Neural MCTS with LLM Guidance for Effective Program Synthesis on Abstraction and Reasoning Corpus

Jinwoo Jeon[1], Seongwoong Shim[1], Sejin Kim[2*], Sungdong Kim[2*], Byung-Jun Lee[1*]

[1]Artificial Intelligence, Korea University, Anam-ro 145, Seoul, 02841, Seongbuk-gu, Republic of Korea.
[2]Artificial Intelligence, Gwangju Institute of Science and Technology, 123 Cheomdangwagi-ro (Oryong-dong), Buk-gu, Gwangju, 61005, Republic of Korea.

*Corresponding author(s). E-mail(s): sejinkim@gist.ac.kr; sundong@gist.ac.kr; byungjunlee@korea.ac.kr;
Contributing authors: kevin04087@korea.ac.kr; ssw030830@korea.ac.kr;

Abstract

The Abstraction and Reasoning Corpus (ARC) has become a prominent benchmark for assessing whether AI systems can perform human-like reasoning. Recent efforts on ARC have increasingly adopted program synthesis paradigms based on domain-specific languages (DSL). To tackle the combinatorial challenges of program synthesis, leading methods employ pretrained models to produce DSL program sequences and apply efficient search algorithms during inference. Monte Carlo Tree Search (MCTS) stands out among these, as it enables effective navigation of vast search spaces while maintaining a principled equilibrium between exploration and exploitation. In this work, we introduce a neural MCTS algorithm customized for ARC-AGI, which amplifies exploration by directing the search toward promising solutions via learned neural guidance, yielding robust performance on demanding ARC tasks. However, pretrained models frequently underperform on certain tasks owing to insufficient exposure to particular DSL tokens or structures during pretraining. To mitigate this, we additionally propose an LLM-guided technique, in which an LLM recommends pertinent tokens to prune the search space, thereby allowing the solver to prioritize previously overlooked DSL elements. Integrating the o4-mini model with our approach, we successfully solve nearly all DSL-solvable tasks in the ARC-AGI-1 evaluation

benchmark, highlighting the effectiveness of our neural MCTS framework guided by LLM-based token recommendations.

Keywords: Program Synthesis, Abstraction and Reasoning Corpus, Neural Search

1 Introduction

Human-like reasoning represents one of the most challenging frontiers in artificial intelligence research. The Abstract Reasoning Corpus (ARC; Chollet 1) benchmark addresses this challenge by providing a systematic framework for evaluating machine reasoning capabilities that mirror human cognitive processes. The ARC-AGI-1 dataset contains 1,000 tasks that challenge systems to infer abstract transformation rules from a few demonstrated input-output pairs, then apply these rules to previously unseen test instances. Figure 1 presents examples of ARC reasoning tasks, each showing the input-output training pairs and test cases. The computational constraints inherent in ARC—where solutions must be found within strict time and memory limits—highlight the need for efficient representation and search methods. This combinatorial challenge has motivated the development of Domain-Specific Languages (DSL) that can systematically explore the space of possible transformation rules.

Early work on synthesis of DSL-based programs for ARC relied mainly on heuristic search methods [2, 3]. These methods focus on applying the DSL to object-centric tasks and use various constraints during the search to reduce the search space. However, they exhibit significant performance degradation on ARC-evaluation tasks that require beyond the limited scope of object-centric training tasks [4]. Recently, neural network are trained to genearte DSL program sequences, and find correct solutions by using efficient search methods [4, 5]. However, these methods are heavily dependent on neural DSL generators, which excel at solving tasks akin to those encountered during training but typically falter on markedly distinct, novel tasks [5].

To address the limitations of prior work, we propose a neural Monte Carlo Tree Search (MCTS) algorithm, which allows effective exploration in the vast space of DSL programs. Our approach is specifically designed for the ARC-AGI benchmark. Concretely, we incorporate a lightweight value network to enhance search efficiency. During the search, partial programs are evaluated, their returns computed, and the resulting pairs stored in a replay buffer. The value network is trained on these partial program-return pairs via expectile regression, enabling efficient off-policy value function estimation with implicit policy improvement over diverse partial programs. Although our method is applicable only to environments with deterministic transitions, such as ARC-AGI, it enables robust value function training, providing direct supervision to all partial programs and effective credit assignment in sparse-reward settings. Empirically, our approach achieves a 70% relative improvement compared to a baseline that assigns value of the programs based on the pixel-level similarity between their output grid and target grid, instead of estimating with value network.

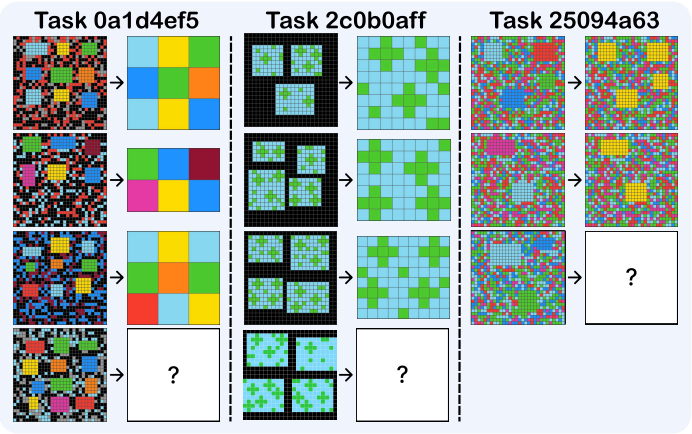


Fig. 1: The ARC benchmark furnishes a small set of training examples to infer an underlying transformation rule, which is then evaluated on a test example. This figure depicts three tasks from an ARC-AGI-1 evaluation set, all unsolved by prior approaches, yet correctly resolved by our method.

To further enhance search efficiency, we leverage large language models (LLMs) such as GPT-4.1 and o4-mini to guide DSL token selections within MCTS. Concretely, we prompt these LLMs to generate solutions directly for ARC tasks based on input/output grids and extract plausible DSL candidate tokens from their reasoning chains. While the generated solutions are frequently incomplete or erroneous, the underlying high-level reasoning often suffices to identify appropriate DSL tokens. These LLM-suggested tokens are subsequently merged with candidates derived from neural MCTS without LLM assistance, and the aggregated pool is further refined through subsequent MCTS iterations. This hybrid methodology demonstrates strong generalization to novel tasks that diverge from those seen during the DSL sampler's pre-training, all without requiring additional training. In summary, our contributions are threefold:

- We introduce a neural MCTS algorithm that is highly effective in deterministic environments, specifically tailored for the ARC-AGI benchmark.
- We propose an LLM-based DSL recommendation technique that substantially boosts MCTS performance.
- We attain 95% near-perfect results, solving DSL-solvable tasks on the ARC-AGI-1 evaluation set, using our method with o4-mini.

2 Related Work

2.1 Heuristic based program synthesis

ARGA [2] adopts an object-centric perspective, representing input images as abstract graphs and searching for valid programs within a DSL specifically tailored for these graph representations. The challenge of combinatorial search complexity is addressed

by applying constraint acquisition, state hashing, and Tabu search. GPAR [3] addresses the challenges of the ARC benchmark by formulating each task as a generalized planning problem in Planning Domain Definition Language(PDDL), enhanced with object-centric functions and ARC-specific constraints. However, this approach is inherently limited to object-centric tasks. ARGA solves 49 problems in the ARC-AGI-1 training set, but only 10 problems in the more challenging ARC-AGI-1 evaluation set, highlighting a significant performance gap.

2.2 Neural networks for program synthesis

As an alternative, recent studies have explored search methods that leverage neural network models trained to learn DSLs. Bidir-Synth [6] initially trains the model on randomly generated programs and subsequently fine-tunes it during the search phase using the REINFORCE [7] algorithm. To further reduce search time, the method incorporates bidirectional search. Bober-Irizar and Banerjee [4] accelerate program search on ARC by adapting DreamCoder [8] with a wake-sleep cycle, where the wake phase searches for candidate programs and the sleep phase refines the DSL and search strategy using a convolutional GrammarNet. GridCoder [5], the current state-of-the-art approach, notes that relying solely on randomly generated tasks for training can result in incoherent and unnatural tasks. Accordingly, instead of relying on random tasks, this method employs purposefully designed tasks with explicit objectives—such as tiling or split-merge operations—to provide structured supervision for the model. During inference, it employs transformer-based probability prediction over DSL tokens and utilizes bootstrapping to obtain robust distributions for program enumeration.

