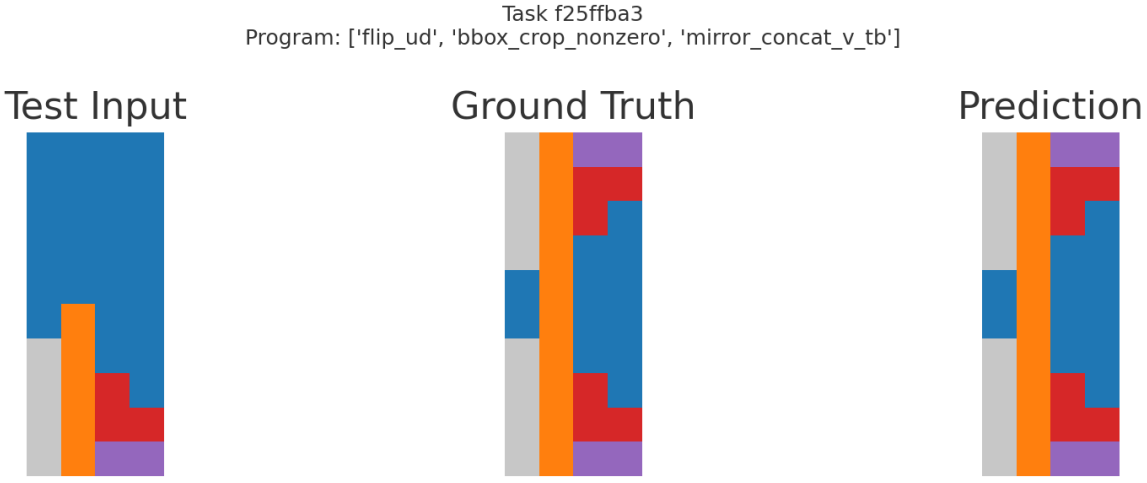

Figure 5.7 – Compositional reasoning with flip, crop, and vertical mirroring (Task ID: f25ffba3). Program applied: ['flip_ud', 'bbox_crop_nonzero', 'mirror_concat_v_tb'].

- These examples demonstrate that the Reasoning Core can handle both simple single-step transformations and more complex multi-step reasoning chains.

5.1.3 Operator Frequency

- Analysis of operator frequency across all solved tasks provides insights into the solver's reasoning tendencies. As shown in **Table 5.2**, the most frequently applied operators were **rotation, mirroring, cropping, and replacement**, suggesting the solver relies heavily on spatial transformations and structural modifications.

