Optimized Concept Learning in BEN

Better identifying when transformations apply in a program synthesis agent

Jonas Duifs[1]
Supervisors: Dr. Sebastijan Dumančić[1]**, Dekel Zak**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 25, 2026

Name of the student: Jonas Duifs
Final project course: CSE3000 Research Project
Thesis committee: Dr. Sebastijan Dumančić, Dekel Zak, Prof.dr. Arie van Deursen

Abstract

The Abstraction and Reasoning Corpus (ARC) challenges AI systems to induce general rules from as few as two to four examples. BEN is a program synthesis agent that tackles ARC through a Divide, Align, and Conquer pipeline inspired by analogical reasoning, but its concept learning phase, which synthesizes Boolean rules governing when transformations apply, must frequently choose between candidate rules of equal structural complexity. Because ARC's extreme data scarcity makes such ties highly prevalent, the system's coarse notion of complexity provides insufficient nuance to distinguish between candidates, leaving it vulnerable to overfitting.

We present three optimizations to BEN's concept learner, built on a reimplementation of BEN in Julia. These optimization provide better nuance in complexity at different stages of the pipeline, helping the system more accurately mimic human analogical reasoning. A *Rule Generality* heuristic uses cross-transformation frequency analysis to discount broadly applicable features. A *DNF Model Score* leverages cumulative solver objective values as a semantic tie-breaker between structurally equivalent rules. A *Simple Minimal Transformation Coverage* heuristic extends the greedy set-cover procedure, with secondary criteria based on correspondence ambiguity and transformation complexity. An ablation study on 400 ARC-AGI-1 tasks shows these optimizations increase accuracy from 32 to 36 solved tasks with no regressions and negligible computational overhead. Analysis of rule complexity reveals that generated DNFs consistently require only a single clause, suggesting that the current performance bottleneck lies in the upstream pipeline rather than in concept complexity itself.

1 Introduction

A long-standing goal in computer science is program synthesis: the ability of a system to autonomously generate an executable program that satisfies a given high-level specification. The computational value of this automated problem-solving could be immense. With the rise of AI, some major advances have been made in this field, as the ability of AI models to write their own programs based on user input is impressive.

However, as the Abstraction and Reasoning Corpus (ARC) dataset [2] has shown, there are major limitations to what AI systems can achieve. The ARC dataset consists of tasks (Figure 1) that are grid-based visual puzzles lacking explicit instructions or context. As there are only a few input-output demonstrations, these tasks require recognizing abstract concepts that AI systems currently struggle to process. In the original ARC paper, this capability is described as "fluid intelligence": the ability to reason, solve novel problems, and adapt to new situations. It seems likely that AI models are not suited well to solve problems like those in ARC without extreme computation costs. Not only in its training, but also in the inference for each specific task.

BEN [7] is a prototype system that has a very different approach to tackling the problems that are in ARC. It tries to follow a more human approach to solve these abstract problems, inspired by *analogical reasoning*. It follows a Divide, Align and Conquer approach. First, it splits the input and output into individual objects. Then, it matches input objects to their potential output counterparts. Next, it searches for a transformation that successfully maps the inputs to those outputs. Finally, BEN enters a concept learning phase. Having found the transformations, it must discover the underlying "rules" that dictate when they apply. These rules function essentially as conditional `if`-statements within the final generated program, ensuring transformations are only applied to the correct objects.

While highly promising as one of the fastest ways to solve ARC tasks, BEN's original

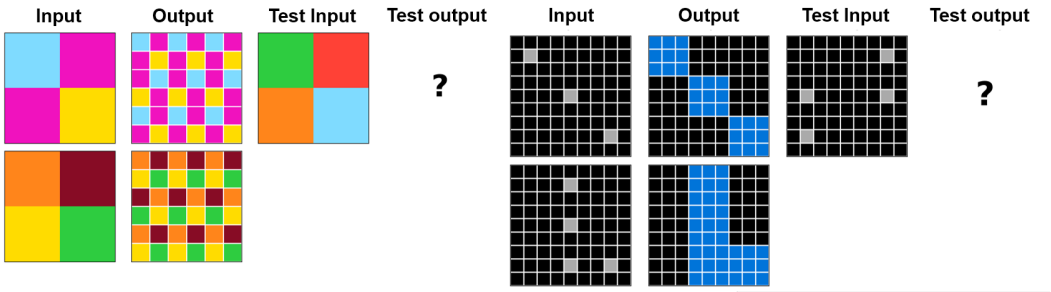

Figure 1: Two examples of ARC-AGI-1 tasks. The solver must deduce the underlying logic from the given input-output demonstrations to generate a correct output.

implementation left little time for optimization. Therefore, this paper focuses specifically on this final concept learning phase. By altering and optimizing BEN's rule-discovery mechanism, this paper aims to improve BEN's overall efficiency in both speed (computation cost) and accuracy (number of successful tasks).

2 Background

2.1 The ARC Benchmark

The ARC-agi-1 dataset is one of the premier benchmarks for AI systems to test their fluid intelligence. In the past 1.5 years systems like Claude, Gemini, ChatGPT have gotten vastly more accurate in solving its tasks. These models have made advancements, but it seems likely there has been some optimization for this benchmark as well. OpenAI highlighted the test-time reasoning capabilities of their o3 model by showcasing its breakthrough performance on the ARC-AGI benchmark during a live event [3].

