Hypothesis Generation in ARC Tasks: Leveraging Human Cognitive Insights for Enhanced Abstract Reasoning

Abhinay Krishna Bodi, Peter Skovorodnikov, Nikolas Prasinos, Xinrui Hu

***Abstract*—In recent years, abstract reasoning benchmarks such as the Abstraction and Reasoning Corpus (ARC) have emerged as critical tools for evaluating cognitive reasoning capabilities in AI systems. However, state-of-the-art models like GPT-4 continue to struggle to solve ARC tasks. In this study, we introduce a novel approach that combines automated hypothesis generation with iterative refinement and task-solving mechanisms. Leveraging the H-ARC dataset, which contains human-generated solutions to ARC problems, our method aims to enhance the plausibility of generated hypotheses while improving task accuracy. Through systematic evaluation, we demonstrate improvements in both hypothesis plausibility and task performance, underscoring the feasibility of enhancing capabilities using cognitive reasoning with open-source models. These findings contribute to the broader understanding of how AI systems can emulate human-like reasoning in abstract tasks.**

Keywords: Cognitive Modeling, Code Generation, Hypothesis Generation, Abstract Reasoning Corpus, H-ARC Dataset

I. Introduction

The Abstraction and Reasoning Corpus (ARC), introduced by Chollet (2019) [1], has emerged as a pivotal benchmark for evaluating the abstract reasoning capabilities of artificial intelligence (AI) systems. In contrast to conventional datasets that focus on pattern recognition from large-scale data, ARC uniquely challenges models to deduce transformation rules from a limited set of input-output examples and apply these rules to novel, unseen tasks. This paradigm shift positions ARC as a critical dataset for assessing generalizable reasoning, a core attribute of human cognition.

Despite ARC's potential, recent studies, such as Wang et al. (2023) [2], have highlighted persistent challenges. Even advanced language models, including GPT-4, often fail to generate accurate and coherent natural language hypotheses required for solving ARC tasks. This shortcoming underscores a broader gap between current AI systems and human-level abstract reasoning. Prior work, including LeGris et al. (2024) [3], has demonstrated the value of human-generated annotations in mitigating this gap by improving both model predictions and task performance. These findings point to the necessity of integrating human cognitive insights into AI systems to enhance their reasoning capabilities.

To address these limitations, our approach introduces a scalable and robust framework that leverages parallel processing to evaluate multiple hypotheses and generate solutions simultaneously. A feedback-driven mechanism iteratively refines the generated hypotheses and code by identifying and rectifying errors in predictions and execution. By combining the efficiency of pre-trained language models with the cognitive insights embedded in human-generated data, this framework enhances task-solving accuracy and plausibility.

This work focuses on improving hypothesis generation, identified as a critical bottleneck in solving ARC tasks. By integrating human annotations with prompt-based optimization techniques, our project advances the exploration of AI's potential to emulate human-like abstract reasoning. This study contributes to bridging the gap between artificial and human cognition, reinforcing the role of ARC as a benchmark for evaluating progress in this domain.

II. Related Work

The Abstraction and Reasoning Corpus (ARC), introduced by Chollet [1], serves as a crucial benchmark for assessing the generalization and abstract reasoning capabilities of artificial intelligence (AI) systems. Chollet posited that ARC tasks emulate human problem-solving processes, emphasizing generalization over pattern recognition or rote memorization. This makes ARC a valuable tool for evaluating AI systems' ability to deduce transformation rules and apply them to novel, unseen problems.

Despite significant progress in large language models like GPT-3 and GPT-4, challenges persist when applying these models to abstract reasoning tasks. Wang et al. [2] explored hypothesis generation for ARC tasks using GPT-4 and identified a key limitation: the inability to generate plausible and accurate hypotheses for solving these tasks. This highlights the gap between current AI capabilities and the demands of generalizable reasoning required by ARC.

In an effort to address these limitations, LeGris et al. [3] introduced the H-ARC dataset, an annotated extension of ARC containing human-generated descriptions of the transformations used to solve tasks. Their work demonstrated that incorporating human insights, such as natural language explanations, significantly improved hypothesis plausibility and task-solving accuracy. These findings underscored the importance of human-annotated

data in bridging the reasoning gap between AI and humans. The H-ARC dataset serves as a cornerstone for our project, enabling a deeper exploration of hypothesis generation and validation in abstract reasoning tasks.

