Thesis for Bachelor's Degree

The Abstraction and Reasoning Corpus Learning Environment for Reinforcement Learning

Hosung Lee

School of Electrical Engineering and Computer Science

Gwangju Institute of Science and Technology

2025

학사학위논문

추상화와 추론을 위한 강화학습 환경

이호성

전기전자컴퓨터공학부

광주과학기술원

2025

The Abstraction and Reasoning Corpus Learning Environment for Reinforcement Learning

Advisor: Sundong Kim

by

Hosung Lee

School of Electrical Engineering and Computer Science

Gwangju Institute of Science and Technology

A thesis submitted to the faculty of the Gwangju Institute of Science and Technology in partial fulfillment of the requirements for the degree of Bachelor of Science in the Electrical Engineering and Computer Science Concentration

Gwangju, Republic of Korea

Dec 6, 2024

Approved by

Professor Sundong Kim

Committee Chair

The Abstraction and Reasoning Corpus Learning Environment for Reinforcement Learning

Hosung Lee

Accepted in partial fulfillment of the requirements for

the degree of Bachelor of Science

Dec 6, 2024

Committee Chair _____

Prof. Sundong Kim

Committee Member _____

Prof. Kangil Kim

Dedicated to my family.

BS/EC 20215169 Hosung Lee. The Abstraction and Reasoning Corpus Learning Environment for Reinforcement Learning. School of Electrical Engineering and Computer Science. 2025. 30p. Advisor: Prof. Sundong Kim.

Abstract

This paper introduces ARCLE, an environment designed to facilitate reinforcement learning research on the Abstraction and Reasoning Corpus (ARC). Addressing this inductive reasoning benchmark with reinforcement learning presents these challenges: a vast action space, a hard-to-reach goal, and various tasks. ARCLE offers structured state space and action space for ARC. ARCLE's action space consists of pixel-based and object-oriented operations that transform state space, which keeps states of editions of the input grid of each ARC task's demonstration pair. We demonstrate that an agent with proximal policy optimization (PPO) is capable of learning individual tasks through ARCLE. Adopting non-factorial policies and auxiliary losses led to performance enhancements, effectively mitigating issues on huge action space and goal attainment. Based on these insights, we propose several research directions and motivations for using ARCLE, including MAML, GFlowNets, and World Models.

BS/EC 20215169 이호성. 추상화와 추론을 위한 강화학습 환경. 전기전자컴퓨터공학부. 2025. 30p. 지도교수: 김선동.

국 문 요 약

본 논문에서는 Abstraction and Reasoning Corpus (ARC) 해결에 강화학습을 활용하기 위한 환경인 ARCLE을 고안한다. 인공지능의 귀납 추론 능력 벤치마크인 ARC를 강화학습의 방법론으로 접근하는 것은, 넓은 행동 공간에서의 문제해결, 도달하기 어려운 목표의 달성과 다중 작업 학습과 같은 강화학습의 주요 문제들을 제시한다. ARCLE에는 ARC의 예시 입출력 쌍의 입력 그리드를 편집하는 것으로 표현되는 상태공간과, 이 상태를 조작하는 픽셀 기반 및 오브젝트 기반 행동 공간이 정의되어 있다. 본 논문에서 구현하는 proximal policy optimization (PPO) 기반 강화학습 에이전트는 ARCLE 환경에서 ARC에 포함된 문제를 개별적으로 학습할 수 있음을 확인하였다. 또한, 구현한 강화학습 에이전트에 non-factorial 정책과 보조 손실함수를 적용함으로써 성능을 향상시켰으며, 이에따라 넓은 행동 공간과 목표 달성 문제를 효과적으로 해결할 수 있었다. ARCLE과 구현한 강화학습 에이전트에서 얻은 통찰에 기반하여, 본 논문에서는 ARCLE을 활용한 앞으로의 연구 방향과 질문을 제기한다.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis introduces ARCLE (ARC Learning Environment) as a reinforcement learning (RL) environment designed for the Abstraction and Reasoning Challenge (ARC) benchmark [1]. This benchmark assesses agents' ability to infer rules from given grid pairs and predict the outcome for a test grid, as illustrated in Figure 1.1. ARC is designed to test abstraction and reasoning skills, making it a touch benchmark within the domain. Despite various attempts to conquer ARC's complexities through program synthesis and reasoning using large language models, RL-based approaches are surprisingly rare (Section 2.1). This scarcity is presumed by the lack of a dedicated RL environment tailored for ARC. To fill this gap, ARCLE based on Gymnasium [2] is created to tackle the benchmark.

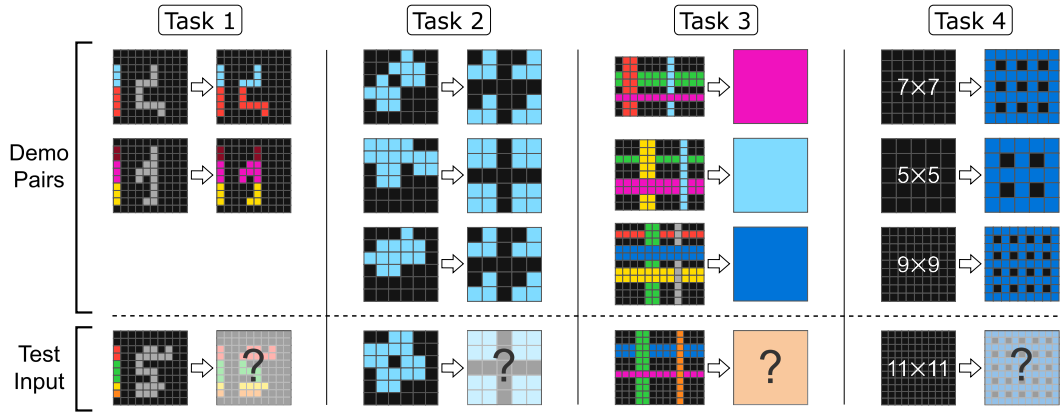

Figure 1.1: Four different ARC tasks are presented, each requiring analysis through its provided demonstration pairs. The identified rule from this analysis must then be applied to a test input grid to produce the answer (test output) grid, which is currently blurred for demonstration purposes. Task 1 modifies all gray grids within a row to match the color found in the far-left column of that same row. Task 2 relocates four identical cyan objects appropriately, each no larger than 2×2 in size. Task 3 determines the color of the topmost line in a stack of overlapping horizontal and vertical lines, and outputs a single pixel of this color.

From RL's standpoint, ARC is considered highly challenging. The typical difficulties include (1) a vast action space, (2) a hard-to-reach goal, and (3) a variety of tasks. While other RL benchmarks (e.g., robotics, financial trading, recommender systems, video games) might feature one of these challenges, ARC encompasses all, showing its difficulty. ARCLE is designed to help researchers navigate these challenges, offering a unique testbed for RL strategies.

Vast action space ARC stands out with its vast action space by allowing a variety of actions such as coloring, moving, rotating, or flipping pixels. This feature creates a large set of possibilities, complicating the development of optimal strategies for RL agents. Such a vast action space demands innovative approaches to navigate effectively.

Hard-to-reach goal ARC tasks are uniquely challenging because success is measured by the ability to replicate complex grid patterns accurately, not by reaching a physical location or endpoint. This requires a deep understanding of the task rules and an ability to apply them precisely. Designing effective reward systems is particularly challenging in this context, as progress is not easily quantified.

Variety of tasks ARC's wide array of tasks necessitates broad generalization, a stark contrast to benchmarks like Atari, which focus on mastering single games.[1] This diversity calls for adaptive and varied strategies, highlighting ARC's demand for advanced RL methods.