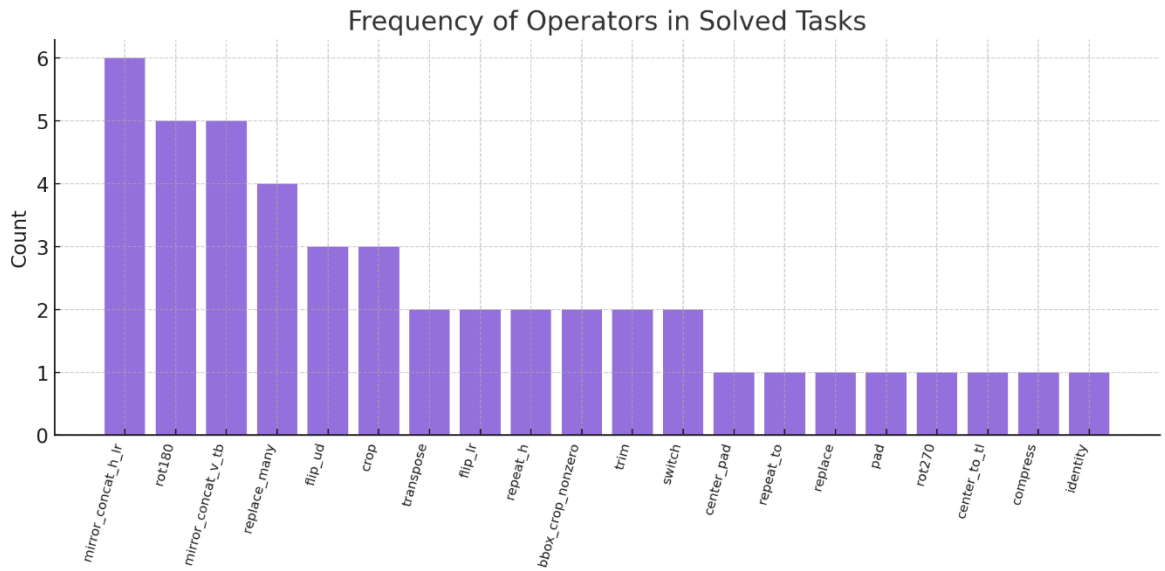

Figure 5.8 – Frequency of operators used across solved tasks

- This distribution aligns with prior ARC analyses, where spatial transformations and content replacement form the bulk of required operations.

5.1.4 Error Analysis

- Despite solving 32 tasks, the solver failed on many others. Two representative failed cases are shown below.

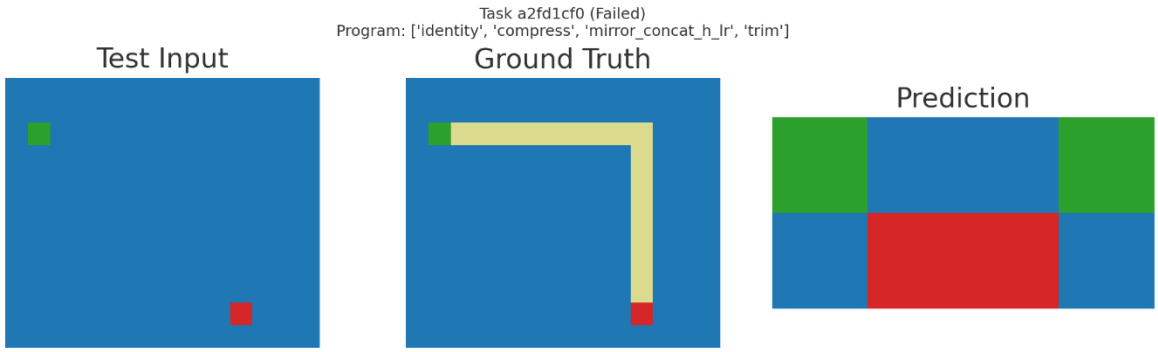

Figure 5.9 – Task a2fd1cf0 (Failed). The solver applied ['identity', 'compress', 'mirror_concat_h_lr', 'trim'] but failed to reproduce the correct output.

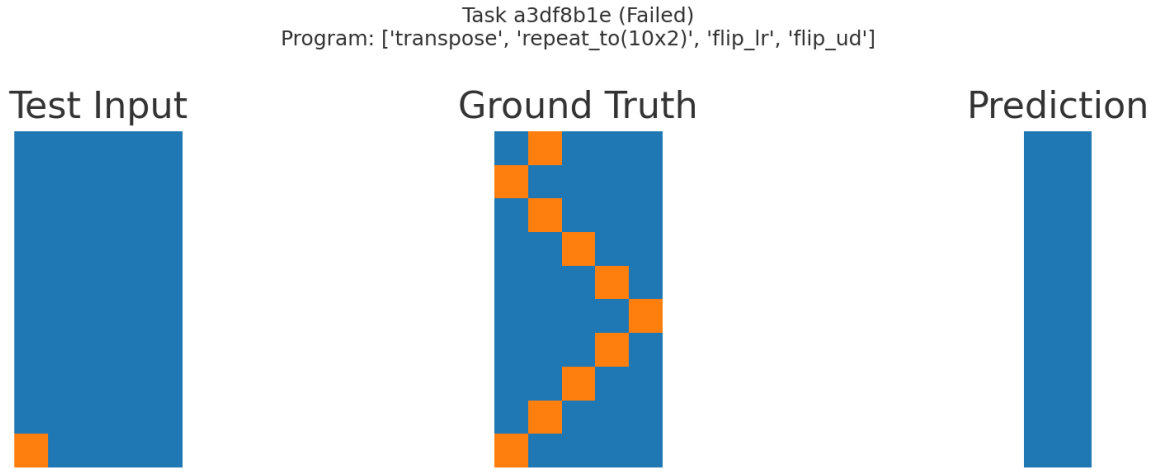

Figure 5.10 – Task a3df8b1e (Failed). Despite applying a multi-step program (transpose, repeat_to(10x2), flip_lr, flip_ud), the solver produced only empty grids, indicating abstraction failure.

