ARCLE: THE ABSTRACT AND REASONING CORPUS LEARNING ENVIRONMENT FOR REINFORCEMENT LEARNING

Hosung Lee[1]*, Sejin Kim[1]*, Seungpil Lee[1], Sanha Hwang[1], Jihwan Lee[1], Byung-Jun Lee[2], Sundong Kim[1]
[1]Gwangju Institute of Science and Technology
[2]Korea University
gitpush-force@gm.gist.ac.kr, sundong@gist.ac.kr

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 a variety of tasks. We demonstrate that an agent with proximal policy optimization can learn individual tasks through ARCLE. The adoption of non-factorial policies and auxiliary losses led to performance enhancements, effectively mitigating issues associated with action spaces and goal attainment. Based on these insights, we propose several research directions and motivations for using ARCLE, including MAML, GFlowNets, and World Models.

1 INTRODUCTION

We introduce ARCLE (ARC Learning Environment) as a reinforcement learning (RL) environment designed for the Abstraction and Reasoning Challenge (ARC) benchmark (Chollet, 2019a). 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. ARC is designed to test abstraction and reasoning skills, making it touch benchmark within the domain. Despite various attempts to conquer ARC's complexities through program synthesis and reasoning with large language models, RL-based approaches are surprisingly rare (Section 2.1). We believe this scarcity is due to the lack of a dedicated RL environment tailored for ARC. To fill this gap, we created ARCLE based on Gymnasium (Towers et al., 2023) to tackle the benchmark.

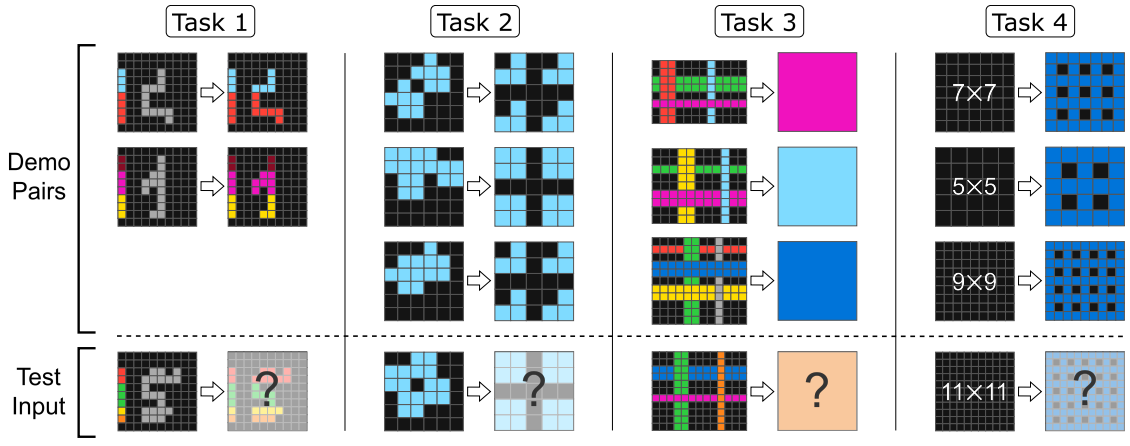


Figure 1: Four different ARC tasks are presented, each requiring analysis through its provided Demo Pairs. The identified rule from this analysis must then be applied to a Test Input grid to produce the answer grid, which is currently blurred for demonstration purposes. The specific rules for each task are as follows: 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. Task 4 transforms the Test Input grid by coloring all but the pixels at the intersections of even-numbered rows and columns in blue.

* Equal Contribution

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. Each ARC task demands not just strategic action but also a nuanced comprehension of the underlying patterns and rules.

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 which 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.

2 Related Works

2.1 Solving ARC

Since the unveiling of the ARC (Chollet, 2019a), approaches ranging from the development of similar benchmarks (Qi et al., 2021; Kim et al., 2022; Xu et al., 2023a) to domain-specific languages and program synthesis (Banburski et al., 2020; Acquaviva et al., 2022; Assouel et al., 2022; Alford et al., 2021; Witt et al., 2023; Ainooson et al., 2023) 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 (Camposampiero et al., 2023; Xu et al., 2023b; Moskvichev et al., 2023; Mitchell et al., 2023; Lee et al., 2024), underscores LLMs' potential in addressing ARC's intricacies.

However, the performance of research utilizing program synthesis and LLMs has not fully met expectations, often due to its logical flaw, called hallucination. This has prompted a pivot towards reinforcement learning as a novel approach, albeit its application to ARC has been limited so far. Notable attempts include the use of RL strategies in program synthesis (Butt et al., 2024) and the exploration of imitation learning (Park et al., 2023). The introduction of ARCLE opens up new possibilities for advancing research on the ARC using RL.

2.2 RL environments similar to ARCLE

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 (Vinyals et al., 2017), 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 (Kauten, 2018), NLE (Küttler et al., 2020)) 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 (Yu et al., 2020), RLBench (James et al., 2020), and CALVIN (Mees et al., 2022), underscoring the complexity and **variety of tasks** that ARCLE introduces to RL research.

[1] Atari benchmarks hosts 57 games, each with its goal. Solutions such as Rainbow DQN (Mnih et al., 2013), R2D2 (Revaud et al., 2019), MuZero (Schrittwieser et al., 2020), and Agent57 (Badia et al., 2020) focus on mastering single games.

3 ARCLE: ARC LEARNING ENVIRONMENT

ARCLE is a reinforcement learning (RL) environment package, implemented in Gymnasium, 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 2 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 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 environment features and defines the ARC-specific general structure of action and state space and user-definable methods. And 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.3), and likewise, `ARCEnv` offers the testing web interface developed by Chollet (2019b). `RawARCEnv` restricts the action space to color modifications or grid size changes, providing a more constrained and monotonic learning environment.

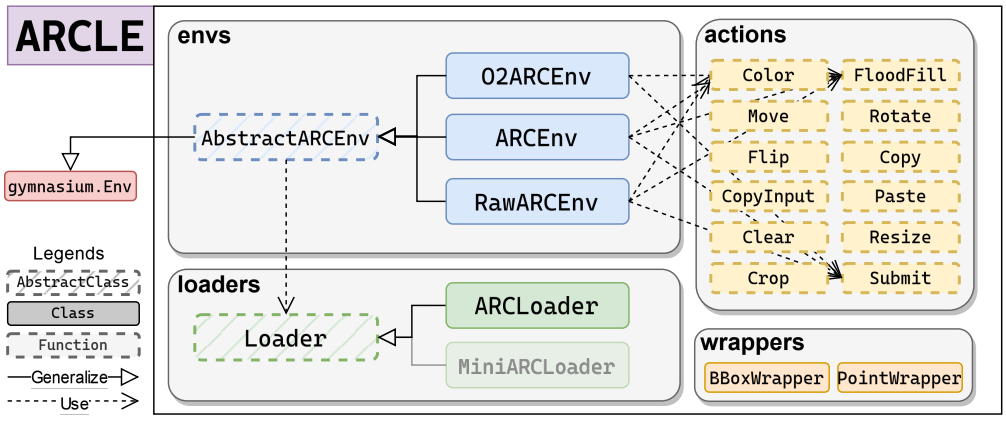


Figure 2: Overall framework of ARCLE.

Next, the **loaders** component provides functionalities to feed the ARC dataset to ARCLE environments. This component consists of the base `Loader` class defining interface requirements to ARCLE environments and its implementations. `ARCLoader` feeds the ARC dataset to any ARCLE environments, defining how the ARC dataset should be parsed from files and how the parsed dataset should be picked. Likewise, to load a similar dataset, one can inherit the `Loader` class and specify how to parse and sample. As an example, we implemented `MiniARCLoader` which loads Mini-ARC dataset (Kim et al., 2022).

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 (Chollet, 2019b).

We focus on explaining `O2ARCEnv` in the following sections, which encompasses most operations by ARCLE.

