

# 000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 ARCMEMO: ABSTRACT REASONING COMPOSITION WITH LIFELONG LLM MEMORY

**Anonymous authors**

Paper under double-blind review

## ABSTRACT

While inference-time scaling enables LLMs to carry out increasingly long and capable reasoning traces, the patterns and insights uncovered during these traces are immediately discarded once the context window is reset for a new query. External memory is a natural way to persist these discoveries, and recent work has shown clear benefits for reasoning-intensive tasks. We see an opportunity to make such memories more broadly reusable and scalable by moving beyond instance-based memory entries (e.g. exact query/response pairs, or summaries tightly coupled with the original problem context) toward **concept-level memory**: reusable, modular abstractions distilled from solution traces and stored in natural language. For future queries, relevant concepts are selectively retrieved and integrated into the prompt, enabling test-time continual learning without weight updates. Our design introduces new strategies for abstracting takeaways from rollouts and retrieving entries for new queries, promoting reuse and allowing memory to expand with additional experiences. We evaluate on ARC-AGI, a benchmark that stresses compositional generalization and abstract reasoning, making it a natural fit for concept memory. Our method yields a **7.5%** relative gain over a strong no-memory baseline, with performance continuing to scale with inference compute. We find abstract concepts to be the most consistent memory design, outscoring the baseline at all tested inference compute scales. Moreover, dynamically updating memory during test-time outperforms fixed settings, supporting the hypothesis that accumulating and abstracting patterns enables further solutions in a form of self-improvement.

## 1 INTRODUCTION

Large language models (LLMs) have made substantial strides in reasoning-intensive tasks with long-form reasoning. However, a notable opportunity lies in the fact that LLM systems are fixed after deployment: new patterns and strategies uncovered during deep reasoning are not yet carried forward once the context is cleared. This contrasts with human approaches to solving complex, compositional reasoning problems, which involve building on prior insights, abstracting patterns, and composing them in new contexts. Augmenting LLMs with memory offers a natural solution to retaining and building on their discoveries.

While external augmentation was initially designed for factuality in knowledge-intensive tasks (e.g. Lewis et al. (2021) with RAG, Zhang et al. (2019) with Knowledge Graphs, *inter alia*), recent work has begun developing memory approaches for reasoning-intensive tasks, storing questions, findings, and mistakes in memory instead of simple facts (Yang et al., 2024). These memories tend to be specific to the problem/experience it was originally derived from (we term these “instance-level” concepts, see Figure 1). Although effective for closely related problems, they have diminished utility for problems superficially different from prior experiences. Other approaches, such as Suzgun et al. (2025), begin addressing the “instance-level” issue by maintaining an evolving summary of previous experience to great effect, but the lack of structure and modularity makes scaling with more experiences challenging. Without selective retrieval, the size of memory is limited, and the problem solving model has the additional burden of picking out the relevant ideas from all past experiences.

We introduce an *abstract concept-level memory* framework to support compositional reasoning for all subsequent queries. We emphasize (1) **abstract concepts** that are more general and separated from their original context to be useful across a larger set of future problems, and (2) **modular**

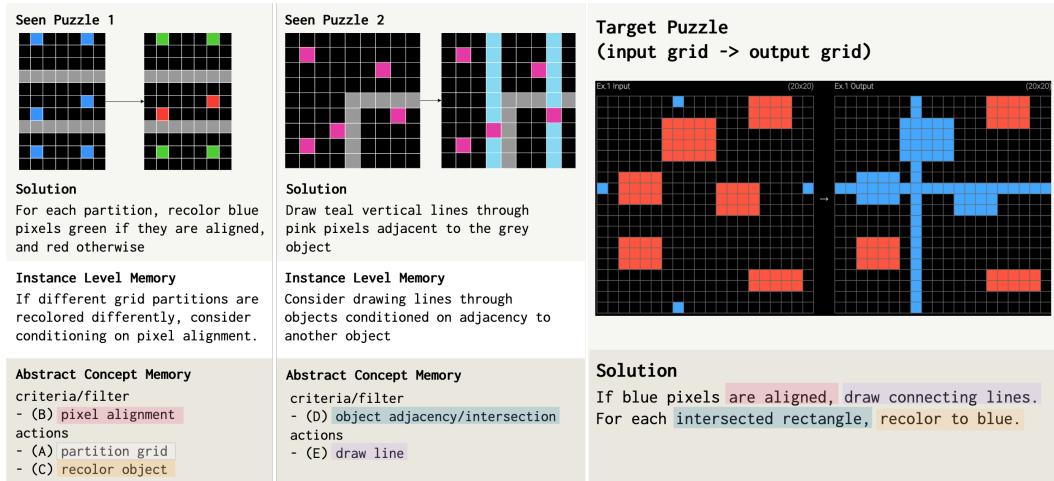


Figure 1: **Instance-Level vs. Abstract Concepts Example.** Each ARC-AGI Chollet (2019) puzzle requires inferring the transformation rule for a set of input/output pixel grids. Here, Puzzle 1 instantiates  $(A \wedge B) \Rightarrow C$  and Puzzle 2 instantiates  $D \Rightarrow E$ . The target puzzle is solved by recombining these ideas  $(B \Rightarrow E, D \Rightarrow C)$ . Instance-level memory tends to store fully composed rules, coupling  $A$  with  $B, C$ , and so on. Transferring to the target then demands both ignoring  $A$  and disentangling/reordering  $B, C$  with  $D, E$ . Abstract memory instead stores  $A, B, C, D, E$  as separate, modular concepts, making them easier to recognize and reassemble in new contexts.

**concepts** that directly promote recombination with other ideas that can be easily built on with new experiences or curated for a target problem. We distill solution traces (both from the system itself or from external sources) into reusable, modular abstractions stored in natural language. At inference time, relevant concepts are selected and integrated into context, enabling test-time continual learning without expensive weight updates. Our design encapsulates two primary operations: (i) writing to memory: formulating concepts for abstraction and generality; and (ii) reading from memory: selecting a subset of concepts for the current problem. We title our method **ArcMemo**—Abstract Reasoning Composition Memory.

As illustrated in Figure 1, instance-level memories often capture an entire solution pattern tied closely to its original problem (e.g., the joint use of ideas  $A, B, C$  in Seen Puzzle 1). Such overspecified entries are less likely to recur in future problems, and even when partially relevant to the Target Puzzle, the agent must still disentangle useful pieces like  $B$  from the original bundle. In contrast, abstract concepts are stored individually with fewer contextual assumptions, making them easier to recognize, adapt, and recombine across superficially different puzzles.

We evaluate on ARC-AGI, where simple pixel-grid operations compose into a vast task space, and solving tasks requires compositional reasoning rather than memorizing individual patterns, making it a natural testbed for ArcMemo. On ARC-AGI-1, our method improves the official score from 55.17 to 59.33 (+7.5% relative gain over a strong no-memory baseline) and is the only memory design we find to outperform the baseline at all inference scales. Our experiments confirm that continually updating memory at evaluation time yields a score improvement that emerges with inference scale via retries: memory updates triggered from a previous pass over the test set may enable new solves in a subsequent pass. Finally, we observe that selecting a subset of memories to include for a particular problem improves performance and reduces token cost, indicating the selection operation is essential beyond allowing memory to grow continually.

## 2 RELATED WORK

Our approach draws on several threads of research in augmenting LLMs with memory but contrasts in (1) target applications (reasoning-intensive vs. knowledge-intensive tasks), (2) the underlying memory modality (text vs. continuous vectors/embeddings), and (3) emphasis on abstraction and modularity for reasoning. We discuss related lines (1) and (2) as well as other related work in Appendix E and emphasize reasoning and abstraction here.

