CONTINUAL-ARC: MEASURING SKILL ACQUISITION OVER A WORKING LIFETIME

Anonymous authors
Paper under double-blind review

ABSTRACT

Agent harnesses have emerged as a promising mechanism for continual learning, alongside test-time parametric learning. However, the current benchmarks that measure such systems have several limitations: they generally focus on a single *applied* domain, they serve each instance only once, and most assume a single family of learner, so memory-based and weight-based systems are rarely compared on the same scale. We propose Continual-ARC, a lifelong stream built from the three ARC-AGI editions, whose tasks need no domain knowledge and which the models under test solve only in part. The same tasks return after dormant gaps with fresh instances, while new tasks keep arriving. On each instance, the learner may pay for the demonstrations or submit an answer, with retries permitted on an incorrect submission. Each instance is scored based on the share of this feedback budget (demonstrations and failed attempts) consumed. Since a learner may keep anything between instances, memory-based and weight-based systems are scored on the same scale. We run three agent harnesses with frozen weights, each with and without persistence, comparing them with two prompting floors and a program-induction method. We also define a second regime in which one system learns from several users simultaneously. Across our experiments, we show that persistence in the harness does generally save on budget, but only modestly. None of them perform measurably better than a floor that simply caches each task's demonstrations. In contrast, a simple method that asks the LLM for one program per task and keeps following demo reproduction outperforms all harnesses by a wide margin. This opens the door to future harness design improvements, and to weight-based continual learning approaches that could teach models to effectively drive harnesses in new environments.

1 INTRODUCTION

Chollet (2019) argued that intelligence should be measured by how efficiently a system turns priors and experience into skill, on tasks that lie largely outside its experience and have a verifiable answer. ARC is the concrete instance of that argument, and on its current edition the gap between what a person solves in a minute and what a model solves from the same demonstrations is still significant (Chollet et al., 2026). For agent systems that work for days and weeks, a more pressing question is whether a skill acquired on one job is still there the next time a similar scenario is met. By design, ARC measures acquisition once, within a single episode, and does not ask this over longer horizons.

The benchmarks that do ask it have three limitations (Section 2). Firstly, each is built around one *applied* domain, such as coding or office work (Wu et al., 2024; Zheng et al., 2025; Joshi et al., 2025), in which the models under test already know a great deal from the pretraining data. A method's gains may come partly from drawing out that knowledge, as reinforcement learning has been shown to do in mathematics (Yue et al., 2025; Shao et al., 2025; Wu et al., 2025), and so may not carry over to a domain that the model is less familiar with. Secondly, each serves a fixed set of instances, each seen once, so the same conceptual task never returns with a fresh input. This means a memorised answer cannot be told apart from a more abstract learned skill. Finally, whilst some permit both (Wu et al., 2024; Zheng et al., 2025), most assume a single family of learner, gradient updates for fine-tuning suites such as MLLM-CTBench (Guo et al., 2025) and prompting and memory for agent benchmarks such as AgentCL and PATH-Bench (Shu et al., 2026; Yang et al., 2026).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We propose Continual-ARC, a lifelong stream built from ARC, where tasks need no knowledge of any domain beyond core-knowledge priors, such as objectness and counting (Chollet, 2019), and the rule of each grid task has to be induced from a few demonstrations. The rules still share their building blocks: the Re-ARC generators write all 400 ARC-AGI-1 training rules with one library of 160 primitives (Hodel, 2024), so a learner that keeps reusable primitives should carry them to tasks it has never met (Appendix A). The models we test solve *some* of these tasks at first sight, but far from all (Appendix A). A model may also know some public ARC-AGI-1 rules from pre-training, and the floors' first exposures measure how much it brings. We expect a method's gains on such tasks to depend less on prior knowledge than gains in coding or office work do. The three editions also rise in difficulty (Chollet et al., 2025; ARC Prize Foundation, 2026), to the point where a model costing a cent per task and a frontier model are best measured on different editions. For ARC-AGI-1, the Re-ARC generators (Hodel, 2024) produce a fresh, verified instance of any training task on demand. A task can then return after a dormant gap as often as a schedule asks, each time with a new input that no cached answer fits. New tasks keep arriving throughout the stream. On each instance, the system may submit an answer and receive a bare verdict, or it may pay for the demonstrations. We score the instance by the share of this feedback budget it used, as Chollet (2019) proposed for an interactive version of ARC. Because a system may carry anything between instances, whether weights or notes, prompting, memory and weight updates all face the same stream. Section 2 sets these properties against those of the closest benchmarks.

We evaluate three agent harnesses, prime-agent (Karten et al., 2026), hermes-agent (Nous Research, 2026) and openclaw (OpenClaw contributors, 2026), running each with every learning mechanism it ships switched on and, as a control, with its persistence switched off. We compare them with two prompting floors and a program-induction method, keeping all weights frozen. On ARC-AGI-1, we find that the harnesses end up level with, or behind, a floor that simply caches each task's demonstrations, while the program-induction method uses 13 to 20 points less of the budget than that floor (Section 5). To see why, we inspected the state that each harness stored. hermes-agent, for example, records general advice on how to approach a task, and openclaw's searches return nothing because its consolidation never writes a memory (Appendix E).

We run only the individual regime, in which each system has a stream to itself, and define a collective regime in which one system serves a hundred users at once (Section 3.5). Our benchmark highlights significant shortcomings in how much off-the-shelf harnesses can "learn" in unfamiliar domains, and opens the door to research on training models to drive harnesses in such domains.

2 Related work

We screened roughly two hundred benchmarks and methods. Table 11 (Appendix F) sets the closest against the five properties that this benchmark combines: a fresh instance on every exposure, recurrence after variable gaps, a task set that grows through the stream, binary feedback that is charged, and a protocol open to prompting, memory and weight updates. We found no work that has all five.

ARC. In the paper that defines ARC, Chollet (2019) lists its weaknesses, one of which motivates this work. He notes that the format is close-ended, and suggests that a better one would let the test-taker query an example generator and submit answers for feedback repeatedly, with the feedback consumed as the score. To our knowledge, this proposal has not been implemented, although Re-ARC (Hodel, 2024) supplies the generators and verifiers it needs, and third-party generators now cover part of ARC-AGI-2 (Lehmann et al., 2026; Moffitt, 2025). The strongest ARC-AGI-1 systems instead adapt their weights at test time and then discard the adaptation, either per task (Akyürek et al., 2024) or in one pass over the evaluation set (Franzen et al., 2025; Pourcel et al., 2025), never meeting a task twice. Similarly, ArcMemo (Ho et al., 2025) writes concepts to memory, but only across a single fixed pass. ARC-AGI-3 (ARC Prize Foundation, 2026) moves to learning over time within a single game, and the harnesses that persist memory across its public games (Karten, 2026; Fox et al., 2026; Bedi, 2026) have saturated that set without scoring the transfer they rely on.

Continual learning. The classic protocol (Lopez-Paz & Ranzato, 2017; van de Ven & Tolias, 2019) is a fixed sequence of tasks with free labels, scored on fixed test sets after each task boundary. Where tasks come back at all, they return as a planned second pass (Wołczyk et al., 2021; Powers et al., 2022; Veniat et al., 2021) or as unannounced revisits (Caccia et al., 2020; Hemati et al., 2023).

Fresh instances are standard in reinforcement learning environments (Cobbe et al., 2020; Küttler et al., 2020) and in generated reasoning suites with verifiers (Stojanovski et al., 2025; Chen et al., 2025; Zeng et al., 2026), but these suites train first and test afterwards, and supervision is free.

Agent benchmarks. The agent benchmarks of 2024 to 2026 share our goal most closely. StreamBench (Wu et al., 2024) gives binary feedback over a single shuffled pass, and LifelongAgentBench (Zheng et al., 2025) brings skills back but not instances. SWE-Bench-CL (Joshi et al., 2025) regroups SWE-Bench by repository. CL-Bench (Asawa et al., 2026) scores a stateful system against a stateless one, and AgentCL (Shu et al., 2026) and PATH-Bench (Yang et al., 2026) run a stream twice or reinsert a probe task. Most of these benchmarks are built on domains such as operating systems and code. All of them draw from a fixed pool of hand-authored instances that appear once each, which lets an instance be memorised, with no tasks ever recurring with new inputs. They also do not grow the task set or charge for supervision, and most evaluate prompting and memory only.

Memory against weights. Every benchmark above serves its stream to a single learner one item at a time, as in our individual regime (Section 3.5), and none serves a system that many users share, as in our collective regime. Verifier suites and the strongest ARC systems do adapt over thousands of instances at once, but in training, with no recurrence and no score charged while the system learns. As a result, the comparison between memory and weights exists only in pieces. Memory-based agents (Shinn et al., 2023; Zhao et al., 2024; Suzgun et al., 2025; Zhou et al., 2025; Zhang et al., 2026) are motivated by the cost or the forgetting of fine-tuning, but they do not run it. On-policy reinforcement learning is reported to forget less than supervised fine-tuning (Shenfeld et al., 2025; Chen et al., 2026), but the evidence is mixed, with reinforcement learning over a sequence of tasks reported to forget severely (Luo et al., 2026). SEAL (Zweiger et al., 2025) runs self-directed weight updates on ARC, and self-improving harnesses (Karten et al., 2026) rewrite their memory and skills, but each is evaluated on its own terms. We built Continual-ARC to make that comparison possible.

3 THE BENCHMARK

The benchmark serves a learner a stream of ARC task instances and charges it for the feedback it uses. We build it up from one instance (Section 3.1) to a stream of instances (Section 3.2) and the measures along this stream (Section 3.3). We define all terms in this section. Table 3 in Appendix A collects the full set of terms.

3.1 ONE INSTANCE

An ARC task is a rule that turns an input grid into an output grid, and an instance of it is one input whose output the learner must produce (on ARC-AGI-3, a task is a game and an instance one of its levels). For each instance, the learner observes the task id, the input grid, and how many attempts and demonstration requests it has used on this instance, but it is told nothing about the stream. It may submit a grid, and the environment answers only correct or incorrect. A correct submission ends the instance, and an incorrect one costs 1. It may instead request demonstrations once per instance, at a cost of $c_d = 1$, which returns the task's demonstration set: three fresh input-output pairs on ARC-AGI-1, and every other public pair of the task under the instance's augmentation on ARC-AGI-2 (Section 3.4). A second request in the same instance is refused and costs nothing. An instance ends on a correct submission or on the eighth wrong attempt, and the learner cannot abandon it. We deliver the verdict on the final submission before any storage step that the system runs, such that what it decides to store can reflect the outcome. When a system gives no readable action after two re-prompts, or is unavailable after the adapter's restarts, it fails the instance (Appendix D).

Let $w_{\max} = 8$ be the attempt cap and $d_{\max} = 1$ the demonstration cap, and on instance i let w_i be the wrong submissions (set to $w_{\max}$ when the instance fails) and d_i the demonstration requests. For the instance, and for a set of instances $\mathcal{I}$, we define the charge, the cap, the share of the budget used, the budget used U and the share solved within the cap P as

$$c_i = w_i + c_d\, d_i, \qquad C_{\max} = w_{\max} + c_d\, d_{\max} = 9, \qquad u_i = 100\,\frac{c_i}{C_{\max}},$$
$$U(\mathcal{I}) = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} u_i, \qquad P(\mathcal{I}) = \frac{100}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} \mathbf{1}[w_i < w_{\max}]. \tag{1}$$

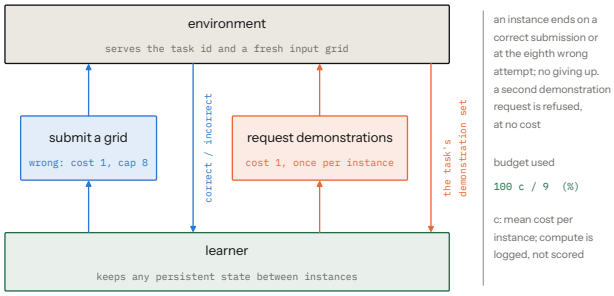

Figure 1: One instance. The learner may submit a grid for a verdict, or pay once for the task's demonstration set. Each wrong attempt and the demonstration request cost one, against a cap of nine. We record what the learner carries between instances, but it does not enter the score.

Here $U = 0$ means that every instance was solved at the first attempt without a demonstration, a task solved at the first attempt after one look uses 11, and $U = 100$ means that every instance failed after a demonstration request. We take U over the stream as the headline number (lower is better), and quote P beside it because neither number determines the other. With eight attempts and a verdict on each, P saturates on the tasks that a model can solve at all, which leaves learning to show in U.

We keep the feedback binary, as ARC Prize does and as a deployed system experiences it, since returning the correct grid on failure would turn the stream into supervised learning. We also expose the task id on every instance to enable the system to properly disambiguate between tasks, as the input grid is not a unique identifier for the task. Examples given in Appendix A.

3.2 A STREAM OF INSTANCES

A learner meets instances as a stream, which we specify as a schedule: a list of slots, each naming a task, an instance seed and a phase (stream or epilogue), served in order. A mode is an order of presentation over the same task set, and we generate every stream with a single process whose named settings are the modes. At step t, an active set $\mathcal{A}_t$ holds at most n_c tasks, from which the slot's task τ_t is drawn uniformly. A task that enters the active set stays for a spell of β instances, after which it rests for ρ instances before it may return. Never-seen tasks arrive every α instances and take the next free place:

$$\tau_t \sim \mathrm{Unif}(\mathcal{A}_t), \qquad \beta \sim \mathrm{Exp}(b), \qquad \rho \sim \mathrm{LogNormal}(\ln m, \sigma), \qquad \alpha \sim \mathrm{Exp}(a). \tag{2}$$

We round spells and rests to whole instances, and each is at least one. The four parameters place a stream on an axis from blocked to structureless, and the three modes are settings of $(n_c, \beta, \rho, \alpha)$,

$$\begin{aligned} \text{sequential: } &(1,\ L/n,\ \infty,\ 0), \qquad \text{random: } (n,\ \infty,\ \cdot,\ 0), \\ \text{workstreams: } &(n_c,\ \mathrm{Exp}(b),\ \mathrm{LogNormal}(\ln m, \sigma),\ \mathrm{Exp}(a)), \end{aligned} \tag{3}$$

where L is the stream length and n is the number of tasks. We give the full process in Algorithm 1 and draw the three modes in Figure 9, both in Appendix B. *Sequential* mode is blocked practice, in which every instance of a task is served before the next task begins and the tasks come in a shuffled order. *Workstreams*, the mode of every track, keeps a few tasks active at once, each worked in bursts with dormant gaps between them. New tasks arrive as a Poisson process. In this mode a task can stay dormant for long enough to be forgotten. *Random* mode is the interleaved i.i.d. control, in which every task is active from the first step. The difference between a system's workstreams and random scores is what burstiness and dormancy cost it. Since the instance seed of a slot is fixed by its position, the modes differ only in which task is served where, and none of them touches the learner's state. We end every mode with the same *epilogue*, in which evaluation tasks never seen before are served once each, to test whether what a learner has built carries to new tasks. Nothing is frozen for it, and the learner is not told that it has begun. To measure forward transfer, we need one more run, the *fresh* reference, which serves the epilogue alone, with the same instance seeds, to a learner whose state is wiped before each instance.

3.3 MEASURING LEARNING

We write e_i for the exposure index of instance i, the number of earlier rounds in which its task appeared, and g_i for its gap, the number of rounds since the task's previous round (in the individual regime, a round is one instance). Over the stream records $\mathcal{S}$, we define

$$U(k) = \text{mean}\{\, u_i : e_i \in B_k \,\}, \quad R(g) = \text{mean}\{\, u_i : e_i > 0,\ g_i \in G_g \,\}, \quad T = \text{mean}\{\, u_i^{\text{ref}} - u_i : i \in \mathcal{E} \,\}, \tag{4}$$

with the exposure buckets B_k and the gap buckets G_g fixed in Appendix A. $U(0)$ is the acquisition cost and $U(k)$ the exposure curve. A system that replays stored demonstrations to a stateless solver moves down this curve only by the demonstration it no longer pays for, while a system that has learned a task falls below the curve of such a replay. $R(g)$ is retention, and forgetting shows as a rise along it. T is forward transfer over the held-out epilogue instances $\mathcal{E}$, the budget saved against the fresh run's u_i^{ref} on the same task and instance seed. Each measure also has a solve-rate twin, which replaces u_i with $\mathbf{1}[w_i < w_{\max}]$. The standard measures of average accuracy, backward transfer and forgetting (Lopez-Paz & Ranzato, 2017) assume task boundaries and a fixed test set per task, neither of which a recurring stream scored by attempts has.