Currently, with the most advanced models, the accuracy has reached high levels (80-90%), but the computation cost has remained particularly high, with around 0.50-1.50 dollars in tokens *per task* at current prices [1]. The test dataset contains around 400 tasks, meaning that running these models on these types of tasks is not feasible in most instances.

BEN, however, does not rely on the massive neural network inference these models rely on. Instead, it utilizes a programmatic approach inspired by human analogical reasoning, making it much faster and cheaper to run. To understand where these differences come from, let us look at the core of BEN's architecture.

2.2 The Original BEN Architecture

We will provide a brief summary of BEN's inner workings. For a complete description, we refer to the original paper [7].

As shown in Figure 2, BEN first decomposes each input-output grid pair into sets of input and output *objects* (connected components identified by a segmentation procedure). These objects are then aligned to establish correspondences between objects based on structural similarity. Each resulting *correspondence* pairs an input object with an output object. BEN then attempts to construct a *transformation*, composed from a library of pre-defined transformation primitives. These transformation map the input object to its corresponding

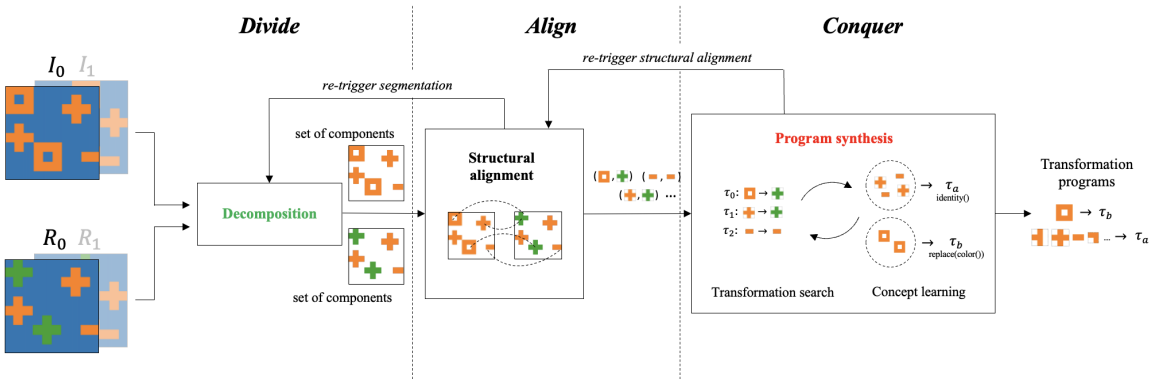

Figure 2: A structural overview of the BEN architecture. Inputs are first decomposed into sets of components and aligned into input-output correspondences. Next, BEN synthesizes a transformation for each correspondence and learns *concepts* (Boolean rules) that dictate when to apply them. Together, these concepts and transformations constitute the final executable program. Image from [7]

output object. Transformations are ranked by their coverage. Meaning the number of correspondences across all training examples that the transformation can successfully explain.

Concept Learning. The concept learning phase iteratively selects transformations and learns Boolean rules that determine which input objects each transformation should be applied to.

In each iteration, BEN selects the transformation with the highest coverage of still-uncovered output objects and splits its correspondences into *unambiguous* (output object produced by exactly one input object) and *ambiguous* (output object produced by more than one input object) sets.

BEN first learns a *concept* from the unambiguous correspondences alone, using the procedure described in Algorithm 1. A concept takes the form of a Disjunctive Normal Form (DNF) formula: a disjunction (OR) of conjunctive clauses, each clause being a conjunction (AND) of literals over object properties. These properties include both intrinsic features (e.g., color, size, position) and relative features (e.g., ranked position or ranked size among objects in the same grid). Each property-value pair generates two binary features: one for equality and one for inequality. This yields a binary feature matrix that serves as input to the DNF learner. The input objects from unambiguous correspondences serve as *positive* examples, all input objects that appear in neither the unambiguous nor the ambiguous set serve as *negative* examples, and the ambiguous input objects are held out.

Algorithm 1 greedily builds the DNF by iteratively finding clauses that cover uncovered positives without covering negatives, repeating three times with previously found literals banned to escape local optima, and retaining the least complex result (fewest total literals).

BEN then resolves ambiguous correspondences: for each ambiguous output object, it tentatively adds each candidate input object as a positive and re-learns the DNF, keeping whichever candidate yields the least complex result. Resolved output objects are removed from remaining transformations' correspondence lists, updating their coverage, and the process repeats with the next-highest-coverage transformation until all output objects are covered or no transformations remain.