**Memory for Test-Time Reasoning and Concept Abstraction.** More recent work has shifted toward reasoning-centric uses of memory, particularly test-time learning. Think-in-Memory records intermediate reasoning steps as structured triples, storing them in a locality-sensitive hash table to enable reuse in multi-step reasoning tasks (Liu et al., 2023). Buffer of Thoughts stores problem specific reasoning templates, retrieved by embedding similarity and instantiated for new inputs (Yang et al., 2024). It categorizes and summarizes solution attempts into templated insights, which are then added to memory. In contrast, Dynamic Cheatsheet maintains a unified buffer that is adapted continuously. With each problem query, the LLM updates this memory blob by rewriting the entire buffer (Suzgun et al., 2025). Rather than retrieving specific entries, the entire cheatsheet is appended to the prompt, functioning as a persistent cache of problem-solving strategies.

**119 ARC-AGI Approaches.** Our work is primarily evaluated against ARC-AGI as a benchmark that  
120 simulates complex reasoning on frontier tasks without requiring frontier-level knowledge. Various  
121 techniques and findings have been developed against this benchmark. Li et al. (2024) demonstrates  
122 the efficacy of synthetic data and the complementary approaches of (i) learning to infer the underlying  
123 transformation via program synthesis and (ii) using test-time weight adaptation to directly learn the  
124 transformation function in the neural net. Akyürek et al. (2025) further investigates the test-time  
125 adaptation method, producing various insights, including that low rank adapters are poorly suited  
126 to retaining practical experience across puzzles. Our program synthesis approach can be viewed  
127 as a memory-augmented version of Wang et al. (2024)'s hypothesis search or Qiu et al. (2024)'s  
128 hypothesis refinement with execution feedback-based retry. Other recent approaches to ARC-AGI  
129 include brain-inspired architectural innovations such as Wang et al. (2025)'s hierarchical reasoning  
130 model, achieving impressive performance through a recurrent architecture that iteratively refines  
131 its output until a halt signal is predicted, together with data augmentation techniques. Finally, the  
132 relative ease of generating new ARC-AGI tasks compared to solving them is exploited in Pourcel  
133 et al. (2025)'s self-improving system that relabels incorrect programs as the correct program for a  
different puzzle and trains a model on its generations this way.

### 3 METHODS

### 3.1 PROBLEM STATEMENT

**Algorithm 1:** Inference with Continually Updating External Memory

**Input:** Dataset  $\mathcal{D} = \{x_i, y_i\}_{i=1}^n$  (labels  $y_i$  optional), External memory  $M$ , Operations MEMREAD, MEMWRITE, GETFEEDBACK, Update interval  $k$ ,

**Output:** Predictions  $\{\hat{y}_i\}$

**for**  $i \leftarrow 1$  **to**  $n$  **do**

```

 $(x_i, y_i) \leftarrow \text{GETITEM}(\mathcal{D}, i);$  // Label  $y_i$  may be absent at inference
 $s_i \leftarrow \text{MEMREAD}(M, x_i);$  // Retrieve relevant memory entries
 $\hat{y}_i \leftarrow \text{LLM\_GENERATE}(x_i, s_i);$  // Predict using LLM + selected memory
if  $i \bmod k = 0$  then
     $f_i \leftarrow \text{GETFEEDBACK}(\hat{y}_i, y_i);$  // test verification, reflection, etc.
     $\text{MEMWRITE}(M, x_i, \hat{y}_i, f_i);$  // Incorporate feedback into memory

```

We formulate a memory system as a collection of memory entries associated with read and write operations. Problem solving with memory augmented LLMs thus requires designing three components as seen in algorithm 1:

1. **Memory Format** (3.2): What is stored in individual entries?
  2. **Memory Write** (3.3): How is memory updated from a reasoning trace?
  3. **Memory Read** (3.4): How is memory used for tackling new problems?

These are generic design considerations for any memory. Our method's novelty lies in the choices made to target (1) abstracting a memory to be less specific to the problem it was derived from (2) enabling memory selection, since greater abstraction makes standard retrieval approaches (embedding similarity thresholding) less effective. We present two such implementations: (1) an open-ended (OE) format that imposes minimal constraints on entry format and uses a simple problem preprocessing

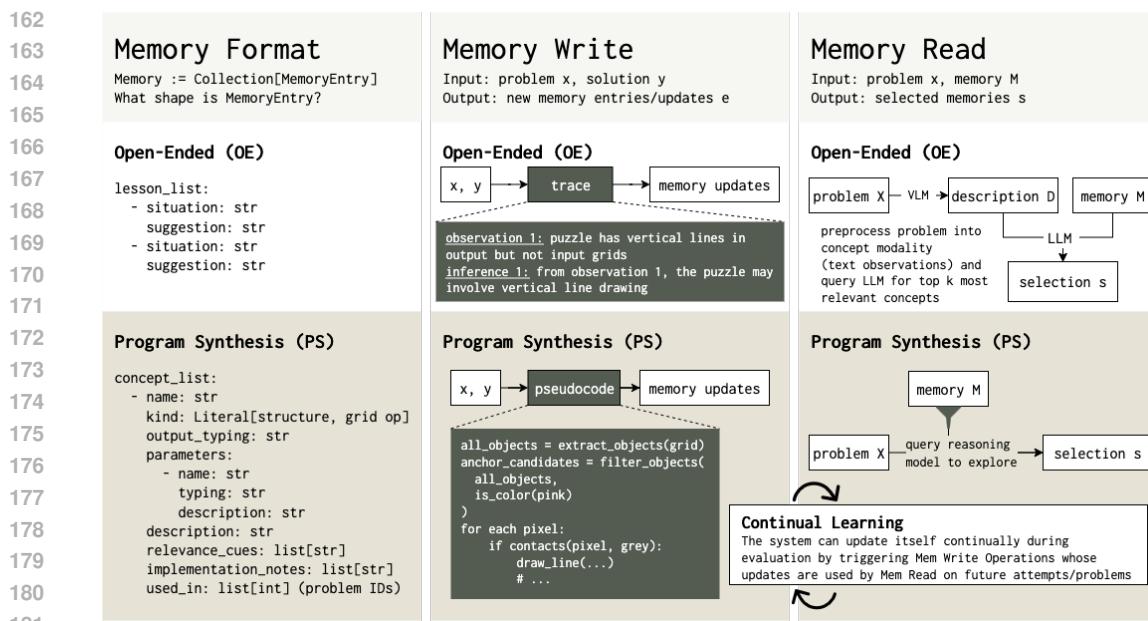


Figure 2: **Method Diagram.** Implementing a memory system requires defining (1) what is stored, (2) how memory is updated, and (3) how memory is used for new queries. The key novelty in this work is emphasizing abstraction and modularity, and the corresponding design changes. In particular, we highlight that parameterization (with higher-order functions allowed and encouraged) promotes abstraction, and typed interface definitions support modularity by showing which concepts can be combined. Since these memory entries are more abstract, they also require more inference to map against new, concrete situations—whether by aligning input against the memory format in a preprocessing query, or leveraging reasoning models to explore in a directed manner.

step for memory selection, and (2) a program-synthesis (PS) inspired format that categorizes and parameterizes concepts and uses reasoning model exploration to select relevant items.

