ConceptSearch: Towards Efficient Program Search Using LLMs for Abstraction and Reasoning Corpus (ARC) (Student Abstract)

Kartik Singhal, Gautam Shroff *

IIIT Delhi, New Delhi 110020, India
{kartik21259, gautam.shroff}@iiitd.ac.in

Abstract

The Abstraction and Reasoning Corpus (ARC) poses a significant challenge to artificial intelligence, demanding broad generalization and few-shot learning capabilities that remain elusive for current deep learning methods, including large language models (LLMs) (Chollet 2019). While LLMs excel in program synthesis, their direct application to ARC yields limited success. To address this, we introduce ConceptSearch, a novel function-search algorithm that leverages LLMs for program generation and employs a concept-based scoring method to guide the search efficiently. Experimental results demonstrate that ConceptSearch outperforms direct GPT-4 prompting, with our novel scoring function boosting efficiency by 30% compared to the baseline Hamming distance. Code at https://github.com/kksinghal/concept-search

1 Introduction

A task $\tau = [(I_{1:n}, O_{1:n}), (I_{t,_{1:n'}}, O_{t,_{1:n'}})]$ consists of set of n demonstration examples $(I_{1:n}, O_{1:n})$ exhibiting a common latent transformation. The goal is to infer that transformation and apply it to each of n' test input grids $I_{t,_{1:n'}}$ to get the corresponding test output grids $O_{t,_{1:n'}}$. Each grid-cell consists of one of 10 symbols (visualized as unique colors) and the grid-size ranges from 1×1 to 30×30 in size. Success is binary, requiring correct predictions for all test examples. The performance is measured by the fraction of tasks solved.

Framing this as a program-synthesis problem, brute-force search and neural-guided search-based methods require careful hand-crafted Domain-Specific Language (DSL) and still scale poorly to complex problems due to combinatorial complexity. Even though humans can solve most of the tasks, to the best of our knowledge, no current machine learning techniques, including Deep Learning, are well-suited to tackle the ARC benchmark.

FunSearch (Romera-Paredes et al. 2023) proposed a function-search algorithm that evolves programs using a large language model (LLM). At each step, the LLM samples and ranks programs from a database, generating new ones based on their scores. This iterative process aims to improve the programs until they converge on a solution.

FunSearch is suited for problems with an efficient evaluator for determining success and a rich scoring feedback quantifying the improvements, instead of binary signal. A trivial scoring function is Hamming distance between the predicted output grid and true output grid, that is, the number of mismatched-pixels, normalized by the size of the grid. However, relying solely on Hamming distance to evaluate programs can be misleading, as superficial resemblance can hide major differences in the program's logic. Therefore, we need our scoring function to capture the concept or logic of the transformation. Our work applies FunSearch to ARC and introduces two novel concept-based scoring functions.

2 Methodology

2.1 ConceptSearch

We adopt the ARC-DSL (Hodel 2024) as our Domain-Specific Language and the provided solutions of training tasks to initialise the program database. In each of the program-generation step, two programs f_1 and f_2 are sampled from P_i using a probability distribution based on the scoring function S. The prompt consists of initial problem context, input grids $I_{1:n}$ of demonstration examples, similar programs f_1 and f_2 along with similarity scores, their realised outputs $f_1(I_{1:n})$ and $f_2(I_{1:n})$, and then finally the desired outputs $O_{1:n}$ of the task.

The generated program f is run with training grids $I_{1:n}$ as input and is evaluated by comparing the dimensions and pixel equivalence with $O_{1:n}$. In case of a syntax error, a feedback loop adds the traceback error in the prompt of the next iteration to potentially fix that syntax. If a program correctly generate the output grids for all demonstration examples, it is evaluated on test grids $(I_{t,_{1:n'}}, O_{t,_{1:n'}})$. If successful, the task is solved. Otherwise, f is added to P and algorithm continues up to 3 evaluation attempts.

We hypothesize that the quality of the programs provided in context directly influences the efficiency of the search process and show that the choice of scoring function S is fundamental to solving complex tasks in a reasonable compute.

2.2 CNN-Based Scoring Function

For demonstration examples $(I_{1:n}, O_{1:n})$ and a given program f, we first compute the realized outputs $f(I_{1:n})$. This yields two sets of transformations: $(I_{1:n}, O_{1:n})$ and

*This work was conducted while Kartik Singhal was an intern and Gautam Shroff was employed at TCS Research

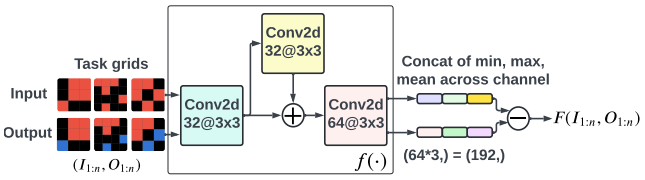

Figure 1: Model architecture trained with classification and contrastive loss to learn meaningful concept representations.