2.3 LLM based method

Xu et al. [9] explored prompt design strategies for large language models (LLMs) on ARC tasks, and found that using object representation solvers such as ARGA yields substantial gains in LLM performance. Bober-Irizar and Banerjee [4] demonstrated that ensembling their approach with GPT-4.0 results in substantial performance gains, noting that the set of problems solvable by the LLM and their own model are complementary. ConceptSearch [10] samples candidate programs using closed source LLM API and selects the best one based on scores from Hamming distance, CNN features, or LLM-based evaluation. Recently, methods have been introduced that apply test-time training (TTT) [11–13], which temporarily fine-tunes LLMs during inference using losses derived from input data. On the ARC-AGI-1 benchmark, TTT significantly improves accuracy, However, this approach comes with significant computational costs, due to the need for massive data augmentation and the overhead of finetuning the model itself.

3 Problem Settings

In this study, we address ARC-AGI-1 problems, represented as a collection of tasks, each comprising a few-shot training set and a test pair. Each problem's training data is denoted by $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^{n}$, where x_i is an input grid, y_i is its corresponding

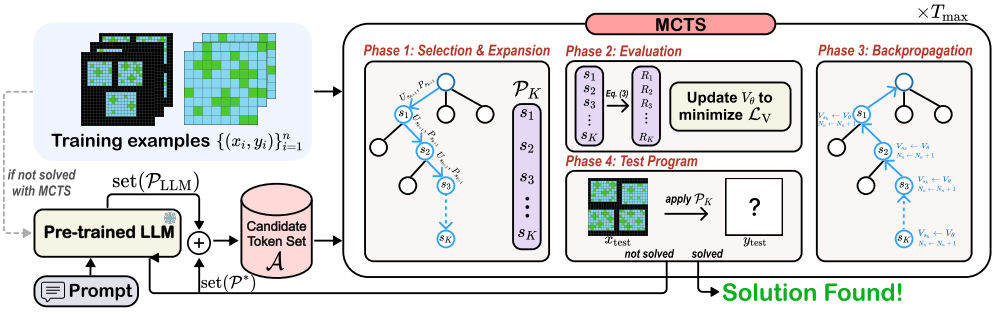


Fig. 2: **Overview of our method.** The neural MCTS iteratively expands the search tree through selection, expansion, evaluation, and backpropagation phases to identify a program that precisely maps training inputs to their outputs. If no solution is found, the DSL tokens from the highest-reward synthesized program are merged with those from the LLM-generated solution, and MCTS is re-executed over this consolidated, reduced DSL set.

output grid, and n represents the number of training examples. For a given test input x_{test}, the objective is to generate a program that accurately predicts the output y_{test}.

We define the DSL as a set of statements $\{s_i\}$. We adopt the DSL introduced by Ouellette [5], which adapts ARC-DSL [14] to be expressed as a syntax tree that can be represented as a linear sequence. This DSL comprises 99 standard tokens and two special tokens: `<NEW LEVEL>` and `<EOS>`. The `<NEW LEVEL>` token denotes the start of a new level within the DSL syntax tree, while `<EOS>` signifies the end of a sequence. A valid sequence of tokens $[s_1, \ldots, s_K]$, concluded with `<EOS>`, forms a program that transforms an input grid according to predefined rules. A problem instance is deemed solved if this program accurately transforms x_{test} into y_{test}.

4 Method

4.1 Neural MCTS for ARC-AGI

We implement Monte-Carlo Tree Search (MCTS) using the practical techniques from MuZero [15], comprising four key phases: *selection*, *expansion*, *evaluation*, and *backpropagation*. The search is guided by two neural models: **a pretrained DSL sampling prior** P, which remains fixed during training, and **a value network** V_θ, which is updated during the search to assess partial programs. During the selection phase, the algorithm navigates internal nodes, selecting the next DSL token (child node) to extend the current program until an `<eos>` token is encountered. If a leaf node is reached without program completion, the expansion phase adds valid DSL tokens as new child nodes. These phases alternate until a valid program is formed. Upon program completion, the evaluation phase updates the value network. In the backpropagation phase, value estimates are propagated up the tree to improve future selections. Our approach is outlined in Figure 2 and Algorithm 1, with details provided below.

4.2 Selection

In the selection phase, the algorithm chooses the child node s_{k+1} with the highest selection score to extend the current partial program $\mathcal{P}_k = [s_1, \ldots, s_k]$. The selection score $U_{s_{k+1}}$ for each child node s_{k+1} is calculated as:

$$U_{s_{k+1}} = \begin{cases} P_{s_{k+1}} \left(1 - \dfrac{N_{s_{k+1}}}{N_{s_k}}\right) & \text{with prob. } \epsilon \\ V_{s_{k+1}} & \text{with prob. } 1 - \epsilon \end{cases} \tag{1}$$

where $P_{s_{k+1}} = P(s_{k+1} \mid \mathcal{P}_k, x_i, y_i)$ is the probability of selecting node s_{k+1} given its ancestors, estimated by a pretrained model and stored during the expansion phase. Here, $N_{s_{k+1}}$ and N_{s_k} denote the visit counts of the child node and its parent, respectively, while $V_{s_{k+1}}$ represents the value estimate of the partial program $\mathcal{P}_k$, updated and stored during the backpropagation phase.

To balance exploration and exploitation, we utilize two distinct selection strategies in an ϵ-greedy manner. With probability ϵ, the algorithm selects a node exploratively; otherwise, it selects exploitatively. In the exploratory case, the selection prioritizes nodes with high model probabilities $P_{s_{k+1}}$ that have been visited less frequently, guiding the algorithm toward promising yet under-explored regions of the search space. In the exploitative case, the algorithm chooses the child node with the highest current value estimate $V_{s_{k+1}}$, focusing on regions previously identified as valuable based on prior evaluations.

4.3 Expansion

Through iterative child node selection, we obtain a partial program sequence $\mathcal{P}_k = [s_1, \ldots, s_k]$, where s_k is a leaf node. We then calculate the probability distribution for the next token, $P(s_{k+1} \mid \mathcal{P}_k, x_i, y_i)$, to transform x_i toward y_i. For each syntactically valid DSL statement s_{k+1} that continues the sequence, a new child node is created. The probability of each new node, $P_{s_{k+1}}$, is stored for use in the selection phase. This selection–expansion cycle continues until a terminal node is reached or a predefined maximum depth is achieved.

4.4 Evaluation

Upon completing the iterative selection-expansion process, yielding a full program sequence $\mathcal{P}_K = [s_1, \ldots, s_K]$, we compute the reward for the program as:

$$r(\mathcal{P}_K, \mathcal{T}) = \frac{1}{n} \sum_{i=1}^{n} \text{sim}(\hat{y}_i, y_i), \tag{2}$$

where $\hat{y}_i$ is the output grid produced by applying the full program sequence to x_i, and $\text{sim}(\cdot, \cdot)$ represents pixel-wise similarity. Thus, the reward reflects how closely the output grid matches the target grid. Since the reward is only received upon program

completion, the discounted return for each partial program is calculated as:

$$R_k = \gamma^{K-k} r(\mathcal{P}_K, \mathcal{T}), \tag{3}$$

where $\gamma \in (0, 1]$ is a discount factor. Afterwards, all partial program sequences and their corresponding returns are stored in a replay buffer $\mathcal{B}$:

$$\mathcal{B} \leftarrow \mathcal{B} \cup \{ (\mathcal{P}_k, R_k) \}_{k=1}^{K}.$$

A mini-batch of M program-return pairs is then randomly sampled from $\mathcal{B}$ to update the value function by minimizing the expectile loss:

$$\mathcal{L}_V = \mathbb{E}_{(\mathcal{P}_k, R_k) \sim \mathcal{B}} \left[L_2^\tau \left(V_\theta(\mathcal{P}_k) - R_k \right) \right], \tag{4}$$

where $L_2^\tau(u) = |\tau - \mathbb{I}(u < 0)|u^2$. With $\tau > 0.5$, this asymmetric loss function reduces the influence of smaller returns R_k, enabling V_θ to better approximate larger returns, corresponding to promising program extensions [16, 17].