**Problem Assumptions.** The main requirement for our memory system is that some feedback is available at test-time (e.g. via test cases or self-reflection). This condition is necessary as retaining patterns from an incorrect or otherwise flawed trace would serve to carry mistakes forward. In other words, some signal is needed to discriminate correct from incorrect traces to ensure only productive ideas and patterns are added to memory. Incorrect traces still likely contain vital signal, but error identification/credit assignment remains an open challenge that we leave to future efforts. Various real-world tasks may satisfy this requirement, as tasks are often specified with either examples showing desired behavior or evaluation criteria defining what outcomes are positive and negative. To give some examples, code completion tools have criteria like compilation success and test pass rate, and medical diagnosis tools have evaluation criteria defined as patient outcomes (e.g., survival rates). In the case of our evaluation benchmark ARC-AGI, puzzles are explicitly presented as a set of input-output examples, which, for our purposes, can serve as automatically verifiable test cases.

### 3.2 MEMORY FORMAT AND ORGANIZATION

Specifying an individual memory entry’s format involves selecting fields useful for retrieval and downstream problem-solving. The organization of memory entries on a collective level is another design surface we leave for future exploration; we use a flat collection of entries for simplicity.

**Open-Ended (OE) Formulation.** We first consider an open-ended approach that imposes minimal structure and defers formatting individual memories to the model. Following the basic lesson format of “under situation X, do action Y”, the only formatting constraint we impose is having distinct situation (X) and suggestion (Y) fields. This separation is designed for abstraction

216	<b>Example Puzzle</b>	
217	<b>Input 1</b>	
218		
219		
220		
221		
222		
223		
224	<b>Output 1</b>	
225		
226		
227		
228		
229		
230		
231		
232		
233	<b>Open Ended (OE)</b>	
234	<b>situation:</b> When the input grid contains only a few distinct nonzero colors and the counts of these colors vary, with the output grid dimensions seeming to relate to those counts.	
235		
236	<b>suggestion:</b> Count the occurrences of each nonzero color. Use these counts to infer the output grid's dimensions (for example, the maximum frequency may define the number of rows, and the number of distinct colors defines the number of columns). Try ordering the colors (such as from highest to lowest frequency) and filling each output column with the corresponding color repeated according to its count, using the background value for any remaining cells.	
237		
238	<b>Program Synthesis (PS)</b>	
239	- concept: unique colors {...}	
240	- concept: count pixels {...}	
241	- concept: sort objects	
242	<b>parameters:</b>	
243	- objects: list[object]	
244	- key: Callable[[object], Any]	
245	<b>relevance cues:</b>	
246	- if output colors or properties	
247	are assigned based on object	
248	ranking (e.g., tallest, smallest)	
249	- if objects need to be processed	
250	in a specific spatial order	
251	(e.g., left-to-right)	
252	<b>implementation notes:</b>	
253	- use Python's sorted() with a	
254	lambda extracting x or y coords	
255		

Figure 3: **Open-Ended (OE) vs. Program Synth (PS) Concept Examples.** An example of concepts from puzzle 9af7a82c abstracted into each concept format. OE defers to the model, while PS imposes structure to encourage abstraction/modularity. Higher order behavior is demonstrated with the “sort objects” concept taking a Callable parameter that specifies specific variations.

(disentangles the core idea from the original context) and modular reuse (situation explicitly describes the conditions where this idea can be reused).

**Program Synthesis (PS) Formulation.** In early iterations of automatic concept summarization from solution traces, we observed outputs to occasionally be overly specific. Consider Figure 3’s OE example that bundles ideas of counting, drawing columns, and sorting into a single entry. This overspecification inherits the same limitations as instance-level concepts: friction when matching a composite idea to new scenarios and redundancy from composite ideas sharing components.

We designed a more structured alternative to the OE format to address these issues and more strongly enforce the ideals of abstraction and modularity. In particular, we took inspiration from software engineering and functional programming, which respectively have existing solutions in modular design for code reuse and higher-level functions for composition. From this analogy, our Program Synthesis (PS) format frames concepts as types, structures, and routines. The primary feature is that each of these concepts is parameterized. Parameterization allows similar concepts to be represented compactly with variations abstracted into parameters. Moreover, type annotations for inputs (parameters) and outputs (return types) encourage modularity and composition: the typed interfaces suggest what ideas fit together and how. Finally, by allowing parameters to be routines themselves (introducing higher-order functions), we encourage further generalization by enabling abstraction over specialized logic and routines rather than just values. Higher-level patterns can be recognized across instances and stored with routine arguments defining the different lower-level logic. A subtle benefit of this more structured formulation is that memory representation can be easily compressed by omitting certain fields (see subsection A.1 for a complete list of PS concept fields).

### 3.3 MEMORY WRITE: CONCEPT ABSTRACTION

Memory’s primary purpose is to persist discoveries and reflections from prior experiences. Converting experiences to memory updates is then the most crucial operation for achieving persistence.

**OE Abstraction.** Abstracting suggestions from a solution trace is straightforward—simply query a model to reflect on the solution trace and summarize specific general ideas that may be reused for future puzzles. To synthesize specific conditions or situational cues to pair with suggestions, ideally, we can directly refer to the original problem-solving process as a series of deductions that derive new facts or explore ideas from existing facts, and link each suggestion (idea) with the prior observations that inferred this suggestion. However, intermediate inferences may be implicit or

270 difficult to extract from a trace, or, in the case of specific commercial models, completely hidden. To  
 271 address this challenge, we generate a post-hoc derivation: an interleaved sequence of observations and  
 272 thoughts/reasoning steps constructed from the input-output of the final solution. These reconstructed  
 273 traces are then used to extract situation-suggestion pairs, forming structured memory entries for  
 274 guiding future analyses. This minimal formulation does not consider the existing contents of memory  
 275 while adding new entries for simplicity and scaling reasons. It defers handling redundancy and  
 276 consolidating ideas to the concept selection and problem-solving phases, respectively.

277 **PS Abstraction.** We find that directly converting solutions into routines often leads to minor  
 278 implementation details being recorded as memories, as opposed to our intended abstract concepts.  
 279 We mitigate this by preprocessing solutions into pseudocode to prioritize higher-level operations  
 280 over low-level implementation details. The main abstraction phase proceeds from the generated  
 281 pseudocode, recording new concepts and revising existing concepts along with their various fields.

282 In the spirit of designing for abstraction, promoting reusability, and minimizing total concept memory  
 283 length, this operation *is* aware of existing concepts with a compressed form of memory included  
 284 in context. We encourage the model to reuse and revise existing concepts by updating concept  
 285 descriptions, parameter lists, relevance cues, and implementation notes. Importantly, we encourage  
 286 higher-order functions with routines as arguments in instructions and few-shot demonstrations.

287 All concept abstraction/memory writing operations are scaffolded with few-shot demonstrations,  
 288 example-rich templates, and comprehensive instructions.

### 291 3.4 MEMORY READ: CONCEPT SELECTION

292 Although state-of-the-art models support longer context windows, their maximum length still limits  
 293 how much information or how many memory entries can be included in a single query. Even without  
 294 this hard limit, including all available memories in context is still undesirable: it can distract the  
 295 model with irrelevant details and flood it with too many unlikely hypotheses to consider. A more  
 296 effective strategy is to introduce a selection mechanism only to include the most relevant subset of  
 297 memory entries at problem-solving time. This allows the memory store to grow continually without  
 298 overwhelming the model’s context window.