Algorithm 1 Optimal Constrained DNF Learner

Input: X: component representations $\{0,1\}^{(n,m)}$, Y: labels of $\{0,1\}^n$
Parameters: j: max number of conjunctions, k: number of search iterations (e.g., 3)
Output: DNF_{opt}
$DNF_{opt} \leftarrow []$
$c_{min} \leftarrow \infty$ ▷ Tracks minimum complexity
$B \leftarrow []$ ▷ Banned clauses to escape local optima
for *iter* in $1..k$ **do**
    $DNF \leftarrow []$
    $P^+ \leftarrow \{o_i \mid o_i \in X \wedge Y[o_i] == 1\}$
    $N^- \leftarrow \{o_i \mid o_i \in X \wedge Y[o_i] == 0\}$
    for i in $1..j$ **do**
        $conj \leftarrow$ solve on P^+ and N^- subject to banned clauses B
        Add $conj$ to DNF
        Remove from P^+ all components covered by $conj$
        if P^+ is empty **then**
            break ▷ Perfect classifier for this iteration, done
        end if
    end for
    $c \leftarrow$ complexity(DNF) ▷ Sum of conjunction lengths
    if $c < c_{min}$ **then**
        $DNF_{opt} \leftarrow DNF$
        $c_{min} \leftarrow c$
    end if
    Add all $conj \in DNF$ to B ▷ Ban clauses for next iteration
end for
return DNF_{opt}

Finally, BEN performs a final concept-learning pass: for each selected transformation, it re-learns the DNF from scratch using the final positive examples and the remaining input objects as negatives. These final concepts, paired with their transformations, constitute the program.

3 Challenges in Concept Learning (Problem Description)

The ARC-AGI-1 benchmark was specifically designed to be difficult for AI systems to solve by enforcing extreme data scarcity, providing only two to four training examples for each task. This data scarcity poses a significant challenge for BEN's concept learning as well, as the solver must frequently deduce multiple entangled concepts simultaneously without overfitting to the small sample size. For instance, in ARC task a85d4709 (Figure 3), the required transformation depends on both the x-coordinate (to determine line color) and the y-coordinate (to determine which line is colored). Because the training examples are so limited, the system must isolate and learn both rules concurrently, even though it never sees all possible variations.

Furthermore, as established in the original BEN paper [7], the inductive bias of ARC tasks is strong: a valid concept must cover all positive examples while strictly excluding any

negative ones. Any DNF formula must achieve perfect classification, as a single misclassified component results in a failed task. Due to the scarcity of data, a complex formula can often achieve this perfect training accuracy simply by memorizing the exact properties of the given examples. However, this overfitted concept will most likely fail when applied to an unseen test input.

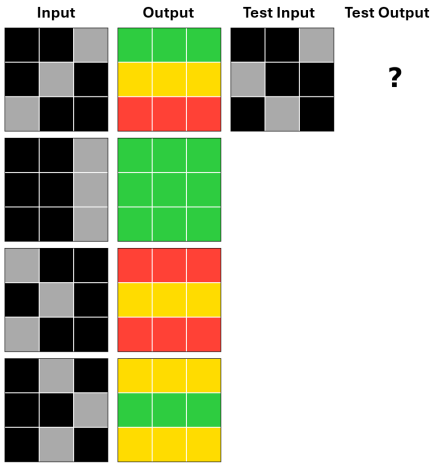

Figure 3: ARC task a85d4709. The transformation depends on both the x-coordinate (determining line color) and the y-coordinate (determining which line is colored). The limited number of examples requires the system to simultaneously learn and isolate both concepts without seeing every spatial variation.

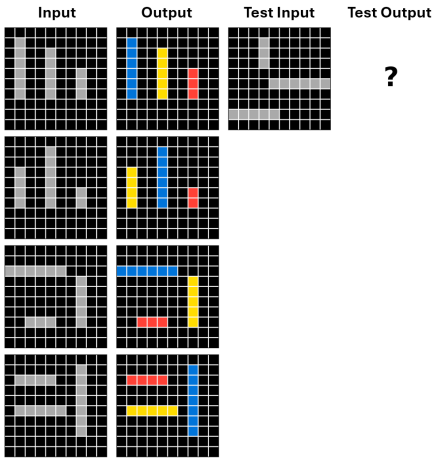

Figure 4: ARC task ea32f347. The transformation depends on object size. However, in the provided training examples, the largest object consistently possesses the highest y-coordinate. This coincidental correlation breaks in the test input, leading to potential overfitting if the highest y-coordinate is prioritized over object size.

Consequently, a concept will only successfully generalize to test inputs if it captures the fundamental, underlying simplicity of the transformation, a process humans inherently perform via analogical reasoning [4]. The objective of concept learning, therefore, is to synthesize the simplest valid rule.

But how is computational simplicity defined in a way that truly mimics human reasoning? When humans solve ARC tasks, they naturally filter out coincidental data and focus on generalized relational properties [5]. This challenge is illustrated by ARC task ea32f347 (Figure 4). Most human observers quickly determine that the transformation rule relies on relative object size [6]. However, within the limited training examples, the largest object coincidentally also has the highest y-coordinate. When evaluating candidate DNFs, the baseline BEN implementation is equally likely to select the relative y position feature as it is to select relative size, because both features yield a valid rule with the same complexity (number of literals). If the solver arbitrarily chooses the y-coordinate feature, the rule will fail on the test input, where this coincidental correlation no longer holds.

Therefore, defining a "simple" concept requires a more sophisticated heuristic than merely minimizing the total number of DNF clauses. To efficiently find when to transform an object, the system must be able to better model this simplicity in some way.

4 Methods

This section outlines the methodological framework and the core algorithmic advancements introduced to BEN. We begin by detailing the reimplementation of the baseline architecture, which transitions the concept learner to a Mixed-Integer Linear Programming (MILP) formulation to give us structural flexibility. Building upon this foundation, we then present a suite of targeted algorithmic optimizations applied to the BEN's concept learning phase.

4.1 Reimplementation in Julia

To establish a more robust and extensible foundation, the core architecture of BEN was completely reimplemented in the Julia programming language to leverage Herb.jl, a novel program synthesis framework.