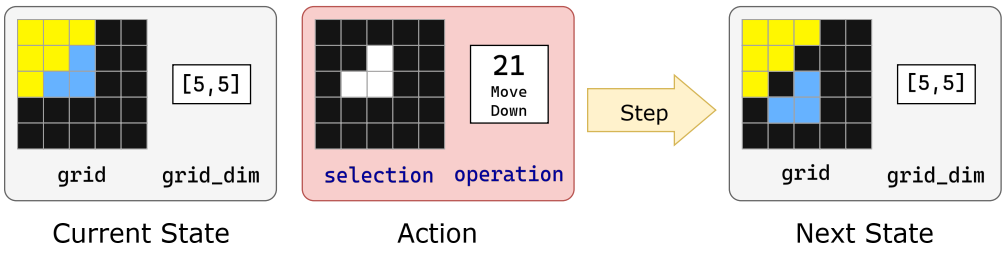


Figure 3: The state transition process of ARCLE.

Table 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

3.1 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 the method of editing (functions in the actions block in Figure 2), 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, we have 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 the introduction of the binary mask selection. Furthermore, by separating `operation` and `selection`, it accommodates the possibility of determining selection conditioned by chosen operation autoregressively.

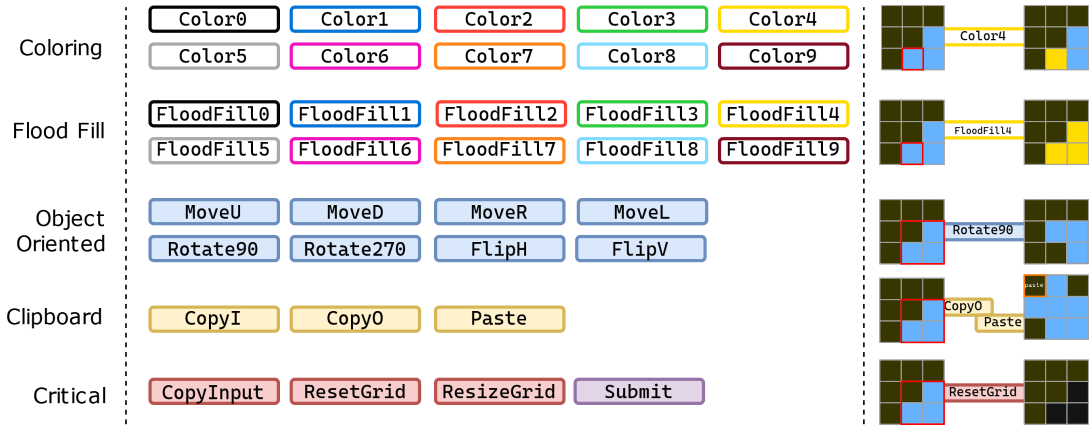


Figure 4: Every `operation` assigned in `O2ARCEnv` (version 0.2.5). Category of operations (left), available operations (middle), and brief examples (right) are shown.

Currently, 35 operations are available in `O2ARCEnv` (Figure 4). When an agent specifies the operation index, the corresponding one is executed. Specifically, operation indices 0–9 represent `Coloring` 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. Actions not present in the original ARC interface (Chollet, 2019b), 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 actions for `Copy` and `Paste`, and 31–34 are assigned to actions that cause breaking changes in the states like duplicating the test input (`CopyInput`), clearing the grid (`ResetGrid`), changing the grid size (`ResizeGrid`), and submitting (`Submit`).

Customization of specific actions by adding or removing them in the same format is freely allowed by subclassing the environments. For a detailed description of every operation in ARCLE, see Appendix A.4.

3.2 States & 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` operations, the environment should include an additional variable 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 1. While the agent performs object-oriented operations 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 Section A.5.

3.3 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.

3.4 Source Code

Since the environments in ARCLE implemented based on Gymnasium (Towers et al., 2023) and are fully written in Python3, users who have used Gymnasium or its predecessor, OpenAI Gym (Brockman et al., 2016), can use it with familiarity. ARCLE is released on GitHub[2] 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.[3] 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.

3.5 API & Sample Usage

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

```python
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.