While most prior work has centered on enhancing the performance of large-scale, pre-trained models, our study takes a novel direction by focusing on iterative hypothesis generation and validation. Our approach prioritizes parallelized evaluation and feedback-driven refinement, leveraging structured processes to improve accuracy and plausibility without relying on additional fine-tuning. By building on the insights from H-ARC and addressing the limitations of existing models, our work contributes to the advancement of abstract reasoning in AI systems.

III. Method

A. *Dataset*

The Human-Annotated Abstraction and Reasoning Corpus (H-ARC) is an extension of the original ARC dataset, designed to incorporate human cognitive insights into abstract reasoning tasks. It comprises action-by-action traces from 1,729 participants solving all 400 training and 400 evaluation tasks from the ARC problem set.

Key Features of the H-ARC Dataset:

- **Comprehensive Human Performance Data**: The dataset includes 15,744 attempts on ARC tasks, capturing detailed step-by-step action sequences.
- **Natural Language Descriptions**: Participants provided explanations for their problem-solving approaches, offering valuable insights into human reasoning processes.
- **Error Analysis**: The dataset documents unique errors and their frequencies, facilitating the study of common mistake patterns in abstract reasoning.
- **State Space Graphs**: Visual representations of the states visited during problem-solving are included, illustrating the pathways taken by participants.

The H-ARC dataset serves as a valuable resource for developing AI systems that emulate human-like reasoning, providing a robust benchmark for evaluating and enhancing machine performance on complex abstract reasoning tasks.

B. *Model Architecture*

We used the Qwen2.5 model, which is renowned for its open-source accessibility, computational efficiency, and cost-effective operation, as the backbone of our project. Qwen employs a Transformer-based architecture that excels in natural language processing tasks, including hypothesis generation and code synthesis. Below is a detailed breakdown of its architecture:

1) **Embedding Layer**:
   - Input tokens are transformed into dense vector representations through embedding layers. This process is mathematically represented as:
   $$E(x_i) = W_e \cdot x_i$$
   Where:
   - x_i: Input token
   - W_e: Embedding weight matrix.
2) **Self-Attention Mechanism**:
   - Qwen's attention mechanism enables it to model dependencies between tokens efficiently. The self-attention mechanism is formulated as:
   $$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$
   Here:
   - $Q = W_q \cdot H$: Query matrix.
   - $K = W_k \cdot H$: Key matrix.
   - $V = W_v \cdot H$: Value matrix.
   - d_k: Dimensionality of the keys.
3) **Multi-Head Attention**:
   - To capture information from multiple subspaces, Qwen uses multi-head attention. This is defined as:
   $$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) \cdot W_o$$
   Where each attention head is computed independently.
4) **Feed-Forward Network (FFN)**:
   - After the attention mechanism, each token representation is passed through a feed-forward network. This is expressed as:
   $$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2$$
   Where W_1, W_2, b_1, b_2 are learned parameters.
5) **Layer Normalization**:
   - Qwen incorporates layer normalization to stabilize training and enhance generalization. This is represented as:
   $$\text{LayerNorm}(x) = \frac{x - \mu}{\sigma} \cdot \gamma + \beta$$
   Where μ and σ are mean and variance, and γ, β are learned scaling parameters.
6) **Decoding Mechanism**:
   - The decoding process generates hypotheses or code token-by-token using autoregressive modeling:
   $$P(y_t | y_{<t}, x) = \text{softmax}(W_o \cdot H_t)$$
   Here H_t represents the hidden state at time t.
7) **Training Objective**:
   - The model is pre-trained using a cross-entropy loss function:
   $$\mathcal{L} = -\sum_{t=1}^{T} \log P(y_t | y_{<t}, x)$$
   This objective ensures accurate and context-aware predictions.

C. Implementation

Our project leverages the Qwen2.5 32B model, hosted on OpenRouter, to solve ARC tasks by executing three main phases: **Hypothesis Generation**, **Python Code Synthesis**, and **Asynchronous Execution**. The implementation ensures efficiency and scalability through structured prompts, iterative feedback, and parallelized processing. Each task consists of multiple training examples and one test example. The test example is evaluated after all training examples pass within the specified number of feedback attempts (`MAX_FEEDBACK_ATTEMPTS`).