300 **OE Selection.** OE-format memory entries were explicitly designed to support memory-based  
 301 selection. The situation clause acts as a semantic hook to identify relevance. The key idea for  
 302 OE Selection is a preprocessing step that leverages model reasoning capability to parse the abstract  
 303 domain input into problem descriptions at different levels of abstraction to facilitate matching against  
 304 our abstract concepts. Since ARC-AGI deals with spatial reasoning, we leverage a vision language  
 305 model for this preprocessing step. We caption each puzzle using a structured prompt that separates  
 306 concrete observations from speculative transformations. This converts spatially rich input into a  
 307 natural language format suitable for matching against stored situations. We then query a model for the  
 308 top- $k$  most relevant entries using the generated description. We explored scoring, thresholding, and  
 309 cumulative similarity approaches (as in top- $p$  sampling). Observing comparable results, we selected  
 310 top- $k$  cutoff for simplicity. We consider this a light-weight approach to converting abstract domain  
 311 inputs into more understandable and memory-aligned representations.

312 **PS Selection.** Standard embedding approaches use a single forward pass per embedding, representing  
 313 a System 1-style fast thinking/intuition-driven understanding from the model. Intuitions may not  
 314 be sufficient for difficult/frontier tasks and domains. Moreover, the connection between a higher-level  
 315 concept and a specific problem instance may not be immediately clear because of the abstraction.

316 To address these challenges, we propose **reasoning-based selection**—using System 2-style thinking  
 317 to deliberately think and explore. In contrast to the more straightforward playbooks curated in the  
 318 OE memory format, the explicitly abstract PS concepts are a set of puzzle pieces with notes on  
 319 identifying if a piece is relevant (relevance cues) and on how various pieces fit together (type annotations).  
 320 To select from this memory, we propose leveraging model-driven exploration instantiated by  
 321 recent models’ long-form reasoning with backtracking. PS Selection instructs a reasoning model to  
 322 systematically explore the problem: first identify initial concepts using relevance cue annotations,  
 323 then attempt to “fill in the details” by determining values or routines to populate these initial concepts’  
 parameters using type annotations to identify which other concepts should be investigated.

324 3.5 CONTINUAL CONCEPT LEARNING  
325

326 A long-held goal in machine learning is to develop lifelong learning systems that continually self-  
 327 improve without manual intervention. A memory system that can leverage the learning signal present  
 328 at test-time via continual updates is one approach to achieve this ambition. Our memory write  
 329 operations are lightweight queries that can ingest solution traces derived from both the system and  
 330 external sources, making continual updates practical at scale.

331 However, continual updates also introduce dependencies on evaluation order. If solving problem  $i$   
 332 induces a memory update that enables problem  $j$  to be solved, then performance differs between  
 333  $(..., x_i, ..., x_j, ...)$  and  $(..., x_j, ..., x_i, ...)$ . Inference batching further complicates this: even if  $x_i$   
 334 precedes  $x_j$ , they may appear in the same batch, so the model attempts  $x_j$  before  $x_i$  has updated  
 335 memory. This introduces an accuracy–throughput trade-off. In all settings, we initialize our external  
 336 memory with seed data (problems and solutions). We explicitly evaluate continual updates in  
 337 subsection 5.3’s experiments, confirming their efficacy. In other experiments, we used fixed memory  
 338 to prioritize throughput and avoid the potentially confounding effect of order dependencies.  
 339

340 4 EXPERIMENTS  
341

342 **Benchmark Selection.** We evaluate our proposed framework on ARC-AGI-1 (Chollet, 2019), a  
 343 benchmark explicitly designed to evaluate intelligence as “efficient acquisition of new skills” instead  
 344 of “fixed possession/memorization of skills.” Each ARC puzzle encodes a transformation rule that  
 345 maps input to output pixel grids. The objective of each puzzle is to infer its rule given several  
 346 examples of input-output grid pairs, and produce the corresponding output grid for several input test  
 347 cases. We find that the abstract domain of pixel grid transforms provides a meaningfully challenging  
 348 testbed to simulate frontier domains without requiring expert knowledge to evaluate trajectories—a  
 349 confluence of desirable properties for evaluating a continual concept learning system.  
 350

351 ARC-AGI-1 contains a public validation split containing 400 puzzles with a difficulty distribution  
 352 matching that of the private evaluation. Following Akyürek et al. (2025), we evaluate a randomly  
 353 selected 100-puzzle subset of the public val split. This makes repeated runs for more stable estimates  
 354 feasible given cost and the sampling variance we observed. Li et al. (2024) manually authored Python  
 355 solutions for 160 puzzles from the public train split to act as “seeds” to recombine into their synthetic  
 356 dataset. We reuse these solutions to seed our memory rather than training data.  
 357

358 **Models.** To build on frontier models, we experiment primarily with OpenAI’s o4-mini. At the  
 359 time of writing, o4-mini is second only to Grok 4, but o4-mini’s lower price puts it on the Pareto  
 360 frontier of cost and performance ARC-Prize (2025). For auxiliary tasks such as concept abstraction  
 361 and non-reasoning selection, we use OpenAI’s GPT-4.1 to conserve token usage. Early experiments  
 362 also evaluated the open-weight DeepSeek R1, which has the benefit of visible thinking traces, but its  
 363 8000 output token limit consistently yielded unfinished solutions in initial testing.  
 364

365 **Evaluation.** While the official evaluation harness queries models to directly predict output grids for  
 366 test cases, we instead use a program synthesis approach that queries for a transformation function to  
 367 convert input to output grids. The code artifact provides more signal for reflection and also allows  
 368 us to test proposed logic against reference pairs for feedback. We evaluate performance under 0, 1,  
 369 and 2 retries with this execution feedback. We follow official ARC-AGI scoring (two attempts per  
 370 puzzle), and account for sampling variance by averaging over extra runs (see details in Appendix B).  
 371 The primary memory baseline we compare against is a re-implementation of Suzgun et al. (2025)’s  
 372 DC-Cu (labeled “cheatsheet”) that uses a frozen memory to match other settings.  
 373

374 5 RESULTS  
375376 5.1 MAIN RESULTS  
377

378 As seen in Table 1, ArcMemo-PS achieves the best official score with standard compute <sup>1</sup> and  
 379 still benefits from additional scaling in both parallel samples and sequential retries. Table 4 shows  
 380

<sup>1</sup>Matching the official evaluation scheme with 2 parallel attempts and 0 retries.

Setting	Oracle@1	<b>Oracle@2 (Official)</b>
qwen3-235b-a22b-instruct	11.00 (1.00)	16.67 (0.57)
deepseek r1	19.50 (7.05)	26.33 (4.02)
claude sonnet 4 (thinking 16k)	45.50 (1.00)	54.17 (0.58)
gemini 2.5 flash (thinking 16k)	9.67 (4.86)	13.83 (4.73)
o4-mini (medium)	46.33 (1.04)	55.17 (3.18)
cheatsheet (o4-mini (medium))	47.50 (2.78)	57.67 (2.52)
ArcMemo-OE (ours, o4-mini (medium))	48.00 (1.00)	56.67 (1.53)
<b>ArcMemo-PS (ours, o4-mini (medium))</b>	<b>49.33 (0.29)</b>	<b>59.33 (0.29)</b>
+ one retry	58.00 (2.29)	67.33 (1.61)
+ two retries	61.67 (3.88)	70.83 (3.06)

Table 1: **Main Results.** ArcMemo-PS yields the best results and scales with additional compute. Bracketed values represent standard deviation. See additional score details in Table 4.

complete details on all settings at all scales, where ArcMemo-PS is the only setting that consistently outperforms the baseline in all inference compute scales. Other memory formulations (Cheatsheet, ArcMemo-OE) situationally improve over the baseline but still underperform it in some regimes.