```python
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)
```

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

[2] https://github.com/ConfeitoHS/arcle

[3] `$ pip install arcle==0.2.5`

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, 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 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 no overlap of irrelevant pixels by the background when object-oriented actions are applied.

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. We speculate that 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. As a preliminary step, we describe the initial results for learning an individual task. 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, we anticipate that it could be trained to solve unlearned tasks using the approaches mentioned above with this agent.

4.1 SOLVING ARC WITH A GIVEN ANSWER: HANDLING THE LARGE DISCRETE STATE-ACTION SPACE

While we expect ARCLE agents to 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. In this Section, we demonstrate the difficulty of obtaining highly performant agents within an ARCLE environment even when the state-action spaces are simplified and the answers are given, and we propose ARCLE-specific auxiliary loss functions and network architectures that can significantly improve agents' performance. Specifically, we 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 we additionally assume 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) We employed the well-known PPO algorithm (Schulman et al., 2017) to train the agents to solve ARCLE with the answers given. Due to the poor generalization ability (Kumar et al., 2021) and learning instability of value-based RL algorithms, recently, PPO has been widely adopted for tuning large neural models (Stiennon et al., 2020; Ouyang et al., 2022). It is an on-policy policy-gradient algorithm that aims to perform a gradient update within the trust region. We gather trajectories and construct a dataset $\mathcal{D} = \{(s_i, a_i, R_i)\}_i$ consisting of state, action, and returns (sum of 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 $\mathcal{D}$:

$$\mathcal{L}^{\text{Baseline}}(\psi) = \mathbb{E}_{\mathcal{D}}\left[\left(r - V_\psi(s)\right)^2\right]$$

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

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 $\text{sg}[\cdot]$ 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.

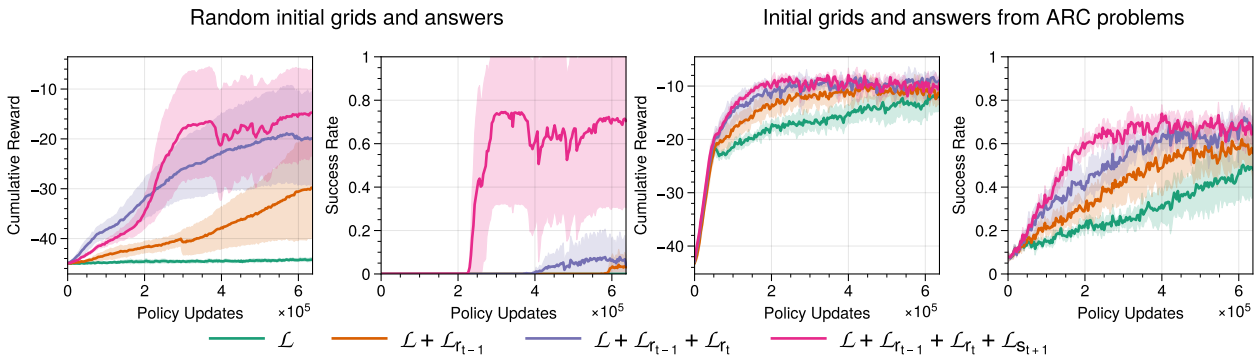


Figure 5: Performance of agents when various auxiliary losses are additionally used are shown. The experiment is repeated four times, and the shaded regions denote 95% confidence intervals.

State Encoder We used a shared state encoder for the policy π_θ and the baseline V_ψ based on a Transformer encoder architecture (Vaswani et al., 2017). 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.

In the following experiments, we only used tasks with `grid_dim` and `answer_dim` less than 5×5 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 (Dosovitskiy et al., 2020). We experimented using two different settings, **(1)** a random setting where we use the randomly generated 5×5 initial grid and goal, and **(2)** an ARC setting where we used initial grids and goals that are equal or smaller than 5×5 from ARC tasks. In the random setting, we need to act precisely due to the vast number of different goals, whereas in the ARC setting, we need 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 (Jaderberg et al., 2016; Lample & Chaplot, 2017). 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, we used the color-equivariant policy architecture that will be detailed in Section 4.1.2.

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. We compute the state-action feature 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 5. 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, we were able 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.

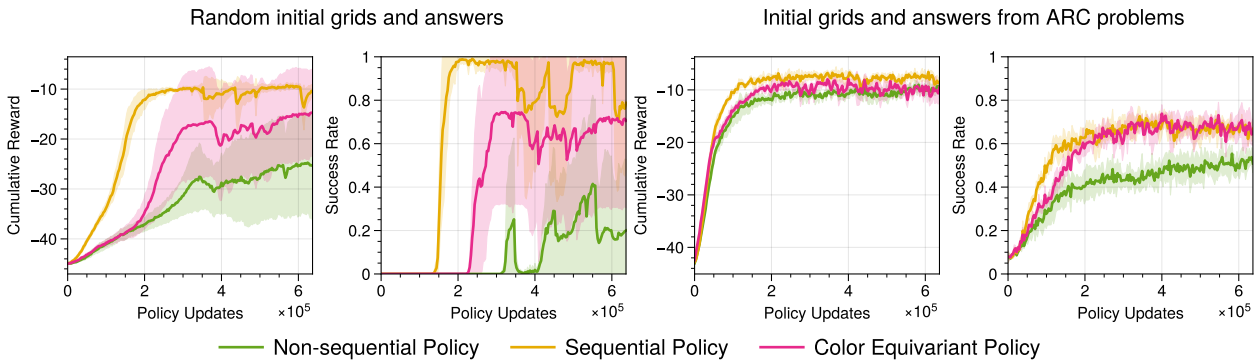


Figure 6: Performance of agents when equipped with different policy architectures. The experiment is repeated 4 times, and the shaded regions denote 95% confidence intervals.

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 $(\texttt{operation} \perp\!\!\!\perp \texttt{selection}) \mid s$ will have limited expressivity in representing such complex multimodal distributions. For all experiments in this section, we used all three auxiliary losses introduced in Section 4.1.1.

To demonstrate the expressivity of different policy architectures, we experimented with the following three architecture types: **(1) Non-sequential policy** assumes $(\texttt{operation} \perp\!\!\!\perp \texttt{selection}) \mid s$. 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 Vinyals et al. (2019). 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, we also experimented **(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. We can achieve color equivariance of the policy 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`. We can then use two different feed-forward networks 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 one forward pass with a few additional `operation` tokens, and it is computationally efficient compared to the sequential policy.

Figure 6 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 we 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. We conducted a continual RL experiment 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.

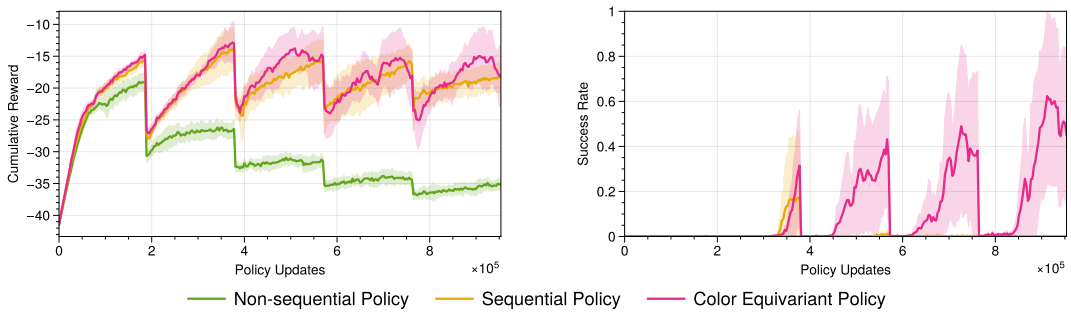


Figure 7: Performance of agents on continual RL task when equipped with different policy architectures. The experiment is repeated 4 times, and the shaded regions denote 95% confidence intervals.

As depicted in Figure 7, all agents experienced a significant drop in performance whenever the number of colors increased. Similar to what we have observed in Figure 6, 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 Future Directions for RL Research in Solving ARC with ARCLE

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 Chollet (2019a), 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), with more details in Appendix A.6.

4.2.1 Meta-RL for Enhancing 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 category of problem, multi-task RL (Wilson et al., 2007) or meta-RL (Finn et al., 2017) 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 task 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 (Bengio et al., 2021), 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 (Chollet, 2019a), 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 (Moskvichev et al., 2023) and parameterizing tasks in each category similarly to XLand (Bauer et al., 2023) 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 we utilized 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 probabilistic 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 (Reale & Russell, 2022), 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.