Hypothesis Generation

The first phase focuses on generating hypotheses to transform input grids into output grids, based on the training examples.

a) Prompt Engineering: The Qwen model is prompted with training examples presented in a structured grid format alongside contextual explanations, derived from the H-ARC dataset. For instance, a prompt might describe the input grid as "Double the size of the grid and replicate the pattern in each quadrant" based on human-annotated descriptions.

b) Mathematical Representation: Given a set of training examples $\mathcal{T} = \{(x_i, y_i)\}$, where x_i represents the input grid and y_i represents the output grid, the hypothesis h is modeled as:

$$h : x_i \mapsto y_i$$

The Qwen model analyzes the semantic relationships within $\mathcal{T}$ and produces natural language hypotheses to describe the transformation logic.

Python Code Synthesis

Once a hypothesis is generated, the Qwen model translates it into Python code that implements the specified transformation.

a) Code Generation: The hypothesis serves as a guiding principle for generating a Python function f, where:

$$f(x) \mapsto y$$

The function f is designed to apply the transformation logic to an input grid x and produce an output grid y. The model is designed to generate `MAX_PROGRAMS` number of possible codes.

b) Iterative Feedback: If the initial code fails to produce correct outputs for the training examples, the process undergoes iterative refinement:

- Errors or deviations are identified during testing.
- Feedback is incorporated into the prompts to guide the Qwen model in revising the code.

This feedback loop continues for up to `MAX_FEEDBACK_ATTEMPTS` until all training examples are successfully solved. Only then is the generated code applied to the test example.

Asynchronous Execution

To optimize performance, the implementation is designed to execute tasks asynchronously, leveraging parallel processing for scalability.

a) Parallel Hypothesis Generation: Multiple hypotheses are generated concurrently for each set of training examples. This is achieved using asynchronous loops, ensuring effective utilization of the model's computational resources.

b) Concurrent Code Evaluation: Python code generated for each hypothesis is executed in parallel. If a hypothesis fails, the corresponding code is iteratively refined until it meets the requirements for all training examples or exceeds the feedback limit.

c) Test Evaluation: The test example is evaluated only after all training examples pass successfully. This ensures that only validated hypotheses and refined code are applied to unseen tasks, maintaining reliability.

This asynchronous architecture minimizes latency, optimizes resource utilization, and supports scalable evaluation across multiple ARC tasks. By combining the advanced reasoning capabilities of Qwen2.5 with parallelized processing.

D. Evaluation Metrics

The evaluation of our system is based on its accuracy in solving ARC tasks, specifically the correctness of the output grids generated by the Python code synthesized from the Qwen model. The evaluation metric is defined mathematically as follows:

1) **Definitions**:
   - Let n represent the total number of test cases.
   - For each test case:
     - $X_{\text{test},i}$: The input grid.
     - $Y_{\text{actual},i}$: The expected (ground truth) output grid.
     - $Y_{\text{pred},i}$: The output grid produced by the synthesized Python code.
2) **Indicator Function**: The indicator function 1 is used to determine whether the predicted output matches the expected output:

$$1[Y_{\text{pred},i} = Y_{\text{actual},i}] = \begin{cases} 1 & \text{if } Y_{\text{pred},i} = Y_{\text{actual},i} \\ 0 & \text{otherwise.} \end{cases}$$

3) **Accuracy Metric**: The accuracy of the system is computed as the ratio of correctly solved tasks to the total number of tasks:

$$\text{Accuracy} = \frac{1}{n} \sum_{i=1}^{n} 1[Y_{\text{pred},i} = Y_{\text{actual},i}].$$

4) **Error Analysis**: For cases where $Y_{\text{pred},i} \neq Y_{\text{actual},i}$, errors are logged and analyzed to identify their source. Common error categories include:
   - Incorrect or incomplete hypothesis generation.
   - Faulty code synthesis that fails to implement the hypothesis correctly.
   - Lack of generalization for unseen test cases.
5) **Iterative Refinement**:
   - If errors are detected, a feedback mechanism is triggered to refine the hypothesis or generated code iteratively.
   - This refinement process aims to minimize $1 -$ Accuracy, i.e., the error rate, over successive iterations.

The evaluation metric quantifies the system's performance in solving ARC tasks, providing a foundation for iterative optimization and benchmarking against alternative approaches.