ARCLE is an environment that helps overcome the challenges of ARC and paves new pathways for AI research, seamlessly linking abstract reasoning in ARC with the adaptability in RL. Our initial experiments highlight the capability of RL to address specific tasks within ARC, indicating the potential necessity for advanced techniques such as meta-RL, generative models, or model-based RL algorithms. Thus, ARCLE stands out as a platform for testing RL solutions, prompting an in-depth exploration of the challenges ARC presents.

[1] Atari benchmarks hosts 57 games, each with its goal. Solutions such as Rainbow DQN [3], R2D2 [4], MuZero [5], and Agent57 [6] focus on mastering single games.

Chapter 2
Related Works

2.1 Solving ARC

Since the unveiling of the ARC [1], approaches ranging from the development of similar benchmarks [7, 8, 9] to domain-specific languages and program synthesis [10, 11, 12, 13, 14, 15] have been explored to extend its applicability and enhance learning strategies. These efforts have deepened our understanding of ARC's challenges, highlighting the complexity of devising comprehensive solutions. The recent shift towards leveraging Large Language Models (LLMs), incorporating strategies from natural language processing to detailed task context integration [16, 17, 18, 19, 20], underscores LLMs' potential in addressing ARC's intricacies.

However, the performance of research on the ARC utilizing program synthesis and LLMs has not fully met expectations, often due to its logical flaw [20]. This has prompted a pivot towards reinforcement learning as a novel approach, albeit its application to ARC has been limited. Notable attempts include using RL strategies in program synthesis [21] and exploring imitation learning [22]. The introduction of ARCLE opens up new possibilities for advancing research on the ARC using RL.

2.2 Related RL Environments

Among the myriad RL environments, those featuring a **vast action** space similar to ARCLE's are prominently observed in game-based settings, such as PySC2 [23], where the diversity of actions, determined by mouse click locations, mirrors the flexible action format in ARC. Similarly, environments designed for recommendation systems (e.g., RecSim, RecoGym) and complex multi-step planning tasks (e.g., Super Mario Bros [24], NLE [25]) may not exhibit wide action spaces at each state but encapsulate the challenge of **hard-to-reach goal** through the necessity of sequential decision-making to

achieve success. In parallel, the breadth of tasks within ARCLE resonates with the diverse objectives found in robotics environments like Meta-World [26], RLBench [27], and CALVIN [28], underscoring the complexity and **variety of tasks** that ARCLE introduces to RL research.

Chapter 3

ARCLE: ARC Learning Environment

3.1 Framework of ARCLE

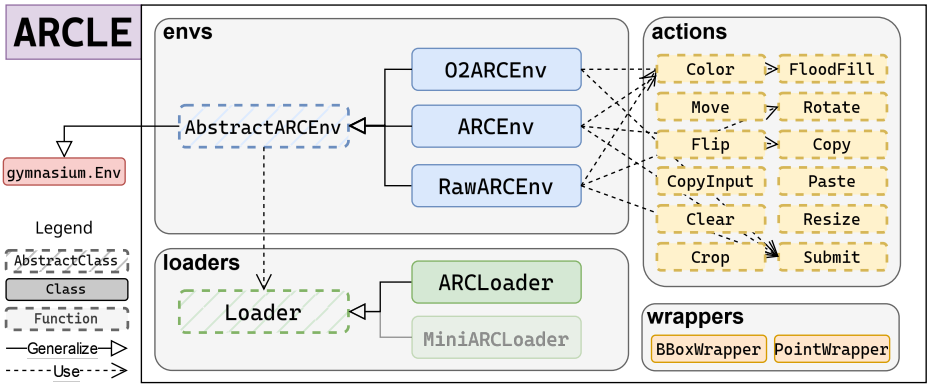

Figure 3.1: Framework of ARCLE. The package consists of components: envs, actions, loaders, and wrappers.

ARCLE is a reinforcement learning (RL) environment package, implemented in Gymnasium [2], designed for RL approaches on Abstraction and Reasoning Corpus (ARC). RL agents on the ARCLE environments learn to solve tasks by selecting actions to edit the grid (to be submitted) to the environment state. As Figure 3.1 illustrates, ARCLE comprises three main components: **envs**, **loaders**, **actions**, and auxiliary **wrappers** which modify the environment's action or state space. The following explanation is based on the terms in Table 3.1.

The **envs** component consists of a base class of ARCLE environments, and its three derivatives. `AbstractARCEnv` inherits Gymnasium's `Env` class to provide reinforcement learning environment features and defines the ARC-specific structure of action and state space and user-definable methods. Its implementations, `O2ARCEnv`, `ARCEnv` and `RawARCEnv` provide embodied action and observation spaces. `O2ARCEnv` constructs the state and action space according to the O2ARC interface (See Appendix A), and

likewise, `ARCEnv` offers the testing web interface developed by François Chollet [29]. `RawARCEnv` restricts the action space to color modifications or grid size changes, providing a more constrained and monotonic learning environment.

Next, the **loaders** component provides functionalities to supply the ARC dataset to ARCLE environments. This component comprises the base `Loader` class defining interface requirements to ARCLE environments and their implementations. `ARCLoader` feeds the ARC dataset to any ARCLE environment and defines how the ARC dataset should be parsed from files and how the parsed dataset should be picked. Likewise, to load a similar dataset to the ARC, one can inherit the `Loader` class and specify how to parse and sample. ARCLE package provides the `MiniARCLoader` which loads Mini-ARC dataset [8] upon an ARCLE environment, as an example usage of `Loader` class.

Last, **actions** component includes a variety of functions capable of changing environment state, called *operation*. Each environment in ARCLE contains several operations to be used in an environment by agents on the environment. Since ARCLE currently implemented actions on the O2ARC interface, it contains more actions (e.g., `Move`, `Rotate`, `Flip`) than the original ARC testing interface [29].

This thesis focuses on explaining `O2ARCEnv` in the following sections, which encompasses most operations by ARCLE.

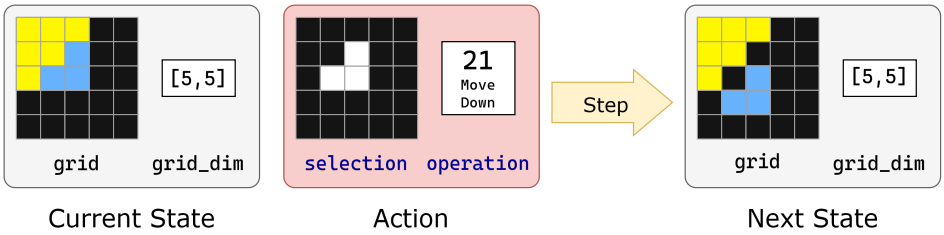

Figure 3.2: The state transition process of ARCLE.

3.2 Actions

Actions in ARCLE are defined to enable editing of the output grid for a given task, consisting of `operation` and `selection`. `operation` represents an integer that specifies

Table 3.1: Variables in action and state spaces and their definition.

Variable Space	Name	Definition
Action	`operation`	Integer index representing edit method of environment state (e.g., `grid`, `clip`)
	`selection`	Binary mask that specifies where a `operation` to be applied
State	`input`	Input grid of demonstration pair or test pair
	`input_dim`	Dimension (height, width) of `input`
	`grid`	Editable output grid of demonstration pair or test pair
	`grid_dim`	Dimension (height, width) of `grid`
	`clip`	Clipboard grid
	`clip_dim`	Dimension (height, width) of `clip`
State (`object_states`)	`selected`	Binary array which represents currently selecting pixels for object-oriented operations
	`active`	Boolean variable of whether last operation was an object-oriented operation
	`object`	Backed-up pixels of specified pixels of `grid` for object-oriented operations
	`object_sel`	Binary mask of exact shape which pixels of `object` that user has specified
	`object_dim`	Dimension (height, width) of bounding box of `object` and `object_sel`
	`object_pos`	Left-top position of bounding box of `obejct` on the `grid`
	`rotation_parity`	Binary value for consistency over serial rotations
	`background`	Pixels remaining in the `grid` excluding
Answer (Hidden to agents)	`answer`	Answer grid of test input grid
	`answer_dim`	Dimension (height, width) of answer grid

the method of editing (functions contained in the actions component in Figure 3.1), and `selection` is a binary mask that denotes the area of the grid affected by the edit.