We observe that ArcMemo-PS’s advantage is most prominent with lower compute regimes. This is consistent with the stated goal “memory for reasoning”: persisting previous experience aims to reduce redundancy from rediscovering ideas found in previous rollouts. The marginal impact of memory is reduced with more inference compute as the model can rediscover ideas through exploration.

Compared to our memory baseline (Cheatsheet), ArcMemo methods are highly competitive, performing favorably in most regimes. The quantitative improvement shown by ArcMemo is supported by qualitative analysis that demonstrates ArcMemo memories are more modular. This qualitative analysis (described more in subsection 5.2) observes that these abstract memories improve concept coverage—more target puzzle ideas are reflected in memory. This difference may be explained by ArcMemo’s modular design being more conducive to maintaining a growing library.

## 5.2 SELECTION ABLATION AND QUALITATIVE RELEVANCE ANALYSIS

		Oracle@k Scores		
retry	setting	k=1	<b>k=2 (Official)</b>	k=3
	0	ArcMemo-PS - selection	<b>49.33 (0.29)</b> 46.83 (5.25)	<b>59.33 (0.29)</b> 55.17 (2.02)
1	ArcMemo-PS - selection	<b>58.00 (2.29)</b> 57.67 (3.79)	<b>67.33 (1.61)</b> 66.00 (3.00)	72.50 <b>73.00</b>
	ArcMemo-PS - selection	<b>61.67 (3.88)</b> 61.33 (3.21)	<b>70.83 (3.06)</b> 70.00 (2.65)	75.50 <b>76.00</b>

Table 2: **Selection Ablation.** Ablating the reasoning-based selection mechanism from ArcMemo-PS reduces overall performance, showing selection is useful for downstream performance.

We perform an ablation experiment removing the selection mechanism from ArcMemo-PS to examine its impact and to compare more directly to the selection-free Cheatsheet methodology. The generally better performance of the with-selection setting (as seen in Table 2) suggests that, in addition to allowing memory to grow without overwhelming context in both the abstraction and inference phases, it also helps downstream problem-solving performance. Moreover, from the token usage plot in Figure 4, we see that concept selection’s performance improvement also comes at higher efficiency—the selection-ablated setting uses far more tokens. However, there still exist certain scale regimes where the no-selection setting has a better score. We attribute this to variance from imperfect selection and note that this indicates headroom for selection/retrieval.

ArcMemo-PS (-selection) is most comparable to Dynamic Cheatsheet’s “cumulative” setting (DC-Cu) as all stored memory is included with each query. While both ultimately solve the same number of puzzles across all runs, we find that the settings differ in 10 puzzles (that is, each system solves 5 puzzles the other does not). We manually analyzed these puzzles and found that only 40% of new solves in the cheatsheet setting were related to memory elements actually in the generated cheatsheet. That is, we were unable to find relevant notes in the cheatsheet to explain 60% of its new solves. In contrast, 100% of new solves from ArcMemo-PS (-selection) can be linked to concept memory contents. While we cannot definitively conclude ArcMemo changes directly induced each of the new solves (given the state of LLM interpretability), this seems to suggest new solves in the ArcMemo settings are potentially more attributable to the memory component compared to sampling variance.

Manual inspection of a subset of reasoning-based selection results used for ArcMemo-PS, finds that while some irrelevant concepts still appear within the selection, the key idea of the target is often included in the selection. The presence of a few distractor concepts appears to be manageable, as the reasoning model itself is capable of exploring with backtracking.

### 5.3 CONTINUAL LEARNING

		Oracle@k Scores		
		k=1	<b>k=2 (Official)</b>	k=3
retry	setting			
0	ArcMemo-OE + continual memory update	48.00 (1.00) 46.33 (1.53)	56.67 (1.53) 56.00 (0.00)	61.00 61.00
1	ArcMemo-OE + continual memory update	56.67 (2.08) 57.17 (3.69)	65.67 (1.53) 65.00 (0.00)	70.00 67.00
2	ArcMemo-OE + continual memory update	60.67 (1.53) <b>62.33 (3.51)</b>	67.67 (2.52) <b>70.00 (1.73)</b>	71.00 <b>72.00</b>

Table 3: **Continual Memory Learning.** Comparing otherwise identical memory systems, we find that with sequential inference compute scale (at high retry depth), continually updating memory with new self-generated solutions leads to improved puzzle-solving performance.

Results from our experiment comparing ArcMemo-OE with a version that updates *during* evaluation (every 10 problems; Table 3) show that our memory system’s performance improves with continual updates—a key requirement for meaningful lifelong learning. In particular, we find that the performance improvement emerges at later iterations in sequential inference scaling. We hypothesize this is because, after more iterations and puzzle-solving passes, new solutions are found, new memories are abstracted, and these new memories help solve other puzzles.

We further discuss token efficiency, concept specificity, and embedding retrieval in Appendix C, and limitations and future work discussion in Appendix D.

## 6 CONCLUSION

In this work, we introduce ArcMemo, a framework designed to support lightweight, lifelong learning on compositional reasoning tasks by emphasizing a higher level of abstraction and modularity. We explore two implementations of memory modules against ARC-AGI, a benchmark specifically designed to resist memorization and to evaluate fluid intelligence instead. Our main findings confirm the efficacy of our approach: ArcMemo-PS outscores comparable methods under the official evaluation protocol and continues to benefit from additional inference compute scale. Moreover, we observe that continual updates benefit memory augmentation over multiple attempts and that selecting a subset of memory for each problem is a crucial component to enable a memory to continually grow without overwhelming the LLM context. This paper’s work represents early attempts toward the main tenets of higher-level abstraction and modularity. Promising future directions include hierarchical designs and consolidation mechanisms that restructure memory. To encourage further investigation, we also release a concept-annotation dataset and configurable puzzle synthesis pipeline, providing resources for evaluating concept representations and advancing abstraction-based memory methods.

486           **ETHICS AND REPRODUCIBILITY STATEMENTS**  
 487

- 488           • **Ethics:** We do not anticipate any ethical concerns with this work, as it focuses on abstract  
 489           reasoning benchmarks without sensitive data.  
 490           • **LLM Usage:** LLMs were used to review and polish small sections of writing.  
 491           • **Reproducibility:** section 3 and section 4 describe our methods and experiments. Further  
 492           experiment details and results are available in the appendix (Appendix B, Appendix A).  
 493           Finally, we include our code repository in the supplemental materials with a README  
 494           explaining code organization and example commands for experiments.  
 495