IV. Results

A. Experimental Setup and Results

Our results are derived from a sample of 100 tasks, constrained by limited computational resources. We set the parameter of `MAX_PROGRAMS` to be equal to 8 and the parameter of `MAX_FEEDBACK_ATTEMPTS` to be equal to 2. To identify the most effective approach, we tested three distinct implementations, comparing their performance.

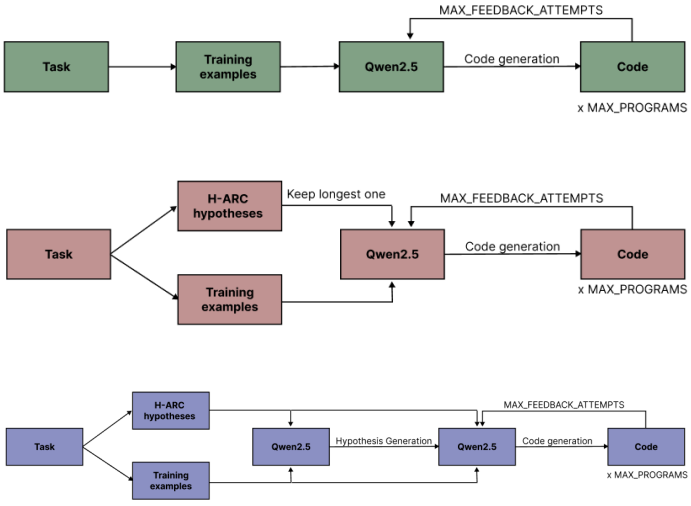

Fig. 1: Experiment 1 (green): Direct Prompting without adding H-ARC information. Experiment 2 (red): Adding the longest H-ARC explanation along with the training examples. Experiment 3 (blue): Generating a synthetic hypothesis by prompting the LLM first and using the new hypothesis to generate code.

Experiment 1: This approach attempted direct prompting the LLM, without any prior contextual information. As anticipated, this implementation produced extremely underwhelming results. Notably, the model failed to solve a single task correctly.

Experiment 2: In this implementation, hypotheses from the H-ARC dataset were incorporated into the prompts for the LLM. However, not all hypotheses were used, as their quality varied; we retained only the longest hypothesis in each case. This approach significantly outperformed the initial implementation but still fell short of achieving satisfactory performance in the broader context of ARC task solving. Specifically, this implementation solved 9 out of 100 tasks.

Experiment 3: In this implementation, we leveraged both the training examples and all hypotheses from the H-ARC dataset to generate a new synthetic hypothesis. This was achieved by prompting the LLM to create a comprehensive hypothesis that encapsulated the information from the human explanations, guided by the training examples. The newly synthesized hypothesis was then used to prompt the LLM to generate code for solving the task. Interestingly, this approach produced results similar to the previous implementation (*Experiment 2*), successfully solving some tasks that the prior approach could not, while failing on tasks that the previous approach managed to solve. However, the overall performance remains insufficient compared to state-of-the-art results, with the model successfully solving 11 out of 100 tasks.

Approaches	**Correct (out of 100)**
Experiment 1	0
Experiment 2	9
Experiment 3	11
Direct prompting (pretrained GPT 4o)	< 5
Direct prompting (fine-tuned LLM)	$\sim$10

TABLE I: Comparison of ARC task-solving approaches and their accuracies, with Direct Prompting performance sourced from the ARC prize Guide

B. Insights and Implications for Future Work

Although the results did not meet the desired performance levels, they provide valuable insights that can serve as a reference for future advancements in the ARC problem-solving challenge.

Firstly, it is evident that directly prompting a pretrained LLM without any contextual input is not an effective approach, as reflected in the very poor performance. This aligns with François Chollet's assertion that LLMs function more like sophisticated lookup tables rather than machines capable of human-like reasoning.

Secondly, the much better results achieved in other experiments highlight the potential of the H-ARC dataset as a valuable resource. Its collection of human explanations can significantly aid the LLM in both hypothesis generation and code generation. However, while H-ARC is useful and indeed increased performance, some of the human explanations were not sufficiently clear or detailed for the LLM to fully grasp the underlying pattern of the ARC tasks at hand. This highlights the importance of high-quality explanations in enabling LLMs to perform complex reasoning tasks. Nevertheless, it is

important to note that even with this improvement, the results remain below the desired threshold. This suggests potential limitations in the way LLMs synthesize new information, further supporting the notion that LLMs do not fully emulate human cognition. An interesting direction for future work would be to fine-tune an LLM with the H-ARC dataset combined with the training examples.