3.4 THREE TRACKS

On **ARC-AGI-1**, the stream draws fresh, verified instances of the 400 training tasks from the ReARC generators (Hodel, 2024), and the held-out epilogue draws on the 400 evaluation tasks. **ARC-AGI-2** draws both from the 114 evaluation tasks new to the second edition. This edition has no generators, and so each instance serves a task's public pairs under a fresh seeded augmentation, a symmetry of the square and a relabelling of colours. In an audit of 40 of its training tasks, about two thirds did not commute with it, and a learned rule must then be re-framed on each instance at the price of about one demonstration (Appendix A). **ARC-AGI-3** is interactive (ARC Prize Foundation, 2026), and on it a game is a task and a level an instance. Since the k-th exposure to a game serves level k modulo the game's number of levels, a re-exposure is a new and typically harder level until the game wraps. We charge a level the actions it consumed against the human baseline for that level, with a cap of five times the baseline.

A *configuration* fixes everything about a stream except the mode and the seed: the track, the number of tasks, the stream and epilogue lengths, the number of users and the four parameters of Equation 2 (Table 6). The ARC-AGI-1 track runs on `arc1-small`, which draws 25 tasks from the 400 and serves 200 stream instances followed by a 25-task epilogue. Four tasks are active at once, spells last three instances and rests thirty, and a new task arrives every two instances. `arc2-small` has the same shape. On both, every task has the same supply of eight instances in every mode, and a task whose supply is spent is finished. The three modes are then the same task visits in three orders. `arc3-small` is six games over 48 instances with a four-game epilogue, served without a supply.

We define a track for each ARC-AGI edition, the schedule of its `small` configuration in workstreams mode at seed 0. Every system meets the same tasks in the same order, with the same instances. We make no track the default, since each edition gives signal for a different class of model (Section 4). ARC-AGI-1 separates cheap models that solve a third to a half of it zero-shot, and ARC-AGI-2 separates models that reach the same band on its calibrated evaluation tasks. Only a few models register on the interactive skill of ARC-AGI-3 at all. The tracks are not commensurable, and we define no pooled score. For comparison, we also run each track's tasks in the other two modes, sequential and random, but none of the larger configurations.

3.5 TWO REGIMES: THE INDIVIDUAL AND THE COLLECTIVE

So far, a single system meets one instance at a time, and whatever it carries may change between any two instances. A deployed system may instead serve many people at once, learning from *all* of them, and the collective regime tests such a system. A configuration has n_u users, each following a separate stream over the same tasks. A round serves one instance per user, all at once, and the system may change its state only between rounds. The individual regime is $n_u = 1$, and any learner can run in either regime, whether it keeps notes or updates its weights. The collective configuration has 100 users for 200 rounds and yields up to 160,000 verdicts, which likely also makes it the more

tractable of the two for a method that learns in its weights. Each user's tasks still arrive and go dormant, and every round is charged (Appendix B and Table 6).

4 LEARNERS

For our experiments, we run the individual regime and we test harness-based learners. The system meets its stream one instance at a time, and we assess what the system does with this experience. We group the learners into four rows. Two prompting floors (F) set the level that any learning method has to beat, and one method (M), a program-induction learner shaped to the domain, shows that the floors can be beaten. We run agent harnesses with every learning mechanism they ship switched on (H), and then again with their persistence switched off (H_0). In each row, the continual-learning method under test is whatever persists between instances.

Prompting floors (F). *Stateless* requests demonstrations on every instance and solves from them and the input, re-prompting with the feedback after a wrong answer up to the cap, and it carries nothing between instances. *Demo-cache* instead requests demonstrations once per task id and stores them, replaying them to the stateless solver on every later instance of that task. Its cost at the last exposure is then the same as at the second, and it serves as the control that separates memory of a task from memory of its demonstrations. Neither floor knows anything about the domain beyond the interface that every learner is given (Appendix C gives the prompts).

Method (M). *Program library* is the demo-cache with one addition. Whenever it has no program for a task, it asks the model for a Python program that maps input to output, keeping the program if it reproduces the stored demonstrations. When the task recurs, the stored program runs first, and after a wrong answer the library asks for a repair against the demonstrations, falling back to the cache's solver when no program survives. We do not count it as a floor, because it assumes two things specific to ARC, namely that the rule is a function of the grid and that the demonstrations can verify it. We include it to show what beating the demo-cache floor looks like on this measure.

Agent harnesses (H). A harness is a complete agent system with its own tools and memory files, which we drive as a black box through the protocol of Section 3.1. We run three harnesses, prime-agent (Karten et al., 2026), hermes-agent (Nous Research, 2026) and openclaw (OpenClaw contributors, 2026). Each instance is a new piece of work, for which the harness gets a fresh conversation, as these systems are meant to be used. Nothing carries over except what the harness persisted itself. We switch on every learning mechanism that a harness ships, namely prime-agent's automatic refinement, hermes-agent's post-turn review of its memory and skill files, and openclaw's skill workshop, memory flush and dreaming. To suit a stream that stands for weeks of work, we run openclaw's dreaming hourly instead of nightly. We also ask prime-agent for its documented cross-session refinement after every instance (Appendix D). Like the program library, a harness may test a candidate against the demonstrations for free, and only the submission is charged. We run each harness in a sandbox that can write only its private state and reach the network only through a tracking proxy, and each starts with no memory files and no bundled skills beyond the mechanics it ships with.

Harnesses without persistence (H_0). In row H_0, we run each harness with its persistence switched off and everything else equal. Before every instance, we stop the system, delete everything it wrote except the adapter's provider configuration, and start it again (Appendix D). Row H_0 keeps the harness's tools and within-instance skill, but none of its memory. Paired with row H on the same schedule, the difference between the two rows gives the value of persistence for that harness.

Models and tracks. We choose the models from the ARC Prize leaderboard at the time of submission. On each track, a preset (a model at one reasoning effort) qualifies when its verified semi-private zero-shot score lies between about 15 and 60 percent, the band in which we expect a harness to have room to learn. One model family, GPT-6 Luna, covers both grid tracks from within the band at the lowest cost on the board, and we use its reasoning effort as the ability axis. We run low, medium and high effort on ARC-AGI-1, the last above the band, and medium, high, xhigh and max effort on ARC-AGI-2. For ARC-AGI-3, we use Claude Opus 5.5 at low effort. The board is measured on the evaluation set, but the ARC-AGI-1 stream serves easier, generated training instances. For

			floors		hermes-agent		openclaw		prime-agent		program
GPT-6 Luna			stateless	demo-cache	stateless	persistent	stateless	persistent	stateless	persistent	library
effort	low	sequential	52.7 ±5.7	47.3 ±5.4	53.0 ±5.0	55.7 ±5.2	47.3 ±5.4	50.7 ±5.0	50.5 ±5.1	46.4 ±5.7	**28.6 ±6.5**
		random	56.0 ±6.1	41.7 ±6.0	51.1 ±5.6	50.0 ±5.3	52.3 ±5.8	48.4 ±5.7	53.2 ±5.0	49.2 ±5.3	**21.6 ±6.1**
		workstreams	58.3 ±5.6	49.6 ±6.2	51.9 ±5.1	56.8 ±5.3	55.1 ±5.5	49.6 ±5.3	52.1 ±5.0	48.6 ±6.0	**28.9 ±6.9**
		epilogue	61.3 ±7.9	55.6 ±8.0	69.8 ±7.7	52.4 ±8.0	63.1 ±7.7	68.4 ±6.9	57.8 ±7.9	62.7 ±7.8	**44.4 ±7.8**
		fresh	54.7 ±8.0	64.0 ±7.6	59.1 ±8.1	61.8 ±7.9	61.3 ±7.7	63.6 ±7.8	64.0 ±7.4	54.2 ±8.2	**48.4 ±7.9**
	medium	sequential	31.6 ±3.6	25.3 ±4.3	27.7 ±3.5	26.1 ±3.2	32.1 ±4.1	28.6 ±3.4	27.3 ±3.0	20.8 ±2.9	**10.4 ±3.1**
		random	32.8 ±3.7	20.8 ±3.5	29.5 ±3.5	26.2 ±3.6	31.4 ±4.0	31.1 ±3.7	30.7 ±3.6	25.2 ±3.5	**9.4 ±3.4**
		workstreams	36.0 ±3.8	27.6 ±4.8	31.1 ±3.8	25.0 ±3.4	29.4 ±3.1	32.4 ±3.6	28.4 ±3.2	23.7 ±4.2	**13.7 ±4.1**
		epilogue	33.8 ±6.6	30.7 ±5.6	32.4 ±5.7	32.9 ±6.3	**24.4 ±3.5**	32.9 ±4.9	28.9 ±5.8	26.2 ±5.1	24.9 ±5.9
		fresh	30.2 ±6.1	**28.4 ±5.3**	30.7 ±5.7	31.1 ±6.5	34.2 ±5.2	28.9 ±5.5	34.2 ±7.1	29.3 ±6.5	29.8 ±6.5
	high	sequential	24.3 ±2.7	15.1 ±2.7	22.8 ±2.5	19.9 ±2.4	22.3 ±2.2	24.6 ±2.8	21.9 ±2.7	14.3 ±1.6	**8.1 ±2.6**
		random	25.7 ±3.1	15.7 ±2.9	24.4 ±3.0	25.6 ±3.6	23.1 ±2.2	25.4 ±2.7	24.6 ±2.7	19.1 ±2.9	**5.3 ±2.2**
		workstreams	26.4 ±2.7	19.0 ±3.9	25.7 ±2.8	21.9 ±2.8	25.1 ±2.7	27.4 ±2.8	24.5 ±2.9	19.7 ±2.7	**7.4 ±2.7**
		epilogue	25.8 ±5.1	25.8 ±5.6	21.3 ±2.6	24.4 ±5.5	30.2 ±4.9	24.4 ±5.1	20.4 ±4.4	**17.3 ±2.4**	20.0 ±5.2
		fresh	20.4 ±4.2	22.2 ±4.9	23.6 ±5.0	22.2 ±4.9	21.3 ±2.1	26.7 ±4.2	27.6 ±5.9	**20.0 ±4.2**	22.2 ±5.1

Table 1: ARC-AGI-1 track, `arc1-small`: the share of the feedback budget used (percent, lower is better) by each system at each reasoning effort. Workstreams is the track's stream, whose epilogue is read against fresh, the same instances with the learner reset before each, where rows H and H_0 run one protocol and their spread shows the noise. Sequential and random are the stream in other orders. Small type is the standard error over tasks, and the best in each row is bold, even within error bars.

	saved against the demo-cache floor		saved by persistence (H against H_0)	
harness	low	medium	low	medium
prime-agent	−1.9 [−9.1, 5.7]	1.3 [−4.7, 8.0]	3.9 [−0.1, 8.1]	5.6 [2.3, 8.8]
hermes-agent	−8.0 [−14.0, −2.1]	−1.2 [−7.0, 5.2]	−2.1 [−4.7, 0.0]	3.7 [1.6, 6.1]
openclaw	−3.4 [−8.1, 2.0]	−6.1 [−10.0, −1.8]	1.9 [−1.0, 5.3]	0.3 [−1.7, 2.4]

Table 2: ARC-AGI-1: budget saved by each harness, in points, paired slot by slot and pooled over the three orders (600 slots), with 95 percent intervals by bootstrap over tasks. A negative value means that the harness used more.

this reason, we also check each preset against the stateless floor's first-exposure solve rate on the stream itself. On ARC-AGI-1, the stateless floor solved 40 percent of first exposures at first answer at low effort and 52 percent at medium, against board scores of 37.7 and 61.0 percent. Both efforts leave room to learn (Appendix A). On ARC-AGI-1, every row runs at every effort and in all three modes, which lets us compare the harnesses on identical weights at each effort. On ARC-AGI-2 and ARC-AGI-3, prime-agent is the only harness. Table 5 gives the rows and orders at each preset and marks the presets outside the band.

5 Results

We report the ARC-AGI-1 track here, and give every table and figure for each track in Appendix G, including ARC-AGI-3, which mostly measures acquisition across a game's levels (Section 3.4).

Memory of the rule. We compare the learners at low and medium effort, where every cell has met all 200 instances of its stream in each of the three orders. We pair each comparison slot by slot on the identical schedule and pool it over the three orders (600 slots), with a 95 percent interval from a bootstrap over tasks. Against the stateless floor, the demo-cache floor saves 9.5 [7.0, 12.2] points of budget at low effort and 8.9 [6.3, 11.4] at medium. This saving is the value of not paying for the demonstrations twice. The program library beats it by a wide margin, saving a further 19.8 [12.0, 28.1] points at low effort and 13.4 [7.3, 20.2] at medium. Over the later exposures of the workstreams stream, the library uses 26 and 11 percent of the budget at low and medium effort, compared with 49 and 26 for the demo-cache. The persistent harnesses use 48 to 56 and 23 to 33.

Memory in the harnesses. None of the three harnesses beats the demo-cache floor (Table 2), and two of them use measurably more. Their persistence, measured against row H_0 on the same slots,

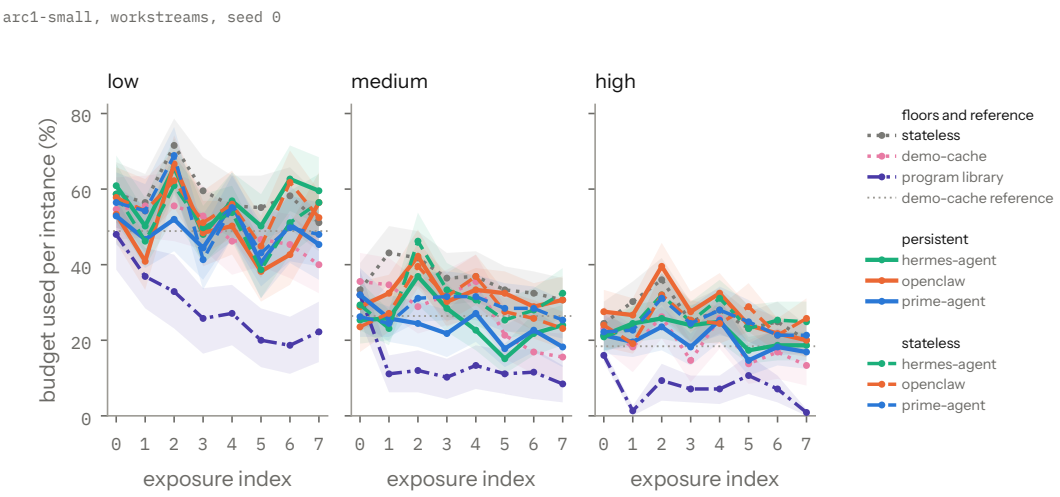

Figure 2: ARC-AGI-1: budget used by exposure index, the number of times the task had been met before, on the track's workstreams stream, with one panel per reasoning effort (lower is better). Floors are dotted and the method dash-dotted. Harnesses are solid with persistence and dashed without, and each line has a band from the 16th to the 84th percentile over task resamples. The thin horizontal line is the demo-cache reference, the mean cost of the demo-cache floor (pink) after its first exposure. By design, that floor saves nothing more once it has cached the demonstrations.