5 CONCLUSION

In this paper, we 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.

ACKNOWLEDGMENTS

This work was supported by the IITP (RS-2023-00216011, No. 2022-0-00311), the National Research Foundation (RS-2023-00240062), Artificial Intelligence Graduate School Program (No. 2019-0-00079, No. 2019-0-01842), and GIST (AI-based Research Scientist Project) funded by the Ministry of Science and ICT, Korea.

REFERENCES

Samuel Acquaviva, Yewen Pu, Marta Kryven, Theodoros Sechopoulos, Catherine Wong, Gabrielle Ecanow, Maxwell Nye, Michael Tessler, and Joshua B. Tenenbaum. Communicating Natural Programs to Humans and Machines. In *NeurIPS*, 2022.

James Ainooson, Deepayan Sanyal, Joel P. Michelson, Yuan Yang, and Maithilee Kunda. A Neurodiversity-Inspired Solver for the Abstraction & Reasoning Corpus (ARC) Using Visual Imagery and Program Synthesis. *arXiv:2302.09425*, 2023.

Simon Alford, Anshula Gandhi, Akshay Rangamani, Andrzej Banburski, Tony Wang, Sylee Dandekar, John Chin, Tomaso Poggio, and Peter Chin. Neural-Guided, Bidirectional Program Search for Abstraction and Reasoning. In *Complex Networks*, 2021.

Rim Assouel, Pau Rodriguez, Perouz Taslakian, David Vazquez, and Yoshua Bengio. Object-centric Compositional Imagination for Visual Abstract Reasoning. In *ICLR Workshop on the Elements of Reasoning: Objects, Structure, and Causality*, 2022.

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the Atari Human Benchmark. In *ICML*, 2020.

Andrzej Banburski, Anshula Gandhi, Simon Alford, Sylee Dandekar, Sang Chin, and Tomaso Poggio. Dreaming with ARC. In *NeurIPS Workshop on Learning Meets Combinatorial Algorithms*, 2020.

David Barrett, Felix Hill, Adam Santoro, Ari Morcos, and Timothy Lillicrap. Measuring Abstract Reasoning in Neural Networks. In *ICML*, 2018.

Jakob Bauer, Kate Baumli, Feryal Behbahani, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, Vibhavari Dasagi, Lucy Gonzalez, et al. Human-Timescale Adaptation in an Open-Ended Task Space. In *ICML*, 2023.

Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A Survey of Meta-Reinforcement Learning. *arXiv:2301.08028*, 2023.

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow Network based Generative Models for Non-Iterative Diverse Candidate Generation. In *NeurIPS*, 2021.

Yoshua Bengio, Salem Lahlou, Tristan Deleu, Edward J. Hu, Mo Tiwari, and Emmanuel Bengio. GFlowNet Foundations. *arXiv:2111.09266*, 2023.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540*, 2016.

Natasha Butt, Blazej Manczak, Auke Wiggers, Corrado Rainone, David Zhang, Michaël Defferrard, and Taco Cohen. CodeIt: Self-Improving Language Models with Prioritized Hindsight Replay. *arXiv:2402.04858*, 2024.

Giacomo Camposampiero, Loïc Houmard, Benjamin Estermann, Joël Mathys, and Roger Wattenhofer. Abstract Visual Reasoning Enabled by Language. In *CVPR*, 2023.

François Chollet. On the Measure of Intelligence. *arXiv:1911.01547*, 2019a.

François Chollet. ARC Testing Interface, 2019b. URL https://github.com/fchollet/ARC/blob/master/apps/testing_interface.html.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*, 2020.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *ICML*, 2017.

David Ha and Jürgen Schmidhuber. World Models. *arXiv:1803.10122*, 2018.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering Diverse Domains through World Models. *arXiv:2301.04104*, 2023.

Felix Hill, Adam Santoro, David GT Barrett, Ari S Morcos, and Timothy Lillicrap. Learning to Make Analogies by Contrasting Abstract Relational Structure. *arXiv:1902.00120*, 2019.

Michael Hodel. ARC: Where Do We Stand Today?, 2023. URL https://lab42.global/arc-article/.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement Learning with Unsupervised Auxiliary Tasks. In *ICLR*, 2016.

Moksh Jain, Emmanuel Bengio, Alex-Hernandez Garcia, Jarrid Rector-Brooks, Bonaventure F. P. Dossou, Chanakya Ekbote, Jie Fu, Tianyu Zhang, Micheal Kilgour, Dinghuai Zhang, Lena Simine, Payel Das, and Yoshua Bengio. Biological Sequence Design with GFlowNets. In *ICML*, 2022.

Moksh Jain, Sharath Chandra Raparthy, Alex Hernandez-Garcia, Jarrid Rector-Brooks, Yoshua Bengio, Santiago Miret, and Emmanuel Bengio. Multi-Objective GFlowNets. In *ICML*, 2023.

Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. RLBench: The Robot Learning Benchmark & Learning Environment. *RA-L*, 2020.

Aysja Johnson, Wai Keen Vong, Brenden M. Lake, and Todd M. Gureckis. Fast and Flexible: Human Program Induction in Abstract Reasoning Tasks. In *CogSci*, 2021.

Christian Kauten. An OpenAI Gym Environment for Super Mario Bros, 2018. URL https://github.com/Kautenja/gym-super-mario-bros.

Subin Kim, Prin Phunyaphibarn, Donghyun Ahn, and Sundong Kim. Playgrounds for Abstraction and Reasoning. In *NeurIPS Workshop on Neuro Causal and Symbolic AI*, 2022.

Aviral Kumar, Rishabh Agarwal, Tengyu Ma, Aaron Courville, George Tucker, and Sergey Levine. DR3: Value-Based Deep Reinforcement Learning Requires Explicit Regularization. In *ICLR*, 2021.

Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The Nethack Learning Environment. In *NeurIPS*, 2020.

Guillaume Lample and Devendra Singh Chaplot. Playing FPS Games with Deep Reinforcement Learning. In *AAAI*, 2017.

Seungpil Lee, Woochang Sim, Donghyeon Shin, Sanha Hwang, Wongyu Seo, Jiwon Park, Seokki Lee, Sejin Kim, , and Sundong Kim. Reasoning Abilities of Large Language Models: In-Depth Analysis on the Abstraction and Reasoning Corpus. *arXiv:2403.11793*, 2024.

Kanika Madan, Jarrid Rector-Brooks, Maksym Korablyov, Emmanuel Bengio, Moksh Jain, Andrei Nica, Tom Bosc, Yoshua Bengio, and Nikolay Malkin. Learning GFlowNets from Partial Episodes for Improved Convergence and Stability. In *ICML*, 2023.

Nikolay Malkin, Moksh Jain, Emmanuel Bengio, Chen Sun, and Yoshua Bengio. Trajectory Balance: Improved Credit Assignment in GFlowNets. In *NeurIPS*, 2022.

Mikołaj Małkiński and Jacek Mańdziuk. A Review of Emerging Research Directions in Abstract Visual Reasoning. *Information Fusion*, 2023.

Jacek Mańdziuk and Adam Żychowski. DeepIQ: A Human-Inspired AI System for Solving IQ Test Problems. In *IJCNN*, 2019.

Yutaka Matsuo, Yann LeCun, Maneesh Sahani, Doina Precup, David Silver, Masashi Sugiyama, Eiji Uchibe, and Jun Morimoto. Deep Learning, Reinforcement Learning, and World Models. *Neural Networks*, 152:267–275, 2022.

Laura E Matzen, Zachary O Benz, Kevin R Dixon, Jamie Posey, James K Kroger, and Ann E Speed. Recreating Raven's: Software for Systematically Generating Large Numbers of Raven-Like Matrix Problems With Normed Properties. *Behavior Research Methods*, 42(2):525–541, 2010.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A Benchmark for Language-Conditioned Policy Learning for Long-Horizon Robot Manipulation Tasks. *RA-L*, 2022.