Thirdly, the parameters `MAX_PROGRAMS` and `MAX_FEEDBACK_ATTEMPTS` were found to be critical for the model's performance. Specifically, increasing these parameters yielded better results. However, due to computational constraints, we were unable to test higher values, such as the 64 programs generated by Wang et al. [2] Despite this, our findings underscore the model's ability to learn and improve by iteratively correcting past errors.

V. Cognitive Science

This project is deeply rooted in cognitive science, drawing inspiration from human processes of abstract reasoning, hypothesis generation, and problem-solving. Cognitive science examines how humans reason and solve problems by generalizing abstract rules from limited examples and applying them to novel tasks. The Abstraction and Reasoning Corpus (ARC) embodies these principles, challenging AI systems to infer transformation rules and solve unseen problems in ways that go beyond pattern recognition.

Our approach attempts to emulate human reasoning by leveraging large language models to generate plausible hypotheses informed by human-generated insights. The H-ARC dataset, which contains detailed human annotations and step-by-step solutions to ARC problems, provides a foundation for this process. Just as humans analyze minimal data to infer generalizable rules, our system generates natural language hypotheses that mimic this human cognitive ability. These hypotheses guide the transformation logic, enabling AI to approach abstract reasoning tasks in a structured and human-like manner.

A defining feature of human cognition is its flexibility and adaptability. When faced with errors or unexpected results, humans iteratively refine their mental models, learning from feedback to adjust their hypotheses and strategies. This iterative feedback loop is central to our system, which mirrors this process by refining hypotheses and transformation rules through repeated cycles of feedback and adjustment. By incorporating this human-like learning mechanism, the system captures an essential aspect of cognitive reasoning: the ability to learn and improve from mistakes.

Cognitive science offers a wealth of insights into how humans generalize, adapt, and solve complex problems—insights that continue to shape advancements in AI. While our current implementation demonstrates the potential of these principles, it also points to opportunities for further improvement. By deepening the integration of cognitive science concepts, such as incorporating richer contextual reasoning, exploring multi-modal data inputs, or modeling hierarchical thinking, AI systems can become more aligned with human reasoning.

This work highlights the importance of cognitive science in guiding the development of AI systems capable of abstract reasoning. The iterative incorporation of human-like processes, including hypothesis generation and feedback-driven refinement, serves as a promising pathway for achieving more robust and generalizable reasoning in AI. With continued advancements in this interdisciplinary approach, we are optimistic that future systems will achieve even greater alignment with human cognition and solve increasingly complex abstract reasoning tasks.

VI. Conclusion

This project successfully demonstrated the application of hypothesis generation and code synthesis to address ARC tasks using the Qwen model. By drawing inspiration from human cognitive processes and leveraging the structured guidance provided by the H-ARC dataset, our system showcased the potential of integrating human-like reasoning into AI systems. The iterative learning and feedback-driven refinement processes enabled the model to adapt and improve its outputs, reflecting a fundamental aspect of human cognition—learning through experience and correction.

While the achieved accuracy of $\sim$11% highlights the inherent complexity of ARC tasks, it also underscores the promise of combining cognitive science principles with advanced language models. Techniques such as incorporating human explanations, iterative hypothesis generation and feedback integration proved effective in enhancing the system's ability to reason abstractly, laying a solid foundation for future exploration.

Looking ahead, deeper integration of cognitive science concepts can further bridge the gap between human and machine reasoning. Strategies like hierarchical reasoning, context-aware hypothesis generation, and multi-modal inputs may enable models to approach abstract reasoning tasks with greater sophistication. Additionally, advancing our understanding of human cognitive processes can inspire novel architectures and training methodologies that align more closely with human problem-solving strategies.

In conclusion, this project not only advances the capability of AI systems to tackle abstract reasoning tasks but also contributes to the broader endeavor of emulating human cognition in AI. By simulating key elements of human reasoning, we take a meaningful step toward developing AI systems that are more intuitive, adaptable, and capable of addressing complex, novel challenges in the future.

References

[1] F. Chollet, *On the Measure of Intelligence*, 2019.

[2] R. Wang, et al., *Hypothesis Search: Inductive Reasoning with Language Models*, 2023.
[3] S. LeGris, et al., *H-ARC: A Robust Estimate of Human Performance on the Abstraction and Reasoning Corpus Benchmark*, 2024.