496           **REFERENCES**  
 497

- 498       Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and  
 499       Jacob Andreas. The surprising effectiveness of test-time training for few-shot learning, 2025. URL  
 500       <https://arxiv.org/abs/2411.07279>.  
 501
- 502       ARC-Prize, 2025. URL <https://arcprize.org/leaderboard>.  
 503
- 504       Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning  
 505       to retrieve, generate, and critique through self-reflection, 2023. URL <https://arxiv.org/abs/2310.11511>.  
 506
- 507       Jinheon Baek, Sujay Kumar Jauhar, Silviu Cucerzan, and Sung Ju Hwang. Researchagent: Iterative re-  
 508       search idea generation over scientific literature with large language models. *ArXiv*, abs/2404.07738,  
 509       2024. URL <https://api.semanticscholar.org/CorpusID:269042844>.  
 510
- 511       Léon Bottou and Yann LeCun. Large scale online learning. *Advances in neural information processing systems*, 16, 2003.  
 512
- 513       Léon Bottou and Yann LeCun. On-line learning for very large data sets. *Applied stochastic models in business and industry*, 21(2):137–151, 2005.  
 514
- 515       Aydar Bulatov, Yuri Kuratov, and Mikhail S. Burtsev. Recurrent memory transformer, 2022. URL  
 516       <https://arxiv.org/abs/2207.06881>.  
 517
- 518       Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building  
 519       production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*,  
 520       2025.  
 521
- 522       François Chollet. On the measure of intelligence, 2019. URL <https://arxiv.org/abs/1911.01547>.  
 523
- 524       Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang,  
 525       Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory, 2025. Work in  
 526       progress.  
 527
- 528       Huanang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong  
 529       Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, Hongru Wang, Han Xiao, Yuhang Zhou, Shaokun Zhang,  
 530       Jiayi Zhang, Jinyu Xiang, Yixiong Fang, Qiwen Zhao, Dongrui Liu, Qihan Ren, Cheng Qian,  
 531       Zhenhailong Wang, Minda Hu, Huazheng Wang, Qingyun Wu, Heng Ji, and Mengdi Wang. A  
 532       survey of self-evolving agents: On path to artificial super intelligence, 2025. URL <https://arxiv.org/abs/2507.21046>.  
 533
- 534       Pengfei He, Zitao Li, Yue Xing, Yaling Li, Jiliang Tang, and Bolin Ding. Make llms better zero-shot  
 535       reasoners: Structure-orientated autonomous reasoning. *ArXiv*, abs/2410.19000, 2024a. URL  
 536       <https://api.semanticscholar.org/CorpusID:273638104>.  
 537
- 538       Zexue He, Leonid Karlinsky, Donghyun Kim, Julian McAuley, Dmitry Krotov, and Rogerio Feris.  
 539       Camelot: Towards large language models with training-free consolidated associative memory,  
 2024b. URL <https://arxiv.org/abs/2402.13449>.

- 540 Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. Hiagent: Hier-  
 541 archical working memory management for solving long-horizon agent tasks with large language  
 542 model. *arXiv preprint arXiv:2408.09559*, 2024.
- 543
- 544 Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song,  
 545 and Denny Zhou. Large language models cannot self-correct reasoning yet, 2024. URL <https://arxiv.org/abs/2310.01798>.
- 546
- 547 Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can llms actually  
 548 correct their own mistakes? a critical survey of self-correction of llms, 2024. URL <https://arxiv.org/abs/2406.01297>.
- 549
- 550 Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal,  
 551 Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe  
 552 Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021. URL <https://arxiv.org/abs/2005.11401>.
- 553
- 554 Wen-Ding Li, Keya Hu, Carter Larsen, Yuqing Wu, Simon Alford, Caleb Woo, Spencer M. Dunn,  
 555 Hao Tang, Michelangelo Naim, Dat Nguyen, Wei-Long Zheng, Zenna Tavares, Yewen Pu, and  
 556 Kevin Ellis. Combining induction and transduction for abstract reasoning, 2024. URL <https://arxiv.org/abs/2411.02272>.
- 557
- 558 Lei Liu, Xiaoyan Yang, Yue Shen, Binbin Hu, Zhiqiang Zhang, Jinjie Gu, and Guannan Zhang.  
 559 Think-in-memory: Recalling and post-thinking enable llms with long-term memory, 2023. URL  
<https://arxiv.org/abs/2311.08719>.
- 560
- 561 Yitao Liu, Chenglei Si, Karthik Narasimhan, and Shunyu Yao. Contextual experience replay for  
 562 self-improvement of language agents. *arXiv preprint arXiv:2506.06698*, 2025.
- 563
- 564 Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon,  
 565 Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder,  
 566 Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative  
 567 refinement with self-feedback, 2023. URL <https://arxiv.org/abs/2303.17651>.
- 568
- 569 Ali Modarressi, Abdullatif Köksal, Ayyoob Imani, Mohsen Fayyaz, and Hinrich Schütze. Memllm:  
 570 Finetuning llms to use an explicit read-write memory. *arXiv preprint arXiv:2404.11672*, 2024.
- 571
- 572 Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E.  
 573 Gonzalez. Memgpt: Towards llms as operating systems, 2024. URL <https://arxiv.org/abs/2310.08560>.
- 574
- 575 Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and  
 576 Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023. URL  
<https://arxiv.org/abs/2304.03442>.
- 577
- 578 Julien Pourcel, Cédric Colas, and Pierre-Yves Oudeyer. Self-improving language models for evolu-  
 579 tionary program synthesis: A case study on arc-agi, 2025. URL <https://arxiv.org/abs/2507.14172>.
- 580
- 581 Linlu Qiu, Liwei Jiang, Ximing Lu, Melanie Sclar, Valentina Pyatkin, Chandra Bhagavatula, Bailin  
 582 Wang, Yoon Kim, Yejin Choi, Nouha Dziri, and Xiang Ren. Phenomenal yet puzzling: Testing  
 583 inductive reasoning capabilities of language models with hypothesis refinement, 2024. URL  
<https://arxiv.org/abs/2310.08559>.
- 584
- 585 Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and  
 586 Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL  
<https://arxiv.org/abs/2303.11366>.
- 587
- 588 Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic  
 589 cheatsheet: Test-time learning with adaptive memory, 2025. URL <https://arxiv.org/abs/2504.07952>.
- 590

- 594 Guan Wang, Jin Li, Yuhao Sun, Xing Chen, Changling Liu, Yue Wu, Meng Lu, Sen Song, and  
 595 Yasin Abbasi Yadkori. Hierarchical reasoning model, 2025. URL <https://arxiv.org/abs/2506.21734>.  
 596
- 597 Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and  
 598 Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023a.  
 599 URL <https://arxiv.org/abs/2305.16291>.  
 600
- 601 Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah D. Goodman.  
 602 Hypothesis search: Inductive reasoning with language models, 2024. URL <https://arxiv.org/abs/2309.05660>.  
 603
- 604 Xintao Wang, Qian Yang, Yongting Qiu, Jiaqing Liang, Qi He, Zhouhong Gu, Yanghua Xiao, and  
 605 W. Wang. Knowledgpt: Enhancing large language models with retrieval and storage access on  
 606 knowledge bases. *ArXiv*, abs/2308.11761, 2023b. URL <https://api.semanticscholar.org/CorpusID:261076315>.  
 607
- 608 Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers,  
 609 2022. URL <https://arxiv.org/abs/2203.08913>.  
 610
- 611 Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E. Gonzalez,  
 612 and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models, 2024.  
 613 URL <https://arxiv.org/abs/2406.04271>.  
 614
- 615 Zhengyan Zhang, Xu Han, Zhiyuan Liu, Xin Jiang, Maosong Sun, and Qun Liu. Ernie: Enhanced  
 616 language representation with informative entities, 2019. URL <https://arxiv.org/abs/1905.07129>.  
 617

## 619 A IMPLEMENTATION DETAILS

### 621 A.1 PS MEMORY FORMAT

623 Each PS format memory entry contains:

- 625 • **Title:** a succinct label for the underlying idea.
- 626 • **Description:** elaboration on behavior and role.
- 627 • **Kind:** whether this concept encodes a type/structure/routine.
- 628 • **Parameters:** a list of fields that parametrize this concept.
- 629 • **Output Typing:** specifies what the output for this routine is, to suggest how various routines  
 630 can plug into other routines.
- 631 • **Relevance Cues:** much like the `situation` field in the OE concepts, we consider the  
 632 context in which this concept is relevant.
- 633 • **Implementation Notes:** suggestions on how to implement the concept in actual code.