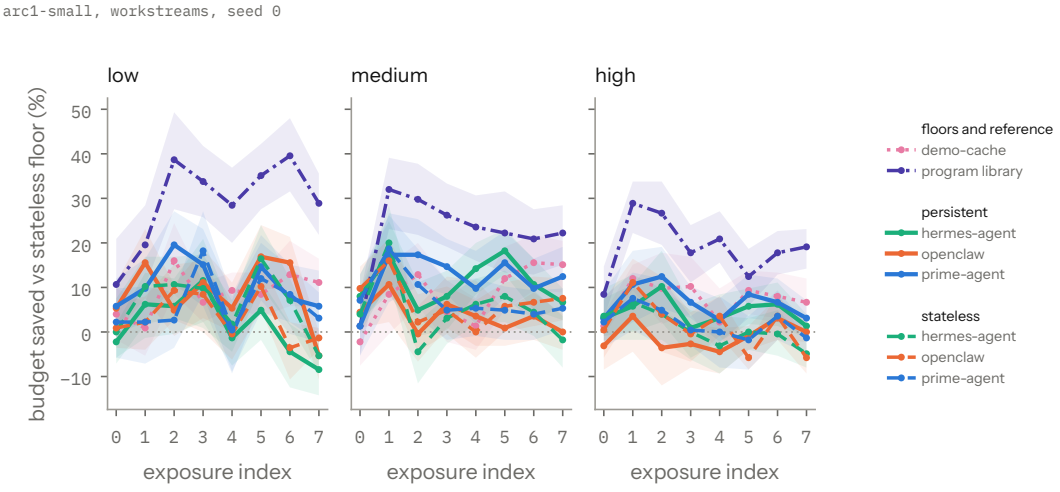

Figure 3: ARC-AGI-1: budget saved against the stateless floor on the same slot, by exposure index, on the workstreams stream of Figure 2, with one panel per effort (higher is better, and zero is the floor). Pairing on the slot removes the effect of the schedule, such as how hard the drawn instances are. A line that rises with exposure shows memory, and the gap between a harness's solid and dashed lines is the value of its persistence. Line styles and bands are as in Figure 2.

is measurable at medium effort for prime-agent and hermes-agent, but it is worth a few points at most, and on the point estimates less than the cached demonstrations save the stateless floor. The state that each harness stored shows why (Appendix E). The three are built for lessons that transfer across unrelated work, none ties its store to the task id, and each breaks at a different step between an instance and its return. prime-agent writes a prose entry per task after every instance, but it shows the model only a fixed digest of its store, in which the entry for a returning task is often out of view,

and the model rarely retrieves it. hermes-agent always shows the model what it stored, but it stores general advice on how to approach a task and almost nothing about the task itself. openclaw searches its history by task id, but its consolidation never writes a memory and old transcripts are pruned, and so these searches almost never return an earlier answer. All three also pay for the demonstrations again on most instances, which the demo-cache floor never does, and this alone costs each of them more than its whole distance from that floor.

Transfer. Transfer to the held-out epilogue is within noise for every system. We compare each system with the fresh reference on the same tasks and instances, pooled over the epilogues of the three orders. The program library saves 1.3 [-7.3, 10.1] points at low effort and 6.5 [-0.1, 14.8] at medium, and no harness's interval excludes zero. The only interval that does is the demo-cache's at low effort, 6.2 [0.4, 13.6], and since that floor carries nothing to a task it has not met, we take it as one false positive among ten intervals. The library's advantage appears to be memory of the rules of the tasks it has met, and we see no sign of it carrying over to new tasks. We attribute this to the learners, since ARC rules share their low-level operations, but the library keeps one whole program per task and no harness stores reusable primitives (Appendix A).

6 Discussion

Agent systems now carry memory and skills from one job to the next, and whether they learn from that experience matters more as they take on longer work. Continual-ARC measures this where a gain depends less on what the model already knows. Its tasks return with fresh instances and its feedback is charged on one scale for any learner, so it tells a system that remembers a task's rule apart from one that only caches the demonstrations. A system that relearns a task on each return pays for the same feedback every time, and the budget used records exactly that cost.

By this measure, the three harnesses we tested do not yet learn, and none falls below the demo-cache floor that a simple program library beats (Section 5). They are built for lessons that carry across unrelated work, and none keeps the demonstrations it has paid for or an executable rule. The one that writes notes per task, prime-agent, often fails to show them to the model when the task returns. We suspect that a harness would gain most from keeping verified programs keyed on the task and refactoring them into reusable primitives, which ARC's shared building blocks should let it carry to new tasks (Appendix A).

The benchmark is built to stay useful as such systems improve, since a memorised answer does not fit a fresh instance and the three tracks span models from a cent per task to the frontier, with larger configurations defined for systems that outgrow the small ones. Its floors also give harness developers a simple test, since a mechanism that does not take a returning task below the demo-cache floor has not learned it. Its limits are those of ARC's public tasks. The stream measures generalisation within a task scope that developers know, and on ARC-AGI-1 the epilogue returns each task's original demonstrations, which are worth more than three fresh pairs. We also do not quantify each task's generalisation difficulty, which Chollet (2019) lists as ARC's first weakness.

We see the largest opening in the model that drives the harness, since a stronger driver already appears to get more from the same harness. Persistence is worth more to prime-agent and hermes-agent at medium effort than at low (Table 2), and on ARC-AGI-2 it saves prime-agent 7 to 11 points at medium effort and above. On ARC-AGI-3, driven by Claude Opus 5.5 in one run of each row, only prime-agent's persistent row used less on levels it met a second time (Appendix G). Frontier models are expensive, however, and many deployments need a model on their own infrastructure for privacy and data residency. Training a smaller model to drive a custom harness in a custom domain, learning what to store and when to retrieve it, would bring harness capabilities to such deployments. Continual-ARC is a natural testbed for it, since its domain is one the model largely does not know and its charged verdicts are the reward the training needs. Its collective regime supplies up to 160,000 of them per configuration while still scoring every one (Section 3.5), and it puts a model trained between rounds on the same scale as a harness that rewrites itself between rounds. Better harnesses and better-trained models will both need a measure on which caching and recall cannot pass for learning, and Continual-ARC was built to be that measure.

AI USE STATEMENT

ICLR 2027 asks authors to say which parts of a paper were produced with generative AI and how they were checked. This section does not count toward the page limit.

We used an AI coding assistant (Claude, Anthropic) throughout this work. It drafted the abstract and every section of this paper from the author's notes, which record each design decision, the reason for it, the objection a reviewer would raise, and the answer. The author edited every draft. It carried out the literature search behind Section 2 and Appendix F, screening roughly two hundred works. The author checked every entry in the tables against the work's own abstract, and every comparative claim in the paper against those notes. It proposed parts of the experimental grid of Section 4, and the author decided on them. It wrote most of the code in the three repositories that we will release, including the ARC-AGI-2 and ARC-AGI-3 environments and the harness adapters, under tests and smoke runs that the author reviewed. It also launched and monitored the runs on remote machines at the author's direction. Finally, it computed the analyses of Section 5 and Appendices E and G from the run records, including the paired comparisons and their intervals, and it read the harnesses' stored state for Appendix E. It also made the per-task calls of the augmentation audit in Appendix A. Every table and figure is regenerated from the run directories when the paper is built. A script in the paper's repository recomputes the numbers quoted in Section 5 and the paired comparisons of Appendices E and G.

No AI was used to generate data. Every instance in the stream comes from the Re-ARC generators, a deterministic augmentation or the ARC-AGI game engine. No AI was used to prove claims, and the equations in Section 3 are definitions. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work involves no human subjects and no personal data. Every task comes from the public ARC-AGI tasks and games or from the Re-ARC generators, which the software fetches from their sources and does not redistribute. ARC-AGI-3 levels are charged against the human baselines that ARC Prize publishes.

The harnesses we evaluate write and run their own code, and one reached the evaluation itself in an early run. Section 4 and Appendix D describe how we confine each one, and we recommend the same to anyone running agents on the benchmark.

Our findings are negative for three publicly available harnesses, none of them from the author's employer, which builds agent systems. We run each as its authors ship it, with every learning mechanism it offers switched on, and we trace each shortfall to a specific step in its design (Appendix E). The adapters are in the supplementary material, so that the harnesses' authors can check how their systems were run.

Continual learning in deployment also raises questions of privacy that this benchmark does not address. In the collective regime, one system learns from many users at once, and what it keeps from one user could reach another. Our streams are abstract puzzles with no user data, and the benchmark does not measure leakage between users.

REPRODUCIBILITY STATEMENT

Section 3 defines the tracks and their measures, and Appendices A and B give every configuration and the process that generates each schedule (Algorithm 1). Schedules are deterministic, and on a track every system meets the same instances in the same order. Appendix C gives every text that a learner is shown, and Appendix D describes how we run each harness, with the versions of every component. Section 4 and Appendix A give the model presets and the rule that chose them. The supplementary material holds an anonymised copy of the code of the benchmark and of every learner, including the harness adapters, with the source of this paper and the records of every run it reports, from which its build regenerates every table and figure. We will release the code publicly with the paper. The models are sampled through provider APIs, so a rerun will differ from ours by about the spread we report, such as the gap between rows H and H_0 in the fresh column of Table 1.

REFERENCES

Adaptive Agent Team, Jakob Bauer, Kate Baumli, Satinder Baveja, Feryal Behbahani, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, Vibhavari Dasagi, Lucy Gonzalez, Karol Gregor, Edward Hughes, Sheleem Kashem, Maria Loks-Thompson, Hannah Openshaw, Jack Parker-Holder, Shreya Pathak, Nicolas Perez-Nieves, Nemanja Rakicevic, Tim Rocktäschel, Yannick Schroecker, Jakub Sygnowski, Karl Tuyls, Sarah York, Alexander Zacherl, and Lei Zhang. Human-timescale adaptation in an open-ended task space. *arXiv preprint arXiv:2301.07608*, 2023.

Qingyao Ai, Yichen Tang, Changyue Wang, Jianming Long, Weihang Su, and Yiqun Liu. MemoryBench: A benchmark for memory and continual learning in LLM systems. *arXiv preprint arXiv:2510.17281*, 2025.

Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. The surprising effectiveness of test-time training for few-shot learning. *arXiv preprint arXiv:2411.07279*, 2024. v1 titled "The Surprising Effectiveness of Test-Time Training for Abstract Reasoning".

Simon Alford, Anshula Gandhi, Akshay Rangamani, Andrzej Banburski, Tony Wang, Sylee Dandekar, John Chin, Tomaso Poggio, and Peter Chin. Neural-guided, bidirectional program search for abstraction and reasoning. In *Complex Networks & Their Applications X*, 2021. arXiv:2110.11536.

ARC Prize Foundation. ARC-AGI-3: A new challenge for frontier agentic intelligence. *arXiv preprint arXiv:2603.24621*, 2026.

Parth Asawa, Christopher M. Glaze, Gabriel Orlanski, Ramya Ramakrishnan, Benji Xu, Asim Biswal, Vincent Sunn Chen, Frederic Sala, Matei Zaharia, and Joseph E. Gonzalez. Continual learning bench: Evaluating frontier AI systems in real-world stateful environments. *arXiv preprint arXiv:2606.05661*, 2026.

Ashpreet Bedi. Learning machines: the memory that cracked ARC-AGI-3 public set. Agno blog, 2026. URL `https://www.agno.com/articles/learning-machines`.

Jörg Bornschein, Alexandre Galashov, Ross Hemsley, Amal Rannen-Triki, Yutian Chen, Arslan Chaudhry, Xu Owen He, Arthur Douillard, Massimo Caccia, Qixuang Feng, Jiajun Shen, Sylvestre-Alvise Rebuffi, Kitty Stacpoole, Diego de las Casas, Will Hawkins, Angeliki Lazaridou, Yee Whye Teh, Andrei A. Rusu, Razvan Pascanu, and Marc'Aurelio Ranzato. NEVIS'22: A stream of 100 tasks sampled from 30 years of computer vision research. *arXiv preprint arXiv:2211.11747*, 2022.

Massimo Caccia, Pau Rodriguez, Oleksiy Ostapenko, Fabrice Normandin, Min Lin, Lucas Caccia, Issam Laradji, Irina Rish, Alexandre Lacoste, David Vazquez, and Laurent Charlin. Online fast adaptation and knowledge accumulation (OSAKA): a new approach to continual learning. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020. arXiv:2003.05856.

Yuxuan Cai, Yipeng Hao, Jie Zhou, Hang Yan, Zhikai Lei, Rui Zhen, Zhenhua Han, Yutao Yang, Junsong Li, Qianjun Pan, Tianyu Huai, Qin Chen, Xin Li, Kai Chen, Bo Zhang, Xipeng Qiu, and Liang He. Building self-evolving agents via experience-driven lifelong learning: A framework and benchmark. *arXiv preprint arXiv:2508.19005*, 2025.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. In *International Conference on Learning Representations (ICLR)*, 2019. arXiv:1812.00420.

Howard Chen, Noam Razin, Karthik Narasimhan, and Danqi Chen. Retaining by doing: The role of on-policy data in mitigating forgetting. In *International Conference on Machine Learning*, 2026. arXiv:2510.18874.

Jiangjie Chen, Qianyu He, Siyu Yuan, Aili Chen, Zhicheng Cai, Weinan Dai, Hongli Yu, Qiying Yu, Xuefeng Li, Jiaze Chen, Hao Zhou, and Mingxuan Wang. Enigmata: Scaling logical reasoning in large language models with synthetic verifiable puzzles. In *Advances in Neural Information Processing Systems*, 2025. arXiv:2505.19914.

Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. In *International Conference on Learning Representations*, 2019. arXiv:1810.08272.

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

François Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. ARC-AGI-2: A new challenge for frontier AI reasoning systems. *arXiv preprint arXiv:2505.11831*, 2025.

François Chollet, Mike Knoop, Gregory Kamradt, and Bryan Landers. ARC Prize 2025: Technical report. *arXiv preprint arXiv:2601.10904*, 2026.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020. arXiv:1912.01588.

Shibhansh Dohare, J. Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A. Rupam Mahmood, and Richard S. Sutton. Loss of plasticity in deep continual learning. *Nature*, 632(8026): 768–774, 2024. doi: 10.1038/s41586-024-07711-7.

Alexis Fox, Junlin Wang, Paul Rosu, and Bhuwan Dhingra. PRO-LONG: Programmatic memory enables long-horizon reasoning. *arXiv preprint arXiv:2607.20064*, 2026.

Daniel Franzen, Jan Disselhoff, and David Hartmann. Product of experts with LLMs: Boosting performance on ARC is a matter of perspective. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025. arXiv:2505.07859.

Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

Tianyi Guan, Yiding Wang, Haotong Yang, Siyuan Cao, Shirui Liu, Yi Hu, Jiaqi Li, and Muhan Zhang. ContinualSkillBench: Can LLM agents truly evolve their capabilities? *arXiv preprint arXiv:2608.03874*, 2026.

Haiyun Guo, Zhiyan Hou, Yandu Sun, Jinghan He, Yu Chen, Yuzhe Zhou, Yuheng Jia, Jinqiao Wang, and Tat-Seng Chua. MLLM-CTBench: A benchmark for continual instruction tuning with reasoning process diagnosis. *arXiv preprint arXiv:2508.08275*, 2025.

Hamed Hemati, Andrea Cossu, Antonio Carta, Julio Hurtado, Lorenzo Pellegrini, Davide Bacciu, Vincenzo Lomonaco, and Damian Borth. Class-incremental learning with repetition. In *Proceedings of the 2nd Conference on Lifelong Learning Agents (CoLLAs)*, 2023. arXiv:2301.11396.

Hamed Hemati, Lorenzo Pellegrini, Xiaotian Duan, Zixuan Zhao, Fangfang Xia, Marc Masana, Benedikt Tscheschner, Eduardo Veas, Yuxiang Zheng, Shiji Zhao, Shao-Yuan Li, Sheng-Jun Huang, Vincenzo Lomonaco, and Gido M. van de Ven. Continual learning in the presence of repetition. *Neural Networks*, 2025. arXiv:2405.04101; CLVision 2023 challenge report.

Matthew Ho, Chen Si, Zhaoxiang Feng, Fangxu Yu, et al. ArcMemo: Abstract reasoning composition with lifelong LLM memory. *arXiv preprint arXiv:2509.04439*, 2025.

Michael Hodel. Addressing the abstraction and reasoning corpus via procedural example generation. *arXiv preprint arXiv:2404.07353*, 2024.

Sihang Jiang, Lipeng Ma, Zhonghua Hong, Keyi Wang, Zhiyu Lu, Tengfei Wang, Shisong Chen, Jinghao Zhang, Tianjun Pan, Weijia Li, Jiaqing Liang, and Yanghua Xiao. SEA-Eval: A benchmark for evaluating self-evolving agents beyond episodic assessment. *arXiv preprint arXiv:2604.08988*, 2026.

Thomas Joshi, Shayan Chowdhury, and Fatih Uysal. SWE-Bench-CL: Continual learning for coding agents. *arXiv preprint arXiv:2507.00014*, 2025.

Seth Karten. Continual harness: An efficient self-improving agent on ARC-AGI-3. Prime Intellect write-up, 2026. URL https://continual-harness.github.io/.