However, to determine *when* to apply these synthesized transformations, the system relies on a DNF concept learner. For this reimplementation, the DNF learner was rebuilt to utilize a MILP solver. MILP is a mathematical optimization framework that minimizes or maximizes a defined linear objective function subject to linear constraints, utilizing discrete integer variables (in our case, binary decisions representing the inclusion of specific logical clauses).

This is a departure from the Constraint Programming (CP) solver used in the original BEN architecture. CP is designed for constraint satisfaction, efficiently navigating search spaces to find any feasible solution that strictly obeys a complex set of logical rules. While the original CP solver successfully guaranteed the exactness required to find valid ARC concepts, it lacked a mechanism to easily weigh the qualitative value of one valid concept against another.

Our MILP formulation shifts the focus from pure constraint satisfaction to explicit cost optimization. While it maintains the strict exactness of the original architecture, transitioning to an MILP objective function allows us to dynamically penalize specific features and clauses. This provides the necessary mathematical flexibility to implement the heuristic-driven optimizations described below.

4.2 Algorithmic Optimizations

Building upon this new baseline, several optimizations were introduced to the Concept Learner. These improvements are designed to guide BEN toward rules that more accurately mimic human analogical reasoning, where we try to prioritize structural generality and simplicity over localized memorization.

4.2.1 MILP Objective Formulation

As established, at the base of the majority of the optimizations in the concept learner is the objective function for the MILP solver. In the original BEN implementation, during each iteration of the greedy rule-finding search, its CP solver was tasked with finding a single conjunction that maximizes the coverage of remaining positive examples, while minimizing the number of selected features. Now, with the use of an MILP solver, we preserve this primary objective of maximizing coverage, but we simultaneously minimize the *weighted complexity* of the selected features. The function is formulated to mathematically guarantee that covering an additional positive example will always supersede the penalty of adding another feature to the rule. This is formalized in the following objective function:

$$\max \left(1000 \sum_{i \in U^+} Y_i - 10 \sum_{j=1}^{m} w_j C_j \right) \tag{1}$$

Where:

- U^+ represents the subset of positive examples that remain uncovered in the current iteration.
- $Y_i \in \{0, 1\}$ is a binary decision variable indicating whether the i-th uncovered positive example is successfully covered by the proposed rule.
- m is the total number of available component features.
- $C_j \in \{0, 1\}$ is a binary decision variable indicating whether the j-th feature is actively included in the conjunction.
- w_j represents the dynamically assigned cost penalty for the j-th feature, further expanded on below.

The large constant weight (1000) applied to the coverage term, relative to the penalty term (10), effectively establishes a lexicographic optimization priority. The solver is strictly incentivized to maximize the number of positive examples covered first; the feature cost penalty acts as a secondary tie-breaker to ensure that among all rules providing maximum coverage, the solver selects the one with the lowest structural complexity.

4.2.2 Rule Generality

To synthesize more intuitive, human-like programs and avoid the overfitting described previously in Figure 4, the concept learner is designed to favor the reuse of underlying properties across different clauses. Humans naturally rely on a consistent set of core relational properties to explain a task, rather than inventing entirely new rationales for every individual transformation [5]. To computationally model this preference for feature reuse, we implemented a two-pass learning architecture in the final stage of the concept learning phase that scales penalties based on high-level properties.

In the first pass, the system generates a DNF classifier for every transformation without scaling feature costs. Following this, a global frequency analysis is conducted to determine how often each base property (e.g., color, x-coordinate) was utilized across all of the DNFs.

In the second pass, these extracted property frequencies are used to dynamically adjust the cost penalty (w_j) for each feature during the final MILP optimization. The cost is adjusted via linear interpolation, ensuring that frequently used features receive a discounted penalty (down to a minimum of 60% of their initial cost), while previously unused features retain their maximum penalty multiplier of 1.0. This scaling is formalized as:

$$w_j = c_j \times \left(1.0 - \left(\frac{f_{p(j)}}{f_{max}} \right) \times 0.4 \right)$$

Where w_j is the final weight of the j-th feature, c_j is the initial base cost of the feature, $p(j)$ denotes the underlying base property of that feature, $f_{p(j)}$ is the observed frequency of that base property during the first pass, and f_{max} is the highest frequency recorded for any single property.

By explicitly lowering the objective cost of features derived from common properties, the MILP solver is mathematically incentivized to converge to more general concepts. This resolves the tie-breaking issue illustrated in Figure 4. If the solver must choose between a localized coincidence (e.g., the highest y-coordinate) and a globally relevant feature (e.g., relative size), the frequency analysis ensures that the broadly applicable property carries a discounted cost weight. Consequently, the optimized MILP solver naturally selects the "general" rule over the overfitted alternative.

4.2.3 DNF Model Score as a Complexity Heuristic

Because BEN decomposes complex ARC tasks into numerous specific subproblems, the transformation search frequently yields competing alignments or distinct search paths that produce candidate DNFs with the exact same minimum clause count. When candidate complexities are strictly equal, the baseline BEN architecture relies on arbitrary selection to resolve these ties, which can easily lead to overfitted or incorrect programs.