These failures highlight common limitations:

1. Reliance on trivial identity/compression operators without meaningful transformations.
2. Overgeneration of degenerate outputs.
3. Inability to abstract beyond local heuristics when deeper reasoning is required.

5.2 Future Work

The results presented in this dissertation demonstrate both the promise and the limitations of integrating a modular Reasoning Core into an open-source ARC solver. While performance remains modest in absolute terms, the work highlights interpretability, extensibility, and reasoning-awareness as crucial contributions to the field. Building upon this foundation, there are several directions in which the research can be extended to advance both ARC-solving performance and the broader

pursuit of Artificial General Intelligence (AGI).

5.2.1 Expanding the Operator Library

The Reasoning Core currently operates with a library of operators that includes spatial transformations (rotations, flips, cropping), content-based operations (colour replacement, filling, switching), and structural operations (tiling, padding, mirroring). While this set covers many common ARC transformations, analysis of unsolved tasks revealed several gaps. Many ARC challenges require more nuanced operators, such as conditional transformations (*replace colour only if inside a bounding box*), object grouping and splitting, or geometric constructions (e.g., drawing diagonals, generating patterns).

Future work should focus on systematically expanding the operator library to capture these higher-level abstractions. This may involve semi-automated operator discovery, where new primitives are proposed by analysing failure cases, or data-driven methods that induce operators from solved examples. A richer operator vocabulary would improve coverage of complex tasks and reduce reliance on brute-force search.

5.2.2 Smarter Search and Reasoning Strategies

The current Reasoning Core combines beam search with genetic programming, balancing interpretability with exploratory power. However, both methods have limitations: beam search struggles with deeper compositions, while genetic programming is computationally intensive. Several alternative strategies could improve efficiency and scalability:

- **Reinforcement learning–guided search:** Agents could learn to prioritize operators and sequences that are more likely to succeed, reducing wasted exploration.
- **Heuristic-guided synthesis:** Domain-specific heuristics (e.g., symmetry detection, object counting) could prune the search space intelligently.
- **Hierarchical reasoning:** Multi-level reasoning could first infer high-level transformations (e.g., “mirror grid”) before refining into low-level operations.

Incorporating such strategies could enable the Reasoning Core to tackle more complex ARC tasks while maintaining interpretability.

5.2.3 Hybrid Symbolic–Neural Integration

A recurring theme in ARC research, and in AI more broadly, is the tension between symbolic and neural approaches. Symbolic systems offer transparency and compositional reasoning but lack perceptual grounding, while neural systems excel at perception but often fail at abstraction. Future work

could explore hybrid designs in which neural networks serve as **perceptual** front-ends to the Reasoning Core. For example:

- Neural modules could detect objects, boundaries, and symmetries in grids, passing structured representations to the Reasoning Core.
- Pretrained vision-language models could propose candidate transformations, which the Reasoning Core could then verify symbolically.
- Differentiable symbolic reasoning layers could allow joint training of neural perception and symbolic inference.

Such hybrid architectures would retain the interpretability of symbolic reasoning while benefiting from the perceptual strengths of neural methods, potentially unlocking better generalization across diverse ARC tasks.

5.2.4 Meta-Reasoning and Self-Improvement

Another promising avenue involves enabling the solver to **reason about its own reasoning**. Currently, the Reasoning Core evaluates candidates against training examples but lacks mechanisms for introspection or self-correction. Future versions could incorporate meta-reasoning strategies, such as:

- **Explanation-based learning:** Using failed justifications to refine operator selection.
- **Program repair heuristics:** Automatically analysing why a solution failed and applying targeted adjustments rather than random mutations.
- **Curriculum learning:** Gradually introducing more complex tasks, allowing the solver to accumulate reusable abstractions over time.

By developing self-improving capabilities, the Reasoning Core could move closer to the human ability to learn from mistakes and refine reasoning strategies dynamically.