Seth Karten, Alex L. Zhang, Kevin Thomas, Sebastian Müller, Elie Bakouch, Daniel Auras, Mika Senghaas, Fares Obeid, Konstantin Dunas, Johannes Hagemann, and Sami Jaghouar. Prime agent: A self-improving RLM harness. *arXiv preprint arXiv:2608.23552*, 2026.

Heinrich Küttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020. arXiv:2006.13760.

Hosung Lee, Sejin Kim, Seungpil Lee, Sanha Hwang, Jihwan Lee, Byung-Jun Lee, and Sundong Kim. ARCLE: The abstraction and reasoning corpus learning environment for reinforcement learning. In *Conference on Lifelong Learning Agents (CoLLAs)*, 2024. arXiv:2407.20806.

Jens Lehmann, Syeda Khushbakht, Nikoo Salehfard, Nur A Zarin Nishat, Dhananjay Bhandiwad, Andrei Aioanei, and Sahar Vahdati. ARC-TGI: Human-validated task generators with reasoning chain templates for ARC-AGI. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, 2026. arXiv:2603.05099.

Zhiqiu Lin, Jia Shi, Deepak Pathak, and Deva Ramanan. The clear benchmark: Continual learning on real-world imagery. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2022. arXiv:2201.06289.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems (NeurIPS) Datasets and Benchmarks Track*, 2023. arXiv:2306.03310.

Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition. In *Proceedings of the 1st Conference on Robot Learning (CoRL)*, 2017. arXiv:1705.03550.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems 30 (NeurIPS)*, 2017. arXiv:1706.08840.

Mao-Lin Luo, Zhe-Xu Wang, Zi-Hao Zhou, Bo Ye, Jian Zhao, Min-Ling Zhang, and Tong Wei. Rl forgets! towards continual policy optimization. *arXiv preprint arXiv:2607.04364*, 2026.

Michael D. Moffitt. ARC-GEN: A mimetic procedural benchmark generator for the abstraction and reasoning corpus. *arXiv preprint arXiv:2511.00162*, 2025.

Alexander New, Megan Baker, Eric Nguyen, and Gautam Vallabha. Lifelong learning metrics. *arXiv preprint arXiv:2201.08278*, 2022.

Nous Research. Hermes agent. https://github.com/NousResearch/hermes-agent, 2026. Version 0.20.2, accessed September 2026.

Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, Nat McAleese, Nathalie Bradley-Schmieg, Nathaniel Wong, Nicolas Porcel, Roberta Raileanu, Steph Hughes-Fitt, Valentin Dalibard, and Wojciech Marian Czarnecki. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021.

OpenClaw contributors. OpenClaw. https://openclaw.ai, 2026. Version 2026.8.1, accessed September 2026.

Emmanouil Antonios Platanios, Abulhair Saparov, and Tom Mitchell. Jelly bean world: A testbed for never-ending learning. In *International Conference on Learning Representations (ICLR)*, 2020. arXiv:2002.06306.

Julien Pourcel, Cédric Colas, and Pierre-Yves Oudeyer. Self-improving language models for evolutionary program synthesis: A case study on ARC-AGI. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025. arXiv:2507.14172.

Sam Powers, Eliot Xing, Eric Kolve, Roozbeh Mottaghi, and Abhinav Gupta. CORA: Benchmarks, baselines, and metrics as a platform for continual reinforcement learning agents. In *Proceedings of The 1st Conference on Lifelong Learning Agents (CoLLAs)*, volume 199 of *Proceedings of Machine Learning Research*, pp. 705–743, 2022. arXiv:2110.10067.

Rulin Shao, Shuyue Stella Li, Rui Xin, Scott Geng, Yiping Wang, Sewoong Oh, Simon Shaolei Du, Nathan Lambert, Sewon Min, Ranjay Krishna, Yulia Tsvetkov, Hannaneh Hajishirzi, Pang Wei Koh, and Luke Zettlemoyer. Spurious rewards: Rethinking training signals in RLVR. *arXiv preprint arXiv:2506.10947*, 2025.

Idan Shenfeld, Jyothish Pari, and Pulkit Agrawal. Rl's razor: Why online reinforcement learning forgets less. *arXiv preprint arXiv:2509.04259*, 2025.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, 2023. arXiv:2303.11366.

Yiheng Shu, Bernal Jiménez Gutiérrez, Saisri Padmaja Jonnalagedda, Yuguang Yao, Huan Sun, and Yu Su. AgentCL: Toward rigorous evaluation of continual learning in language agents. *arXiv preprint arXiv:2606.02461*, 2026.

Zafir Stojanovski, Oliver Stanley, Joe Sharratt, Richard Jones, Abdulhakeem Adefioye, Jean Kaddour, and Andreas Köpf. Reasoning gym: Reasoning environments for reinforcement learning with verifiable rewards. In *Advances in Neural Information Processing Systems*, 2025. arXiv:2505.24760.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. *arXiv preprint arXiv:2504.07952*, 2025.

Tristan Tomilin, Meng Fang, Yudi Zhang, and Mykola Pechenizkiy. COOM: A game benchmark for continual reinforcement learning. In *Advances in Neural Information Processing Systems 36 (NeurIPS) Datasets and Benchmarks Track*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/d61d9f4fe4357296cb658795fd7999f0-Abstract-Datasets_and_Benchmarks.html.

Tristan Tomilin, Luka van den Boogaard, Samuel Garcin, Constantin Ruhdorfer, Bram Grooten, Fabrice Kusters, Yali Du, Andreas Bulling, Mykola Pechenizkiy, and Meng Fang. MEAL: A benchmark for continual multi-agent reinforcement learning. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. arXiv:2506.14990.

Gido M. van de Ven and Andreas S. Tolias. Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*, 2019.

Tom Veniat, Ludovic Denoyer, and Marc'Aurelio Ranzato. Efficient continual learning with modular networks and task-driven priors. In *International Conference on Learning Representations (ICLR)*, 2021. arXiv:2012.12631; introduces the CTrL benchmark.

Xiao Wang, Yuansen Zhang, Tianze Chen, Songyang Gao, Senjie Jin, Xianjun Yang, Zhiheng Xi, Rui Zheng, Yicheng Zou, Tao Gui, Qi Zhang, and Xuanjing Huang. TRACE: A comprehensive benchmark for continual learning in large language models. *arXiv preprint arXiv:2310.06762*, 2023.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H. Chi, Chi Wang, Shuo Chen, Fernando Pereira, Wang-Cheng Kang, and Derek Zhiyuan Cheng. Evo-Memory: Benchmarking LLM agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*, 2025.

Maciej Wołczyk, Michał Zając, Razvan Pascanu, Łukasz Kuciński, and Piotr Miłoś. Continual world: A robotic benchmark for continual reinforcement learning. In *Advances in Neural Information Processing Systems 34 (NeurIPS)*, 2021. arXiv:2105.10919.

Cheng-Kuang Wu, Zhi Rui Tam, Chieh-Yen Lin, Yun-Nung Chen, and Hung-yi Lee. StreamBench: Towards benchmarking continuous improvement of language agents. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024), Datasets and Benchmarks Track*, 2024. arXiv:2406.08747.

Mingqi Wu, Zhihao Zhang, Qiaole Dong, Zhiheng Xi, Jun Zhao, Senjie Jin, Xiaoran Fan, Yuhao Zhou, Huijie Lv, Ming Zhang, Yanwei Fu, Qin Liu, Songyang Zhang, and Qi Zhang. Reasoning or memorization? unreliable results of reinforcement learning due to data contamination. *arXiv preprint arXiv:2507.10532*, 2025.

Jundong Xu, Qingchuan Li, Jiaying Wu, Yihuai Lan, Shuyue Stella Li, Huichi Zhou, Bowen Jiang, Lei Wang, Jun Wang, Anh Tuan Luu, Caiming Xiong, Hae Won Park, Bryan Hooi, and Zhiyuan Hu. EvoArena: Tracking memory evolution for robust LLM agents in dynamic environments. *arXiv preprint arXiv:2606.13681*, 2026.

Dong Yan, Jian Liang, Dapeng Hu, Ran He, Nicholas Jing Yuan, Qi Zhang, and Tieniu Tan. AgentStream: How well do self-evolving LLM agents perform under streaming tasks? *arXiv preprint arXiv:2608.00155*, 2026.

Xidong Yang, Xingyi Zhang, Wenhao Li, Wenyan Liu, Junjie Sheng, Yun Hua, Wei Yin, Tao Fang, Chuyun Shen, and Xiangfeng Wang. PATH-Bench: Path-dependent evaluation of lifelong agents. *arXiv preprint arXiv:2608.01149*, 2026.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in LLMs beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

Zhiyuan Zeng, Hamish Ivison, Yiping Wang, Lifan Yuan, Shuyue Stella Li, Zhuorui Ye, Siting Li, Jacqueline He, Runlong Zhou, Tong Chen, Chenyang Zhao, Yulia Tsvetkov, Simon Shaolei Du, Natasha Jaques, Hao Peng, Pang Wei Koh, and Hannaneh Hajishirzi. Rlve: Scaling up reinforcement learning for language models with adaptive verifiable environments. In *International Conference on Machine Learning*, 2026. arXiv:2511.07317.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017. arXiv:1703.04200.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026. arXiv:2510.04618.

Zihan Zhang, Meng Fang, Ling Chen, and Mohammad-Reza Namazi-Rad. CITB: A benchmark for continual instruction tuning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023. arXiv:2310.14510.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024. arXiv:2308.10144.

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. LifelongAgentBench: Evaluating LLM agents as lifelong learners. *arXiv preprint arXiv:2505.11942*, 2025.

Shanshan Zhong, Yi Lu, Jingjie Ning, Yibing Wan, Lihan Feng, Yuyi Ao, Leonardo F. R. Ribeiro, Markus Dreyer, Sean Ammirati, and Chenyan Xiong. SkillLearnBench: Benchmarking continual learning methods for agent skill generation on real-world tasks. *arXiv preprint arXiv:2604.20087*, 2026.

Huichi Zhou, Yihang Chen, Siyuan Guo, Xue Yan, Kin Hei Lee, Zihan Wang, Ka Yiu Lee, Guchun Zhang, Kun Shao, Linyi Yang, and Jun Wang. Memento: Fine-tuning llm agents without fine-tuning llms. *arXiv preprint arXiv:2508.16153*, 2025.

Adam Zweiger, Jyothish Pari, Han Guo, Ekin Akyürek, Yoon Kim, and Pulkit Agrawal. Self-adapting language models. *arXiv preprint arXiv:2506.10943*, 2025.

A BENCHMARK DETAILS

ARC-AGI-1. We draw the stream from the 400 ARC-AGI-1 training tasks through the Re-ARC generators (Hodel, 2024), pinned by commit. Given a task id and a seed, the generator produces an input grid and the verifier produces its output, and we keep only the pairs that the verifier accepts, discarding any demonstration whose input equals the test input. A learner cannot cache outputs, since every instance is fresh, and caching demonstrations has a price. The held-out epilogue draws on the 400 ARC-AGI-1 evaluation tasks, which have no generators, and each of them appears once, with its original demonstrations available on request.

Why the epilogue can show transfer. ARC tasks differ in their rules but share the operations from which the rules are built. The Re-ARC generators (Hodel, 2024), from which the ARC-AGI-1 stream is drawn, implement all 400 training rules with one library of 160 primitive functions, and 53 of these primitives appear in ten or more of the generators, among them filling cells, shifting objects and filtering sets of cells (`scripts/rearc_primitives.py`). A learner that refactors what it learns into task-specific compositions of reusable low-level primitives, and keeps the primitives, should therefore meet a held-out task with much of the machinery that the task needs already in place. Library learning has shown this on ARC, where abstractions that DreamCoder built from solved tasks let it solve progressively harder ones (Alford et al., 2021), and concept-level memory distilled from solution traces raised a language model's score over a no-memory baseline (Ho et al., 2025). None of the learners in this paper keeps such primitives. The program library stores one whole program per task, and the harnesses store prose, general advice or a skill for searching their history (Appendix E), and so we attribute the absence of transfer in Section 5 to these learners.

term	meaning
task	an ARC rule, known to the learner by its id; on ARC-AGI-3, a game
instance	one input of a task to solve: generated fresh on ARC-AGI-1, freshly augmented on ARC-AGI-2, a level on ARC-AGI-3
slot	one entry of a schedule, naming a task, an instance seed and a phase
phase	the stream, or the epilogue that follows it
epilogue	held-out tasks never seen before, served once each after the stream and unannounced
schedule	the ordered list of slots a system is served, fixed by a configuration, a mode and a seed
configuration	the track, the numbers of tasks and users, the stream and epilogue lengths and the workstreams parameters, such as `arc1-small` (Table 6)
mode	the order of presentation over the same tasks: sequential, random or workstreams (Section 3.2)
regime	individual (one user, whose system may change between any two instances) or collective (many users, served in rounds, with changes only between rounds)
track	one fixed schedule per ARC-AGI edition: the `small` configuration in workstreams mode with its epilogue, in the individual regime, at seed 0
fresh reference	the epilogue served alone, with the learner reset before each instance, against which forward transfer is measured
exposure index	the number of earlier rounds in which an instance's task appeared
gap	the number of rounds since the task's previous round
row	a kind of learner: prompting floors (F), the method (M), and harnesses with persistence (H) and without it (H_0)
preset	a model at one reasoning effort
cell	one run: a row at one preset in one mode

Table 3: The terms of Sections 3 and 4.

ARC-AGI-2. The second edition has 1,000 public training and 120 evaluation tasks, and no generators. Its training set is mostly the first edition (767 of the 1,000 are ARC-AGI-1 tasks, as are 6 of the 120 evaluation tasks), and it is uncalibrated. The leaderboard measures a model on the calibrated evaluation set. For this reason, we draw both the stream and the epilogue from the 114 evaluation tasks new to ARC-AGI-2, with 25 for the stream and 25 held out from the remaining 89. The stream serves each task's public pairs under a seeded augmentation, one of the eight symmetries of the square and a relabelling of colours 1 to 9. We draw the augmentation afresh on every instance and share it between the test pair and the demonstrations served with it. A cached output is useless under a fresh augmentation, and only a rule learned up to symmetry and relabelling carries

mode	active	spell	rest	arrivals
sequential	1	all its instances	never returns	next at once
workstreams	n_c	$\mathrm{Exp}(b)$	$\mathrm{LogNormal}(\ln m, \sigma)$	$\mathrm{Exp}(a)$
random	all	never ends	none	all at start

Table 4: The three modes as settings of one process, seeded by configuration and seed, so every stream is deterministic and the instance seed of slot t is the same in every mode.

over from one instance to the next (Akyürek et al., 2024). With only its public pairs to draw on, the demonstration request returns every other pair under the instance's augmentation, and a second request could only repeat the first. The augmentation assumes that a task's rule commutes with symmetry and relabelling. To test this, we audited 40 randomly drawn training tasks from images of their demonstration pairs, and the per-task calls, several of which are borderline, are in the supplementary material. Of these, 12 depend on orientation and 21 give a fixed colour a distinguished role. In all, 26 fail at least one of the two and 14 commute with both. We expect the cost to be bounded, because every instance is internally consistent and one demonstration request in most cases pins the frame. On a non-commuting task, a perfect learner uses about 11 percent of the budget. The audit is of training tasks, and if the evaluation tasks fail to commute as often, no system's exposure curve on this track can fall much below 7 percent. Generators would remove this offset, but none exist for the evaluation tasks, and ARC-GEN (Moffitt, 2025), at commit a15cbdb4, covers 891 of the 1,000 training tasks and none of the evaluation set. The epilogue serves its tasks unaugmented.

ARC-AGI-3. The third edition is interactive (ARC Prize Foundation, 2026). A game has several levels, in which the agent sees a 64 by 64 frame and acts through up to seven actions, and the edition scores efficiency as the actions taken against a human baseline for each level. On this track a game is a task and a level is an instance, and successive exposures to a game climb through its levels and then wrap, as Section 3.4 describes. In the small configuration, 48 stream instances fall on six games of six to nine levels each. The process without a supply serves some games more often than others, and three of the six games wrap in the seed-0 stream. On this track, the exposure curve mixes level difficulty with what is retained, and only a level met a second time separates the two (Appendix G). A configuration that gives every game more exposures than it has levels would separate them throughout, and we leave this to a larger run. An instance ends when the level is completed, or when actions plus resets reach a cap of five times the human baseline for that level, at most 1,000 actions. We charge a solved instance its actions plus resets divided by that baseline, and a failed instance the cap. The score then has the form of Equation 1 with $C_{\max} = 5$, and the complement of u_i has the shape of the edition's score, an efficiency that is zero for a level not completed. Recurrence on this track tests whether the skill of playing a level is kept, since levels are fixed. We pin the 25 public games by version and hold the epilogue games out from them.