Suvir Mirchandani, Fei Xia, Pete Florence, Brian Ichter, Danny Driess, Montserrat Gonzalez Arenas, Kanishka Rao, Dorsa Sadigh, and Andy Zeng. Large Language Models as General Pattern Machines. *arXiv:2307.04721*, 2023.

Melanie Mitchell, Alessandro B Palmarini, and Arseny Moskvichev. Comparing Humans, GPT-4, and GPT-4V On Abstraction and Reasoning Tasks. *arXiv:2311.09247*, 2023.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602*, 2013.

Arseny Moskvichev, Victor Vikram Odouard, and Melanie Mitchell. The ConceptARC benchmark: Evaluating Understanding and Generalization in the ARC Domain. *TMLR*, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training Language Models to Follow Instructions with Human Feedback. In *NeurIPS*, 2022.

Jaehyun Park, Jaegyun Im, Sanha Hwang, Mintaek Lim, Sabina Ualibekova, Sejin Kim, and Sundong Kim. Unraveling the ARC Puzzle: Mimicking Human Solutions with Object-Centric Decision Transformer. In *ICML Workshop on Interactive Learning with Implicit Human Feedback*, 2023.

Yonggang Qi, Kai Zhang, Aneeshan Sain, and Yi-Zhe Song. PQA: Perceptual Question Answering. In *CVPR*, pp. 12056–12064, 2021.

Christopher Reale and Rebecca Russell. Learning and Understanding a Disentangled Feature Representation for Hidden Parameters in Reinforcement Learning. *arXiv.2211.16315*, 2022.

Jerome Revaud, Philippe Weinzaepfel, César De Souza, Noe Pion, Gabriela Csurka, Yohann Cabon, and Martin Humenberger. R2D2: Repeatable and Reliable Detector and Descriptor. *arXiv:1906.06195*, 2019.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model. *Nature*, 2020.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization algorithms. *arXiv:1707.06347*, 2017.

Suyeon Shim, Dohyun Ko, Hosung Lee, Seokki Lee, Doyoon Song, Sanha Hwang, Sejin Kim, and Sundong Kim. O2ARC 3.0: A Platform for Solving and Creating ARC Tasks, 2024. URL https://o2arc.com.

Kimberly L Stachenfeld, Matthew M Botvinick, and Samuel J Gershman. The Hippocampus as a Predictive Map. *Nature Neuroscience*, 20(11):1643–1653, 2017.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. In *NeurIPS*, 2020.

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, 2023. URL https://github.com/Farama-Foundation/Gymnasium.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All You Need. *NeurIPS*, 30, 2017.

Karel Veldkamp, Hannes Rosenbusch, Luca Thoms, and Claire Stevenson. Solving ARC Visual Analogies with Neural Embeddings and Vector Arithmetic: A Generalized Method. *arXiv:2311.08083*, 2023.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft II: A New Challenge for Reinforcement Learning. *arXiv:1708.04782*, 2017.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster Level in StarCraft II using Multi-Agent Reinforcement Learning. *Nature*, 2019.

Ke Wang and Zhendong Su. Automatic Generation of Raven's Progressive Matrices. In *IJCAI*, 2015.

Taylor Webb, Zachary Dulberg, Steven Frankland, Alexander Petrov, Randall O'Reilly, and Jonathan Cohen. Learning Representations That Support Extrapolation. In *ICML*, 2020.

Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-Task Reinforcement Learning: A Hierarchical Bayesian Approach. In *ICML*, 2007.

Jonas Witt, Stef Rasing, Sebastijan Dumančić, Tias Guns, and Claus-Christian Carbon. A Divide-Align-Conquer Strategy for Program Synthesis. *arXiv:2301.03094*, 2023.

Yudong Xu, Elias B. Khalil, and Scott Sanner. Graphs, Constraints, and Search for the Abstraction and Reasoning Corpus. In *AAAI*, 2023a.

Yudong Xu, Wenhao Li, Pashootan Vaezipoor, Scott Sanner, and Elias B Khalil. LLMs and the Abstraction and Reasoning Corpus: Successes, Failures, and the Importance of Object-based Representations. *arXiv:2305.18354*, 2023b.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning. In *CoRL*, 2020.

Chi Zhang, Feng Gao, Baoxiong Jia, Yixin Zhu, and Song-Chun Zhu. RAVEN: A Dataset for Relational and Analogical Visual Reasoning. In *CVPR*, 2019.

David W. Zhang, Corrado Rainone, Markus Peschl, and Roberto Bondesan. Robust Scheduling with GFlowNets. In *ICLR*, 2023a.

Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron Courville, Yoshua Bengio, and Ling Pan. Let the Flows Tell: Solving Graph Combinatorial Optimization Problems with GFlowNets. In *NeurIPS*, 2023b.

Wenhe Zhang, Chi Zhang, Yixin Zhu, and Song-Chun Zhu. Machine Number Sense: A Dataset of Visual Arithmetic Problems for Abstract and Relational Reasoning. In *AAAI*, 2020.

A Appendix

A.1 Abstraction and Reasoning Corpus (ARC)

With increasing interest in human-like AI, attempts to measure intelligence are being made. However, how can intelligence be quantified? Previous research has defined intelligence as the ability to solve various types of problems with limited data and experience (Chollet, 2019a).

$$I^{opt}_{IS,\,scope} = \underset{T \in scope}{Avg}\,[\omega_{T,\Theta} \cdot \Theta \sum_{C \in Cur^{opt}_{T}} [P_C \cdot \frac{GD^{\Theta}_{IS,T,C}}{P_{IS,T} + E^{\Theta}_{IS,T,C}}]]$$

In the above equation, $\omega_{T,\Theta} \cdot \Theta$ represents the weights, C denotes a single curriculum (training data), P_C is the probability of the curriculum occurring, $GD^{\Theta}_{IS,T,C}$ signifies the generalization difficulty of solved problems, $P_{IS,T}$ represents prior knowledge, and $E^{\Theta}_{IS,T,C}$ denotes the model's experience. Note that intelligence gets bigger when prior knowledge and experience get smaller and generalization difficulty gets bigger, just as the definition of intelligence proposed above. Therefore, it was argued that benchmarks for evaluating intelligence must meet three criteria: (1) solvable with limited prior knowledge alone, (2) composed of diverse problem types, and (3) quantitatively measurable.

ARC is a benchmark proposed to quantitatively measure the intellectual capabilities of computers. Each task in ARC consists of 2–5 demo pairs with both inputs and outputs given, along with one test input grid. The goal is to infer the solution to the test input grid by deducing rules from the examples. Demo inputs and outputs can vary in size from a minimum of 1×1 grid to a maximum of 30×30 grid, with each grid capable of being colored with 10 different colors. One other property of ARC is that it is solvable with just four types of prior knowledge: objectness, goal-directedness, counting, and geometry and topology (Chollet, 2019a). Thus, ARC is considered a fair intelligence assessment scale because it requires **relatively simple rules and limited prior knowledge** to solve tasks, while also necessitating the inference of **various rules** and enabling **numerical evaluation** of whether a task can be solved or not.