To decisively resolve these ties and avoid arbitrary selection, the optimized system evaluates candidate DNF classifiers using a hierarchical, multi-objective criterion rather than relying solely on structural complexity. While minimizing the total number of generated clauses remains the primary heuristic, the system employs an aggregate "model score" as a secondary semantic tie-breaker. This score, derived from the cumulative objective values achieved by the MILP solver across its iterations, ensures that among equally simple rules, the one utilizing the most heavily favored and generalizable features is selected. This semantic evaluation is applied at two distinct architectural levels:

First, during the local multi-start search of an optimal DNF, the metric evaluates the outputs of the different search iterations, ensuring the system selects the optimal, most cohesive rule for a given set of correspondences.

Second, at a global architectural level, the metric serves as a heuristic for final correspondence disambiguation. The preceding transformation search phase frequently yields ambiguous alignments, where multiple distinct input objects could theoretically map to the same output. To isolate the genuine causal relationship, the system evaluates the optimal DNF classifiers generated for each competing correspondence.

Consequently, the optimized system has a more nuanced measure of simplicity, enabling it to select the most conceptually elegant rule even when competing classifiers have identical structural lengths.

4.2.4 Simple Minimal Transformation Coverage

In the baseline BEN architecture, the rule finding phase employs a greedy set cover approach to build the final solution program. It iteratively selects the transformation program that successfully reconstructs the highest number of uncovered output objects. However, due to the inherent scarcity of training examples in ARC tasks, a genuinely correct transformation often covers a mathematically small number of correspondences (e.g., two to four). Consequently, ties in coverage frequency are highly prevalent, leading the baseline system to arbitrarily choose between transformations that cover an equal number of output objects.

As detailed in our problem description, the strict inductive bias of ARC tasks demands fundamentally simple concepts to ensure successful generalization to unseen test inputs. Arbitrary selection at this stage directly undermines this objective. If the system inadvertently prioritizes an overly complex transformation simply to resolve a tie, it forces the concept learner to justify an inherently overfitted, brittle rule.

To prevent this and align the chosen transformations with the requirement for conceptual simplicity, we introduce a hierarchical, three-tier evaluation heuristic for selecting the optimal transformation during each iteration:

1. **Maximum Output Coverage (Primary):** The system strictly prioritizes the transformation that maximizes the number of currently uncovered output objects it can successfully reconstruct.

2. **Minimum Ambiguity (Secondary):** If competing transformations yield identical coverage, the system evaluates the correspondence ambiguity generated by each transformation. Ambiguity is quantified as the difference between the total number of valid component correspondences and the number of unique output objects they map to (i.e., the frequency of many-to-one mappings). By selecting the transformation with the fewest ambiguities, the system inherently prioritizes mappings with a higher degree of causal certainty.

3. **Minimum Transformation Complexity (Tertiary):** If both coverage and ambiguity are equal, the system applies the principle of parsimony to the transformation itself. It selects the candidate with the lowest structural complexity, defined by the length of its symbolic expression (AST node count). This ensures the system inherently favors simpler, more generalizable transformations.

By replacing the baseline's arbitrary tie-breaking with this cascading logic, the optimization ensures that the concept learning phase is consistently fed the cleanest, simplest, and least ambiguous transformation data possible.

5 Experiments

To empirically measure the impact of the proposed concept learning optimizations, we conducted an ablation study. By systematically removing individual components, we can isolate the effect of each algorithmic enhancement on BEN's accuracy, complexity, and computational efficiency of its programs.

5.1 Experimental Setup

Guided by this methodology, our evaluation is structured around four targeted research questions:

- **Q1. What is the overall impact of the combined algorithmic optimizations?**
  To answer Q1, we compare the **Baseline (Julia-BEN)**—the reimplemented Julia version utilizing the standard concept learner without optimizations—against **Optimized BEN**, the complete system incorporating all enhancements detailed in Section 3.

- **Q2. What is the isolated impact of prioritizing rule generality?**
  To answer Q2, we compare Optimized BEN against **Ablation 1**, a configuration that specifically excludes the two-pass frequency analysis and dynamic feature cost scaling.

- **Q3. How does semantic correspondence disambiguation affect performance?**
  To answer Q3, we compare Optimized BEN against **Ablation 2**, which excludes the global DNF model score heuristic, forcing the system to revert to arbitrary selection when resolving ties in structural complexity.
- **Q4. What is the impact of utilizing a hierarchical, simplicity-driven heuristic for transformation selection?**
  To answer Q4, we compare Optimized BEN against **Ablation 3**, which excludes the hierarchical evaluation heuristic and reverts to selecting transformations solely based on arbitrary tie-breaking for output coverage.

To accurately answer these research questions, we evaluate each configuration across three primary metrics: accuracy (percentage of tasks solved), rule complexity (mean clauses and total unique properties utilized), and computational efficiency (measured by both the average rule-finding time in seconds and the number of search iterations required to cover all positive examples). To ensure reliable measurement across these dimensions, all configurations were executed on a single workstation with an AMD Ryzen 9 5900X CPU and 32GB of RAM.

The evaluation was conducted across the 400 tasks of the ARC-AGI dataset using a two-phase execution strategy:

- **Phase 1: Broad Capability Assessment:** All configurations were evaluated across the complete 400-task dataset utilizing a multi-threaded architecture. This phase captures the macro-level accuracy and the structural complexity of the synthesized programs.
- **Phase 2: Deterministic Timing Analysis:** Because multi-threading introduces non-deterministic execution times due to synchronization and thread-locking, computational efficiency was measured in a strictly single-threaded environment. A subset of tasks successfully solved by *Optimized BEN* was re-evaluated three separate times per configuration, with the results averaged to ensure a reliable, isolated measure of computational cost.