Model selection. The band of Section 4 has a ceiling and a floor. Above it, a preset solves nearly every instance at first sight, and its exposure curves would measure memory of answers. Below it, a preset cannot hold the action protocol. The leaderboard's verified score ranks the candidates but does not settle the choice, because its protocol gives demonstrations on the evaluation tasks while ours charges for them. On ARC-AGI-1, our stream also serves generated training instances, which are easier than the evaluation set. We therefore check the stateless floor's first-exposure solve rate on the stream against the verified score, since a preset whose first-exposure rate on the stream approaches one has no room to learn on this track, whatever the leaderboard says. Table 5 lists the presets that run, most of them selected by the rule and the exceptions marked. The selection is scripted from the leaderboard's published data. We take the check from the fresh reference of the floor cells, with 25 tasks per cell and a standard error of about ten points. On ARC-AGI-1, the stateless floor solved 32 percent of tasks with no wrong attempt and 56 percent within the cap at low effort, 52 and 88 percent at medium, and 68 and 96 percent at high, against 37.7, 61.0 and 70.3 on the board. On the ARC-AGI-2 pool of evaluation tasks new to the second edition, it solved 0, 0, 24, 44 and 60 percent at first answer and 4, 40, 60, 68 and 88 percent within the cap at low, medium, high, xhigh and max effort, against 4.6, 18.1, 31.4, 41.9 and 59.3 on the board. The first exposures of the stream cells agree with the fresh reference within their error bars. Pooled over the three orders of the ARC-AGI-1 stream, the stateless floor solved 40 percent of first exposures at first answer and 60 percent within

the cap at low effort, and 52 and 77 percent at medium. The eight attempts with a verdict add 20 and 25 points over the first answer. On ARC-AGI-2, medium and high effort lie inside the band, with xhigh at its upper edge, max above it and low below. We run all four efforts on the board, with max as the ceiling of the effort axis.

track	model, effort	verified	USD/task	rows
ARC-AGI-1	GPT-6 Luna, low	37.7%	0.003	floors, method, all harnesses (H and H_0)
	GPT-6 Luna, medium	61.0%	0.006	floors, method, all harnesses (H and H_0)
	GPT-6 Luna, high†	70.3%	0.009	floors, method, all harnesses (H and H_0)
ARC-AGI-2	GPT-6 Luna, low†	4.6%	0.005	floors (fresh and random)
	GPT-6 Luna, medium	18.1%	0.012	floors, method, prime-agent (H and H_0)
	GPT-6 Luna, high	31.4%	0.023	floors, method, prime-agent (H and H_0)
	GPT-6 Luna, xhigh	41.9%	0.032	floors (fresh), prime-agent (H and H_0)
	GPT-6 Luna, max	59.3%	0.062	floors (fresh), prime-agent (H and H_0)
ARC-AGI-3	Claude Opus 5.5, low†	–		prime-agent (H and H_0)

Table 5: Models, chosen from the ARC Prize leaderboard at the time of submission. By default a track runs the presets of one family inside the 15 to 60 percent zero-shot band. † marks the three presets outside it or not on the board: ARC-AGI-1 at high effort, ARC-AGI-2 at low effort, which runs the floors alone as the low end of the first-exposure check, and Claude Opus 5.5 on ARC-AGI-3, which the board does not list. Verified is the semi-private score at the stated reasoning effort, and USD per task is the leaderboard's cost under that protocol. The ARC-AGI-3 leaderboard publishes no cost, and – marks a preset it does not list. The ARC-AGI-2 efforts are the four in band on the board, and the text gives the stateless floor's first-exposure rate on the pool at each. In the rows column, H is a harness with its persistence on and H_0 is the same harness wiped before every instance. On ARC-AGI-1 every row runs all three orders. On ARC-AGI-2, prime-agent's rows H and H_0 run the workstreams and sequential orders at medium, high, xhigh and max effort. There the floors run the fresh reference at every effort, the workstreams order at medium and high and the random order at low, medium and high, and the method runs the workstreams order at medium and high. GPT-6 Luna is priced at 0.10 USD per million input tokens, 0.50 per million output tokens and 0.01 for a cached read.

Task ids. In continual learning, exposing the task id corresponds to the task-oracle setting. We expose it because recognising that today's grid is the same kind of problem as one from last month is a separate skill from remembering how to solve it, and mixing the two would make the measurement harder to read. An input grid alone also does not determine the task, as Figure 4 shows with pairs of training tasks whose inputs are drawn from overlapping distributions. The same grid is then a valid input to both tasks, and only the demonstrations or the id say which rule applies. A learner may key its memory on the id, as the floors and the harnesses of Section 4 do. A task-free variant would be a different benchmark, in which the learner must first buy enough demonstrations to identify the task, and we leave it to future work.

Observation horizon. As the generators produce fresh instances without end, a configuration's stream length is only the window over which we observe a learner. It places no limit on how much the learner may learn. Since every task's supply is the stream length over the number of tasks, a longer configuration is a different schedule and does not extend a shorter one. Within a run, however, a checkpoint is a prefix, and it never changes when the run is carried further. Every instance is charged, so the budget used against instances is the experience curve and the budget used at a horizon is its mean (Equation 1). A system that becomes competent after fewer instances then uses less of the budget at every checkpoint, which is the kind of efficiency that Chollet (2019) asked for, and the benchmark sets no cap on data. While a learner cannot choose how many instances arrive, it can choose how many demonstrations to buy, which the score charges, and how much of its experience to train on, which we count as compute and log without scoring. The ARC-AGI-3 configurations alone run the stream process without the supply rule, and Table 6 gives the collective configuration's checkpoints.

Task sets. A track's task set is the seed-0 draw of its small configuration, and the draws nest across a track's configurations. The medium set is the small set plus a seeded draw from the rest of the pool,

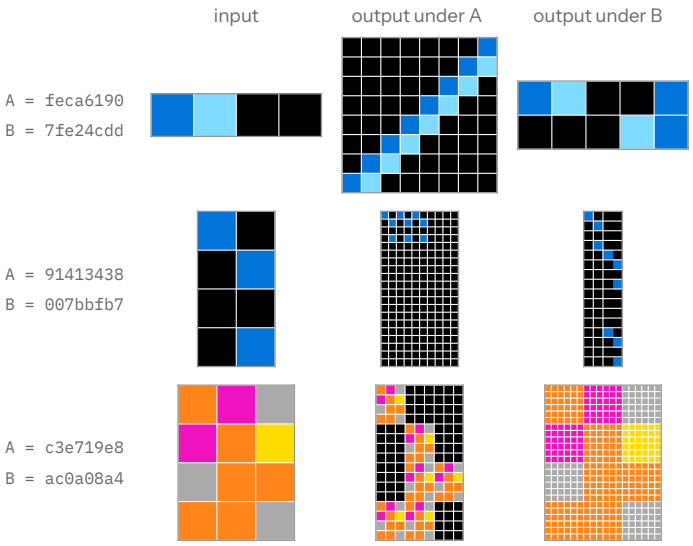

Figure 4: An input grid does not determine its task. Each row shows one input generated for training task A, with the output of task A's rule and the output of task B's rule on the same grid. Both are valid ARC tasks, and only the demonstrations or the id say which rule applies. Top: A (`feca6190`) draws diagonal lines from each coloured cell on a larger black square, and B (`7fe24cdd`) lays out the input and its three rotations as four quadrants. Middle: A (`91413438`) tiles copies of the input, one per coloured cell, and B (`007bbfb7`) places the input's pattern in every tile whose cell is coloured. Bottom: A (`c3e719e8`) places the whole input in every tile whose cell has the most frequent colour. B (`ac0a08a4`) enlarges every cell into a block. The pairs are selected automatically: an input generated for A on which B's verifier runs and returns a different valid grid.

track	config	tasks	stream	epilogue	n_u	n_c	b	m	a
ARC-AGI-1	arc1-small	25	200	25	1	4	3	30	2
	arc1-medium	100	3,000	100	1	5	6	120	20
	arc1-full	400	12,000	400	1	6	6	300	20
	arc1-collective	25	200	25	100	4	3	30	2
ARC-AGI-2	arc2-small	25	200	25	1	4	3	30	2
ARC-AGI-3	arc3-small	6	48	4	1	2	3	8	6

Table 6: Named configurations. Each track is its `small` configuration in workstreams mode, in the individual regime, at seed 0. n_u is the number of users, and the stream and the epilogue lengths are per user. n_c, b, m and a are the workstreams parameters of Table 4, in instances, with $\sigma = 1$ throughout. We define `arc1-collective` here. It has 100 users, each on a stream of the small configuration's shape over its tasks, for 200 rounds and 800 instances per task over the team, so one round gives the system up to 800 graded attempts to learn from before the next. Its budget used is reported at 2,000, 6,000 and 20,000 instances, the ends of rounds 20, 60 and 200. Every configuration has an attempt cap of 8 and one demonstration request that returns the task's set, with $c_d = 1$. On ARC-AGI-3 the cap is five times the level's human baseline.

and the full set is every task, for the stream and the epilogue alike. Figures 5 to 7 draw the small sets, with one thumbnail per task labelled with its id and showing its first demonstration pair, or the first frame of its first level for a game. `scripts/task_sets.py` prints every set from the seed, the medium ones included, and the full configuration is all 400 ARC-AGI-1 training tasks, with all 400 evaluation tasks as its epilogue.

Buckets. The small configurations of the grid tracks serve every task eight times, and so the exposure index of Equation 4 runs from 0 to 7. Every figure shows each index separately, and a final point pools anything beyond 7 on configurations that go further, such as `arc3-small`. The gap buckets G_g are 1 to 10, 11 to 30, 31 to 100, 101 to 300 and over 300 instances.

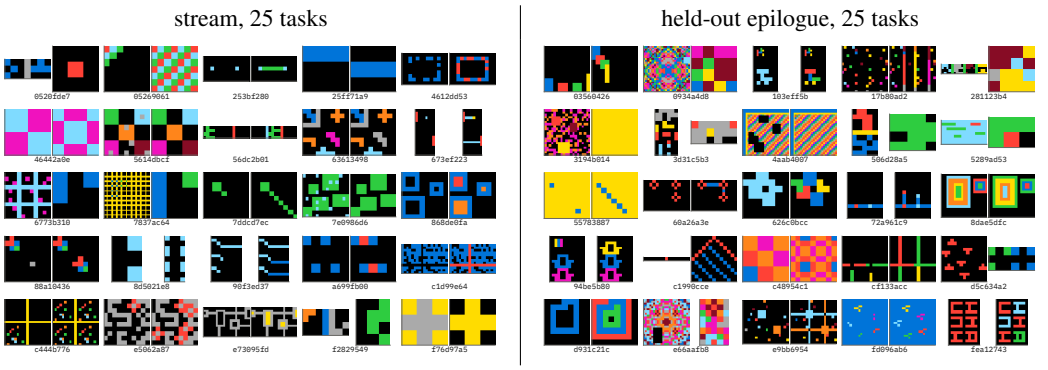

Figure 5: The ARC-AGI-1 track: the 25 stream tasks of `arc1-small` (left) and its 25 held-out epilogue tasks (right). Each thumbnail is the first demonstration pair, input beside output.

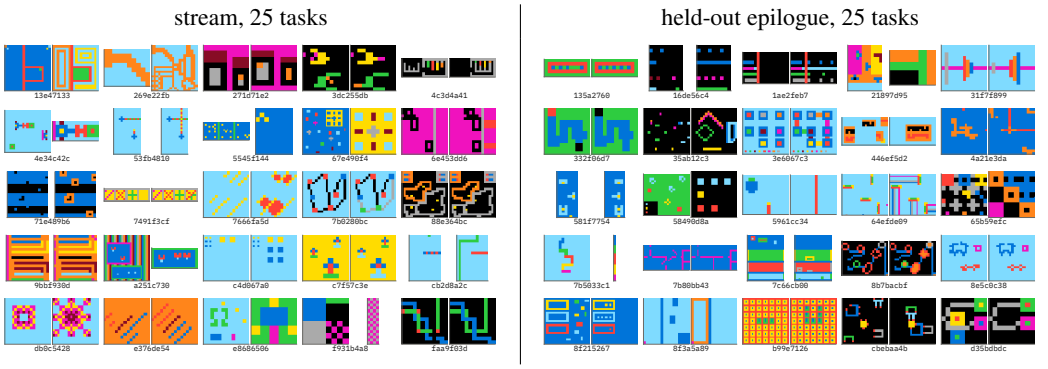

Figure 6: The ARC-AGI-2 track: the 25 stream tasks of `arc2-small` (left), served under augmentation, and its 25 held-out epilogue tasks (right), served unaugmented; all fifty are evaluation tasks new to the second edition.

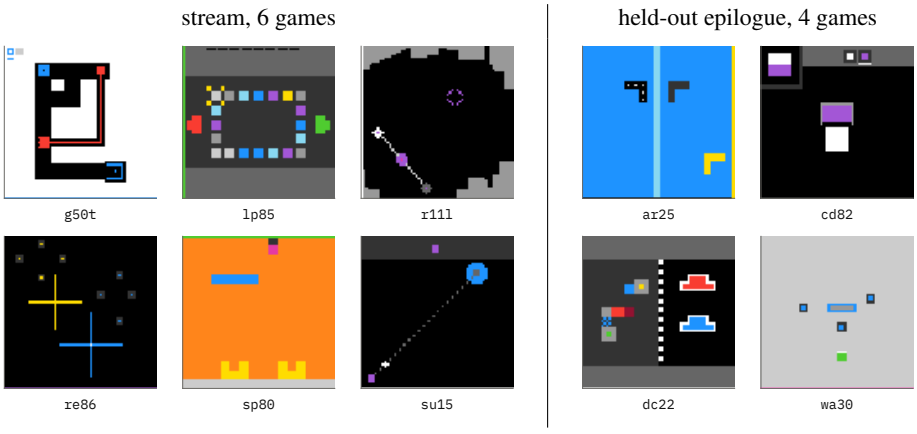

Figure 7: The ARC-AGI-3 track: the six stream games of `arc3-small` (left) and its four held-out epilogue games (right), each shown as the first frame of its first level.

Configurations. On the grid tracks, each task gets the stream length divided by the number of tasks in every mode, which is eight on the small configurations. Sequential mode serves them in one block per task, and the other two modes in the orders their processes produce. On the `arc1-small` workstreams schedule at seed 0, a learner has met 14 tasks by step 50 and all 25 by step 100. The gap before a re-exposure has a median of 5 instances, a 90th percentile of 49 and a maximum of 141.

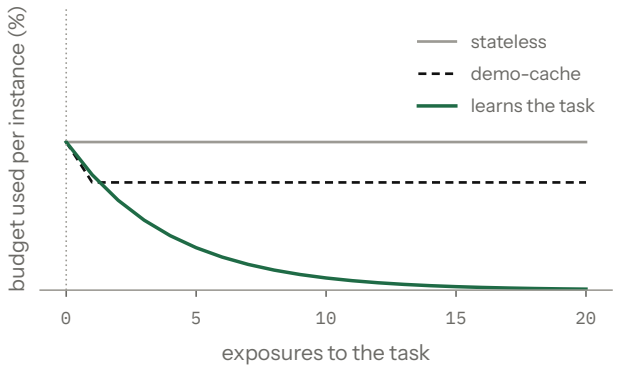

Figure 8: A schematic, with no data, of how the exposure axis separates a stateless solver, a demonstration cache and a system that has learned the task.

We also keep the regimes apart in every table, since one user's stream and a whole team's measure different things.

The exposure axis. ARC measures the cost of the first exposure, and Figure 8 sketches what the exposure axis adds.

Generating data at will. Chollet (2019) also wrote that an ideal benchmark should avoid tasks for which new data can be generated at will, because performance can then be bought with data. We generate data at will deliberately, for two reasons. In the stream, every demonstration and every attempt is charged, and so the measure is the feedback consumed, as he proposed, which leaves no room for the unpaid practice on the test distribution that concerned him. In the epilogue there are no generators at all, which rules out buying final skill with data where we measure transfer. The concern does survive in a weaker form, since Re-ARC covers the public training split, which any developer knows.