Importantly, this efficient off-policy approach to training the value function is feasible solely under deterministic transitions; in stochastic environments, values could be overestimated owing to the optimistic choice of transition samples. This attribute of the return-based expectile loss renders it especially appropriate for DSL search problems, which inherently involve deterministic transitions.

4.5 Backpropagation.

After updating the value network, each node n along the path is updated as follows:

$$N_{s_k} \leftarrow N_{s_k} + 1, \quad V_{s_k} \leftarrow V_\theta(\mathcal{P}_k), \tag{5}$$

for $k = K, \ldots, 1$. If the complete program sequence $\mathcal{P}_K = [s_1, \ldots, s_K]$ successfully transforms x_i into y_i for $i = 1, \ldots, n$, it is accepted as a solution; otherwise, the process is repeated.

4.6 LLM Guidance to Reduce Search Space

While neural guidance using a pretrained DSL sampler and a value network enables effective exploration, the search space remains vast, and these small models may favor incorrect DSL tokens for complex tasks. In contrast, Large Language Models (LLMs) are recognized for their robust reasoning capabilities, making them a key approach for ARC tasks [4, 9, 11].

However, despite their strength in high-level reasoning, LLMs often err in low-level decisions, frequently leading to incorrect solutions unless a prohibitively large LLM is applied. To address this, we propose leveraging LLMs to recommend suitable DSL tokens for the current task based on their high-level reasoning. LLMs can identify DSL tokens overlooked by the pretrained model. Although LLM-generated programs are frequently incorrect, the tokens they use are often plausible for a correct program. Incorporating these tokens into MCTS can significantly reduce the search space.

Algorithm 1 LLM-guided neural MCTS

Input: train examples $\{(x_i, y_i)\}_{i=1}^{n}$; test pair $(x_{\text{test}}, y_{\text{test}})$
Output: sequence of DSL tokens $\mathcal{P}_K = [s_1, \ldots, s_K]$

1: Set the candidate token set to the complete token set $\mathcal{A} \leftarrow \mathcal{D}$
2: Set best program so far: $\mathcal{P}^* \leftarrow [\,]$
3: Initialize search tree
4: Pick training example (x_i, y_i) to build tree
5: **for** MCTS iterations $t = 1$ to $T_{\max}$ **do**
6: $\quad \mathcal{P}_0 \leftarrow [\,]$
7: $\quad$ **for** $k = 0$ to $K_{\max}$ **do**
8: $\quad\quad$ **if** $s_k =$ `<EOS>` **then break**
9: $\quad\quad$ **if** s_k is a leaf **then**
10: $\quad\quad\quad$ ***Expansion***: add valid next tokens $s_{k+1} \in \mathcal{A}$ with $P(s_{k+1} \mid \mathcal{P}_k, x_i, y_i) > 0$ to tree
11: $\quad\quad$ **end if**
12: $\quad\quad$ ***Selection***: set the next node maximizing Eq. (1),

$$s_{k+1} = \arg\max U_{s_{k+1}}$$

13: $\quad\quad \mathcal{P}_{k+1} \leftarrow [\mathcal{P}_k; s_{k+1}]$
14: $\quad$ **end for**
15: $\quad K \leftarrow k$
16: $\quad$ ***Evaluation***: compute the reward $r(\mathcal{P}_K, \mathcal{T})$ and returns $\{R_k\}_{k=1}^{K}$ according to Eq. (2-3)
17: $\quad \mathcal{B} \leftarrow \mathcal{B} \cup \{(\mathcal{P}_k, R_k)\}_{k=1}^{K}$
18: $\quad$ **if** $r(\mathcal{P}_K, \mathcal{T}) > r(\mathcal{P}^*, \mathcal{T})$ **then** $\mathcal{P}^* \leftarrow \mathcal{P}_K$
19: $\quad$ Update value function minimizing Eq. (4)
20: $\quad$ ***Backpropagation***: update visited count and value (Eq. (5))
21: $\quad$ **if** $\mathcal{P}_K$ yields y_i from x_i for $i = 1, \ldots n$ **then**
22: $\quad\quad$ **return** $\mathcal{P}_K$
23: $\quad$ **end if**
24: **end for**
25: **if** $\mathcal{A} = \mathcal{D}$ **then**
26: $\quad$ # Not solved, reduce search space with LLM.
27: $\quad$ Suggest a new program using LLM:

$$\mathcal{P}_{\text{LLM}} = \text{LLM}(\mathcal{P}^*, \mathcal{T}, \mathcal{D})$$

28: $\quad$ Update the candidate token set:

$$\mathcal{A} = \text{set}(\mathcal{P}^*) \cup \text{set}(\mathcal{P}_{\text{LLM}})$$

29: $\quad$ **goto** line 3
30: **end if**
31: **return** $\mathcal{P}^*$

Specifically, if no valid solution is found after a predefined number of MCTS iterations, we create a new candidate DSL set for MCTS by combining the DSL tokens from the program with the highest return so far, $\text{set}(\mathcal{P}^*) \subseteq \mathcal{D}$, and the DSL tokens

Method	Accuracy
DreamCoder [4]	5/49
GridCoder [5]	33/49
Ours (neural MCTS)	**34/49**
GPT-4.1	5/49
DreamCoder + GPT-4.1	9/49
Conceptsearch [10]	17/49
GridCoder + GPT-4.1	36/49
Ours + GPT-4.1 (LLM-guided)	**39/49**
o4-mini	42/49
DreamCoder + o4-mini	42/49
GridCoder + o4-mini	44/49
Ours + o4-mini (LLM-guided)	**47/49**

Discount factor (γ)	Accuracy
0.0	28/49
0.50	31/49
0.70	34/49
0.90	34/49

Table 1: (**Left**) Comparison of our method with baselines, grouped by the LLM used in each algorithm. (**Right**) Accuracy for different discount factors.

from the LLM-generated solution, $\text{set}(\mathcal{P}_{\text{LLM}}) \subseteq \mathcal{D}$, forming $\mathcal{A} = \text{set}(\mathcal{P}^*) \cup \text{set}(\mathcal{P}_{\text{LLM}})$. We then rerun MCTS using the reduced candidate set $\mathcal{A}$, which is much smaller than the full DSL set $\mathcal{D}$.

5 Experiment

To evaluate the effectiveness of our method, we perform experiments guided by key research questions: **RQ1.** How does our method's performance compare to other baselines? **RQ2.** What is the contribution of each component in our proposed method to its overall performance? **RQ3.** How does using an LLM to reduce the search space impact MCTS performance across iterations? **RQ4.** How cost-efficient is the LLM-guided Neural MCTS compared to existing baselines? **RQ5.** How robust is our value learning framework under stochastic execution dynamics?

5.1 Experiment Settings

As a baseline for neural network-based program synthesis, we use DreamCoder [4] and GridCoder [5], recent state-of-the-art models designed for ARC-AGI-1 tasks. We additionally selected conceptsearch [10], a state-of-the-art code generation method based on API call for LLM guidance baseline.

We evaluate all methods on a set of 49 ARC-AGI-1 evaluation tasks. These benchmark tasks were selected to ensure methodological comparability rather than based on specific task-level characteristics.

Our approach relies on a DSL-pretrained model released by GridCoder, which currently represents the state of the art in DSL-based ARC program synthesis. Accordingly, we adopt the same set of 49 evaluation tasks used in the GridCoder preprint, enabling a direct and fair comparison under identical DSL and evaluation conditions.