One of the key features of ARC is its requirement for high levels of abstraction and reasoning compared to other benchmarks. Previous research comparing benchmarks for evaluating visual reasoning abilities noted that ARC stands out because it requires understanding abstract images and rules, evaluates by generating answers, and can present unseen tasks (Małkiński & Mańdziuk, 2023). Due to these characteristics, as of 2023, state-of-the-art models demonstrate approximately 30% accuracy (Johnson et al., 2021). When compared to the fact that human performance is around 80%, it becomes evident how challenging the benchmark is. This is why ARC gathers attention in the pursuit of human-like AI research.

Table 2: Alignment of Abstract Visual Reasoning tasks with its taxonomy (Małkiński & Mańdziuk, 2023). The problems and their corresponding benchmarks are cataloged under the following four dimensions of the taxonomy: Input shapes, Hidden rules, Target tasks, and Specific challenges.

Dataset	Geometric Shapes	Abstract Shapes	Explicit Rules	Abstract Rules	Classify	Generate	Domain Transfer	Extrapolate
ARC (Chollet, 2019a)		✓		✓		✓	✓	✓
Sandia (Matzen et al., 2010)	✓		✓		✓			
Synthetic (Wang & Su, 2015)	✓		✓		✓			
G-set (Mańdziuk & Żychowski, 2019)	✓		✓		✓			
RAVEN (Zhang et al., 2019)	✓		✓		✓		✓	
PGM (Barrett et al., 2018)	✓		✓		✓		✓	✓
Hill et al. (Hill et al., 2019)	✓		✓		✓		✓	✓
G1-set (Mańdziuk & Żychowski, 2019)	✓		✓		✓		✓	
S1-set (Mańdziuk & Żychowski, 2019)	✓		✓		✓		✓	
MNS (Zhang et al., 2020)	✓		✓		✓			
VAEC (Webb et al., 2020)	✓		✓		✓			✓
DOPT (Webb et al., 2020)	✓		✓		✓			✓

A.2 Significance of Solving ARC using Reinforcement Learning

Just as ARC has significant implications for the field of reinforcement learning (RL), RL also holds great importance for the ARC and Artificial General Intelligence (AGI) research areas. This is because RL methodology 1) has characteristics suitable for solving ARC compared to other approaches, 2) has high possibility of leading to research on human reasoning, which is the aim of ARC, and 3) facilitates the utilization of research resources that have been employed in other fields.

The Suitability of RL for Solving ARC RL possesses suitable characteristics for solving ARC. ARC can be understood as a program synthesis benchmark that involves composing complex solutions from simple skills (Chollet, 2019a). Previous attempts to solve ARC support this claim. Conventional deep learning approaches like autoencoders show performance below 10% (Veldkamp et al., 2023), as they are specialized in learning a single skill for solving a specific task. Similarly, large language models that have achieved success across various domains also exhibit around 15% performance (Mirchandani et al., 2023), likely due to their limitation in finding combinations of step-by-step skills. The highest performance has been achieved by program synthesis methods that search for combinations of human-designed Domain Specific Languages (DSLs) (Hodel, 2023). While these results support that ARC indeed requires program synthesis components, the current methods using manually designed DSLs are limited in their vulnerability to unseen tasks and human bias. On the other hand, RL is a research field specialized in solving complex problems by composing simple actions. Methods like MuZero (Schrittwieser et al., 2020) have successfully found effective action combinations in environments like the game of Go. The strong compositional capability makes RL more suitable for solving ARC than other approaches.

Potential for Expanding into Human Reasoning Research RL approach to solving ARC is expected to provide a significant foundation for theoretical inquiries into AGI, as RL methodologies are similar to how humans solve tasks. While there have been several attempts to solve ARC, existing methods differ from the strategies typically employed by intelligent agents for general problem-solving. Recent research methods, spearheaded by large language models, have been argued to diverge from human reasoning processes (Mitchell et al., 2023). In contrast, RL has consistently yielded findings suggesting its biological similarity to human reasoning processes (Matsuo et al., 2022; Stachenfeld et al., 2017). Additionally, the use of intuitive rewards and policies allows for a transparent examination and approach to tasks compared to other machine learning methods. This transparency in understanding how actual reasoning occurs is one of the crucial characteristics that enables research into the nature of reasoning. Therefore, research on ARC through RL will not only aim to solve the tasks but also present an opportunity to gain insights into how humans reason from a biological perspective.

Acquisition of diverse research resources ARC is a benchmark created in 2019, and its solution methods have not been deeply studied yet. Supporting RL environment that can solve ARC has the effect of easily introducing resources from other machine learning research fields into ARC. RL allows the application of deep learning models and techniques developed in other fields such as vision and natural language processing as its policy function, making it advantageous for utilizing existing resources. These characteristics will further facilitate the application of recently spotlighted methods such as meta-learning, continual learning, and multi-task learning to ARC. Consequently, rather than being limited to finding RL-specific solutions, it provides a great opportunity to test various methodologies and resources together.

Despite these advantages, efforts to tackle ARC tasks using RL have been limited, mainly due to a lack of appropriate RL environments. ARCLE maintains the inherent difficulty of ARC that requires solving general tasks with minimal prior knowledge and experience while preserving the strength of the RL approach in its similarity to human reasoning by incorporating the actions humans use when solving ARC. Furthermore, the action space consisting of low-level actions and the environment that allows for various variations offer high potential for utilization in meta-learning and continual learning research. Owing to these characteristics, ARCLE will not only contribute to the field of RL research but also make significant contributions to research on ARC and reasoning intelligence.

A.3 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 (Kim et al., 2022; Shim et al., 2024)[4], an improvement upon the initial testing web interface developed by Chollet (2019b). 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 (Park et al., 2023). 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 transformed O2ARC into a reinforcement learning environment for solving ARC tasks by agents. In Section A.6, we explore the possibility of using alternative RL algorithms to solve ARC tasks.