Compute. Each record carries whatever the learner reports, as a per-instance difference in cumulative counters, including tokens, wall-clock seconds, bytes of persistent state and dollars, and, for a learner that trains on its traces, the number of instances it has trained on. The baselines report these through the accounting layer so that they do not count for themselves. We log compute without scoring it, because we do not know the exchange rate between a weight update and a hundred thousand prompt tokens, and any fixed rate would decide the comparison in advance.

B SCHEDULE GENERATOR DETAILS

We generate every schedule deterministically from the configuration name, the mode and the seed, and a track is the seed-0 stream of its small configuration. Since we derive instance seeds by hashing the schedule seed, the phase and the slot index, slot t carries the same instance seed in every mode and always yields the same instance. We also ensure that the test input of a slot never appears among the demonstrations served for it.

The stream process. Algorithm 1 generates every stream, keeping up to n_c tasks active and, at each step, sampling one active task uniformly and emitting a slot for it. A task becomes active for a spell of β instances, and when the spell is spent it rests for ρ instances before it is ready to be re-activated, unless $\rho = \infty$, in which case it is finished. Never-seen tasks arrive every α instances and take priority over waiting tasks for the next free place. When no task is ready and no arrival is due, we wake the soonest-waking dormant task early so that the stream never idles, and we pull unseen tasks ahead of schedule only when no other task exists. On the grid tracks, every task has a supply of s instances, and a spell never exceeds what the task has left. A task whose supply is spent is finished and does not rest. On ARC-AGI-3 there is no supply, and $s = \infty$. Every spell, rest and

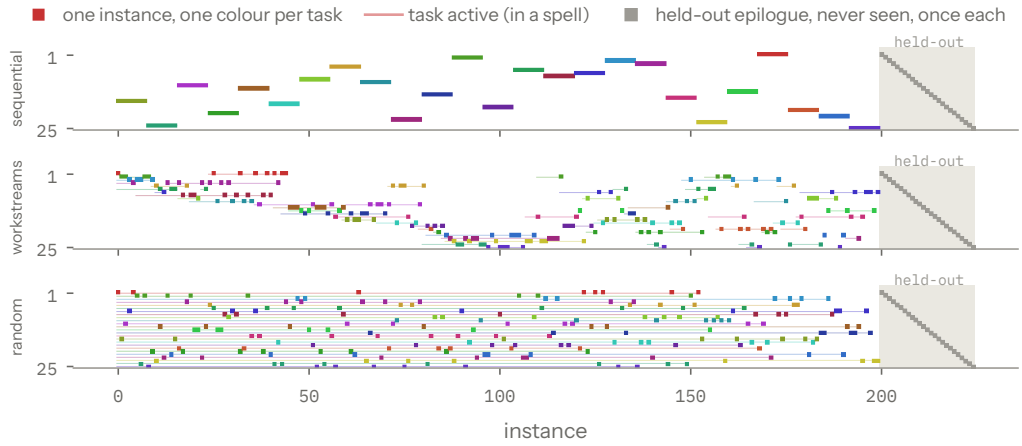

Figure 9: The three modes on `arc1-small`, seed 0. Each mark is one instance, with one row per task in the same order in all three panels. A thin line in the task's colour marks a spell in which the task is active. All three modes end with the same held-out epilogue (grey).

inter-arrival time is a fresh draw from the distribution that the mode sets for it, and all draws come from one generator seeded by the schedule seed and the mode name.

The three modes are settings of $(n_c, \beta, \rho, \alpha)$, as in Equation 3:

- **sequential**: $n_c = 1$, β is the constant L/n (the stream length over the number of tasks), $\rho = \infty$ and α is the constant 0, so the process reduces to the tasks in a shuffled order, each for L/n consecutive instances.
- **workstreams**: n_c from Table 6, $\beta = \max(1, \lfloor \mathrm{Exp}(b) + \frac{1}{2} \rfloor)$, $\rho = \max(1, \lfloor \mathrm{LogNormal}(\ln m, \sigma) + \frac{1}{2} \rfloor)$ and $\alpha \sim \mathrm{Exp}(a)$.
- **random**: $n_c = n$, $\beta = \infty$ and $\alpha = 0$ (ρ is never drawn), so every task is introduced at $t = 0$ and stays until its supply is spent, and each step is a uniform draw over the tasks with instances left.

Users and rounds. A configuration carries a number of users n_u. With $n_u = 1$, the schedule is one stream and every slot forms a round. With $n_u > 1$, every user has a separate schedule, drawn by the same process with a separate seed over the tasks and held-out tasks drawn at the configuration's seed. User 0, which keeps the configuration's seed, meets exactly the individual stream at that seed. Round r serves slot r of every user at once, and the learner acts on every user's instance with the state that the round began with. Each instance runs from its first action to its verdict at its user's pace, and the round ends when every user's instance has. We compute the exposure index and gap of a slot over rounds and over the whole team, so that two users who meet a task in the same round share both. Every record also carries the count for its user. At $n_u = 1$, both are the definitions of Section 3.3, unchanged. In this regime, the fresh reference wipes the learner's state before every round.

Epilogue and the fresh reference. The epilogue serves the held-out tasks in a shuffled order, once each, with nothing reset, and the fresh reference serves the same slots with the reset flag set on every one. Figure 9 and Table 4 show the three modes on the small configuration.

C Prompts

The texts below are everything a learner is shown, as they appear in the baselines repository. Appendix D describes how we run each harness, with the software versions and the model presets.

Algorithm 1 The stream. Parameters: concurrency n_c; spell β, rest ρ (possibly ∞) and inter-arrival time α, each a fresh draw at every use; supply s per task (possibly ∞); length L.

$\mathcal{Q} \leftarrow$ shuffled task list (never seen); $s_\tau \leftarrow s$ for every task τ; $\mathcal{A} \leftarrow \varnothing$ (active, with remaining spell); $\mathcal{D} \leftarrow \varnothing$ (dormant, with wake step); $\mathcal{W} \leftarrow \varnothing$ (ready); $t_{\text{next}} \leftarrow 0$
for $t = 0, \ldots, L-1$ **do**
  move every task in $\mathcal{D}$ with wake step $\leq t$ to $\mathcal{W}$
  while $|\mathcal{A}| < n_c$ **do**
    if $\mathcal{Q} \neq \varnothing$ and $t \geq t_{\text{next}}$ **then**
      $\tau \leftarrow \text{pop}(\mathcal{Q})$; $t_{\text{next}} \leftarrow t + \alpha$
    else if $\mathcal{W} \neq \varnothing$ **then**
      $\tau \leftarrow$ uniform draw from $\mathcal{W}$, removed
    else if $\mathcal{D} \neq \varnothing$ **then**
      $\tau \leftarrow$ the task in $\mathcal{D}$ with the earliest wake step, removed
    else if $\mathcal{Q} \neq \varnothing$ **then**
      $\tau \leftarrow \text{pop}(\mathcal{Q})$
    else
      break
    end if
    $\mathcal{A}[\tau] \leftarrow \min(\beta, s_\tau)$
  end while
  $\tau \leftarrow$ uniform draw from $\mathcal{A}$; emit slot $(\tau, \text{seed}(t))$; $\mathcal{A}[\tau] \leftarrow \mathcal{A}[\tau] - 1$; $s_\tau \leftarrow s_\tau - 1$
  if $\mathcal{A}[\tau] = 0$ **then**
    remove τ from $\mathcal{A}$; **if** $\rho < \infty$ and $s_\tau > 0$ **then** $\mathcal{D}[\tau] \leftarrow t + 1 + \rho$
  end if
end for

The texts below are reproduced from the baselines repository at commit `8dcca8e`, with lines longer than the page wrapped at spaces; placeholders in braces are filled at run time with the values named in Section 3.

Grid protocol: rules shown once per session (demo cost 1)

```
You are being evaluated on Continual-ARC, a lifelong stream of ARC-style grid
  puzzles.
Each puzzle instance has a task id and a test input grid. Tasks recur across the
stream with fresh inputs, so anything you learn about a task id is worth keeping.

On every message you must choose exactly one action and put it as a JSON object
on the last line of your reply:
  {"action": "submit", "grid": [[0, 1, 2], [3, 4, 5]]}   submit an output grid
  {"action": "request_demos"}                            ask for demonstration pairs
    for this task
Grids are lists of rows; cells are integers 0-9; dimensions are 1-30.

Scoring: every wrong submission costs 1 and every demo request costs 1;
an instance ends when a submission is correct or when the attempt cap is reached,
and reaching the cap is charged the full cap. Feedback on a submission is only
correct/incorrect. There is no way to abandon an instance and nothing is shared
between instances: each is scored on its own. Lower cost is better, so prefer
remembering a task's rule over re-requesting demos, and prefer requesting demos
over guessing blindly; a wrong submission always costs less than the cap.
```

Grid serialisation (a 3 by 3 example)

```
0 1 0
1 1 1
0 1 0
```

Grid protocol: re-prompt after an unparseable reply

```
I could not find a valid action in your reply ({reason}). Reply again with exactly
  one of
{{"action": "submit", "grid": [[...]]}}, or {{"action": "request_demos"}}
as a JSON object on the last line.
```

Grid protocol: closing message at the end of every instance (verdict filled in)

```
{verdict}

This instance is finished. No action is needed for it; reply with the single word:
  done.
```

Grid protocol: notice after a turn exceeds its budget

```
Your previous turn was stopped because it exceeded the time or model-call budget for
  a single action. Decide now from what you already know, without further
  exploration.
```

ARC-AGI-3 protocol: rules shown once per session

```
You are being evaluated on Continual-ARC-3, a lifelong stream of ARC-AGI-3 games.
Each game has levels; an instance is one level of one game. You see the current
frame (a 64x64 grid of colours 0-15, one hex digit per cell, row by row, top row
first) and the actions the game accepts. You must discover what the game wants by
acting. Games recur across the stream, so anything you learn about a game id is
worth keeping.

On every message choose what to do and put it as a JSON object on the last line
of your reply:
  {"action": "act", "id": 1}                       one simple action (ids 1-5 and 7)
  {"action": "act", "id": 6, "x": 12, "y": 40}     one click at column x, row y
    (0-63)
  {"action": "act", "ids": [1, 1, 2, 5]}           a sequence of up to 20 simple
    actions,
                                                   applied in order; it stops early
                                                     if the
                                                   level completes or the game ends
  {"actions": [{"id": 6, "x": 3, "y": 4}, {"id": 1}]}   a sequence with clicks in it
  {"action": "reset"}                              reload the current level
Actions 1-4 usually move (up, down, left, right), 5 usually selects or interacts,
6 clicks, 7 undoes where supported; the meaning is the game's to define. Every
action in a sequence is charged; you see the frame after the sequence.

Scoring: an instance costs the actions you spend (resets included) divided by the
human baseline for its level, up to a cap of several times that baseline (the
observation states your cap). Completing the level ends the instance at that
cost; reaching the cap ends it at the maximum cost. There is no way to abandon an
instance and nothing is shared between instances: each is scored on its own, so
keep exploring until the level completes or the cap ends it. After a game over
you must reset to continue.
```

ARC-AGI-3 protocol: re-prompt after an unparseable reply

```
I could not find a valid action in your reply ({reason}). Reply again with exactly
  one of
{{"action": "act", "id": N}} (with "x" and "y" for id 6), {{"action": "act", "ids":
  [...]}}, or {{"action": "reset"}}
as a JSON object on the last line.
```

Prompting floors: system prompt of the stateless and demo-cache solvers

```
You solve ARC-style grid puzzles. A grid is a list of rows; cells are integers 0-9;
  each
dimension is 1-30. You are shown demonstration input/output pairs that share one
  hidden
transformation rule, then a test input. Work out the rule and produce the output
  grid for
the test input. Your reply must end with exactly one JSON object on its own line:
{"grid": [[0, 1, 2], [3, 4, 5]]}
Only the last JSON object in your reply is read; it must be the complete output
  grid.

Reply with the JSON grid line only, no explanation.
```

Method row M: system prompt of the program-library learner

```
You solve ARC-style grid puzzles by writing programs. A grid is a list of rows of
  integers
0-9. You are shown demonstration input/output pairs that share one hidden
  transformation
rule. Write a Python function `transform(grid)` that returns the output grid for any
  input
of this task. Use only the standard library. Reply with exactly one fenced ```python
  block
containing the complete function and nothing else that needs to run.
```

D HARNESS IMPLEMENTATION DETAILS

This appendix gives the details of how we run the three harnesses.

D.1 VERSIONS

component	version	identity	driven over	installed from
prime-agent	0.9.5	release `0.9.5-linux-x64-bc4b0ed7...` (no git hash in the release)	RPC on stdin/stdout	vendor installer
hermes-agent	0.20.2	git `0c3452bf75ce6aecc603954d44c8d470d2a20c15`	ACP, `hermes-acp`	checkout, editable install
openclaw	2026.9.5	git `ec9c1a1`	ACP, `openclaw acp`	npm package
Agent Client Protocol client	0.12.1	`agent-client-protocol` on PyPI		
bubblewrap	0.9.0	Ubuntu 24.04.4, kernel 6.8		
continual-arc (benchmark)	0.1.0	git `87966df` to `7257858`		
continual-arc-baselines (adapters)	0.1.0	git `5f36700` to `8dde7b2`		
Re-ARC		git `e5b7f1d06362a76f9d3b8c25154ff1fafca897ce`		
ARC-AGI-1		git `399030444e0ab0cc8b4e199870fb20b863846f34`		
ARC-AGI-2		git `f3283f727488ad98fe575ea6a5ac981e4a188e49`		
ARC-AGI-3 games		25 public games pinned by version in `arc3/games.py`		

Table 7: Software behind the harness rows, as installed on every runner. Our two repositories reach the runners by file copy, so we give each as the range of commits that its reported runs can have used. The range starts at the commit that added the closing message, which every reported harness run counts, or at the commit that added the even supply of instances, which every reported stream schedule shows. It ends at the last commit before the last launch. The prompt texts are the same across the range (Appendix C). opencode is not in the grid, because it can read skill and instruction files but has no mechanism that writes any, leaving it nothing to learn with.

We ran every cell on five Hetzner `cpx62` instances (16 shared vCPU, 30 GB) in one region, where a cell is one benchmark process plus the harness's process tree, and each family of cells had a runner to itself. All model calls go to OpenRouter, which chooses the serving provider for each request by price-weighted load balancing and then keeps a cached conversation on the provider that served it. We set no provider preference and no fallback. Every request carries the cell's attempt id as OpenRouter's `session_id` so that cells with identical openings are not treated as one session. The run also records which provider served each call.

D.2 A CONVERSATION PER INSTANCE

Each instance is a new piece of work. At the start of every instance, we open a fresh conversation with the harness and send the protocol rules again, and at the end of the instance, once the harness's end-of-work steps have run (below), we close the conversation. Nothing carries from one instance to the next except what the harness persisted itself, which is the global harness store, skills and workspace files for prime-agent, the memory and skill files for hermes-agent, and the memory notes, skills and dream diary for openclaw. This protocol follows the intended use of these systems, one session per job with anything worth keeping written to files, and under it a row measures only what the harness chose to keep.

We also ran the alternative, one conversation per cell for the whole stream, and found that it measures something else. The transcript then carried everything a harness knew about a recurring task, so its re-exposure gains were in-context learning that every compaction and restart wiped, and prime-agent's refinements lived in a session-local store that a restart discarded. Two harnesses also got worse as their context grew over the stream, which reflects compaction more than learning. A short run of the per-instance protocol then confirmed, from each harness's files, that a session opens with only our observation, that each review fires on its designed cadence and persists, and that the next session loads what was persisted.

D.3 HOW EACH HARNESS IS DRIVEN AND CONFIGURED

The closing message. Every instance ends with one more turn in its conversation, carrying the verdict on the final submission ("Your submission was CORRECT. Instance solved with 1 wrong attempt(s).", or the cap-reached line) followed by "This instance is finished. No action is needed for it; reply with the single word: done." We do not read the reply. Our first version of the protocol had no such turn. The verdict then reached the system at the top of its next message, which belonged

to the next instance and a new conversation, and by then every storage step had run. prime-agent's refine and the post-turn reviews of hermes-agent and openclaw stored trajectories without their outcome, and we added the closing message to fix this. It costs one short model call per instance, which we journal without scoring, and every record counts it as `system_closings`.