### 636 A.2 EXPERIMENT PARAMETERS

638 To build on frontier models, we experiment primarily with OpenAI’s o4-mini (max tokens=32000,  
 639 reasoning\_effort=medium). For auxiliary tasks such as concept abstraction and non-reasoning  
 640 selection, we use OpenAI’s GPT-4.1 (temperature=0.3, max tokens=1000) to reduce token usage. We  
 641 looked into evaluating the open-source DeepSeek R1, which also has a transparent thinking process,  
 642 but the output 8000 token limit consistently yielded unfinished solutions.

## 643 B EVALUATION DETAILS

646 We use ARC-AGI’s official scoring metric, which we term `oracle@k` (each test case is scored  
 647 separately, and if any of  $k$  candidates pass, full credit is given for that test case). The ARC-AGI  
 official evaluation has  $k = 2$ , but to mitigate sampling variance, we sample 3 runs for each setting

and report the average single run score, average oracle@2 score, and their standard deviation. We investigate several alternate scoring protocols (requiring a single program to solve all test cases, accumulating test case solve rates, and requiring references also to be solved) and include the results in Table 5.

Here is a precise definition of our scoring procedure for a single problem  $P$ . Let...

- $X$  be the size  $n$  set of attempts on problem  $P$
- $T$  be the set of test cases for problem  $P$
- $C$  represent a particular  $k$ -subset of  $X$ , in notation:  $C \in [X]^k$

$$\text{score} = \frac{1}{|[X]^k|} \sum_{C \in [X]^k} z(C)$$

where  $z$  is the problem score by a  $k$ -attempt ensemble

$$z(C) = \frac{1}{|T|} \sum_{t_i \in T} \mathbb{1}\{\exists c \in C \text{ Verify}(c, t_i)\}$$

where

- $c$  is a single program attempt in the  $k$ -attempt ensemble,
- $t$  is a test case,
- and `Verify` returns true if the program successfully passes the test case.

iteration	setting	Individual Run Scores			Oracle@k Scores		
		run0	run1	run2	k=1	<b>k=2</b> (Official Score)	k=3
0	baseline	46.00	45.50	47.50	46.33 (1.04)	55.17 (3.18)	59.50
	cheatsheet	48.00	44.50	50.00	47.50 (2.78)	57.67 (2.52)	<b>64.00</b>
	ArcMemo-OE (ours)	49.00	47.00	48.00	48.00 (1.00)	56.67 (1.53)	61.00
	ArcMemo-PS (ours)	49.50	49.00	49.50	<b>49.33 (0.29)</b>	<b>59.33 (0.29)</b>	63.50
1	baseline	57.00	55.50	61.00	57.83 (2.84)	66.67 (3.82)	71.50
	cheatsheet	56.00	56.50	57.50	56.67 (0.76)	65.67 (1.44)	70.50
	ArcMemo-OE (ours)	59.00	55.00	56.00	56.67 (2.08)	65.67 (1.53)	70.00
	ArcMemo-PS (ours)	58.50	60.00	55.50	<b>58.00 (2.29)</b>	<b>67.33 (1.61)</b>	<b>72.50</b>
2	baseline	59.50	59.00	65.00	61.17 (3.33)	69.00 (2.65)	73.00
	cheatsheet	65.00	61.00	61.00	<b>62.33 (2.31)</b>	<b>71.33 (1.53)</b>	<b>76.00</b>
	ArcMemo-OE (ours)	62.00	59.00	61.00	60.67 (1.53)	67.67 (2.52)	71.00
	ArcMemo-PS (ours)	60.50	66.00	58.50	61.67 (3.88)	70.83 (3.06)	75.50

Table 4: **Full o4-mini Results.** The right partition of the table containing aggregate scores shows the inference scaling: going down represents sequential retry, going right represents adding parallel samples. ArcMemo-PS is the only setting that outperforms the baseline consistently at every tested scale.

## C FURTHER DISCUSSION

**Token Efficiency** We find that our system’s token efficiency is similar to the baseline. We observe that memory-augmented runs tend to increase output token usage. While this seems to counter the intuition that retrieving rather than regenerating certain ideas would save tokens, we hypothesize that introducing memory leads the model to explore more hypotheses. In other words, selection inaccuracies (false positives) seem to be manageable by the reasoning models, but at the cost of increased token usage. Full results are plotted in Figure 4.

iteration	setting	run0	run1	run2	k=1	k=2	k=3
0	baseline	46.00	45.00	47.00	46.00 (1.00)	54.67 (3.21)	59.00
	cheatsheet	48.00	44.00	50.00	47.33 (3.06)	57.67 (2.52)	64.00
	ArcMemo-OE	49.00	47.00	48.00	48.00 (1.00)	56.67 (1.53)	61.00
	ArcMemo-PS	49.00	49.00	49.00	49.00 (0.00)	59.00 (0.00)	63.00
1	baseline	57.00	55.00	61.00	57.67 (3.06)	66.33 (4.04)	71.00
	cheatsheet	56.00	56.00	57.00	56.33 (0.58)	65.33 (1.15)	70.00
	ArcMemo-OE	59.00	55.00	56.00	56.67 (2.08)	65.67 (1.53)	70.00
	ArcMemo-PS	58.00	60.00	55.00	57.67 (2.52)	67.00 (1.73)	72.00
2	baseline	59.00	59.00	65.00	61.00 (3.46)	69.00 (2.65)	73.00
	cheatsheet	65.00	61.00	61.00	62.33 (2.31)	71.33 (1.53)	76.00
	ArcMemo-OE	62.00	59.00	61.00	60.67 (1.53)	67.67 (2.52)	71.00
	ArcMemo-PS	60.00	66.00	58.00	61.33 (4.16)	70.33 (3.06)	75.00

Table 5: **Strict Scoring.** While the official evaluation scheme allows different test cases solved by different attempts to be ensembled, in contrast, the strict scoring regime only marks a puzzle as solved if a single attempt (generated program) solves all test cases.

**Concept Specificity Investigation** As part of the development of our concept representations, we conducted small-scale experiments using tiny ( $n=10$ ) subsets of the validation split to investigate the level of detail needed for concepts to be useful. Manually writing maximum detail situation-suggestion style concepts and using iteratively LLM-summarized versions (for a total of 5 levels of specificity for each concept), we found the reasonably expected result that higher levels of specificity correspond to better solve rates. Maximum specificity solved 4/10 with a generally decreasing pattern with lower specificity. While this result was entirely expected, the main goal of the experiment was to determine how much we could compromise on concept detail to improve the retrieval aspect. There is a tension between retrieval and puzzle solving in the sense that puzzle solving benefits from more detailed suggestions, but retrieval struggles to apply them to new situations when the concept’s relevant situation is so precisely defined.

**Embedding-based Retrieval Experiments** Early in testing, we investigated standard embedding-based retrieval approaches. Following the ArcMemo-OE setting, we used a VLM-generated caption to query a vector database of ArcMemo-OE-style concept embeddings. While computationally cheap compared to autoregressive generation (especially so when leveraging long-form reasoning), we saw generally poor retrieval results. Using OpenAI’s embeddings API and an o3-mini puzzle-solving backbone proved ineffective, lowering the score from 0.26 to 0.22, marking a 15% reduction. Qualitative analysis found that retrieved concepts were largely irrelevant to the puzzle and seemed to over-index on lexical similarity. Moreover, a small set of broadly defined (non-specific) concepts was found to be grossly overrepresented.