By defining ARCLE's actions through `operation` and `selection` as illustrated by the action in the middle of Figure 3.2, ARCLE standardized various types of actions within the same structure. Notably, the actions in ARCLE can affect a single pixel, contiguous multiple pixels, or even non-contiguous pixels, accommodating these possibilities through employing the binary mask selection. Furthermore, by separating `operation` and `selection`, it accommodates the possibility of determining selection conditioned by the chosen operation autoregressively.

Currently, 35 operations are available in `O2ARCEnv` (Figure 3.3). When an agent specifies the operation index, the corresponding one is executed. Specifically, operation indices 0–9 represent to `Color` the selected pixels (by `selection`) with one color among the ten colors used in ARC, while 10–19 denote a `Flood Fill` based on Depth-First Search (DFS) in the selected pixels. Object-oriented operations not present in the original ARC testing interface [29], such as `Move`, `Rotate`, and `Flip`, are assigned to 20–23 (up, down, right, left), 24–25 (counterclockwise, clockwise), and 26–27 (horizontal, vertical), respectively. Additionally, 28–30 correspond to `Copy` and `Paste`, and 31–34 are assigned to operations that cause breaking changes in the states like duplicating the test

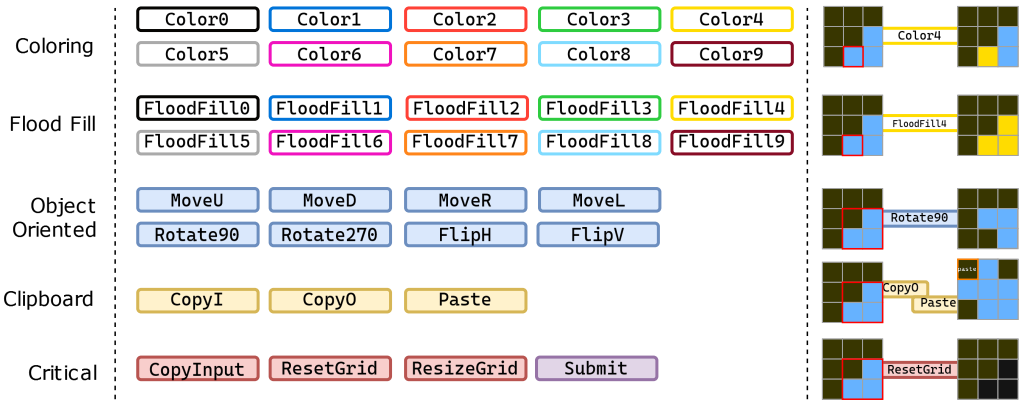

Figure 3.3: Every `operation` assigned in `O2ARCEnv` (version 0.2.5). The categories of operations (left), available operations (middle), and application examples of operations (right) are shown.

input (`CopyInput`), clearing the grid (`ResetGrid`), changing the grid size (`ResizeGrid`), and submitting (`Submit`). Subclassing the environments allows customizing available operations by adding or removing them in the same format.

3.3 States and Observations

All environments included in ARCLE are designed with the assumption to be Markov Decision Processes (MDP). Therefore, every parameter used in changing the environment's state is given to agents in the environment, so observations and states can be considered equivalent. The basic state space of an environment within ARCLE consists of the `input` and `grid`. Input represents the test input grid of an ARC task, so it is fixed unless a new task is assigned to an environment. Grid is initially set as the test input grid of a task, and an agent edits this by selecting actions.

Depending on which operations an environment adopts, the state of the environment can be different. For instance, if an environment includes `Copy` operation, the environment should include additional variables of the copied part: `clip`. Hence in `O2ARCEnv`, more variables are included in the state, to support `Copy` and object-oriented operations such as `Move`. These object-oriented actions from the O2ARC interface are supplemented with `selected`, `object`, `object_pos` and `background`. Descriptions of these variables are depicted in 3.1. While the agent performs object-oriented opera-

tions in a row, `object` and `background` works as two layers; `object` is overlayed on the `background` at `object_pos`. For the detailed mechanism described in next section.

3.4 Two-Layer Mechanism for Object-Oriented Actions

The actions implemented in ARCLE, such as `Move`, `Rotate`, `Flip`, are object-oriented actions that act on the pixels selected by the agent, i.e., the objects. Simplifying the implementation of these actions to merely move the selected pixels could lead to issues as illustrated in Figure 3.4.

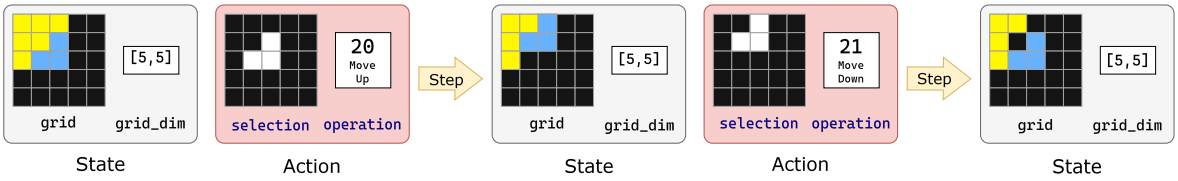

Figure 3.4: The edge case with consecutive `Move` actions without two-layer mechanism.

Figure 3.4 demonstrates the issue with a simplistic implementation of the `Move` action, depicted through the process of an agent performing two `Move` actions. In the first `Move` action, the gray pixels (object) included in the `selection` move upwards, overlapping with the yellow pixels directly above the object, and the pixels vacated by the object's movement are filled with the background color, black. When the gray object is moved back down in the second `Move` action, the pixels previously painted with the background color during the first move overlap with the object, and similarly, the vacated pixels are filled with the background color, black. As a result, the yellow pixels that overlapped with the object during the first `Move` action are changed to the background color, black. Thus, a simple implementation of object-oriented actions can lead to the disappearance of pixels that overlap as the object moves.

To prevent information loss during the movement of objects, ARCLE's object-oriented actions are implemented using a two-layer mechanism. This approach is inspired by the way people typically lift and move objects, dividing the state space's `grid` into an `object` layer, which includes currently selected pixels, and a `background` layer, which comprises the rest of the pixels. Actions are performed on the `object` layer,

which is then placed over the `background` layer to create the final `grid`. This two-layer mechanism for object-oriented actions utilizes variables stored in the `object_states` dictionary within the state space, and the detailed operation process is as Figure 3.5.

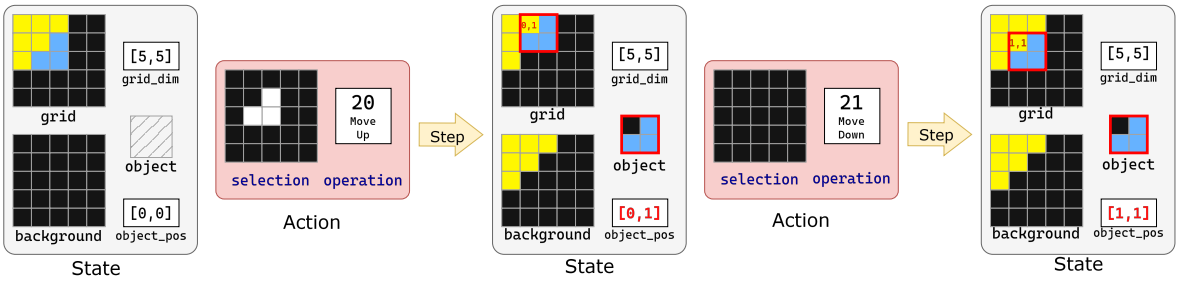

Figure 3.5: The edge case with serial two-layer mechanism `Move` actions.