Common to all three. The observation of Section 3.1 is the user turn, and after parsing the harness's reply for an action (see readable replies below), we send the feedback as the next user turn (Appendix C has the texts). We run every harness as one long-lived process per cell, started lazily on the first action, with a fresh session per instance as above. The harness is pointed at a local reverse proxy as its model endpoint, with a placeholder key. The proxy holds the real credential and journals every request and reply, with tokens, cached tokens, charge, latency and provider. The proxy forwards the body unchanged, except that it asks the provider to report usage, adds the run's reasoning effort and session id where the harness set none, and answers the model-metadata probe with the declared context window. We alias task ids per run. A harness sees `task-<8 hex>`, derived from the real id with a salt drawn once per run, which stays stable within the run and means nothing outside it. Each harness runs in a bubblewrap sandbox in which the whole filesystem is read-only, and in which the benchmark source, its data, every virtual environment and the whole run tree of every cell on the runner are masked with empty tmpfs. Only the harness's private state directory, a private HOME and a private TMPDIR are writable, and its bridge script is readable. The sandbox also has a private PID namespace and a private network namespace, whose only route out is a UNIX-socket bridge to the proxy on loopback (openclaw may also listen on one loopback port for the gateway it runs for itself). The environment is an allowlist of `PATH`, locale, `TERM`, `TZ`, the proxy address and the placeholder key, and the harness inherits no other variable. A program written by the program library runs in a sandbox of the same kind.

Row H_0. Before every instance, we stop the harness and delete everything it wrote, from its home directory, memory and skill files to its sessions, workspace and temporary files. Only the adapter's provider configuration survives, and we then start the harness again. We also skip its end-of-instance storage step and idle gap, since nothing could survive them.

Resource counters. Every record carries two families of counters. We use the proxy's counters (`llm_*`, for requests, input, output and reasoning tokens, and cost) everywhere in this paper. Every call passes through the proxy, input is the whole prompt including cached tokens, and an instance's value is the change in the counter while the instance was open. We also record the harness's usage reports (`system_*`) as received, but these are not comparable across harnesses. ACP defines its usage field as the session total, openclaw reports none, and prime-agent's RPC usage is the count kept by its client, a few percent of the prompt tokens that the proxy sees. No figure or table uses them.

prime-agent. We start it as `prime-agent -mode rpc -provider arc-proxy -model <id> -thinking <level> -cwd <workspace> -session-dir <sessions> -offline -daemon-socket <private socket>`, with `-thinking` carrying the preset's reasoning effort. In the agent directory, we write a `models.json` that declares one provider (the proxy, OpenAI-completions API) with one model marked as reasoning, a context window of 262,144 tokens and a maximum reply of 65,536 tokens. We also write a `settings.json` that disables the web-search bundled skill and the Linear, Notion and Prime Intellect integration skills. Every other bundled skill (edit, compact, goal, refine, skill-creator) belongs to the harness's mechanics and stays. Its Python-backed skills are installed in the kernel virtual environment, of which each cell holds a writable copy so that skills the system writes can install too. prime-agent reads its AGENTS.md from the workspace, as shipped, and we open a new session over RPC at every instance. Its automatic refinement runs at its default, every 25 assistant turns, gated by the harness's review step. This refinement writes the session-local store, which is lost when the session ends, and so we also ask at the end of every instance for the cross-session refinement that prime-agent documents, `refine.run(global_=True)` in its kernel. This rewrites the global harness store (prompt notes, memories, skill descriptions, sub-agent specifications) that every later session loads. The request is a line of text in the conversation, as a user would type it, and we wait for the refinement to complete before closing the session.

hermes-agent. We start it as `hermes-acp` from the checkout's virtual environment, with `HERMES_HOME` set to a private directory that holds a `config.yaml` written by us. The config sets provider `openai` at the proxy's address, model discovery off, a context length of 262,144, a 65,536-token reply cap, approvals off (`HERMES_YOLO_MODE=1`) and title generation off. Its post-turn background review of memory and skill files stays on, as shipped, and fires after any turn once ten tool iterations have accumulated. The `.no-bundled-skills` marker stops hermes from seeding its bundled skill library, and so `HERMES_HOME` starts with empty skill and memory directories and no messaging channel. Its `MEMORY.md`, `USER.md` and skills index are injected into every new session's system prompt, and we open a new ACP session at every instance.

openclaw. We start it through a launcher script that runs `openclaw gateway run --auth token --bind loopback` on a private port, waits for it, and then runs `openclaw acp --session agent:main:acp-bridge --no-prefix-cwd` on the inherited stdio, and the bridge's exit ends the gateway. `OPENCLAW_STATE_DIR` is private and holds the config that we write, which sets up a local gateway with token auth and one agent with a private workspace, bootstrap skipped, heartbeat off and openclaw's sandbox off, since ours encloses it. The agent uses the `coding` tool profile, with filesystem access limited to the workspace and full exec on the gateway host. The config declares one provider (the proxy, OpenAI-completions API, 600-second request timeout, private network allowed) with one model at a 262,144 context window and a 65,536 reply cap. With `skills.allowBundled` naming nothing, no bundled skill loads, and `skills.workshop.autonomous.mode` is `auto`. We enable the memory plugin's `dreaming` and set its sweep (`frequency`) to hourly, from a nightly default, to match the time scale of the stream, and `OPENCLAW_SKIP_CHANNELS` keeps every messaging channel unloaded. The bridge runs on one fixed session key, and so the fresh conversation per instance is openclaw's in-place reset, `/reset`, which drops the transcript and keeps the workspace, memory files and skills. After every instance we idle for 40 seconds, since openclaw's experience review, the step that turns a session into a skill proposal, starts only after 30 seconds of quiet with no run active, which a stream of back-to-back prompts would never give it.

D.4 Budgets, re-prompts and restarts

rule	prime-agent	hermes-agent, openclaw
action budget	300 s or 15 model calls per turn	600 s per turn
follow-up after a cut turn	300 s or 6 calls, with the budget notice	600 s, with the budget notice
hard ceiling on one turn	1,800 s	the action budget
idle watchdog	600 s with no RPC event and no proxy traffic	300 s with no update and no proxy traffic
quiet probe	every 15 s of silence, ask the worker directly	
final-reply grace	20 s	
cancel grace		60 s for the turn to end after cancel
provider errors	prime-agent's own retries, then 5 re-sent prompts with 60 s doubling backoff	the harness's own retries
unusable reply	two re-prompts, then the action is a give-up	same
restarts	2 per action, files intact, then a give-up	same
end of instance	global refine, waited for	openclaw: 40 s idle
startup	300 s	300 s
declared context window, reply cap	262,144, 65,536	262,144, 65,536

Table 8: Adapter rules. A turn cut for budget does not fail the instance. The harness keeps what it said, and its next prompt carries the notice of Appendix C, so it must decide from what it has. A give-up is scored as a failed instance, charged the cap, and counted in the run's resources, and no learner may choose it. A restart opens a new session, which under the per-instance protocol costs the harness nothing it would have kept.

We set the action budgets ourselves, and the benchmark does not score time. The budgets exist to stop a turn that would otherwise never end, and the idle watchdog handles a harness that falls

silent. The declared window of 262,144 tokens is 2^{18}, but the models in the grid advertise larger windows, 1,050,000 tokens for GPT-6 Luna and 1,000,000 for Claude Opus 5.5. Under the per-instance protocol a conversation holds one instance, a few tens of thousands of tokens at most, and so the window is far from binding, affecting only what each harness tells its model about itself.

Readable replies. Our first parser was strict, and in early runs it refused replies that held a well-formed action in an unexpected shape, most often wrapped as a string inside the runtime's own message envelope. Since the benchmark is not a test of output formatting, we now count a reply as readable if it holds an action object anywhere, parsed leniently (stray or missing brackets, single quotes, trailing commas), with the action and the grid under any of the usual keys or inside a runtime's argument object or envelope. A bare grid or rows of digits also count as readable. A garbage grid under an explicit submit is a submission, and the environment charges it. Run on 31,202 prime-agent replies from early runs, the lenient parser reads every reply that a strict JSON parser reads in the same way and leaves two unreadable.

D.5 What the endpoint says, and what each harness does with it

Since the model endpoint is the proxy, everything a harness learns about the model comes from the provider entry that we write for it, plus whatever it asks the endpoint. hermes-agent trusts the context length it probes from the endpoint over its configuration, and we therefore have the proxy answer the probe with the declared window. openclaw defeats the provider's prompt cache through the way it builds each request, appending a short volatile *internal context* message (active exec sessions, subagents, media tasks) as a second consecutive user message at the end of every request. On OpenAI's automatic prefix cache, only the shared tools-and-system prefix of about 19k tokens is then ever read from cache, and the rest is written afresh on every call. We confirmed this by replaying captured requests, finding that with the trailing message removed, the next request read 50k of 54k tokens from cache, against 19k as sent. We leave the harness as it comes and report the cost, noting that under the per-instance protocol, the context rewritten on each call is only one instance long.

D.6 Model presets

A preset names one model at one reasoning effort, the pair that the ARC Prize leaderboard verifies. We pass it to the harness as the model id and to the proxy as a default reasoning effort on every request, applied only where the harness did not set one. The grid presets are GPT-6 Luna at low, medium, high, xhigh and max effort and Claude Opus 5.5 at low effort (Table 5). The floors and the program library call the model at the preset's reasoning effort with a completion limit of 128,000 tokens, the model's maximum. We first set it at 12,000, but at high effort and above the reasoning alone used all of it, and replies were cut off before the grid.

E What the harnesses persist, and what of it carries

Every reading in this appendix comes from the ARC-AGI-1 cells of Appendix G, which run with the closing message of Appendix D so that the verdict of an instance reaches the harness before its storage step. We examined prime-agent in all of its ARC-AGI-1 cells, hermes-agent through its session database, and openclaw through its session log, which keeps the last 119 to 138 instances of each cell.

Stores and retrieval. For each row, Table 9 lists the store that its design writes between instances and the path by which the store reaches the model when a task returns. These are properties of the designs, which we took from their code and configuration (Appendix D), and they do not depend on any run. The three harnesses are built for lessons that transfer across unrelated work, and none of them ties its store to the task id, as the method row does. The rules of the stream tell each harness that tasks recur and that a remembered rule is cheaper than a demonstration request, and we examine below what each does with this.

Harnesses without persistence. The stateless floor makes one model call per attempt and asks for the whole grid in that call. A harness, in contrast, has tools that the floor lacks, code execution

system	store written between instances	written by	how it reaches the model on a re-exposure
program library (M)	one Python program per task, kept only if it reproduces the demonstrations, plus the demonstrations bought	the learner, after every instance	run first; a model call only when it fails
prime-agent	memories, prompt notes, skills, sub-agents	a refinement step after every instance	a digest of the store, capped in size, injected at session start; a search tool
hermes-agent	skill files and a user memory	a background review after a turn with ten or more tool iterations	the agent reads its skills at session start or on demand
openclaw	a consolidated memory, skills, workspace files	a scheduled consolidation over transcripts and a skill workshop	skills loaded at session start; memory search over the consolidated memory and recent transcripts

Table 9: What each row is built to persist between instances and how it reaches the model when a task returns, read from the designs. Table 10 gives what the stores held and delivered on the cells.

above all. It can parse the demonstrations into arrays, compute the changed cells and write the candidate transformation as a program, which it then runs on the test input after checking it against the demonstration inputs. Building the grid with a program removes the transcription errors that a large grid invites. A check against the demonstrations also catches a wrong rule before it costs an attempt. This advantage lies within the instance, and so it appears at the first exposure, does not grow with exposure, and survives the wipe of row H_0. Paired with the stateless floor at first exposure and pooled over the three orders, the stateless harnesses save between 6 and 8 points of the cap at medium effort. At low effort they save between -2 and $+5$, within noise, because the low-effort model mostly answers by reading the grid and seldom runs code. hermes-agent, for instance, ran code 1.3 times per instance at low effort and 4.4 times at medium.

The value of persistence. Whether a store can answer a task depends on what kind of store it is. An executable artefact per task, tested before it is stored, answers the task at zero model cost and cannot be misapplied, and the method row keeps such an artefact. A rule written in words answers the task only through a model that trusts and applies it. If the rule ends by telling the agent to confirm it against the demonstrations, it also costs the very demonstration request it was meant to save. A lesson about method, such as how to analyse demonstration pairs or when to stop guessing, improves every instance a little, without favouring re-exposures. Finally, a retrieval path that shows a recent window of a capped digest, or that depends on a search the agent must think to make, may not deliver even a stored rule when the task returns. From the designs alone, we would expect only a store that holds a tested, executable rule per task to bend the exposure curve below the demo-cache reference.

prime-agent. The refinement ran after every one of the 224 instances of the low-effort workstreams cell. It left one memory per task, 49 entries for the 50 tasks that the stream and its epilogue served, each about 500 characters of prose keyed on the task id. Of these, 30 state a rule and 19 record that the task is unresolved and which guesses to avoid. At medium effort, every entry states a rule. At the start of every session, prime-agent injects a digest of fixed size into the model's context, holding the prompt notes, six memory entries as one-line previews followed by a count of the rest ("+42 more"), and the last ten refinement summaries. The entry for the task at hand appears in the digest only if its position in the store puts it there. In sequential mode the previous refinement concerns the same task, and the digest names it on 85 to 96 percent of re-exposures. In workstreams mode it names it on 74 percent of re-exposures in the first half of the run and on 47 percent in the second, as the store outgrows the window (67 and 54 percent at medium). The model can also reach the store from the kernel, through an overview call with a cap on entries, a search by task id, and the state file. When its entry was not in the digest, the low-effort model went to the store on 24 percent of re-exposures and never retrieved the entry for the task. The medium-effort model went on 97 percent and retrieved the entry on a quarter of them, calling the capped overview 55 times for every 2 searches by id. On the low-effort workstreams cell, budget used was 58 percent at first exposure,

43 percent on a re-exposure whose entry was in the digest, and 56 percent on one whose entry was not. At medium effort the figures were 32, 24 and 27 percent. A seen entry saves about 15 points at low effort and a few at medium, and an unseen one saves none. Paired against its H_0 row on the re-exposure slots of all three orders, persistence saves 4.1 points at low effort (95 percent interval -0.1 to $+8.6$ by bootstrap over tasks, 525 slots) and 5.7 at medium ($+2.0$ to $+9.2$). These are close to the 3.9 and 5.6 of Section 5, which include the first exposures. We attribute this limit, at least in part, to the representation. A seen entry describes the rule in words, with one instance's specifics in it, and so the model must re-derive and re-execute the transformation. It then saves about a quarter of the budget it used at first exposure, while the method row's program, which runs first, saves most of it.

hermes-agent. Its store holds general method and almost nothing about individual tasks. After the low-effort workstreams stream, its memory file sits at its cap of about 2,200 characters, holding six restatements of the output protocol and one line that names a task. Its user profile holds four more restatements. Its one skill, which its background review rewrote after nearly every instance, has 33 reference files of general advice on reading demonstrations and recovering from a wrong attempt, none of which names a task. The skill reaches the model reliably, having been loaded on 174 of 175 re-exposures. The past sessions are searchable, and at medium effort the user profile tells the agent to search them when a task id recurs. At low effort the agent never did so, and at medium effort it did on 36 of 175 workstreams re-exposures, always by task id, and the earlier transcript came back every time. These retrievals did not change the cost, since budget used was 24 percent on those re-exposures and 25 percent on the rest, the medium-effort model solving most tasks at first sight. Its persistence instead changes how it attempts a task, with 1.35 wrong attempts per instance at medium effort against 1.79 for its stateless row, and 4.11 against 3.67 at low effort. On the workstreams stream alone, persistence saves it $+6.1 \pm 1.4$ points at medium effort and -4.8 ± 2.2 at low (one standard error, as in Table 14). Pooled over the three orders, Section 5 gives 3.7 and -2.1. We suspect that advice on method helps only a model that is able to act on it, although we have not tested this directly.

openclaw. Dreaming, openclaw's scheduled consolidation, ran every hour of the stream and never promoted a memory. Neither cell has a `MEMORY.md`, each dreaming report ends with no candidate truths, and dreaming wrote only an 11 KB dream diary in prose. The skill workshop wrote one general skill, whose first step is to search the retained history for the task id. At low effort the agent followed it, searching on all 94 re-exposures in its log on the workstreams stream, half of them by task id. Of the 86 re-exposures of a task it had solved before, the earlier correct answer came back once, and in the random and sequential orders it came back on none of 170. Because old session transcripts are pruned, a search for a task finds the dream diary and the current session but not the earlier solution. The searching is also costly: the persistent row read 442 thousand tokens per first exposure, against 196 thousand for its stateless row. At medium effort the agent searched on 4 of 153 re-exposures. On the workstreams stream alone, persistence saves it $+5.4 \pm 2.8$ points at low effort and -3.0 ± 1.6 at medium. Section 5 pools the three orders and gives 1.9 and 0.3. We do not know what produces the low-effort saving on this stream, but it cannot be the recall of an earlier answer, since almost none came back.