Although the number of tasks is smaller than the full ARC-AGI-1 benchmark, this subset constitutes a meaningful and controlled evaluation setting for assessing neural

program synthesis methods, as all tasks are fully solvable within the shared DSL used by both the baselines and our method.[1]

For the neural networks in our neural MCTS, we adapt the vision-language model (VLM) from Ouellette [5], which takes the input grid, output grid, and current program sequence as inputs, and autoregressively predicts the next DSL token. The value network, a lightweight transformer encoder, processes embedded DSL program sequences independently of the pretrained model. We apply mean pooling to the outputs and generate a scalar value in $(0, 1)$ using a linear output head with sigmoid activation.

For LLM guidance, we employed GPT-4.1 and o4-mini to assess the impact of LLMs with varying reasoning capabilities. As baselines, we first evaluated each LLM using a text-based grid format, adhering to the prompt template from the official ARC AGI Benchmarking repository.[2] We also created additional baselines by ensembling each LLM with program synthesis baselines, reporting results to ensure a fair comparison with our proposed LLM-guided approach. For each task, only one LLM API call was made. In the case of conceptsearch, the original implementation used the Gemini 1.5 API; however, we employed GPT-4.1 instead to ensure consistency across all baselines. In the original ConceptSearch implementation, evaluation is primarily conducted on a set of 50 ARC training tasks followed by [9]. our focus is instead on the ARC-AGI-1 evaluation split. Accordingly, we evaluate all methods on a set of 49 ARC-AGI-1 evaluation tasks, which provides a more meaningful and standardized benchmark for assessing generalization and comparative performance across methods.

We assessed accuracy three times, adhering to the protocol outlined by Bober-Irizar and Banerjee [4]. For ensembling with LLMs and baselines, we assigned lower weights to the three candidate solutions generated solely by the LLM and higher weights to those from baselines, as the latter only submit solutions when a program correctly solves all training instances. Solutions were placed into a priority queue based on their weights, and the top three candidates were selected. A task was deemed solved if at least one of these three candidates provided the correct answers.

For detailed training settings, hyperparameter configurations, and the prompt template used for LLM guidance, please see the Appendix B.

5.2 How does our method's performance compare to other baselines?

Table 1 (left) presents the performance of our method and baselines on the 49 selected tasks. When evaluated using neural MCTS alone, our approach slightly surpasses GridCoder. Specifically, we found that our model solves all tasks handled by GridCoder and additionally resolves the object-selector task (`2c0b0aff`), which involves counting 4-connected objects within each rectangular object. GridCoder generates candidate programs through bootstrapping, selecting the sequence with the highest joint token probability. As a result, DSL tokens with low probabilities (e.g., `<count>` with a probability of 0.05), though critical, are often omitted. In contrast, our explicit exploration strategy identifies solutions that incorporate these essential but low-probability tokens (See Figure 3).

[1] The list of selected tasks is available at https://github.com/SimonOuellette35/GridCoder2024.
[2] https://github.com/arcprize/arc-agi-benchmarking

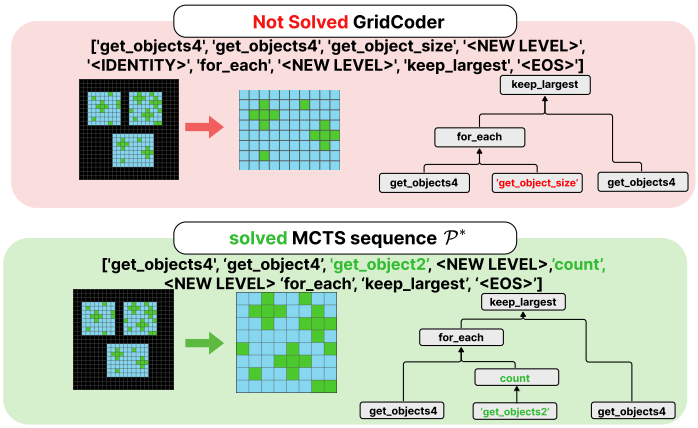


Fig. 3: Neural MCTS achieves successful resolution of task 2c0b0aff, which remains unsolved by GridCoder due to the absence of essential DSL primitives such as `count` and `get_object2`.

For LLM-only baselines, GPT-4.1 solves 5 of the 49 tasks, while o4-mini successfully solves 42, demonstrating superior performance. Ensembling our LLM-guided approach with GPT-4.1 results in solving 39 tasks, compared to 36 tasks when ensembling GPT-4.1 with GridCoder. Notably, our ensemble approach solves five additional tasks beyond the 34 achieved by the value-guided method. Of these, three tasks are solved exclusively by GPT-4.1, while our LLM-guided method uniquely solves two challenging tasks (`25094a63`, `0a1d4ef5`). Although o4-mini performs strongly, it leaves seven tasks unsolved. Our method addresses five of these: two tasks (`1a6449f1`, `1c56ad9f`) via neural MCTS, and three tasks (`25094a63`, `0a1d4ef5`, `d56f2372`) exclusively through LLM guidance. In total, our approach solves 47 out of 49 tasks, achieving near-perfect performance on the benchmark.

While we acknowledge that reasoning-oriented LLMs already achieve strong performance—o4-mini with simple prompting solves 42 out of 49 tasks as shown in Table 1. Our Neural MCTS–only search nevertheless solves 34 out of 49 tasks without invoking any LLM API calls, already marginally outperforming GridCoder.

When combined with LLM guidance, Neural MCTS enables the system to solve three additional tasks(`25094a63`, `0a1d4ef5`, `d56f2372`) that remain unsolved by o4-mini prompting alone, demonstrating a clear synergistic effect. These results show that Neural MCTS plays a complementary and essential role in program synthesis.

5.3 What is the contribution of each component in our proposed method to its overall performance?

Table 2 illustrates the impact of using a value function and its training approach on overall performance. Without value network training, relying solely on pixel-wise similarity, the model solves only 20 out of 49 tasks due to limited utilization of accumulated information from prior MCTS iterations not utilizing replay buffer. Training the value network on full program sequences increases performance to 28 tasks, but limits

Method	Accuracy
Train value on partial programs (**proposed**)	**34/49**
Train value on complete programs only	28/49
Use pixel-wise similarity directly (w/o value)	20/49

Table 2: Ablation analysis of removing each method.

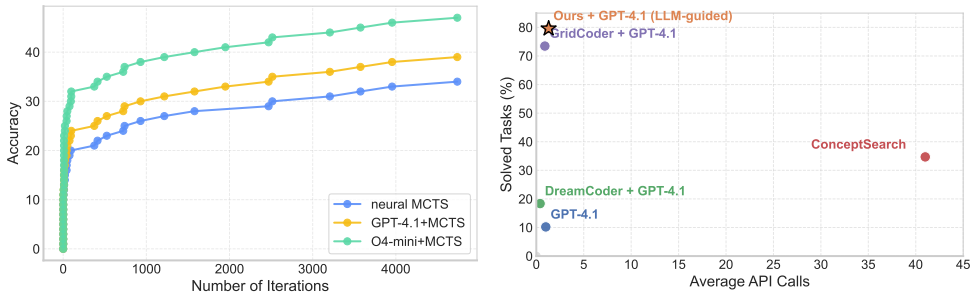


Fig. 4: (**Left**) Effect of iteration count on solved tasks in our method. Integrating GPT-4.1 or o4-mini guidance with neural MCTS enables a greater number of tasks to be solved within a fixed iteration budget, compared to neural MCTS alone. (**Right**) Comparison of performance versus average API calls. Our method attains the best performance with the low API cost.

effective value updates for partial sequences. Allowing the value network to leverage partial rewards further improves performance to 34 tasks, demonstrating that partial supervision enhances performance.

Table 1 (right) illustrates the impact of various discount factors γ on performance. Generally, smaller values of γ lead to reduced performance, as they overly prioritize shorter programs and largely disregard the similarity between outer grids and the target grid.