At the start of an object-oriented action, the `active` variable in the dictionary is set to 1, and the pixels designated by the agent's `selection` in the action space are stored in `object`, while the rest are stored in `background`. The top-left coordinate of the bounding box surrounding the `object` is saved in `object_pos`. If the action performed is not object-oriented, the `active` variable is set to 0, and the `object` is reset. When `active` is 1, meaning an object-oriented action was performed previously, and the agent performs another object-oriented action, only `object`, `object_pos`, or `object_dim` change depending on the type of action, while `background` remains unchanged. Upon completing an object-oriented action, `background` and `object` merge using the information from `object_pos`, and the result is stored in `grid`. For example, in the two-layer mechanism, the `Move` action is implemented such that only the location of the `object` changes, altering only the value of `object_pos` during the action, while the rest of the variables in the dictionary, like `object` or `object_dim`, do not change.

3.5 Rewards

The built-in reward currently offered in ARCLE is the sparse reward. This reward grants 1 when the agent performs the `submit` action and the state space's grid exactly matches the task's answer grid, and 0 if even a single pixel differs. This sparse reward approach can hinder the learning of an agent whose total reward sum remains 0 as

there is a unique answer per task. To counteract this, an auxiliary reward was designed and utilized in the subsequent Section 4.1. This auxiliary reward adds a penalty term based on the ratio of the number of incorrect pixels to the total pixels, guiding the agent to learn in a direction that minimizes the number of pixels differing from the correct grid. Identifying a reward setting superior to this auxiliary reward setup, i.e., one that can be universally applied across all ARC tasks aware environment's action space (e.g., object-oriented operations), requires further research.

3.6 Source Code and Sample Usage

Since the environments in ARCLE implemented based on Gymnasium [2] and are fully written in Python3, users who have used Gymnasium or its predecessor, OpenAI Gym [30], can use it with familiarity. ARCLE is released on GitHub[1] under the terms of the Apache-2.0 License, as well as uploaded to the PyPI (Python Package Index), so the ARCLE can be easily installed by the `pip` command.[2] Without modifying the source code, one can still create custom ARCLE-based environments by subclassing provided environments in ARCLE or wrapping with the wrapper classes. Please note that ARCLE is currently being continuously updated, so users may need to check the version. In this paper, our descriptions and experiments are based on version 0.2.5.

Using Gymnasium API, an ARCLE environment can be created. Listing 1 is the most basic usage of the ARCLE environment loop. In the code, the `O2ARCEnv` is created and its `reset` method is called to initialize the environment to the input grid state of a random ARC task. If `adaptation` is `True` in the reset options, the environment samples and initializes states and answers as a demonstration pair, otherwise, it initializes as a test input pair. Next, within a loop, the `sample` function from the Gymnasium API is executed to select a random action, and the `step` function applies this action to the current state. Finally, if the state reaches the correct solution, the `reset` function is executed again to start the loop over with a new ARC task.

An example of using a bounding box form for `selection` (in action space) instead

[1] `https://github.com/ConfeitoHS/arcle`
[2] `$ pip install arcle==0.2.5`

Listing 1: Basic Usage of an ARCLE environment (`O2ARCEnv`) with Gymnasium API. Action is randomly sampled.

```
import arcle
import gymnasium as gym

env = gym.make('ARCLE/O2ARCEnv',
↪  render_mode='ansi')
obs, info = env.reset(options={'adaptation':
↪  True})

for _ in range(1000):
    action = env.action_space.sample()
    obs, reward, term, trunc, info =
    ↪  env.step(action)
    if term or trunc:
        obs, info = env.reset()
```

Listing 2: BBox wrapping of the environment. Action is randomly sampled once, resulting in a 5-tuple. Continuing code from Listing 1.

```
from arcle.wrappers import BBoxWrapper

env = BBoxWrapper(env)

obs, info = env.reset()
action = env.action_space.sample()
print(action) # 5-tuple: (y1, x1, y2, x2, op)
```

of a raw binary mask, is shown in Listing 2. The environment is wrapped using a `BBoxWrapper` from ARCLE. As a result, the random action returned by the `sample` function consists of a tuple of five numbers, the first four values representing the bounding box of the `selection` and the last value specifies `operation`. While configuring `selection` as a raw binary mask for a grid of size $H \times W$ offers the advantage of allowing free-form object configurations, it also poses the problem of having a very large action space of $O(2^{HW})$. On the other hand, configuring `selection` as a bounding box simplifies it to four integers, reducing the action space to $O(H^2W^2)$, but it limits the shape of the object to a rectangle. This restriction is in place that necessitates the selection of background pixels when dealing with non-rectangular objects. However, ARCLE actions differentiate between zero-valued pixels, which are considered blank, and non-zero pixels. This distinction ensures that when the object is isolated from other pixels, there will be less overlap of irrelevant pixels by the background when object-oriented actions are applied.

Chapter 4
ARCLE Benchmarks

This chapter explains the process through which an agent learns to solve synthetic tasks using the ARCLE environment. To ultimately solve ARC, the agent must acquire the ability to tackle unseen tasks through the learning process of tasks provided in the training dataset. Approaches like meta-RL, generative models (e.g. GFlowNet), and model-based RL algorithms (e.g. World Models) may be necessary to solve tasks not observed during training. The initial results of learning an individual task are described as a preliminary step. The PPO-based agent learns the input/output grid pairs presented in one of the ARC tasks. If a method can be designed for the agent to understand and learn from these tasks, it could be trained to solve unlearned tasks using the approaches mentioned above with this agent.

4.1 Solving ARC with a Given Answer

While it is expected that ARCLE agents be better at imitating the cognitive process of human problem-solving, training an RL agent for ARCLE itself additionally becomes a difficult challenge due to its large discrete state-action space. This section demonstrates the difficulty of obtaining highly performant agents within an ARCLE environment even when the state-action spaces are simplified the answers are given and ARCLE-specific auxiliary loss functions and network architectures to be proposed can significantly improve agents' performance. Specifically, these architectures use `operations` of 0–9 only with rectangular-shaped `selection` only (in a bounding box representation), and consequently, the sufficient information for decision making (i.e., the state s) becomes `(grid, grid_dim, answer, answer_dim)` as assuming answers to be given. We expect the methods introduced here to be used to help train ARCLE agents for the original ARC, where the answers are not provided and state-

action spaces are more complex.

Proximal Policy Optimization (PPO) The well-known PPO algorithm [31] is employed to train the agents to solve ARCLE with the answers given. Due to the poor generalization ability [32] and learning instability of value-based RL algorithms, recently, PPO has been widely adopted for tuning large neural models [33, 34]. It is an on-policy policy-gradient algorithm that aims to perform a gradient update within the trust region. Trajectories of the environment is collected and a data set $\mathcal{D} = \{(s_i, a_i, R_i)\}_i$ is constructed consisting of the state, the action, and the returns (the sum of the discounted rewards starting from the state). Then, the policy is updated according to the following losses ($\mathcal{L} = \mathcal{L}^{\text{Baseline}} + \mathcal{L}^{\text{PPO}}$) with samples from $\mathcal{D}$:

$$\mathcal{L}^{\text{Baseline}}(\psi) = \mathbb{E}_{\mathcal{D}}\left[(r - V_\psi(s))^2\right]$$
$$\mathcal{L}^{\text{PPO}}(\theta) = \mathbb{E}_{\mathcal{D}}\left[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)}\left(r - \texttt{sg}[V_\psi(s)]\right), \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\text{old}}(a|s)}, 1-\epsilon, 1+\epsilon\right)\left(R - \texttt{sg}[V_\psi(s)]\right)\right)\right],$$

where V_ψ is a value function that works as a baseline that reduces the gradient variance, π_{old} is a policy used to gather the trajectories, and $\texttt{sg}[\cdot]$ is a stop-gradient operator. $r \in [-1, 0]$ is a reward from a dense reward function that penalizes the agent by the ratio of incorrect pixels of the next state.