Figure 8 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. The detail about the actions of O2ARC and ARCLE is explained in Section A.4.

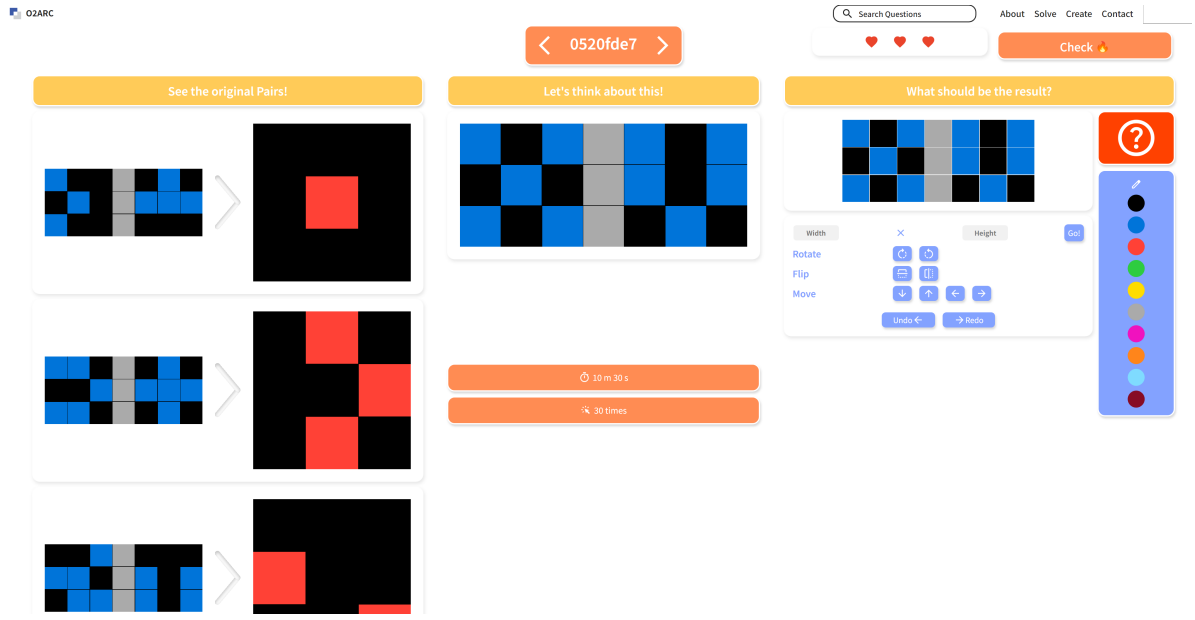


Figure 8: O2ARC Solve page. The left side shows demo input and output grid pairs, the center demonstrates the test input grid, and the right side consists of the result grid and available actions.

[4] https://o2arc.com

A.4 EVERY OPERATION INCLUDING ARCLE

Figure 9 shows all the `operations` currently present in ARCLE. As we mentioned before, the environments in ARCLE support only a subset of `operation`, tailored to their respective purposes. For instance, the simplest environment, `ARCRawEnv`, includes every Coloring `operation` and two Criticals (`ResizeGrid` and `Submit`).

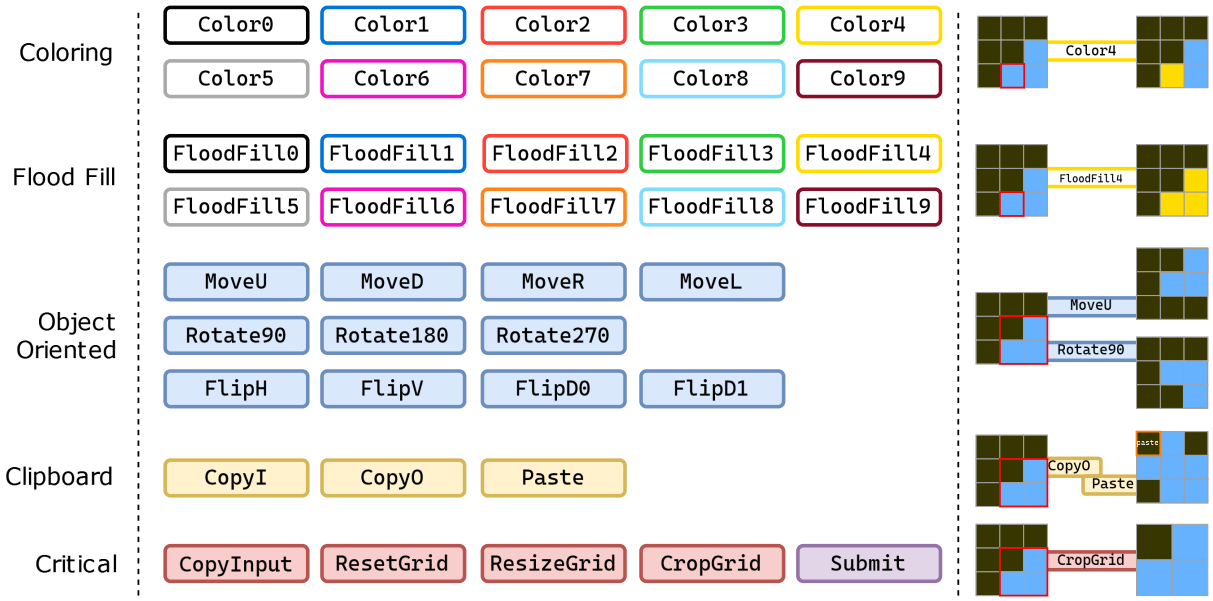


Figure 9: Every operations in ARCLE (version 0.2.5).

The operations in the ARCLE **actions** component are categorized into five groups: Coloring, Flood Fill, Object-Oriented, Clipboard, and Critical operations. **Coloring** operations (`Color0`–`Color9`) change the color of selected pixels specified in `selection`. **Flood Fill** (`FloodFill0`–`FloodFill9`) alters the color of multiple pixels sharing the same color of `selection` pixels, as depicted in the right side of Figure 9. Two groups each consist of ten operations corresponding to ten distinct colors, defined in the ARC.

Object-oriented operations refers to `Move`, `Rotate`, and `Flip` added to O2ARC, by regarding pixels within the grid as object(s). `MoveU`, `MoveD`, `MoveR`, and `MoveL` shifts selected pixels in the specified direction (up, down, right, left). `Rotate90`, `Rotate180`, `Rotate270` turn them by 90, 180, or 270 degrees counterclockwise respectively. `FlipH`, `FlipV`, `FlipD0`, `FlipD1` mirror object(s) horizontally, vertically, or diagonally (major and minor diagonal). If the `selection` is non-rectangular, object-oriented operations first calculate the bounding box of `selection` and operate in advance. Black pixels (zero-valued) are considered blank or transparent pixels, therefore those pixels do not affect other pixels by overlapping while performing sequences of object-oriented operations.

Clipboard operations involves the clipboard of the state, `clip`, copying from selected and pasting to the grid. `CopyI` copies pixels from test input grid (`input` in state space) specified by `selection` of action. On the other hand, `CopyO` copies pixels from the editing grid (`grid` of the state). `Paste` overlay the pixels of the `clip` state, on the editing grid (`grid` of the state). Pasting location is specified by the left-top corner of the bounding box of `selection` binary array.

Critical operations significantly modify the grid; every operation affects the editing grid by replacing the input grid (`CopyInput`), clearing its pixels to black (`ResetGrid`), changing the grid's size (`ResizeGrid`), and cropping the grid (`CropGrid`). `ResizeGrid` changes the grid's height and width as indices of bottom-right pixels of the `selection`. `CropGrid` directly puts selected pixels into `grid`, with resizing `grid` as a bounding box of selected pixels. Lastly, `Submit` operation, highlighted in purple, submits the current editing grid to compare with the answer of the assigned ARC task.

A.5 Two-Layer Mechanism of 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 10.

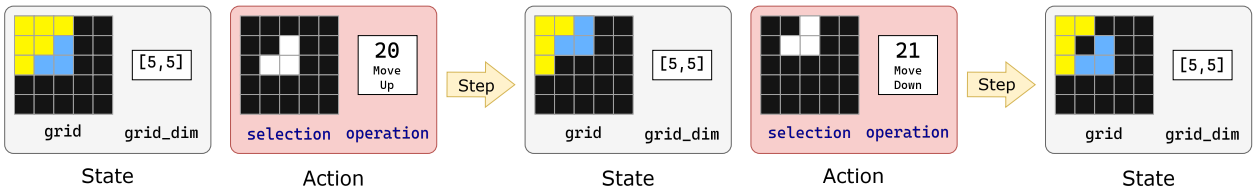


Figure 10: The edge case with serial simple `Move` actions.

Figure 10 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, we implemented ARCLE's object-oriented actions 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 11.

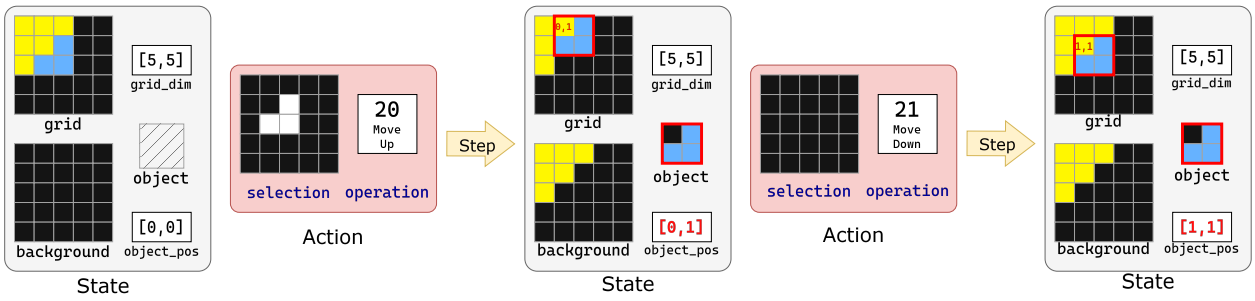


Figure 11: The edge case with serial two-layer mechanism `Move` actions.

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.

A.6 Other RL Algorithms Trials

For future convenience of application, we provide three types of reinforcement learning models using ARCLE. These include Meta-RL, which has shown success in solving various types of tasks, GFlowNet, which excels in exploring wide search spaces, and World Model, which has strengths in analyzing complex domains. Such applications demonstrate that ARCLE can be utilized not only for standard reinforcement learning algorithms that demand strict reward and action specifications.

A.6.1 Meta-RL Algorithms

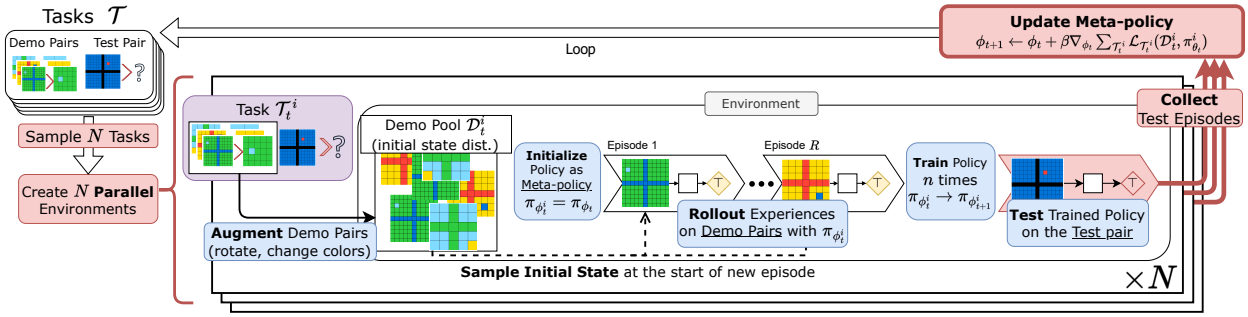


Figure 12: Learning Procedure of MAML integrated with ARCLE to solve ARC.

Model-Agnostic Meta-Learning We introduce the architecture of MAML (Finn et al., 2017), the most famous parameterized policy gradient (PPG) (Beck et al., 2023) meta-RL algorithm, integrated with ARCLE follows the training process as described in Figure 12. At time t, when a subset of the various tasks $(\mathcal{T}_1, \cdots, \mathcal{T}_N)$ stored in ARCLE is sampled, it generates augmented demo pairs $\mathcal{D}_t^i$ from the demo pair available for task $\mathcal{T}_t^i$. Each inner loop holds a policy π_{ϕ_t} created with the current initial parameter ϕ_t, and trains by sampling and rolling out augmented demos $(\mathcal{D}_t^i)$. The updated parameter ϕ_{t+1}^i during the training process is obtained as per the following Equation 1.

$$\phi_{t+1}^i = \phi_t - \alpha \nabla_{\phi_t} \mathcal{L}_{\mathcal{T}_t^i}(\pi_{\phi_t}) \tag{1}$$

Afterward, in the stage known as meta-testing, attempts are made to solve each task using the policy (π_{ϕ_t}) from each inner loop. The outcomes of these attempts are then utilized to update the parameters of the meta policy, by the Equation 2.

$$\phi_{t+1} = \phi_t - \beta \nabla_{\phi_t} \sum_{\mathcal{T}_t^i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_t^i}(\pi_{\phi_{t+1}^i}) \tag{2}$$

Expected Usage Meta-RL is a type of meta-learning where conventional RL is used in the inner loop. This approach allows us to view the ARC task from a meta-learning perspective, where the goal is to train models on tasks with minimal information and then evaluate them on new tasks. If we use models capable of effectively learning an individual ARC task in the inner loop, like the PPO-based model proposed in this study, meta-learning will enable these models to learn how to solve a variety of ARC tasks. Consequently, such trained models will also gain the ability to solve untrained tasks.

However, there are concerns that the inner loop of meta-RL may struggle to learn effectively due to ARC tasks being composed of very limited information (3–5 demo grid pairs and a test input grid). Additionally, the vast action space of ARC could also hinder learning. To address these issues, RL techniques such as offline meta-RL or hindsight experience replay might be necessary. Particularly for offline meta-RL, a buffer containing a large number of trial records might be required. Collecting such records through an interface like O2ARC or utilizing methods such as data augmentation could be potential solutions.