## D LIMITATIONS AND FUTURE WORK

### D.1 LIMITATIONS

Following Akyürek et al. (2025), our evaluation focuses on a 100-task subset of ARC-AGI-1, selected to make large-scale experimentation feasible under compute and cost constraints. This choice was further motivated by the substantial sampling variance we observed—identical prompts can yield noticeably different scores, requiring multiple runs to obtain stable estimates. As a result, our study concentrates on a relatively small frontier of puzzles where memory augmentation can yield new solves, limiting the absolute magnitude of observable gains.

In addition, ARC-AGI-2 was released mid-project as a harder successor benchmark, with state-of-the-art performance still below 30%. Re-running all baselines and experiments on ARC-AGI-2 was infeasible within scope due to cost and time constraints, but extending memory-based approaches to ARC-AGI-2 remains an important direction for future work.

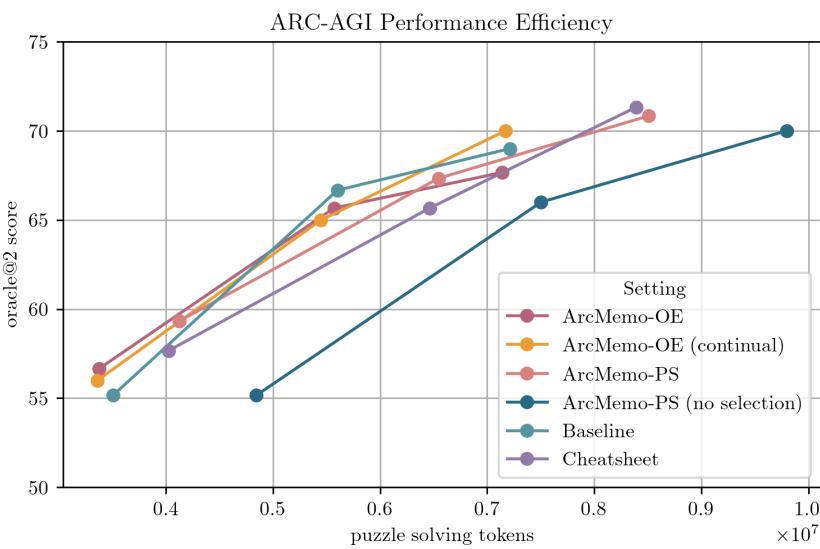


Figure 4: Token Efficiency Plot: Comparing various settings’ official score (oracle@2) on the public validation subset to the tokens used by the reasoning model for strictly puzzle solving.

## D.2 FUTURE WORK

Several directions follow naturally from this work. One is to broaden evaluation to larger and more challenging puzzle sets, including those unsolved by the baseline, to test both new solve potential and consistency on borderline cases. Another is to explore order-robust update strategies and curriculum designs that reduce sensitivity to problem ordering while preserving inter-problem learning benefits. Beyond these extensions, a key avenue is hierarchical abstraction: moving from additive concept updates toward consolidation operations that restructure the memory across multiple experiences. Finally, our current design choices for representation and retrieval were made under cost constraints; richer optimization may yield substantial improvements.

To facilitate further research, we release a data resource: hand-annotated concepts for difficult puzzles, together with a configurable puzzle synthesis pipeline. This resource is intended to support isolated evaluation of concept representations and selection mechanisms, as well as development of methods for concept abstraction. We hope it provides a useful foundation for continued exploration of abstract reasoning memory.

## E ADDITIONAL RELATED WORK

**Knowledge Base Augmented LLMs for Factuality.** Early efforts to enhance language models with external memory primarily targeted knowledge-intensive tasks. Retrieval-Augmented Generation (RAG) introduced the idea of retrieving documents from an external store based on embedding similarity to augment generation (Lewis et al., 2021). Follow-up work like KnowledGPT incorporated formal knowledge bases, allowing LLMs to execute structured queries through “Program-of-Thought” prompting for multi-hop reasoning (Wang et al., 2023b). More recently, MemLLM proposed inline memory operations—allowing the model to read from and write to memory directly during generation—enabling continual adaptation without re-training (Modarressi et al., 2024).

**Embedding Space Memory.** Another stream of work aims to overcome the limited context window of transformers through architectural interventions that compress information from long sequences into continuous space. Memorizing Transformers stored latent key-value vectors from past inputs and retrieved them via a kNN mechanism to be attended to alongside the current context (Wu et al., 2022). The Recurrent Memory Transformer extended this by incorporating a writable memory updated over time, making memory content responsive to ongoing computation (Bulatov et al., 2022). He et al. (2024b) blended these ideas by applying a moving average to update memory slots, allowing

810 for both similarity-based retrieval and continual adaptation He et al. (2024a). MemGPT introduced  
 811 a hierarchical memory system with a working (RAM-like) memory and a larger archival memory,  
 812 simulating long-term information management in a more structured fashion (Packer et al., 2024).  
 813

814 **Parameter-Free Test-time Learning.** Test-time learning (sometimes referred to as online learning)  
 815 describes methods that update predictions at inference time Bottou & LeCun (2003; 2005). However,  
 816 trends towards (1) massive model sizes and (2) black box models accessed through APIs make  
 817 continually updating parameters computationally impractical or downright impossible. In light of  
 818 this, LLM test-time learning research sought to pursue parameter-free adaptation by using LLMs’  
 819 remarkable capacity for instruction following and self-reflection. Methods such as Reflexion (Shinn  
 820 et al., 2023), Self-Refine (Madaan et al., 2023), and Self-RAG (Asai et al., 2023), *inter alia* attempt  
 821 to correct mistakes at test-time by way of self-reflection. Follow-up work (Huang et al., 2024; Kamoi  
 822 et al., 2024) finds that self-correction methods depend on the availability of some extrinsic feedback or  
 823 verification mechanism to reliably improve end performance and that these methods produce dubious  
 824 results absent these grounded sources of feedback. Still, self-reflection remains a key component of  
 825 self-evolving agents (Gao et al., 2025).

826 **Memory for Agentic Skill Acquisition.** Memory also plays a crucial role in LLM-based agents.  
 827 Park et al. (2023) structured memory as a stream of episodic observations and distilled high-level  
 828 summaries from them to guide decision-making. Contextual Experience Replay discretizes trajectories  
 829 into “experiences” (containing environment dynamics from a trajectory and skills storing actions  
 830 from said trajectory) and retrieves relevant blocks to guide new episodes (Liu et al., 2025). HiAgent  
 831 organizes working memory (within a single query) hierarchically by subgoals, summarizing and  
 832 replacing traces to improve long-horizon efficiency (Hu et al., 2024). External-memory systems  
 833 such as Mem0 and MemP manage a persistent store with explicit add–update–prune operations  
 834 (MemP casts control as an MDP), yielding gains on dialogue and tool-use/planning tasks (Chhikara  
 835 et al., 2025; Fang et al., 2025). MemP in particular is a concurrent work that also demonstrates the  
 836 efficacy of abstraction, although it targets rewriting action sequences as step sequences in explicitly  
 837 agentic settings, contrasting with ArcMemo’s general problem-solving focus. Voyager uses LLMs to  
 838 autonomously acquire and store skills while exploring Minecraft, with a growing library of reusable  
 839 code snippets written by the agent itself Wang et al. (2023a). Other systems, like the ResearchAgent,  
 840 use memory initialized from domain-specific corpora, such as scientific papers, to ground query  
 841 responses in high-utility prior knowledge (Baek et al., 2024).

842  
 843  
 844  
 845  
 846  
 847  
 848  
 849  
 850  
 851  
 852  
 853  
 854  
 855  
 856  
 857  
 858  
 859  
 860  
 861  
 862  
 863