State encoder A shared state encoder for the policy π_θ is used, and the baseline V_ψ based on a Transformer encoder architecture [35]. Each pixel of `grid` and `answer` is encoded as a token by taking a summation over corresponding position, color, and token type embeddings, where token type embedding informs whether it belongs to `grid` or `answer`. Depending on `grid_dim` and `answer_dim`, the tokens with an inactive pixel are masked so that it is not attended by other tokens. Each function gets its own special token(s) and feed-forward network to pass the extracted state feature from its token. The baseline V_ψ use a single special token for its scalar output whereas the policy π_θ uses two or more tokens for representing both `operation` and `selection`, which will be detailed in Section 4.1.2.

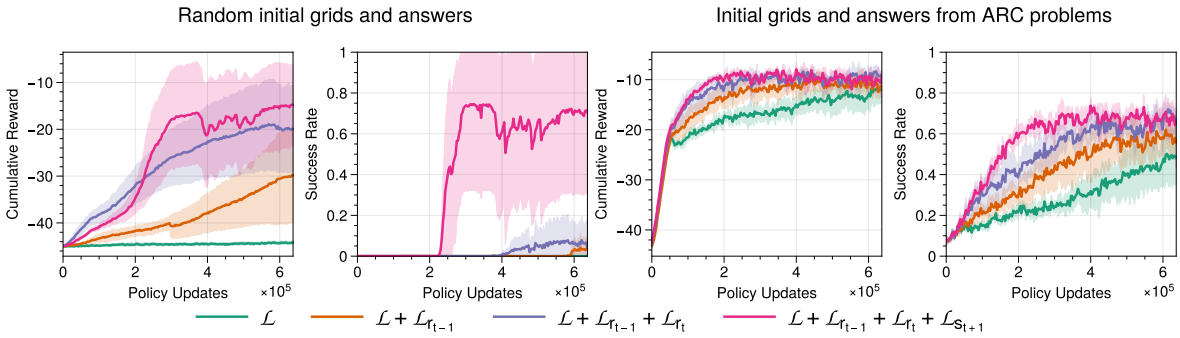

Figure 4.1: Performance of agents when various auxiliary losses are additionally used are shown. The experiment is repeated four times, and the shaded regions denote 95% confidence intervals.

In the following experiments, tasks with `grid_dim` and `answer_dim` less than 5×5 are only used due to the computational demand of the current state encoder architecture. However, it can be alleviated by using more scalable architecture, e.g., a patch as a token instead of a pixel as a token [36]. Using two different settings, **(1)** a random setting where the randomly generated 5×5 initial grid and goal are used, and **(2)** an ARC setting where we used initial grids and goals that are equal or smaller than 5×5 from ARC tasks, are experimented. In the random setting, a policy needs to act precisely due to the vast number of different goals, whereas, in the ARC setting, a policy needs to adapt to various grid sizes.

4.1.1 Learning better representation through auxiliary loss functions

Using an auxiliary loss function to predict important information has been a widely used approach for better generalizable representation and faster training [37, 38]. We experimented three different auxiliary losses, **(1)** $\mathcal{L}_{r_{t-1}}$ predicting the previous reward r_{t-1} from the current state s_t, **(2)** $\mathcal{L}_{r_t}$ predicting the current reward r_t from the current state-action (s_t, a_t), and **(3)** $\mathcal{L}_{s_{t+1}}$ predicting the next state s_{t+1} from the current state-action (s_t, a_t). All three functions are deterministic, and they are highly informative as they are correlated to either the value function or the action-value function. For policy architecture, the color-equivariant policy architecture is used which will be detailed in Section 4.1.2.

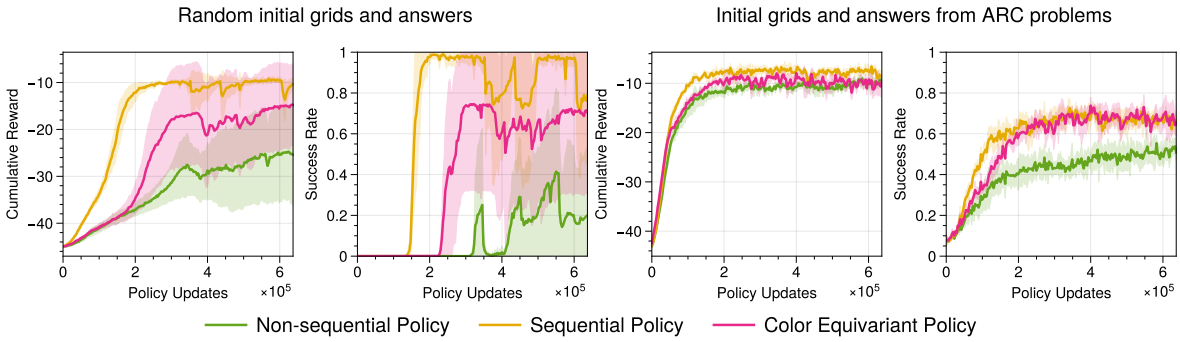

Figure 4.2: Performance of agents when equipped with different policy architectures. The experiment is repeated 4 times, and the shaded regions denote 95% confidence intervals.

While the first auxiliary loss $\mathcal{L}_{r_{t-1}}$ can be easily adopted by additionally training a feed-forward network on top of the extracted state feature from the special token of V_ψ, the other two auxiliary losses require the state-action feature that is not utilized in conventional PPO. The state-action feature are computed by performing a forward propagation again with additional action embedding tokens after sampling an action from a policy. The prediction for the loss $\mathcal{L}_{r_t}$ is done on top of the last action token that embeds `selection`, and the prediction for the loss $\mathcal{L}_{s_{t+1}}$ is done on top of tokens that represent each pixel of `grid`.

The results are reported in Figure 4.1. Note that the vanilla PPO agent was not able to learn anything in the random setting despite the vastly simplified state-action space, demonstrating the difficulty of training an agent for ARCLE. While all of the experimented auxiliary losses improve the learning of the agent, it can be seen that adopting auxiliary features and adopting state-action feature-based auxiliary features make a significant difference in performance. Only with all three of these auxiliary losses, it was possible to get three agents out of four that achieved a success rate larger than 95% in the random setting. On the other hand, in the ARC setting, auxiliary losses were able to help, but their effect was relatively less dramatic.

4.1.2 Non-factorizable policy architecture

It can be observed that the two main components of the action space of ARCLE, `operation` and `selection`, are intertwined with each other and cannot be separately decided. For example, the optimal `selection` for coloring a pixel, or rotating an object will be completely different. This observation shows that the considerate choice of policy architecture is necessary, as conventional factorized policy assuming (`operation` $\perp\!\!\!\perp$ `selection`) | s will have limited expressivity in representing such complex multimodal distributions. For all experiments in this section, all three auxiliary losses introduced in Section 4.1.1 are used.

To demonstrate the expressivity of different policy architectures, experiments on the following three architecture types: **(1) Non-sequential policy** assumes (`operation` $\perp$ $\perp$ `selection`) | s, are conducted. This policy is implemented by using two special tokens for `operation` and `selection` with two feed-forward networks on top of extracted features from those tokens. **(2) Sequential policy** does not assume conditional independence, by making the decision of `selection` dependent on sampled `operation`, similar to the RNN policy of [39]. This policy requires two forward passes to sample an action, one for sampling `operation` from its special token and one for sampling `selection` from the token embedding the sampled `operation`, and therefore it is more computationally demanding.