5.4 How does using an LLM to reduce the search space impact MCTS performance across iterations?

Figure 4 (left) presents the performance of our approach as a function of the number of search iterations, which is capped at a maximum of 5,000. The results compare neural MCTS, Ensemble method of MCTS with GPT-4.1 guidance, and MCTS with o4-mini guidance. Notably, increasing the maximum number of iterations for neural MCTS beyond 5,000 does not yield further improvements in performance. This saturation suggests that the limitation lies in the ability of the pretrained model itself, rather than the efficiency of the search. Consequently, these results highlight the need to incorporate LLM guidance to overcome this limitation. The performance progressively improves from neural MCTS to GPT-4.1 and further to o4-mini. This trend indicates that LLM-guided recommendations can enhance the efficiency of the search: tasks that

Method	$p = 0.1$	$p = 0.2$	$p = 0.3$
Random rollout sampling	30/49	27/49	23/49
Mean return over rollouts (**proposed**)	**32/49**	**30/49**	**28/49**

Table 3: Robustness comparison under stochastic transition. We report the number of solved tasks out of 49.

could not be solved by neural MCTS within 5,000 iterations become tractable when leveraging LLM suggestions. Consequently, the search space is reduced, enabling a greater number of tasks to be solved within the search budget.

5.5 How much performance can be achieved per unit cost?

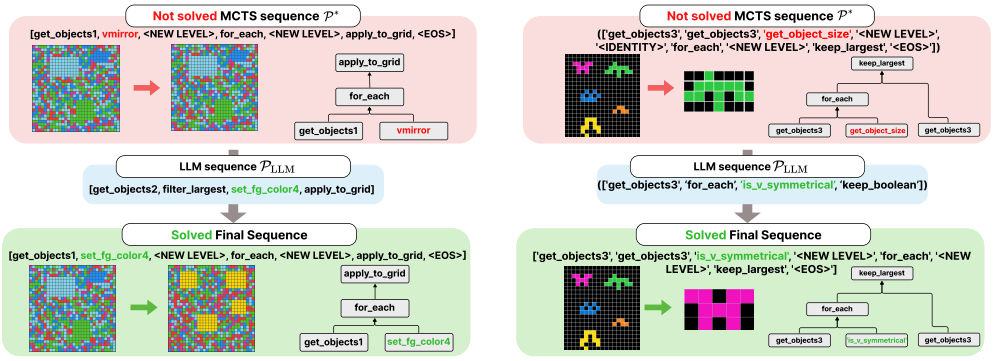
Figure 4 (right) represents illustrates how many tasks are solved as the average number of API calls increases using GPT-4.1 API guidance. Although GPT-4.1 and DreamCoder + GPT-4.1 have lower API costs, their performance is poor, whereas Our method achieve strong results with comparably low API calls. ConceptSearch shows competitive performance among baselines but requires substantially more API calls, implying higher computational cost and revealing a limitation of existing state of the art code generation models.

5.6 Robustness under stochastic transition

We evaluate the robustness of our value learning framework under stochastic execution dynamics by introducing probabilistic failures into the DSL interpreter. At each primitive execution step, the transition fails with probability p and the state remains unchanged, while with probability $1-p$ the transition executes deterministically. Under such stochastic execution, we estimate program values using multiple rollouts. We compare two strategies for value target construction: (i) using a single randomly sampled rollout outcome, and (ii) using the mean return across multiple rollouts as the regression target. As shown in Table 3, using mean returns consistently outperforms random sampling and exhibits significantly more graceful degradation as the level of stochasticity increases. These results suggest that relying on a single rollout introduces stochastic bias and high-variance targets in value estimation. In contrast, averaging returns across multiple rollouts effectively mitigates this stochastic bias, leading to more stable and reliable training signals. Although stochastic transitions inherently introduce bias, our use of mean returns over multiple rollouts effectively mitigates its impact.

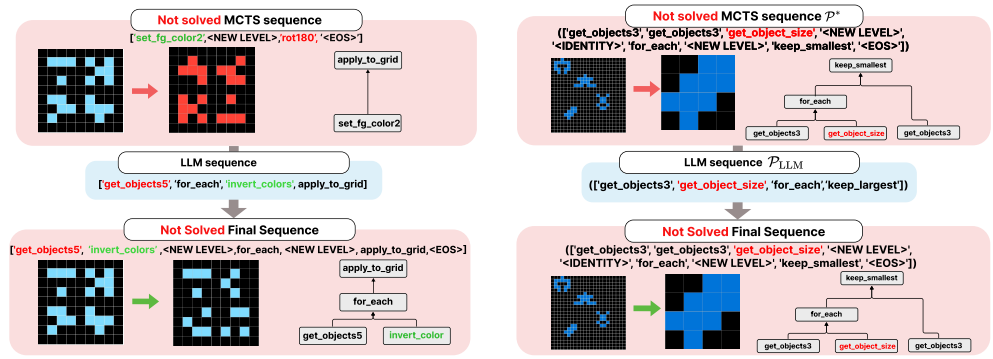
6 Case Study

We examine both the successful examples (`25094a63`, `d56f2372`) and the failure cases (`1c0d0a4b`, `73ccf9c2`) among the ARC tasks attempted with our method under the guidance of o4-mini.



(a) Task 25094a63 (b) Task d56f2372

Fig. 5: Examples of tasks solved using the LLM-guided method. (a) is successfully solved by leveraging the `set_fg_color4` primitive to modify the foreground color yellow, while (b) makes use of `is_v_symmetrical` to reason about the vertical symmetry present in the input grid.



(a) Task 1c0d0a4b (b) Task 73ccf9c2

Fig. 6: Examples of tasks unsolved using the LLM-guided method. (a) fails due to the absence of DSL primitive both `set_fg_color2` and `get_object6`, while (b) `is_v_symmetrical` is not included.

6.1 Task 25094a63

Figure 5a depicts our method's successful execution for task `25094a63`, where all rectangular objects are detected within the grid and colored yellow. Analysis of the neural MCTS-predicted program reveals that it executes a sequence involving rectangular object extraction (`get_object1`), vertical reflection (`vmirror`), and iterative application across the grid (`for_each`, `apply_to_grid`). However, this program falls short of solving the task because it omits the essential color-setting operation, `set_fg_color4`, which is required to color all affected objects yellow. The pretrained model's inability

to sample this specific token led to this incomplete solution. Leveraging LLM guidance for DSL programs, we successfully incorporated the necessary `set_fg_color4` token. Subsequent re-execution of MCTS with this correction enabled the model to generate the precise correct output sequence for the task.

6.2 Task d56f2372

Figure 5b illustrates task `d56f2372`. The target solution requires identifying all 8-connected objects of a uniform color (allowing diagonal connectivity) and selecting the sole object that possesses vertical symmetry. The neural MCTS program initially identifies all same-color, 8-connected objects using `get_object3`, measures their sizes via `get_object_size`, and then selects the largest object from those matching the retrieved size list using `keep_largest`. Critically, the pretrained model fails to incorporate the vertical symmetry verification step, a crucial component of the target solution. Leveraging the LLM to provide the `is_v_symmetrical` DSL token and integrating it into the program sequence successfully addressed this limitation. The corrected program filters the 8-connected objects to retain only those exhibiting vertical symmetry (indicated by a Boolean value of 1), subsequently selecting the largest among these. As only one such object exists in the grid, this refined sequence accurately returns the unique symmetric object.

6.3 Task 1c0d0a4b

Figure 6a presents the tasks corresponding to failure cases. The correct program for this task finds all objects that are 3×3 in size (i.e., objects containing colors other than the background color), simply inverts the colors and then changes the resulting color to red.[3] An analysis of the MCTS results reveals its limitations in accurately solving the task. Specifically, the algorithm struggles with precise object detection and, although it attempts to change the color to red, it fails to correctly modify the relevant pixels at the intended locations.The LLM fails to suggest the correct `get_objects6` DSL for detecting 3×3 windows, instead recommending `get_object5`, which leads to an incorrect solution. While it did suggest useful tokens like `invert_color`, the failure to detect the correct objects ultimately prevented task success. One possible explanation for incorrect reasoning in this case may be that the object detection DSLs were introduced solely through textual descriptions in the prompt. Furthermore, In the case of the first training instance, correct outputs are obtained by detecting 8-connected non-background objects using `get_object5`, which may lead to false confidence in its reasoning.