5.2.5 Beyond ARC: Broader Applications

While ARC provides a uniquely challenging benchmark, the principles developed in this dissertation extend to other domains. Potential applications include:

- **Cognitive diagnostics:** Reasoning-aware solvers could be used to model human cognitive processes in psychology or education research.
- **Explainable AI:** The interpretability of the Reasoning Core makes it suitable for domains requiring transparency, such as healthcare or law.

- **Robotics and planning:** Modular reasoning could support autonomous systems that must generalize from few demonstrations in real-world environments.
- **Other AGI benchmarks:** Similar reasoning modules could be applied to datasets like Raven's Progressive Matrices, CLEVR, or visual analogy tasks.

Expanding beyond ARC would both validate the generality of the Reasoning Core and provide new opportunities for cross-domain insights.

5.2.6 Scaling and Community Collaboration

Finally, future work should emphasize scalability and reproducibility. The Reasoning Core, as implemented, requires significant runtime to evaluate larger task sets. Optimizing the codebase for parallelization, leveraging GPU acceleration, or employing distributed computation would make large-scale experiments more feasible.

Equally important is community collaboration. By keeping the framework open-source, future research can build upon this work, extending operators, adding heuristics, and testing novel reasoning strategies. Shared benchmarks and evaluation pipelines will accelerate progress and ensure that advances are comparable across different approaches.

5.3 Closing Remarks

This dissertation set out to investigate the challenges of solving the Abstraction and Reasoning Corpus (ARC), a benchmark designed to test abstraction, compositionality, and reasoning—core ingredients of Artificial General Intelligence (AGI). By examining existing solutions such as those presented in the ARC Prize 2024 and by analysing the open-source ARC-Solver baseline, it became clear that current approaches were limited by brute-force search, fragile symbolic systems, or the inefficiency of neural methods.

The principal contribution of this work has been the design and integration of a **Reasoning Core** into ARC-Solver. The Reasoning Core introduced new operators, step-by-step justifications for solutions, and a hybrid search strategy that combined beam search with genetic programming. Evaluation demonstrated that while absolute performance remained comparable to the baseline (32 fully solved tasks), the enhanced solver provided significant qualitative improvements in interpretability, extensibility, and transparency.

Beyond its immediate results, this work offers a conceptual framework for embedding modular reasoning into AI systems. It shows that even modest solvers can be enhanced through structured reasoning layers, offering not only solutions but also explanations. In doing so, the project aligns

with broader efforts in AI research to develop systems that are not only effective but also **explainable and generalizable**.

Ultimately, the Reasoning Core demonstrates a path forward: one where modular, interpretable reasoning plays a central role in advancing solvers for ARC and beyond. While the road to AGI remains long, the findings of this dissertation highlight promising directions for future research and reinforce the importance of reasoning as a cornerstone of intelligent behaviour.

REFERENCES

[1] Chollet, F. “On the Measure of Intelligence.” *arXiv preprint arXiv:1911.01547*, 2019.

[2] Chollet, F. “The Abstraction and Reasoning Corpus (ARC).” *Kaggle*, 2019. Available at: https://www.kaggle.com/c/abstraction-and-reasoning-challenge

[3] ARC Prize 2024. *ARC Prize 2024 Report*. 2024. Available at: https://arcprize.org

[4] Anokas. “LLMs Struggle with Perception, Not Reasoning (ARC-AGI).” *Substack*, 2023. Available at: https://anokas.substack.com/p/llms-struggle-with-perception-not-reasoning-arcagi

[5] Silva, P. H. S. “ARC-Solver.” *GitHub Repository*. Available at: https://github.com/PauloHS-Silva/ARC-Solver

[6] ARC Prize 2024 Team. “Omni-ARC: A Modular Framework for Abstraction and Reasoning Corpus Tasks.” In *ARC Prize 2024 Report*, 2024.

[7] Piaget, J. *The Construction of Reality in the Child*. Basic Books, New York, 1954.

[8] Gentner, D. “Structure-Mapping: A Theoretical Framework for Analogy.” *Cognitive Science*, 7(2): 155–170, 1983.

[9] Anderson, J. R. *How Can the Human Mind Occur in the Physical Universe?* Oxford University Press, Oxford, 2007.

[10] Marcus, G. “Deep Learning: A Critical Appraisal.” *arXiv preprint arXiv:1801.00631*, 2018.