On the other hand, experiments on **(3) Color-equivariant policy** that takes advantage of the ARCLE task that the same permutation of colors applied to the task and the policy coloring actions results in the equivalent task, i.e., the color equivariance, are conducted. Color equivariance of the policy can be achieved by using several special tokens for policy equal to the number of `operation` to represent them, and by setting a special token of color-related `operation` as a function of color embedding used to represent `grid`. Then two different feed-forward networks are placed on top of extracted features of these `operation` tokens. One gives scalar output per token to be used as logits for deciding the `operation`. The other one is used to get the `operation`-specific `selection` on top of the sampled `operation` token. This policy only requires

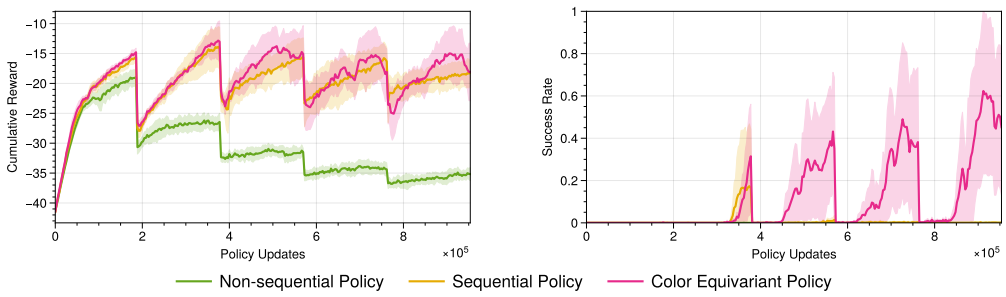

Figure 4.3: Performance of agents on continual RL task when equipped with different policy architectures. The experiment is repeated 4 times, and the shaded regions denote 95% confidence intervals.

one forward pass with a few additional `operation` tokens, and it is computationally efficient compared to the sequential policy.

Figure 4.2 summarizes the result. Overall, it can be observed that sequential policy and color equivariant policy outperform non-sequential policy, showing that conditional dependence is crucial for learning in ARCLE. Sequential policy shows more stable and faster learning compared to color equivariant policy in terms of policy updates. However, considering that sequential policy takes approximately 1.5x training time and 2x inference time, there is a clear trade-off and one can choose from two depending on the situation.

4.1.3 ARCLE as a Continual RL Environment

To address the inherent challenges of the ARCLE environment, it may be necessary to provide an agent with a curriculum. In such scenarios, an agent capable of continuously learning from a changing set of tasks would be beneficial. Experiments are conducted in a continual RL setting to demonstrate the robust learning capabilities of the proposed policy architectures in response to task changes. In this experiment, the initial grids and answers were randomly generated as before, but the number of colors used increased periodically—specifically, across five learning phases with 2, 4, 6, 8, and 10 colors respectively.

As depicted in Figure 4.3, all agents experienced a significant drop in performance whenever the number of colors increased. Similar to what is observed in Figure 4.2, the

non-sequential policy cannot express the complicated dependencies between `operation` and `selection`, and is outperformed by the other two policy architectures. However, within the context of the continual RL experiment, the sequential policy was not able to adapt to the new sets of tasks and recorded a 0% success rate after the second change in the environment. Conversely, the color equivariant policy demonstrated the ability to continuously improve its success rate, illustrating its rapid adaptability, which stems from its architectural design.

4.2 Research Directions

Based on Section 4.1, where we demonstrated the development and initial success of a PPO-based agent within ARCLE, this section aims to show future RL research in addressing the challenges. Inspired by François Chollet [1], we posit that an effective ARC solver must possess advanced abstraction and reasoning abilities. Thus, we propose several research directions using ARCLE (e.g. MAML, GFlowNet, and World Model).

4.2.1 Meta-RL for Enhanced Reasoning Skills

ARC is a multi-task few-shot learning problem: the whole dataset consists of multiple tasks, and each task has few demonstration pairs (supporting set) to infer the output of a test input (query set). ARCLE is in the identical problem but in the RL setting. To manage this, multi-task RL [40] or meta-RL [41] algorithms that foster an agent to experience over a task distribution could be applied. We have focused on developing ideas with meta-RL rather than multi-task RL as it gives a richer optimization. In this setting, the meta-training set and the meta-testing set are the training and evaluation sets given in the ARC dataset.

Meta-RL algorithms on ARCLE should be capable of outputting an RL algorithm that rapidly reasons and produces a policy for each ARC task, without exhaustive searching over actions. The policy is trained to generate valid trajectories from the input to the output grids simultaneously on multiple demonstration pairs in an ARC

task. Then the policy is applied to the test input grid to generate output.

Therefore, Meta-RL endows agents with essential reasoning skills for ARC's diverse tasks, enabling them to quickly adapt to new tasks by autonomously developing learning strategies. Integrating Meta-RL with ARCLE opens new pathways for researchers to devise techniques that allow AI to effectively generalize learning across various tasks, thus embodying the 'learning to learn' principle.

4.2.2 Generative Models as Surrogates for Reasoning

Generative models, particularly GFlowNet [42], offer a novel approach to tackling the reasoning challenges presented by ARC. While an agent is equally given a set of grid operations on the ARCLE, many possible trajectories can lead to a correct answer for an ARC task. Moreover, among demonstration pairs in an ARC task, the detailed trajectories for each pair are varied, as each pair has its own input grid. GFlowNet establishes its policy as a generative model that enables the sampling of actions from it, and the probability of sampling is proportional to the reward-driven objective. Therefore, GFlowNet benefits from not only learning a posterior distribution to include high-reward modes but also from searching multiple modes of a solution space by leveraging probabilistic reasoning to generate diverse possible solutions, in the form of a directed acyclic graph (DAG). This supports a GFlowNet policy to solve the demonstration pairs in one ARC task, although its input grids are different from (but possess the same rule) one another. Moreover, its ability to identify multiple viable solutions for individual ARC tasks underscores its utility for data augmentation with correct solutions, further enhancing its value as a research tool in this domain.

4.2.3 Model-Based RL for Abstraction Skills

Encoding the demonstration pairs based on the core knowledge is a crucial point in establishing a plan to solve an ARC task. Model-based RL, particularly World Models might be a solution to support abstraction in tackling the ARC pairs. Among the ARC tasks, there are dissimilar common rules over all pairs in a task, although, there are

a few categories of core knowledge that a common rule in each task be derived from. Objectness, goal-directness, arithmetic, geometric, and topology are part of them [1], and these can be infused in ARCLE's actions, like `Move` and `Rotate` operations. Since World Models internalize the environment transitions caused by ARCLE's actions to learn an agent on its simulation, it would learn a joint representation of ARCLE's grid pair and actions (containing core knowledge). It encourages a controller in World Model agent to utilize flexible neural representation, rather than hard-coded operations in ARCLE to simulate. In short, World Models would provide neural abstraction skills of the pairs and operations in ARCLE, which supports a controller to search a rule efficiently on the flexible representation.

Hence, developing agents that can construct and utilize these models is a step towards equipping them with the necessary abstraction skills for handling both trained and untrained tasks.

4.2.4 Further Research Questions

Several research questions would advance while tackling ARC with ARCLE. First, the ARC task does not possess an explicit task distribution since individual ARC tasks include a unique rule and the current ARC dataset has only a finite 800 training and evaluation tasks. Categorizing ARC tasks correspondingly to the core knowledge [18] and parameterizing tasks in each category similarly to XLand [43] would be a worthwhile topic that reinforces meta-RL and multi-task learning more promising methods to solve ARC with broader tasks.

Next, ARCLE's action space consists of two sub-action spaces: an integer `operation` and a discrete binary mask `selection`. Handling with `selection` might entail an exponential size of search space: in particular, the size of DAG with the GFlowNet approaches grows enormously to degrade the efficiency of figuring out the correct output grid. One probable setting is that utilizing a sequential policy in 4.1.2, maintaining two networks that produce `operation` and `selection` hierarchically. Then this GFlowNet may maintain a DAG of sampling `operation` only by considering ARCLE as a proba-

bilistic environment. However, the validity of this method is indeterministic.