5.2 Results

As detailed in Table 1, the Optimized BEN and Ablation 2 configurations achieve the highest overall task accuracy, successfully solving 36 out of 400 tasks. The remaining ablations perform marginally worse, with the Baseline and Ablation 3 exhibiting the lowest accuracy. These results indicate that the Model Score as a Complexity Heuristic does not affect the total number of solved tasks. Conversely, Rule Generality and Simple Transformation Coverage contribute positively to the overall accuracy, yielding 1 and 3 additional successful tasks, respectively.

Notably, none of the introduced optimizations cause any previously solved tasks to fail.

Regarding computational overhead, the average execution and rule-finding times remain largely consistent across all configurations. An analysis of the run-to-run execution time variance (Figure 3) reveals a mean absolute difference per task of approximately 0.03 seconds during the rule-finding phase. However, the total execution time mean absolute difference per task is roughly an order of magnitude higher. This discrepancy indicates that the majority of the run-to-run variance originates from the alignment phase and the transformation search, rather than the rule-finding process itself.

Table 1: Ablation Study: Performance comparison (Accuracy, Execution Time, Rule Finding Time). Best values are highlighted in bold.

Configuration	Task Accuracy	Avg. Execution Time (s)	Avg. Rule Finding Time (s)
Baseline (Julia-BEN)	32/400	15.71	0.26
Ablation 1 (No Rule Generality)	35/400	**15.07**	**0.25**
Ablation 2 (No Model Score)	**36/400**	15.47	0.38
Ablation 3 (No Simple Transf.)	33/400	15.69	0.39
Optimized BEN	**36/400**	15.57	0.39

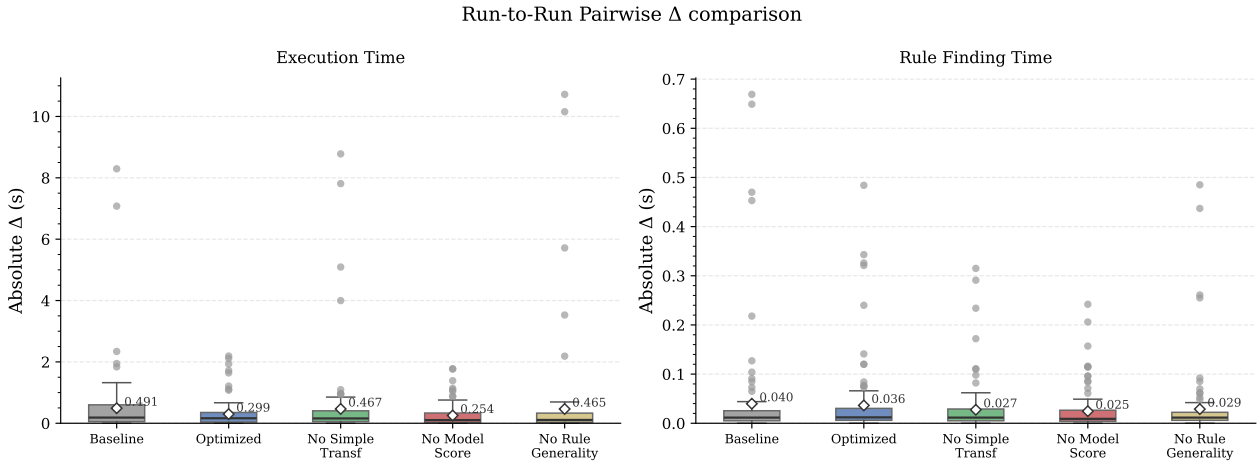

Figure 5: Run-to-run variance in execution time per task. The data demonstrates minimal variance during the rule-finding phase (mean absolute difference ≈ 0.03s across all configurations), whereas total execution time exhibits significantly higher variance. Note: To account for Julia's Just-In-Time (JIT) compilation overhead, the initial task execution is treated as a standard warmup phase and excluded from these measurements.

Rule simplicity Table 2 presents the average clause and unique property counts, which show negligible variation across configurations. This indicates that the generated DNFs maintain a consistent level of structural complexity regardless of the applied optimizations, despite the differences in task accuracy reported in Table 1.

Furthermore, the average number of iterations needed to cover all positive examples with a DNF is approximately 1 across all configurations. The only deviation occurs for the Baseline and Ablation 3, where a single failing task prevents the solver from covering all positive examples for one transformation.

6 Discussion

Performance vs. Computational Cost The results demonstrate that the proposed enhancements to the concept learning phase yield minor but meaningful improvements in overall task accuracy. While an increase of 4 solved tasks may seem numerically small, it rep-

Table 2: Rule Simplicity. **Avg. Clauses per Task** represents the mean number of total clauses generated to solve a task, **Avg. Clauses per DNF** represents the mean length of individual rules. **Avg. Unique Prop. Count** indicates the distinct properties used per task. For all three metrics, lower values denote simpler programs (lower is better). Note: Tasks requiring unconditional transformations (where rules are applied globally to all objects) are excluded, resulting in an evaluated subset of $n = 23$ tasks.