6.4 Task 73ccf9c2

Figure 6b presents the result for this task. In contrast to the previously successful task `d56f2372`, this example requires identifying the 8-connected object of a single color that is not vertically symmetric. Notably, there is only one such non-symmetric object in the grid. The MCTS-generated program identifies all 8-connected objects of

[3] See the full grid pairs at https://o2arc.com/task/1c0d0a4b

the same color, measure their sizes, and returns the smallest one. Obtaining the correct solution requires using the `is_v_symmetrical` DSL in place of `get_object_size`. However, the LLM fails to suggest this DSL token. It does not incorporate symmetry detection. Unlike task `d56f2372`, the our method fails on this task primarily because LLM are known to struggle with identifying the absence of recognizing when an object does not satisfy a given condition-such as being non-symmetric. Prior research has shown that LLM perform well when detecting the presence of features, but consistently underperform in reasoning about their negation [18].

7 Conclusion

In this work, we introduce a neural MCTS framework for program synthesis, combining a pretrained DSL sequence model with a value network to guide the search process. Although the pretrained model alone exhibits limited generalization capabilities when faced with novel tasks, incorporating of a learned value function significantly enhances exploration efficiency and facilitates more informed search strategies. By propagating value estimates across partial sequences, the solver effectively leverages accumulated experience, enabling successful handling of previously unseen task. Moreover, we further improved search effectiveness by employing LLM-based recommendations for generating promising DSL token sequences. This targeted LLM guidance substantially reduces the search space complexity, allowing the framework to efficiently solve tasks that are individually intractable for either neural MCTS or standalone LLM-based methods. Our comprehensive experimental results demonstrate that the fully integrated method, incorporating guidance from o4-mini, achieves near-complete task coverage by successfully solving 47 out of the 49 tasks expressible in the DSL. These findings highlight the synergy between neural-guided search and LLM-generated DSL recommendations, enabling robust and scalable program synthesis.

8 Future Work

Our current method is primarily tailored to ARC-AGI-1 [1], where structured domain-specific languages (DSLs), such as the Hodel DSL, are available and effective. In contrast, ARC-AGI-2 [19] poses a greater challenge, in part because existing DSLs can, in principle, represent ARC-AGI-2 tasks, but often at the cost of significantly longer and more complex programs. A promising direction for future work is to leverage frontier LLMs to induce, refine, or expand DSL primitives, followed by additional pretraining on the resulting extended DSL. Our LLM-guided Neural MCTS framework could then be naturally applied on top of this enriched program space to enable more flexible and scalable program synthesis. We believe that combining LLM-driven DSL expansion with search-based reasoning represents an important step toward addressing the challenges posed by ARC-AGI-2.

Declarations

- **Funding** This work was partly supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the

Korea government (MSIT) (No. RS-2022-II220311, Development of Goal-Oriented Reinforcement Learning Techniques for Contact-Rich Robotic Manipulation of Everyday Objects; No. RS-2024-00457882, AI Research Hub Project; No. RS-2019-II190079, Artificial Intelligence Graduate School Program (Korea University); and No. RS-2025-25410841, Beyond the Turing Test: Human-Level Game-Playing Agents with Generalization and Adaptation), the IITP–ITRC (Information Technology Research Center) grant funded by the Korea government (Ministry of Science and ICT) (IITP-2025-RS-2024-00436857), the NRF (RS-2024-00451162 and RS-2024-00454000) funded by the Ministry of Science and ICT, Korea, the BK21 Four project of the National Research Foundation of Korea, the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (RS-2025-00560367), and the IITP under the Artificial Intelligence Star Fellowship support program to nurture the best talents (IITP-2025-RS-2025-02304828) funded by the Korea government (MSIT).

- **Conflict of interest/Competing interests** We declare that the authors have no competing interests as defined by Springer, or other interests that might be perceived to influence the results and/or discussion reported in this paper.
- **Ethics approval and consent to participate** Not applicable.
- **Consent for publication** Not applicable
- **Data availability** All datasets used in this study are publicly available benchmark datasets. No new proprietary or personal data were generated or analyzed in this work.
- **Materials availability** Not applicable
- **Code availability** The code implementing the proposed method will be made publicly available upon publication.
- **Author contribution** J.J.(Jinwoo Jeon) conducted the overall algorithm design and performed all experiments. S.S.(Seongwoong Shim) prepared the figures and tables in the manuscript. S.K. (Sejin Kim), S.K. (Sungdong Kim), and B.L.(Byung-Jun Lee) supervised the project and revised the manuscripts. All authors reviewed the manuscript.

References

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

[2] Xu, Y., Khalil, E.B., Sanner, S.: Graphs, constraints, and search for the abstraction and reasoning corpus. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 37, pp. 4115–4122 (2023). https://doi.org/10.1609/aaai.v37i4.25527

[3] Lei, C., Lipovetzky, N., Ehinger, K.A.: Generalized planning for the abstraction and reasoning corpus. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, pp. 20168–20175 (2024). https://doi.org/10.1609/aaai.v38i18.29996

[4] Bober-Irizar, M., Banerjee, S.: Neural networks for abstraction and reasoning. Scientific Reports **14**(1), 27823 (2024) https://doi.org/10.1038/s41598-024-73582-7

[5] Ouellette, S.: Towards efficient neurally-guided program induction for arc-agi. arXiv preprint arXiv:2411.17708 (2024)

[6] Alford, S., Gandhi, A., Rangamani, A., Banburski, A., Wang, T., Dandekar, S., Chin, J., Poggio, T., Chin, P.: Neural-guided, bidirectional program search for abstraction and reasoning. In: International Conference on Complex Networks and Their Applications, pp. 657–668 (2021). https://doi.org/10.1007/978-3-030-93409-5_54 . Springer

[7] Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning **8**(3), 229–256 (1992) https://doi.org/10.1007/BF00992696

[8] Ellis, K., Wong, C., Nye, M., Sablé-Meyer, M., Morales, L., Hewitt, L., Cary, L., Solar-Lezama, A., Tenenbaum, J.B.: Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In: Proceedings of the 42nd Acm Sigplan International Conference on Programming Language Design and Implementation, pp. 835–850 (2021). https://doi.org/10.1145/3453483.3454080

[9] Xu, Y., Li, W., Vaezipoor, P., Sanner, S., Khalil, E.B.: Llms and the abstraction and reasoning corpus: Successes, failures, and the importance of object-based representations. arXiv preprint arXiv:2305.18354 (2023)

[10] Singhal, K., Shroff, G.: Conceptsearch: Towards efficient program search using llms for abstraction and reasoning corpus (arc). In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 39, pp. 20506–20513 (2025). https://doi.org/10.1609/aaai.v39i19.34259

[11] Akyürek, E., Damani, M., Zweiger, A., Qiu, L., Guo, H., Pari, J., Kim, Y., Andreas, J.: The surprising effectiveness of test-time training for few-shot learning. arXiv preprint arXiv:2411.07279 (2024)

[12] Franzen, D., Disselhoff, J., Hartmann, D.: Product of experts with llms: Boosting performance on arc is a matter of perspective. In: Forty-second International Conference on Machine Learning, pp. 17657–17671 (2025). https://openreview.net/forum?id=dsBjxI6l8W

[13] Pourcel, J., Colas, C., Oudeyer, P.-Y.: Self-improving language models for evolutionary program synthesis: A case study on arc-agi. In: Forty-second International Conference on Machine Learning, pp. 49659–49688 (2025). https://openreview.net/forum?id=z4IG090qt2

[14] Hodel, M.: ARC-DSL. https://github.com/michaelhodel/arc-dsl (2023)

[15] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., *et al.*: Mastering atari, go, chess and shogi by planning with a learned model. Nature **588**(7839), 604–609 (2020) https://doi.org/10.1038/s41586-020-03051-4

[16] Kostrikov, I., Nair, A., Levine, S.: Offline reinforcement learning with implicit q-learning. In: International Conference on Learning Representations (2022). https://openreview.net/forum?id=68n2s9ZJWF8