Lastly, one might doubt the necessity of World Models in solving ARC to provide richer abstraction. The reason would be that training agents directly in the environment is more straightforward since the environment's dynamics are deterministic, rather than learning the World Models. Nevertheless, in ARC and ARCLE, there is a significant amount of auxiliary information to abstract more than a transition of observation: object information and their topology, symmetry, and so forth. Previous studies have shown that it can capture information such as a hidden gravity parameter in an environment [44], and it is expected that additional useful information for ARC can be extracted as well. One open question brought here is what information a World Model can extract for ARC. In particular, whether it can disentangle and extract information common to every ARC task (e.g., state transition) and task-specific priors (e.g., objectness) in an interpretable form is a research question for the future.

Chapter 5
Conclusion

This thesis introduced ARCLE, an RL environment designed for the ARC benchmark, using the Gymnasium library for direct engagement with ARC's challenges. Our development and application of a PPO-based agent, enhanced with auxiliary losses and non-factorizable policies, have demonstrated ARCLE's possibility of learning and performance improvements in addressing ARC tasks. In detail, auxiliary losses improved learning outcomes, especially evident in random settings where the comprehensive application of all proposed strategies yielded the best performance. The success rate in these settings, and the superior outcomes from applying sequential and color-equivalent policies, underline the importance of strategic `operation` and `selection` processes.

These experimental results show advanced RL methodologies—such as meta-RL, generative models, and model-based RL—to further enhance AI's reasoning and abstraction abilities. Specifically, meta-RL offers the potential to refine AI's reasoning skills by enabling adaptive learning strategies across varied tasks, suggesting a path toward more generalized intelligence. Generative models, by simulating complex reasoning processes, could serve as links between data and sophisticated decision-making in ARC. Model-based RL model could strengthen AI's ability to distill and apply abstract concepts from complex inputs. Thus, further research using ARCLE could elevate AI's learning strategies and expand the boundaries of its current capabilities. We invite the RL community to engage with ARCLE not just to solve ARC but to contribute to the broader endeavor of advancing AI research. Through such collaborative efforts, we can unlock new horizons in AI's ability to learn, reason, and abstract, marking significant progress in the field.

References

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

2. M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," 2023.

3. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv:1312.5602*, 2013.

4. J. Revaud, P. Weinzaepfel, C. De Souza, N. Pion, G. Csurka, Y. Cabon, and M. Humenberger, "R2D2: Repeatable and Reliable Detector and Descriptor," *arXiv:1906.06195*, 2019.

5. J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model," *Nature*, 2020.

6. A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell, "Agent57: Outperforming the Atari Human Benchmark," in *ICML*, 2020.

7. Y. Qi, K. Zhang, A. Sain, and Y.-Z. Song, "PQA: Perceptual Question Answering," in *CVPR*, 2021.

8. S. Kim, P. Phunyaphibarn, D. Ahn, and S. Kim, "Playgrounds for Abstraction and Reasoning," in *NeurIPS Workshop on Neuro Causal and Symbolic AI*, 2022.

9. Y. Xu, E. B. Khalil, and S. Sanner, "Graphs, Constraints, and Search for the Abstraction and Reasoning Corpus," in *AAAI*, 2023.

10. A. Banburski, A. Gandhi, S. Alford, S. Dandekar, S. Chin, and T. Poggio, "Dreaming with ARC," in *NeurIPS Workshop on Learning Meets Combinatorial Algorithms*, 2020.

11. S. Acquaviva, Y. Pu, M. Kryven, T. Sechopoulos, C. Wong, G. Ecanow, M. Nye, M. Tessler, and J. B. Tenenbaum, "Communicating Natural Programs to Humans and Machines," in *NeurIPS*, 2022.

12. R. Assouel, P. Rodriguez, P. Taslakian, D. Vazquez, and Y. Bengio, "Object-Centric Compositional Imagination for Visual Abstract Reasoning," in *ICLR Workshop on the Elements of Reasoning: Objects, Structure, and Causality*, 2022.

13. S. Alford, A. Gandhi, A. Rangamani, A. Banburski, T. Wang, S. Dandekar, J. Chin, T. Poggio, and P. Chin, "Neural-Guided, Bidirectional Program Search for Abstraction and Reasoning," in *COMPLEX NETWORKS*, 2021.

14. J. Witt, S. Rasing, S. Dumančić, T. Guns, and C.-C. Carbon, "A Divide-Align-Conquer Strategy for Program Synthesis," *arXiv:2301.03094*, 2023.

15. J. Ainooson, D. Sanyal, J. P. Michelson, Y. Yang, and M. Kunda, "A

Neurodiversity-Inspired Solver for the Abstraction & Reasoning Corpus (ARC) Using Visual Imagery and Program Synthesis," *arXiv:2302.09425*, 2023.

16. G. Camposampiero, L. Houmard, B. Estermann, J. Mathys, and R. Wattenhofer, "Abstract Visual Reasoning Enabled by Language," in *CVPR*, 2023.

17. Y. Xu, W. Li, P. Vaezipoor, S. Sanner, and E. B. Khalil, "LLMs and the Abstraction and Reasoning Corpus: Successes, Failures, and the Importance of Object-Based Representations," *Transactions on Machine Learning Research*, 2024.

18. A. Moskvichev, V. V. Odouard, and M. Mitchell, "The ConceptARC Benchmark: Evaluating Understanding and Generalization in the ARC Domain," *Transactions on Machine Learning Research*, 2023.

19. M. Mitchell, A. B. Palmarini, and A. Moskvichev, "Comparing Humans, GPT-4, and GPT-4V On Abstraction and Reasoning Tasks," in *AAAI Workshop on Are Large Language Models Simply Causal Parrots?*, 2024.

20. S. Lee, W. Sim, D. Shin, S. Hwang, W. Seo, J. Park, S. Lee, S. Kim, and S. Kim, "Reasoning Abilities of Large Language Models: In-Depth Analysis on the Abstraction and Reasoning Corpus," *arXiv:2403.11793*, 2024.

21. N. Butt, B. Manczak, A. Wiggers, C. Rainone, D. Zhang, M. Defferrard, and T. Cohen, "CodeIt: Self-Improving Language Models with Prioritized Hindsight Replay," *arXiv:2402.04858*, 2024.

22. J. Park, J. Im, S. Hwang, M. Lim, S. Ualibekova, S. Kim, and S. Kim, "Unrav-

eling the ARC Puzzle: Mimicking Human Solutions with Object-Centric Decision Transformer," in *ICML Workshop on Interactive Learning with Implicit Human Feedback*, 2023.

23. O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, *et al.*, "Starcraft II: A New Challenge for Reinforcement Learning," *arXiv:1708.04782*, 2017.

24. C. Kauten, "An OpenAI Gym Environment for Super Mario Bros," 2018.

25. H. Küttler, N. Nardelli, A. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel, "The Nethack Learning Environment," in *NeurIPS*, 2020.

26. T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine, "Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning," in *CoRL*, 2020.

27. S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, "RLBench: The Robot Learning Benchmark & Learning Environment," *IEEE Robotics and Automation Letters*, 2020.

28. O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard, "Calvin: A Benchmark for Language-Conditioned Policy Learning for Long-Horizon Robot Manipulation Tasks," *IEEE Robotics and Automation Letters*, 2022.

29. F. Chollet, "ARC Testing Interface," 2019.

30. G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *arXiv:1606.01540*, 2016.

31. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization algorithms," *arXiv:1707.06347*, 2017.

32. A. Kumar, R. Agarwal, T. Ma, A. Courville, G. Tucker, and S. Levine, "DR3: Value-Based Deep Reinforcement Learning Requires Explicit Regularization," in *ICLR*, 2021.