Configuration	**Avg. Clauses per Task**	**Avg. Clauses per DNF**	**Avg. Unique Prop. Count**
Baseline (Julia-BEN)	3.05	0.99	3.05
Ablation 1 (No Rule Generality)	3.09	1.00	3.00
Ablation 2 (No Model Score)	3.09	1.00	3.00
Ablation 3 (No Hierarchical Transf.)	3.05	0.99	3.05
Optimized BEN	3.09	1.00	3.00

resents a decent relative improvement over the Julia-BEN baseline. Crucially, as highlighted in the execution time analysis, this increase in accuracy incurs a negligible computational penalty. The average execution time across all configurations remains stable around 15.5 seconds per task. The fact that the run-to-run variance exceeds the time added by these optimizations indicates that the computational cost of the enhanced concept learning phase is virtually insignificant. Therefore, for Rule Generality and Simple Transformation Coverage, the accuracy gains strictly outweigh the computational costs.

Robustness of the Baseline and Analysis of Enhancements: The baseline evaluation underscores that the original concept learning formulation, while basic, is highly robust. The ARC dataset introduces a strict inductive bias: a task requires the perfect reconstruction of all outputsm meaning that a valid concept must cover all positive components and strictly exclude negative ones. A single false positive invalidates the solution. This strict separation aligns well with DNF generation. Furthermore, the fact that none of the introduced optimizations cause any previously solved tasks to fail demonstrates that the baseline's underlying logic is not disrupted by these enhancements.

However, when the solver encounters ties between multiple valid DNFs, our optimizations help the agent select the most intuitively "correct" and generalizable option. Notably, despite the negligible variation in rule complexity across configurations (Table 2), the higher task accuracy achieved by Ablation 2 and Optimized BEN suggests that these configurations produce rules that generalize more effectively. For the additionally solved tasks, these models successfully discovered abstractions that generalized robustly from the demonstration examples to the hidden test output.

Simple Minimal Transformation Coverage: This proved to be the most impactful enhancement, enabling the solution of three additional tasks. By applying Occam's Razor, the agent favors transformations that are less complex and cover less ambiguous correspondences. In few-shot environments where a single transformation might only cover a few correspondences, providing a robust metric to determine the "best" minimal transformation significantly improves downstream concept learning.

Rule Generality: Intuitively, when humans solve general problems, they prefer a single broad rule over a disjointed set of hyper-specific properties. Incorporating this preference helps the agent avoid overfitting. (See Figure 4 for a task that benefits from this).

DNF Model Score as a Complexity Heuristic: This optimization did not yield an immediate performance improvement in solved tasks. Because the generated DNFs likely share scores when their complexities are the same, this disambiguation metric is rarely the deciding factor in the current evaluation on ARC-AGI-1. However, it may become relevant if future iterations implement weighted feature scoring on top of frequency costs, which should result in higher variance in DNF scores and thus necessitate semantic tie-breaking.

Simplicity of Generated Concepts The near-unit clause count observed across all configurations (Table 2) suggests that, at least for the tasks BEN currently solves, the underlying concepts are inherently simple. Two interpretations are plausible. First, this may reflect an intrinsic property of ARC: the benchmark is designed around compositional transformations, each governed by a simple rule. Second, this simplicity may be an artifact of BEN's decomposition strategy; by splitting each task into per-object sub-problems, the resulting sub-concepts are naturally narrow in scope.

This observation, combined with the accuracy gap between our reimplementation and the original BEN (32 vs. 90 solved tasks), strongly suggests that the current performance bottleneck lies not in concept complexity but in the upstream pipeline. The baseline Julia-BEN employs a single, naive object segmentation approach, pixels are grouped into objects only if they share the same color and an orthogonal boundary, which fails to capture the underlying visual logic for many ARC tasks. When combined with potential errors in the alignment phase or transformation search, the concept learner is often presented with incorrect correspondences to begin with. Consequently, the true impact of our concept learning optimizations may be partially masked. Resolving these upstream bottlenecks would provide a stronger foundation, potentially amplifying the benefits of the enhancements introduced in this study.

7 Conclusions and Future Work

In this paper, we explored how BEN can more efficiently determine when transformation programs apply, through optimizations in the concept learning phase. By integrating heuristics for Simple Minimal Transformations and Rule Generality, we aimed to improve the agent's ability to select the most accurate and generalizable logic when interpreting ambiguous few-shot examples. Our results demonstrate that these optimizations successfully enhance BEN's task accuracy on the ARC dataset without incurring a meaningful computational penalty. Specifically, encouraging the agent to favor less complex transformations and broader rules allows it to find solutions that generalize more effectively to hidden test cases, leveraging the principles of Occam's Razor within the concept learning phase.

However, our evaluation also indicates that the downstream success of concept learning is still influenced by upstream processing. The baseline Julia-BEN reimplementation currently employs a naive object segmentation approach, which can feed malformed inputs to the alignment and transformation search phases. Consequently, the true potential of our optimized concept learner may be marginally constrained by these earlier stages.