[17] Park, M.-K., Lee, B.-J.: Improving neural machine translation with offline evaluations. In: Park, J.C., Arase, Y., Hu, B., Lu, W., Wijaya, D., Purwarianti, A., Krisnadhi, A.A. (eds.) Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 385–397. Association for Computational Linguistics, Nusa Dua, Bali (2023). https://doi.org/10.18653/v1/2023.ijcnlp-main.25 . https://aclanthology.org/2023.ijcnlp-main.25/

[18] García-Ferrero, I., Altuna, B., Alvez, J., Gonzalez-Dios, I., Rigau, G.: This is not a dataset: A large negation benchmark to challenge large language models. In: Bouamor, H., Pino, J., Bali, K. (eds.) Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, pp. 8596–8615. Association for Computational Linguistics, Singapore (2023). https://doi.org/10.18653/v1/2023.emnlp-main.531 . https://aclanthology.org/2023.emnlp-main.531/

[19] Chollet, F., Knoop, M., Kamradt, G., Landers, B., Pinkard, H.: Arc-agi-2: A new challenge for frontier ai reasoning systems. arXiv preprint arXiv:2505.11831 (2025)

[20] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[21] Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games **4**(1), 1–43 (2012) https://doi.org/10.1109/TCIAIG.2012.2186810

Appendix A Appendix

A.1 Hyperparameter Settings and Templates

We optimized the value network using the AdamW [20] optimizer with a learning rate of 3e-4, betas of (0.9, 0.999), and a weight decay of 1e-2. The number of MCTS iterations was set to a maximum of 5,000. The exploration rate ϵ was set to 0.4, the discount factor γ to 0.9, and the expectile loss for τ to 0.7. The value network employed the same architecture for all tasks, but was trained separately on each task.

We present the full set of available DSL primitives, along with text-based representations of the input–output grid pairs. Subsequently, using the most similar program sequence found by MCTS (which did not solve the task but yielded the highest similarity score), we request the LLM to recommend a new sequence of DSL primitives to solve the task. The LLM DSL recommendation full template used to reduce the search space is as follows(this is example task of 25094a63):

```
**set_fg_color1** (0): set all foreground pixels to 1
**set_fg_color2** (1): set all foreground pixels to 2
**set_fg_color3** (2): set all foreground pixels to 3
**set_fg_color4** (4): set all foreground pixels to 4
**set_fg_color5** (5): set all foreground pixels to 5
**set_fg_color6** (5): set all foreground pixels to 6
**set_fg_color7** (6): set all foreground pixels to 7
**set_fg_color8** (7): set all foreground pixels to 8
**set_fg_color9** (8): set all foreground pixels to 9
**shift_left** (9): shift entire grid one cell left
**shift_right** (10): shift entire grid one cell right
**shift_up** (11): shift entire grid one cell up
**shift_down** (12): shift entire grid one cell down
**vmirror** (13): mirror grid vertically (top <-> bottom)
**hmirror** (14): mirror grid horizontally (left <-> right)
**rot90** (15): rotate grid 90 deg clockwise
**tophalf** (16): extract top half of the grid
**bottomhalf** (17): extract bottom half of the grid
**lefthalf** (18): extract left half of the grid
**righthalf** (19): extract right half of the grid
**symmetrize_left_around_vertical** (20): reflect left half
    across vertical axis
**symmetrize_right_around_vertical** (21): reflect right half
     across vertical axis
**symmetrize_top_around_horizontal** (22): reflect top half
    across horizontal axis
**symmetrize_bottom_around_horizontal** (23): reflect bottom
    half across horizontal axis
**upscale_horizontal_by_two** (24): double width by repeating
     columns
**upscale_vertical_by_two** (25): double height by repeating
    rows
**upscale_by_two** (26): upscale both dimensions by factor 2
**gravitate_right** (27): move non-background pixels to the
    right edge
**gravitate_left** (28): move non-background pixels to the
    left edge
**gravitate_up** (29): move non-background pixels to the top
    edge
```