33. N. Stiennon, L. Ouyang, J. Wu, D. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. F. Christiano, "Learning to summarize with human feedback," in *NeurIPS*, 2020.

34. L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, *et al.*, "Training Language Models to Follow Instructions with Human Feedback," in *NeurIPS*, 2022.

35. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is All You Need," in *NeurIPS*, 2017.

36. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale," in *ICLR*, 2021.

37. M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and

K. Kavukcuoglu, "Reinforcement Learning with Unsupervised Auxiliary Tasks," in *ICLR*, 2016.

38. G. Lample and D. S. Chaplot, "Playing FPS Games with Deep Reinforcement Learning," in *AAAI*, 2017.

39. O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster Level in StarCraft II using Multi-Agent Reinforcement Learning," *Nature*, 2019.

40. A. Wilson, A. Fern, S. Ray, and P. Tadepalli, "Multi-Task Reinforcement Learning: A Hierarchical Bayesian Approach," in *ICML*, 2007.

41. C. Finn, P. Abbeel, and S. Levine, "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks," in *ICML*, 2017.

42. Y. Bengio, S. Lahlou, T. Deleu, E. J. Hu, M. Tiwari, and E. Bengio, "GFlowNet Foundations," *arXiv:2111.09266*, 2023.

43. J. Bauer, K. Baumli, F. Behbahani, A. Bhoopchand, N. Bradley-Schmieg, M. Chang, N. Clay, A. Collister, V. Dasagi, L. Gonzalez, *et al.*, "Human-Timescale Adaptation in an Open-Ended Task Space," in *ICML*, 2023.

44. C. Reale and R. Russell, "Learning and Understanding a Disentangled Feature Representation for Hidden Parameters in Reinforcement Learning," *arXiv.2211.16315*, 2022.

45. S. Shim, D. Ko, H. Lee, S. Lee, D. Song, S. Hwang, S. Kim, and S. Kim, "O2ARC 3.0: A Platform for Solving and Creating ARC Tasks," in *IJCAI Demo*, 2024.

Appendix A
Object-Oriented ARC (O2ARC) Web Interface

Object-Oriented ARC (O2ARC) is a web interface that allows humans to directly solve ARC tasks and collect the process of solving them [8, 45][1], an improvement upon the initial testing web interface developed by François Chollet [29]. Initially, Chollet's testing web interface featured only a basic version involving coloring. However, the version of O2ARC has progressively improved to include object-oriented actions such as movement, rotation, and mirroring. Since actions represent the most intuitive low-level actions conceivable by humans, the sequence of actions (traces) could be used as a dataset reflecting human cognitive processes. The most recent version of O2ARC allows for the collection of traces solved by humans and also includes the ability to create tasks directly, thus evolving into a tool that can aid in the development of general artificial intelligence capable of mimicking human cognitive processes using ARC. The dataset collecting human traces is valuable in the research and development of artificial intelligence capable of human-like thinking.

Previous research has been conducted on whether learning from human traces can solve tasks [22]. This research demonstrated that with a sufficient number of traces solved by humans, it becomes feasible to solve ARC tasks by reflecting human solutions, thereby showcasing the potential of offline reinforcement learning. In line with this, ARCLE has been developed by integrating the actions of O2ARC to investigate whether an agent can address ARC tasks like human thinking. While O2ARC has one of its strengths in collecting human traces, ARCLE has the advantage of being able to train agents using the actions same as O2ARC. Therefore, ARCLE can be seen as having

[1] `https://o2arc.com`

transformed O2ARC into a reinforcement learning environment for solving ARC tasks by agents.

Figure A.1 presents the interface for solving ARC tasks in O2ARC. As depicted, The left side part "See the original pairs" displays demo pairs corresponding to the task, while the center provides the test input grid for the task. Users infer common rules from these examples to guess the appropriate answer for the given input. The right side features a grid space labeled "What should be the result" where users input their answers. Additionally, a palette on the far right offers a selection of 10 colors for ARC. Users can use O2ARC's functionalities to color pixels, select objects, and perform actions such as rotation or copying and pasting. If necessary, they can input integer numbers into width and height cells to resize and submit the correct answer.

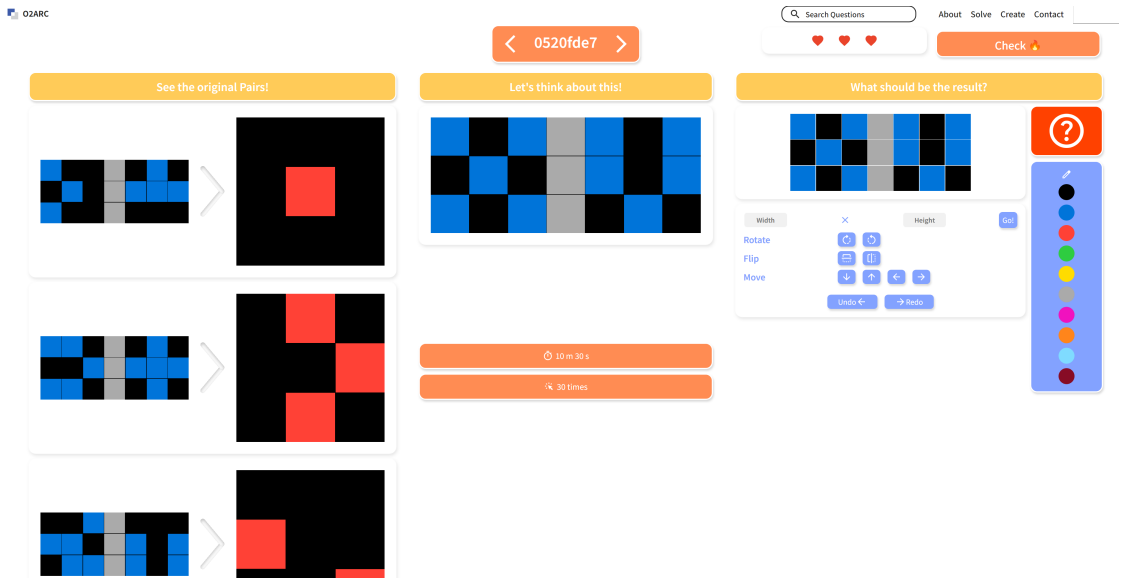

Figure A.1: O2ARC Solve page. The left side shows demo input and output grid pairs, the center demonstrates the test input grid, and the right side consists of the result grid and available actions.

Appendix B
Abbreviations

AI	Artificial Intelligence
AGI	Artificial General Intelligence
RL	Reinforcement Learning
PPO	Proximal Policy Optimization
ARC	Abstraction and Reasoning Corpus
ARCLE	ARC Learning Environment
MAML	Model-Agnostic Meta-Learning
O2ARC	Object-Oriented ARC (Web Interface)
DAG	Directed Acyclic Graph

Acknowledgements

꾸준히 저를 무조건적으로 지지해주시고, 조언해주신 부모님, 할머님께 매우 감사합니다. 학부 재학기간 4년간 저를 응원해주고 격려해주고 지지해준 동아리 선배와 친구 모두에게 감사합니다. 좋아하는 것에 대해 같이 이야기하고, 해커톤도 같이 출전하는 등 다양한 추억을 쌓을 수 있었습니다. 데이터사이언스연구실에서 연구와 개발, 랩 이벤트를 열면서 많은 지원해주신 김선동 교수님과 같이 연구했던 연구실 동료들 모두에게 감사드립니다. 다루기 어려운 문제를 해결하고자 할 때의 어려움과 접근 방법, 새로운 아이디어가 나오는 과정, 논문 쓰는 방법, 영어 발표하는 방법 뿐만 아니라 교내외에의 있는 다양한 기회를 소개해주시고 연결해주셔서 감사합니다.