$(I_{1:n}, f(I_{1:n}))$. Our goal is to compute the distance between these two sets of transformations.

To train a neural network capable of capturing the concept of transformation, we utilize the Concept-ARC dataset (Moskvichev, Odouard, and Mitchell 2023). The dataset categorizes tasks into 16 distinct classes, each representing a specific concept. Our neural-network architecture is inspired by (Bober-Irizar and Banerjee 2024) for handling variable-sized grids (see Figure 1). Due to the variability in grid sizes, we use an aggregator functions (min, max and mean) across channels to produce a constant-size feature vector.

For each input-output pair in the task grids, the pair is passed through the model. The mean of the feature vectors for each pair, after performing a difference operation, is calculated. The final feature vector z, which integrates information from all input-output pairs, effectively represents the underlying transformation rule.

The model F is trained with a dual loss approach, combining cross-entropy and contrastive (triplet-margin) loss to classify tasks into 16 concept classes. Data augmentation, including grid rotation, transposition and random color permutations, was used to improve generalization.

The similarity score between the transformation underlying $(I_{1:n}, O_{1:n})$ and program f is computed as:

$$S_{\text{CNN}}((I_{1:n}, O_{1:n}), f) = \|F(I_{1:n}, O_{1:n}) - F(I_{1:n}, f(I_{1:n}))\| \quad (1)$$

2.3 LLM-Based Natural Language Scoring Function

Instead of using CNN-based feature extractor, we use a pre-trained LLM for this task. LARC (Acquaviva et al. 2023) is a dataset consisting of natural language descriptions on how to solve each task provided by human participants. For solving a task with demonstration examples $(I_{1:n}, O_{1:n})$, our objective is to find a natural language hypothesis that corresponds to these examples. To generate this "goal" hypothesis h_0, we employ a completion task by providing in-context examples consisting of $(I_{1:n}, f_i(I_{1:n}))$ and their NL descriptions.

At this stage, we have natural language hypotheses, h_0 for the desired transformation (goal hypothesis) and h_f for each of the programs in the program database. To derive a feature vector for a natural language hypothesis, we fine-tune a SentenceTransformer F (all-mpnet-base-v2) using the ConceptARC dataset with contrastive loss (BatchAllTripletLoss).

The similarity score between the goal hypothesis h_0 and program description h_f,

$$S_{\text{LLM}}(h_0, h_f) = \|F(h_0) - F(h_f))\| \quad (2)$$

3 Results

The evaluation set includes 50 tasks from the training set, chosen to reduce LLM inference costs and allow comparison with the results in (Xu et al. 2024). In our experiments, we utilize Gemini 1.5 Pro for both program generation and goal hypothesis generation. For the hypothesis generation of program definitions, we employ Gemini 1.5 Flash.

We evaluate our method using three different scoring functions: Hamming distance, CNN-based scoring, and LLM-based natural language scoring (Table 1 and 2).

Method	Accuracy	Mean iters
(Xu et al. 2024) - GPT-4	13/50	-
Ours - Hamming distance	25/50	3.70
Ours - CNN-based	25/50	2.80
Ours - LLM-based	**29/50**	**2.05**

Table 1: Performance Comparison: Our Approach vs. GPT-4 on Direct-Grid Prompting. Mean iterations are the average program iterations for tasks solved by all methods.

	LLM-based	CNN-based
Hamming distance	24.7	29.3
CNN-based	10.3	-

Table 2: Efficiency (%) gain in each column shows improvement over the corresponding row's function based on program iterations, for tasks solved by both methods.

References

Acquaviva, S.; Pu, Y.; Kryven, M.; Sechopoulos, T.; Wong, C.; Ecanow, G. E.; Nye, M.; Tessler, M. H.; and Tenenbaum, J. B. 2023. Communicating Natural Programs to Humans and Machines. arXiv:2106.07824.

Bober-Irizar, M.; and Banerjee, S. 2024. Neural networks for abstraction and reasoning: Towards broad generalization in machines. arXiv:2402.03507.

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

Hodel, M. 2024. ARC-DSL.

Moskvichev, A.; Odouard, V. V.; and Mitchell, M. 2023. The ConceptARC Benchmark: Evaluating Understanding and Generalization in the ARC Domain. arXiv:2305.07141.

Romera-Paredes, B.; Barekatain, M.; Novikov, A.; Balog, M.; Kumar, M. P.; Dupont, E.; Ruiz, F. J. R.; Ellenberg, J. S.; Wang, P.; Fawzi, O.; Kohli, P.; and Fawzi, A. 2023. Mathematical discoveries from program search with large language models. *Nature*, 625(7995): 468–475.

Xu, Y.; Li, W.; Vaezipoor, P.; Sanner, S.; and Khalil, E. B. 2024. LLMs and the Abstraction and Reasoning Corpus: Successes, Failures, and the Importance of Object-based Representations. arXiv:2305.18354.