Configuration	Task Accuracy	Exec. Time (s)	Concept Learn. Time (s)	Unique Prop./Task	Clauses /Task	Clauses /concept
Baseline	32/400	15.71	0.26	3.05	3.05	0.99
No Concept Generality	35/400	**15.07**	**0.25**	3.00	3.09	1.00
No Concept Score	**36/400**	15.47	0.38	3.00	3.09	1.00
No Simple Transf.	33/400	15.69	0.39	3.05	3.05	0.99
Optimized BEN	**36/400**	15.57	0.39	3.00	3.09	1.00

Table 3: All measurements are averages except for accuracy

To address these limitations and build upon the findings of this study, future work should explore three complementary directions.

First, improving the upstream pipeline would provide a stronger foundation for the concept learner. Replacing the current naive segmentation with a more robust and flexible module, and expanding the feature extraction pipeline to capture explicit inter-object relationships (such as containment or adjacency) would allow the concept learner to reason over a richer and more accurate input representation.

Second, the heuristic framework introduced here can be extended further. The frequency-based logic underlying Rule Generality could be applied to transformation selection itself, dynamically guiding the search space when tasks require structurally similar transformations across multiple objects. Similarly, implementing weighted feature scoring alongside frequency costs would create wider variance in DNF scores, making the Model Score heuristic a more effective disambiguation tool.

Third, the observation that successfully solved tasks consistently require only a single clause per DNF suggests a useful diagnostic signal: high clause counts likely indicate upstream errors in segmentation, alignment, or transformation search, rather than genuine concept complexity. Future iterations could leverage this metric as a trigger for backtracking, discarding overly complex hypotheses in favor of re-exploring alternative correspondences. Such a mechanism would create a feedback loop between concept learning and the upstream pipeline, enabling the system to self-correct rather than propagate errors forward.

More broadly, the heuristics validated here address inherent ambiguity in program synthesis beyond this benchmark. Future research could investigate applying this optimised concept learning approach to broader, real-world program synthesis tasks where noise-tolerant rule generation is critical.

8 Responsible Research

Reproducibility The code that was written for this project, and solver configurations used for the described experiments, can be found at https://github.com/Herb-AI under the name DivideAlignConquer.jl and DivideAlignConquerExperiments.jl.

Use of AI tools In accordance with TU Delft policy, the author discloses the use of generative AI tools for language editing and basic coding support (e.g., boilerplate generation and debugging). The author assumes full responsibility for all design choices, experimental methodologies, and final results. No AI-generated content was incorporated without strict manual verification against the underlying data.

References

[1] ARC Prize Foundation. ARC-AGI Leaderboard, 2026. Accessed: 2026-06-20.

[2] François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

[3] François Chollet. Openai o3 breakthrough high score on ARC-AGI-Pub, 12 2024.

[4] Dedre Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2):155–170, 1983.

[5] Alice Johnson, Wai Keen Broderick Vong, Brenden M. Lake, and Todd M. Gureckis. Fast and flexible: Human program induction in abstract reasoning tasks. *arXiv preprint arXiv:2103.05823*, 2021.

[6] Solim LeGris, Wai Keen Vong, Brenden M. Lake, and Todd M. Gureckis. A comprehensive behavioral dataset for the abstraction and reasoning corpus. *Scientific Data*, 12(1):1380, 2025.

[7] Jonas Witt, Sebastijan Dumančić, Tias Guns, and Claus-Christian Carbon. A divide-align-conquer strategy for program synthesis. *arXiv*, 2023.

A Detailed Task-Level Performance

Table 4: Pass/Fail status across configurations for the subset of tasks that succeed under Optimized BEN.

Task Name	Baseline	No Rule Generality	No Model Score	No Simple Transf	Optimized
08ed6ac7	✓	✓	✓	✓	✓
0ca9ddb6	✓	✓	✓	✓	✓
0d3d703e	×	✓	✓	×	✓
1bfc4729	✓	✓	✓	✓	✓
1cf80156	✓	✓	✓	✓	✓
1f85a75f	×	✓	✓	×	✓
23b5c85d	✓	✓	✓	✓	✓
25ff71a9	✓	✓	✓	✓	✓
27a28665	✓	✓	✓	✓	✓
31aa019c	✓	✓	✓	✓	✓
3ac3eb23	✓	✓	✓	✓	✓
3c9b0459	✓	✓	✓	✓	✓
48d8fb45	✓	✓	✓	✓	✓
496994bd	✓	✓	✓	✓	✓
6150a2bd	✓	✓	✓	✓	✓
67385a82	✓	✓	✓	✓	✓
67a3c6ac	✓	✓	✓	✓	✓
68b16354	✓	✓	✓	✓	✓
6e82a1ae	✓	✓	✓	✓	✓
74dd1130	✓	✓	✓	✓	✓
9dfd6313	✓	✓	✓	✓	✓
a61f2674	✓	✓	✓	✓	✓
a79310a0	✓	✓	✓	✓	✓
a9f96cdd	✓	✓	✓	✓	✓
aedd82e4	✓	✓	✓	✓	✓
b1948b0a	✓	✓	✓	✓	✓
b60334d2	✓	✓	✓	✓	✓
b9b7f026	×	✓	✓	×	✓
be94b721	✓	✓	✓	✓	✓
c8f0f002	✓	✓	✓	✓	✓
d037b0a7	✓	✓	✓	✓	✓
d364b489	✓	✓	✓	✓	✓
d511f180	✓	✓	✓	✓	✓
e9afcf9a	✓	✓	✓	✓	✓
ea32f347	×	×	✓	✓	✓
ed36ccf7	✓	✓	✓	✓	✓