A.6.2 GENERATIVE FLOW NETWORK

Generative Flow Network Generative Flow Network (GFlowNet) is a kind of RL algorithm and generative model that can generate desirable trajectories (solutions) using the concept of flow networks (Bengio et al., 2021; 2023). In GFlowNet, there is a proportional relationship between reward and flow, allowing it to be trained with rewards. A reward is used to train the flow using specific loss function (Bengio et al., 2021; Malkin et al., 2022; Madan et al., 2023) to match flow. GFlowNet generates sequences of actions based on the actions defined in the environment, thereby reaching the terminal state. Through this process, it can generate various solutions with high rewards. The author said that GFlowNet is similar to the human recognition process in terms of stacking thoughts. Solving the ARC task needs human-like AI, thus, the concept of GFlowNet could be a suitable approach for addressing ARC.

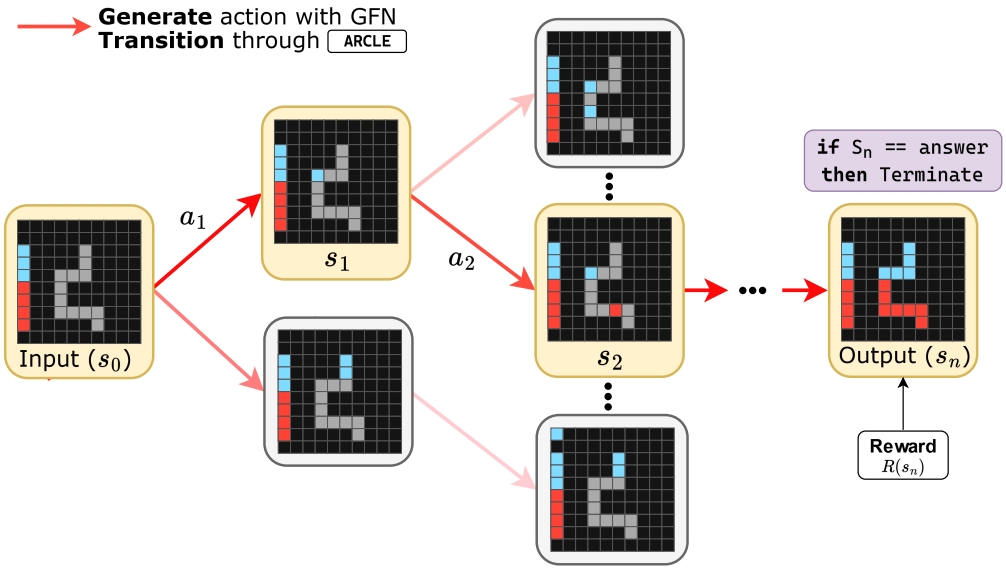
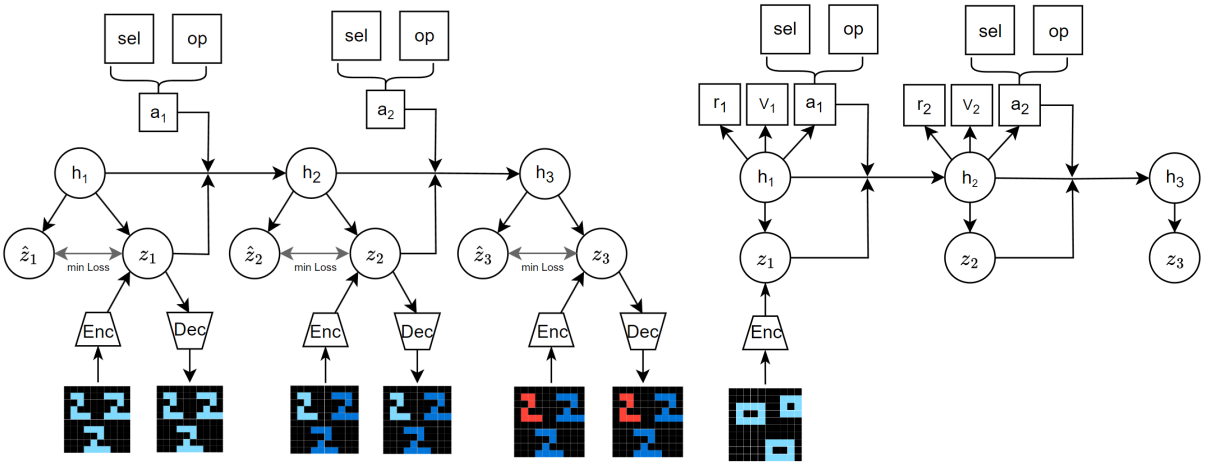


Figure 13: A simple architecture of GFlowNet interacting with ARCLE to find paths (solutions or modes). At each iteration, the GFN generates a succession of actions up to the terminal state, exploring the solution. The color saturation of the arrows indicates high and low flow (deeper color indicates higher flow). In our experiments, we were able to guarantee the DAG by coloring the actions one pixel at a time, and we colored the most desirable solution yellow to show that the answer can be found this way.

Expected Usage ARCLE operates within a discrete action space. Concurrently, GFlowNet, by learning the distribution of actions via a policy network, facilitates the generation of actions while transferring the responsibility for generating next states to ARCLE. This approach enables a transition-based training methodology for generating action sequences. A crucial element of training GFlowNet is the establishment of a Directed Acyclic Graph (DAG) structure, essential for significantly reducing the search space and enhancing learning efficiency. Applying ARCLE to GFlowNet in its basic form could lead to the creation of cycles, thus failing to guarantee DAG structure. A practical method for ensuring a DAG structure involves sequentially coloring one pixel at a time, achievable through a specialized subclass of ARCLE focused on such actions. Considering a $H \times W$ grid size, this leads to a theoretical maximum search space of $10^{H \times W}$. Given GFlowNet's proven efficacy in navigating extensive search spaces in graph combinatorial optimization challenges, (Zhang et al., 2023b), it is expected to outperform other competing algorithms.

The proposed approach aims to construct a DAG to investigate potential solutions, though this methodology mirrors human cognitive processes remains uncertain. Furthermore, even if this method could guarantee DAG structure, there still remains a vast search space of $10^{n \times n}$. Since this strategy yields only one solution, it is harder to find an optimal path in contrast to other applications of GFlowNet where solutions encompass a spectrum of possibilities across various paths (Jain et al., 2022; 2023; Zhang et al., 2023a;b). Therefore, for GFlowNet's effective application in solving ARCLE challenges, it is imperative to not only fully utilize ARCLE's actions but also to devise a strategy for DAG construction. Finding a proper way to build a DAG structure could reduce search space significantly and guarantee training effectively.

A.6.3 WORLD MODEL



(a) An World Model Architecture (b) An Actor Critic Architecture

Figure 14: An architecture of DreamerV3 (Hafner et al., 2023), an advanced version of World Model, solving ARC.

World Model World Model (Ha & Schmidhuber, 2018) is a model-based reinforcement learning algorithm designed to predict models in complex domains where it's difficult to create state spaces and transition functions. To achieve this, the World Model consists of three parts: vision, memory, and controller. Vision is responsible for generating a latent state from given visual images. For instance, in games like Minecraft, where 3D spatial information, items, and various in-game statuses are included in the image, vision transforms it into a vector representing the current state. Memory stores information to predict the next state using the latent state provided by vision. It's known to store prior knowledge inherent to the environment, such as gravity or friction, in vector form. After the learning process, vision and memory function as a kind of model that predicts states and transitions. The controller part is responsible for generating appropriate actions using the given model.

$$\phi_{t+1} = \phi_t - \alpha \nabla_{\phi_t} \mathcal{L}_{dynamics}(p_{\phi_t}(z_t|x_t)), q_{\phi_t}(z_t|h_t)) \tag{3}$$

In the above equation, $p_{\phi_t}(z_t|x_t)$ represents vision function which makes latent state z_t given input image x_t, whereas $q_{\phi_t}(z_t|h_t)$ represents memory function which makes latent state z_t given transition prediction h_t. Subsequently, updated vision and memory are utilized to train the controller using the latent state z_t and transition prediction h_t provided by each, as follows.

$$\theta_{t+1} = \theta_t - \beta \nabla_{\theta_t} \mathcal{L}_{controller}(\pi_{\theta_t}) \tag{4}$$

Expected Usage The lack of knowledge about which information to extract from input and output grids, as well as the necessary prior knowledge for solving specific types of tasks, is a significant challenge in solving ARC. The Vision part of the World Model excels at extracting important information from visual inputs, while the Memory part is strong at extracting prior knowledge. Therefore, in future ARC research, the architecture of the World Model is expected to play a crucial role. To facilitate future research efforts, this study provides a simple implementation of DreamerV3(Hafner et al., 2023), a version of the World Model, applied to ARC.

Furthermore, existing research on World Models has predominantly focused on scenarios where a single agent performs a single task. However, to address the ARC problem, which involves a diverse array of tasks, it might be necessary to consider approaches that incorporate pretraining techniques or apply meta-learning methods. These strategies could potentially enable an agent to adapt to and perform multiple tasks by leveraging prior knowledge or learning how to learn across different tasks.