gravitate_down (30): move non-background pixels to the bottom edge
gravitate_left_right (31): split gravitate in left/right halves
gravitate_top_down (32): split gravitate in top/down halves
topthird (33): extract top 1/3 of the grid
vcenterthird (34): extract vertical center third
bottomthird (35): extract bottom 1/3 of the grid
leftthird (36): extract leftmost third
hcenterthird (37): extract horizontal center third
rightthird (38): extract rightmost third
cellwiseOR (39): OR operation per cell across two grids
cellwiseAND (40): AND operation per cell across two grids
cellwiseXOR (41): XOR operation per cell across two grids
cellwiseDifference (42): difference per cell (a & not b)
cellwiseNOR (43): NOR operation per cell across two grids
vconcat (44): concatenate two grids vertically
hconcat (45): concatenate two grids horizontally
color_change (46): change all pixels of one color to another
invert_colors (47): swap major/minor foreground colors
first_quadrant (48): extract top-left quadrant
second_quadrant (49): extract top-right quadrant
third_quadrant (50): extract bottom-left quadrant
fourth_quadrant (51): extract bottom-right quadrant
hfirstfourth (52): extract top half of left half
hsecondfourth (53): extract top half of right half
hthirdfourth (54): extract bottom half of left half
hlastfourth (55): extract bottom half of right half
vfirstfourth (56): extract left half of top half
vsecondfourth (57): extract right half of top half
vthirdfourth (58): extract left half of bottom half
vlastfourth (59): extract right half of bottom half
rot180 (60): rotate grid 180 deg
rot270 (61): rotate grid 270 deg clockwise
duplicate_top_row (62): duplicate the first (top) row
duplicate_bottom_row (63): duplicate the last (bottom) row
duplicate_left_column (64): duplicate the first (left) column
duplicate_right_column (65): duplicate the last (right) column
get_objects1 (67): extract rectangular frame objects
get_objects2 (68): extract 4-connected same-color objects

```
**get_objects3** (69): extract 8-connected same-color objects
**get_objects4** (70): extract 4-connected non-bg objects
**get_objects5** (71): extract 8-connected non-bg objects
**compress_objects_linear** (72): this generates an output
    grid from a list of objects by concatenating them to each
    other in horizontal or vertically fashion, automatically
    determined from their relative positions.
**compress_objects_quad** (73): this generates an output grid
     from a list of objects by concatenating them to each
    other in a rectangular fashion (mostly 2x2, 2x3, 3x2 and 3
    x3 patterns), automatically determined from their relative
     positions.
**compress_objects_quad_pad** (74):  similar to compress
    objects quad, but introduces one row and column of black
    padding between the concatenated objects.
**apply_to_grid** (75): paste objects back into original grid
**for_each** (76): apply a function to each object
**remove_outline** (77):  removes the top row, bottom row,
    left column, right column.
**shear_grid_left** (78): shear grid leftwards
**shear_grid_right** (79): shear grid rightwards
**shear_grid_zigzag** (80): zigzag shear effect
**get_object_size** (81): count pixels in an object
**count** (82): count number of objects
**filter_largest** (83): zero out all but largest object
**keep_largest** (84): select largest object
**filter_smallest** (85): zero out all but smallest object
**keep_smallest** (86): select smallest object
**get_pixels** (87): list all pixel colors
**is_h_symmetrical** (88): test horizontal symmetry
**is_v_symmetrical** (89): test vertical symmetry
**logical_not** (90): boolean negation
**keep_boolean** (91): keep objects where condition True
**filter_boolean** (92): zero out objects where condition
    True
**get_major_pixel** (93): get pixel of major color
**get_minor_pixel** (94): get pixel of minor color
**insert_outline** (95): add one-pixel border
**upscale_by_three** (96): upscale both dimensions by factor
    3
**cellwiseOR_list** (97): OR across a list of grids
**get_objects6** (98): extract the object square style
(3*3,5*5, containing not background color)

Example 1 Input:
```

```
3 3 2 6 3 6 8 8 8 2 3 3 3 3 3 6 3 3 8 8 1 2 2 6 3 3 2 8 1 1
8 2 1 6 3 1 8 3 1 8 3 8 8 1 3 2 8 3 8 8 3 1 3 1 8 3 2 6 2 6
1 8 3 1 8 8 8 8 8 8 1 3 2 3 3 6 6 2 3 6 2 2 2 2 2 2 2 3 2 3
1 2 8 1 8 8 8 8 8 8 3 8 1 2 1 1 2 3 8 3 2 2 2 2 2 2 2 2 1 3
1 3 8 1 8 8 8 8 8 8 3 2 3 3 8 3 1 1 3 3 2 2 2 2 2 2 2 6 8 3
3 3 8 2 8 8 8 8 8 8 3 3 1 3 2 3 6 1 1 2 2 2 2 2 2 2 2 3 6 2
8 2 1 1 8 8 8 8 8 8 3 3 6 1 3 1 8 1 2 1 2 2 2 2 2 2 2 2 3 8
1 3 3 6 8 8 8 8 8 8 8 1 2 8 2 8 2 1 3 3 2 2 2 2 2 2 2 2 8 6
3 1 3 8 3 2 3 8 1 3 1 8 1 3 1 2 3 1 8 6 2 1 3 1 1 8 3 1 6 3
2 8 6 3 1 3 8 2 1 3 2 3 3 3 3 1 8 3 3 6 2 8 2 2 6 2 1 6 2 3
8 8 2 2 3 1 1 3 2 3 3 8 2 3 3 8 8 6 6 2 1 2 6 2 3 3 3 2 6 3
6 3 2 2 8 3 2 3 3 1 3 2 2 3 2 6 3 2 2 1 1 2 1 8 6 3 2 1 8 2
8 6 2 8 2 2 2 3 3 8 1 1 3 1 6 1 3 2 8 3 8 3 3 3 3 3 3 1 8 1
8 8 2 8 8 6 8 6 3 8 6 1 3 2 8 3 6 6 2 6 3 8 3 3 3 3 3 3 8 1
1 8 2 6 2 8 1 3 6 3 8 2 2 3 6 1 6 2 8 3 8 3 3 3 3 3 3 2 2 3
3 2 8 1 1 3 2 2 2 6 8 3 8 8 1 2 8 6 1 3 1 2 3 3 3 3 3 2 2 2
8 3 8 1 2 3 8 6 3 3 3 1 6 3 1 2 1 3 2 3 2 8 3 3 3 3 3 8 3 3
6 2 3 8 6 2 2 1 8 8 1 1 1 1 1 1 1 6 6 2 1 6 3 1 6 8 3 1 2 3
2 1 2 1 8 2 3 2 6 8 1 1 1 1 1 1 1 1 1 1 3 6 2 1 2 2 2 3 3 1
1 8 8 2 8 2 2 2 3 1 1 1 1 1 1 1 1 8 2 3 2 3 6 6 2 3 3 3 6 3
2 6 8 3 6 1 3 8 3 6 1 1 1 1 1 1 1 2 3 3 3 1 6 3 3 6 1 3 2 2
6 8 6 2 3 2 6 3 3 1 1 1 1 1 1 1 1 3 3 8 1 6 3 3 3 8 1 8 2 3
6 3 1 3 6 6 1 6 3 8 1 1 1 1 1 1 1 1 2 1 6 3 3 8 1 8 3 8 2 1
3 2 2 3 1 1 2 3 8 6 1 3 3 1 8 3 1 8 8 3 8 3 1 8 8 1 1 2 1 8
3 2 3 6 1 8 3 6 3 3 2 2 1 3 6 3 2 3 8 3 8 3 2 2 2 2 3 3 1 6
2 8 6 2 2 1 8 3 1 6 8 2 3 2 3 2 3 3 3 3 2 2 2 8 6 8 3 6 1 3
6 2 3 2 3 3 8 3 3 6 2 2 3 3 8 8 1 3 1 2 8 3 8 3 3 3 6 1 2 2
2 3 2 1 2 6 3 1 8 3 1 6 2 3 8 2 6 1 1 1 3 6 8 1 2 8 6 2 3 2
2 1 8 2 6 3 8 2 3 6 8 8 2 8 8 3 2 3 1 6 8 2 6 3 2 3 2 1 8 3
1 6 3 1 6 6 3 1 2 8 8 1 8 1 3 3 1 2 6 8 3 1 6 8 3 8 3 1 1 8
```

Example 1 output:

...

...

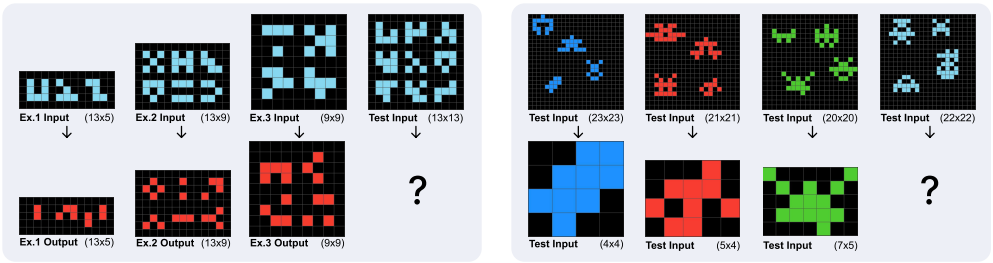
High Similarity Program Sequence (NOT the correct answer, but close):

– Sequence 1 (similarity 0.875): get_objects1 –> vmirror –> <NEW LEVEL> –> for_each –> <NEW LEVEL> –> apply_to_grid –> <EOS>

The above program sequence(s) are close to the correct answer, but not exactly correct. Please use them as a reference, analyze what is missing or incorrect, and propose a new program sequence that solves the task correctly.

```
1. Propose a program sequence (a list of DSL primitives and
   their arguments, in order) that can transform the given
   input(s) to the output(s). Please use the minimal set of
   DSL primitives necessary to achieve the transformation.
2. For each primitive in your proposed program sequence,
   provide a one-sentence rationale explaining why it is
   required for the transformation.
3. For each example, check if your proposed program sequence
   can actually transform the input to the output. Briefly
   explain for each example whether the transformation is
   correct, and if not, what is missing or incorrect.
Please return your answer as JSON with the following keys:
- program_sequence: [ ... ]
- justification: { prim_name: rationale, ... }
- example_verification: { example_idx: verification_result,
  ... }
```

A.2 Full Grid Pairs for Failure Cases



(a) Task 1c0d0a4b (b) Task 73ccf9c2

Fig. A1: (a) Extract the 3×3 object tile containing blue pixels, invert its colors, and then recolor it to red, (b) Select objects that are not vertically symmetric among all 8-connected objects.

We will provide examples of tasks that could not be solved using LLM guided of using O4-mini. Figure A1 are the input-output grid pairs for task 1c0d0a4b and task 73ccf9c2.

For task 1c0d0a4b, as shown in the training pairs, This task involves extracting a 3×3 object tile containing blue pixels (`get_objects6`), inverting its colors(`invert_colors`), and then converting its color to red(`set_fg_color2`). For task 73ccf9c2, the objective is to select the object among all 8-connected objects(`get_objects3`) that is not vertically symmetrical(`is_v_symmetrical`). In this case, there is only one such object present in each example.

A.3 Theoretical Perspective

Our approach is built upon the standard Monte Carlo Tree Search (MCTS) framework, which iteratively refines search decisions through repeated simulation, evaluation, and backpropagation. Under sufficient exploration and an increasing number of simulations, this process is commonly understood to lead to progressively more stable and informed search behavior [21].Within this framework, the large language model (LLM)-derived policy prior and the learned value function are incorporated as heuristic guidance mechanisms. Their role is to improve search efficiency in large and sparse program spaces by prioritizing plausible program continuations and focusing computation on promising regions of the search space. Importantly, these components do not alter the underlying evaluation or backpropagation dynamics of MCTS, but instead influence the order in which candidate nodes are explored. As a consequence, the overall search process retains the iterative refinement characteristics of MCTS, while benefiting from heuristic guidance that reduces the effective search space.