Demonstrations bought again. Although each harness stores something after every instance, none of them keeps the demonstrations it has already paid for, as the demo-cache floor does, and so each pays for them again. On the workstreams stream, where 175 of the 200 instances are re-exposures, hermes-agent and openclaw request the demonstrations 0.90 to 1.00 times per instance, and prime-agent 0.88 times at low effort and 0.67 at medium. At one ninth of the cap per request, the extra requests cost each harness between 6 and 10 points of budget on this stream, more than its whole distance from the demo-cache floor in every cell.

F Related work: extended table

In Table 11, we place the benchmarks and methods closest to this one, out of those we screened, against the properties that this benchmark combines. The survey notes, with more detail on each work, are in the supplementary material. Other benchmarks cover aspects that ours does not, such

harness	store after the low-effort workstreams stream	what reached the model on a re-exposure	saved vs own H_0	
			low	medium
prime-agent	one prose memory per task (49), six prompt notes	its task's entry, in the fixed digest on 74 to 47 percent of re-exposures	$+3.6 \pm 3.6$	$+4.7 \pm 2.4$
hermes-agent	memory at its cap, six protocol notes; one general skill with 33 references	the general skill, on 174 of 175; an earlier transcript, on 36 of 175 at medium, worth nothing	-4.8 ± 2.2	$+6.1 \pm 1.4$
openclaw	no memory file; an 11 KB dream diary; one general skill	an earlier correct answer, on 1 of 86 re-exposures of a solved task	$+5.4 \pm 2.8$	-3.0 ± 1.6

Table 10: What each harness stored, what of it reached the model when a task returned, and what persistence saved against the same harness wiped before every instance, in points of the cap on the workstreams stream alone, $\pm$ one standard error over tasks (Table 14). Section 5 gives the savings pooled over the three orders, with 95 percent intervals.

as drift in real domains (Lomonaco & Maltoni, 2017; Lin et al., 2022), natural-language feedback (Ai et al., 2025) and side effects on general ability (Wang et al., 2023).

G ALL RESULTS

We give every table and figure for each track here, each view once, with the line styles and bands of Figure 2. Where a track's models are the efforts of one family, the panels run from the least to the most effort.

G.1 ARC-AGI-1

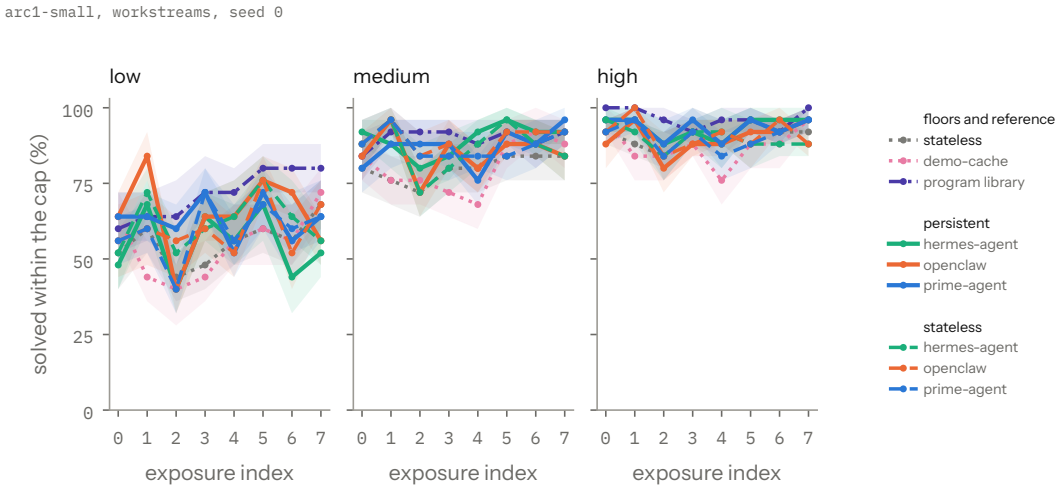

Figure 10: ARC-AGI-1, share of instances solved within the cap by exposure index, one panel per effort, on the track's workstreams stream, the solve-rate twin of Figure 2, with the same bands over task resamples. The leftmost point of each line is the first-exposure solve rate, the check of Section 4 that a preset has room to learn. The two views differ where a system learns to solve but keeps paying for attempts and demonstrations on the way, and where it fails outright.

G.2 ARC-AGI-2

On this track only prime-agent runs both variants. When we pair them on the same slots and pool over the workstreams and sequential orders, its persistent row saves 7.0 [2.2, 12.0] points of the cap

benchmark	domain	instances	recurrence	growth	feedback	learners	score
Supervised continual learning							
Permuted/Split MNIST (Goodfellow et al., 2013; Zenke et al., 2017)	image classes	fixed	none	no	labels	W	accuracy
GEM (Lopez-Paz & Ranzato, 2017)	image classes	fixed	none	no	labels	W	ACC, BWT, FWT
A-GEM (Chaudhry et al., 2019)	image classes	fixed	none	no	labels	W	accuracy, forgetting, LCA
CTrL (Veniat et al., 2021)	image streams	fixed	one revisit at end	no	labels	W	accuracy, transfer
OSAKA (Caccia et al., 2020)	few-shot classes	fixed pools	hidden revisits	new tasks	labels after	W	cumulative online acc.
CIR (Hemati et al., 2023; 2025)	image classes	generated streams	class repetition	classes	labels	W	average accuracy
CORe50 (Lomonaco & Maltoni, 2017)	objects (video)	fixed	sessions	classes	labels	W	accuracy curve
CLEAR (Lin et al., 2022)	temporal images	fixed	none	no	labels	W	in/next-domain acc.
NEVIS'22 (Bornschein et al., 2022)	106 vision tasks	fixed	none	yes (chronological)	labels	any	accuracy vs compute
Loss of plasticity (Dohare et al., 2024)	thousands of tasks	fixed	none	unbounded	labels, reward	W	online accuracy
Continual RL and generated environments							
Continual World (Wołczyk et al., 2021)	manipulation	random init	second pass (CW20)	no	dense reward	W	perf., forgetting, transfer
CORA (Powers et al., 2022)	Atari, Procgen, MiniHack	levels	fixed cycles	no	reward	W	continual eval, forgetting
COOM (Tomilin et al., 2023)	3D ViZDoom	fixed	none	no	reward	W	perf., forgetting, transfer
MEAL (Tomilin et al., 2026)	Overcooked	generated layouts	none	no	reward	W	perf., forgetting, plasticity
L2M metrics (New et al., 2022)	any	scenario	learning blocks	no	reward	any	PM, FT, BT, RP, SE
LIBERO (Liu et al., 2023)	manipulation	fixed	none	no	50 demos, free	W	FWT, NBT, AUC
Jelly Bean World (Platanios et al., 2020)	never-ending grid	generated	no tasks	no	reward	W	reward over time
Procgen (Cobbe et al., 2020)	16 games	generated levels	not continual	no	reward	W	sample eff., generalisation
XLand (Open Ended Learning Team et al., 2021; Adaptive Agent Team et al., 2023)	3D task space	generated	trials	no	reward	W	held-out task success
BabyAI (Chevalier-Boisvert et al., 2019)	grounded language	generated	curriculum	no	reward, demos	W	demos per level
Reasoning Gym (Stojanovski et al., 2025)	100+ reasoning tasks	generated	train/test	no	verifier	W	accuracy
Enigmata (Chen et al., 2025)	36 puzzles	generated	train/test	no	verifier	W	accuracy
RLVE (Zeng et al., 2026)	400 verifiable envs	generated, adaptive	training curriculum	no	verifier	W	accuracy
Language agents							
StreamBench (Wu et al., 2024)	SQL, code, tools, QA	fixed, one pass	none	no	binary	P, M	final accuracy
LifelongAgentBench (Zheng et al., 2025)	DB, OS, KG	fixed	skills recur	no	success	P, M	success rate
SWE-Bench-CL (Joshi et al., 2025)	GitHub issues	fixed	none	no	tests	M	ACC, forgetting, BWT
MemoryBench (Ai et al., 2025)	QA, legal, academic	fixed	none	no	simulated user	M	normalised effectiveness
CL-Bench (Asawa et al., 2026)	six real domains	fixed	poker stages	no	reward, tests	P, M	gain over stateless
SkillLearnBench (Zhong et al., 2026)	real skill tasks	fixed, 2–6 each	none	no	verifier, self	P, M	skill quality, accuracy
AgentStream (Yan et al., 2026)	six agent benchmarks	fixed, one pass	none	no	execution, self-evaluation	P, M	task accuracy
AgentCL (Shu et al., 2026)	code, research, BabyAI	fixed	one second pass	no	none	M	plasticity, stability gains
PATH-Bench (Yang et al., 2026)	code, tools	fixed	probe every 6–12	no	none	M	AP, FWT, BWT
Evo-Memory (Wei et al., 2025)	QA, math, embodied	fixed	similar tasks	no	correctness	M	accuracy, ordering
ContinualSkillBench (Guan et al., 2026)	five domains	fixed	none	no	evaluator	P, M	reward delta
SEA-Eval (Jiang et al., 2026)	web, OS	variable slots	repeats, variants	no	outcome	P, M	success, efficiency
EvoArena (Xu et al., 2026)	terminal, SWE	fixed chains	evolved versions	no	execution	M	chain accuracy
StuLife (Cai et al., 2025)	simulated college	fixed	exams revisit	no	scores, text	P, M	GPA, retention
TRACE (Wang et al., 2023)	8 NLP tasks	fixed	none	no	labels	W	OP, BWT, side effects
CITB (Zhang et al., 2023)	dialogue tasks	fixed	none	no	labels	W	ROUGE, FWT, BWT
ARC							
ARC-AGI-1 (Chollet, 2019)	grids	fixed	none	no	none (2 tries)	any	accuracy
ARC-AGI-2 (Chollet et al., 2025)	grids	fixed	none	no	none (2 tries)	any	accuracy, cost/task
ARCLE (Lee et al., 2024)	grid editing MDP	fixed	none	no	step reward	W	per-task success
ArcMemo (Ho et al., 2025)	ARC-AGI-1 (100)	original	none	no	train pairs	M	accuracy
SOAR / PoE (Pourcel et al., 2025; Franzen et al., 2025)	ARC-AGI-1 eval	original	none (one batch)	no	train pairs	W	accuracy
ARC-AGI-3 (ARC Prize Foundation, 2026)	grid games	fixed games	none scored	no	observations	any	RHAE
Continual-ARC	ARC-AGI-1 to 3	generated, augmented, levels	bursts, variable gaps	yes	binary, paid demos; actions (ARC-AGI-3)	P, M, W	budget used, solved

Table 11: Extended comparison. Instances: whether each exposure draws a fresh instance. Recurrence: whether and how tasks come back. Growth: whether the task set grows through the stream. Learners: which families the protocol admits, P prompting, M external memory, W weight updates; LCA is learning-curve area, RHAE is ARC-AGI-3's human-relative action efficiency.

			floors		hermes-agent		openclaw		prime-agent		program
GPT-6 Luna			stateless	demo-cache	stateless	persistent	stateless	persistent	stateless	persistent	library
effort	low	workstreams	58.3 ±5.6	49.6 ±6.2	51.9 ±5.1	56.8 ±5.3	55.1 ±5.5	49.6 ±5.3	52.1 ±5.0	48.6 ±6.0	**28.9 ±6.9**
		epilogue	61.3 ±7.9	55.6 ±8.0	69.8 ±7.7	52.4 ±8.0	63.1 ±7.7	68.4 ±6.9	57.8 ±7.9	62.7 ±7.8	**44.4 ±7.8**
		sequential	52.7 ±5.7	47.3 ±5.4	53.0 ±5.0	55.7 ±5.2	47.3 ±5.4	50.7 ±5.0	50.5 ±5.1	46.4 ±5.7	**28.6 ±6.5**
		epilogue	54.7 ±7.8	59.1 ±8.0	61.8 ±7.7	64.4 ±7.8	67.1 ±7.2	59.6 ±8.2	64.0 ±7.8	65.8 ±8.3	**43.6 ±7.5**
		random	56.0 ±6.1	41.7 ±6.0	51.1 ±5.6	50.0 ±5.3	52.3 ±5.8	48.4 ±5.7	53.2 ±5.0	49.2 ±5.3	**21.6 ±6.1**
		epilogue	55.6 ±7.8	58.7 ±8.0	65.3 ±7.7	59.6 ±7.6	64.9 ±6.9	61.3 ±7.7	66.2 ±7.1	54.7 ±8.1	**53.3 ±8.3**
		fresh	54.7 ±8.0	64.0 ±7.6	59.1 ±8.1	61.8 ±7.9	61.3 ±7.7	63.6 ±7.8	64.0 ±7.4	54.2 ±8.2	**48.4 ±7.9**
	medium	workstreams	36.0 ±3.8	27.6 ±4.8	31.1 ±3.8	25.0 ±3.4	29.4 ±3.1	32.4 ±3.6	28.4 ±3.2	23.7 ±4.2	**13.7 ±4.1**
		epilogue	33.8 ±6.6	30.7 ±5.6	32.4 ±5.7	32.9 ±6.3	**24.4 ±3.5**	32.9 ±4.9	28.9 ±5.8	26.2 ±5.1	24.9 ±5.9
		sequential	31.6 ±3.6	25.3 ±4.3	27.7 ±3.5	26.1 ±3.2	32.1 ±4.1	28.6 ±3.4	27.3 ±3.0	20.8 ±2.9	**10.4 ±3.1**
		epilogue	33.3 ±6.7	28.0 ±5.9	24.0 ±6.1	25.3 ±5.0	27.1 ±5.3	36.0 ±5.3	28.0 ±5.3	33.8 ±5.9	**22.2 ±5.1**
		random	32.8 ±3.7	20.8 ±3.5	29.5 ±3.5	26.2 ±3.6	31.4 ±4.0	31.1 ±3.7	30.7 ±3.6	25.2 ±3.5	**9.4 ±3.4**
		epilogue	36.9 ±6.0	30.2 ±5.0	27.1 ±5.1	30.2 ±5.7	28.0 ±5.5	30.7 ±5.5	27.1 ±5.8	30.7 ±6.1	**22.7 ±5.4**
		fresh	30.2 ±6.1	**28.4 ±5.3**	30.7 ±5.7	31.1 ±6.5	34.2 ±5.2	28.9 ±5.5	34.2 ±7.1	29.3 ±6.5	29.8 ±6.5
	high	workstreams	26.4 ±2.7	19.0 ±3.9	25.7 ±2.8	21.9 ±2.8	25.1 ±2.7	27.4 ±2.8	24.5 ±2.9	19.7 ±2.7	**7.4 ±2.7**
		epilogue	25.8 ±5.1	25.8 ±5.6	21.3 ±2.6	24.4 ±5.5	30.2 ±4.9	24.4 ±5.1	20.4 ±4.4	**17.3 ±2.4**	20.0 ±5.2
		sequential	24.3 ±2.7	15.1 ±2.7	22.8 ±2.5	19.9 ±2.4	22.3 ±2.2	24.6 ±2.8	21.9 ±2.7	14.3 ±1.6	**8.1 ±2.6**
		epilogue	21.3 ±4.8	22.7 ±4.2	24.9 ±4.5	24.0 ±4.3	21.3 ±3.9	26.2 ±4.8	23.1 ±4.2	**20.9 ±4.6**	21.8 ±5.1
		random	25.7 ±3.1	15.7 ±2.9	24.4 ±3.0	25.6 ±3.6	23.1 ±2.2	25.4 ±2.7	24.6 ±2.7	19.1 ±2.9	**5.3 ±2.2**
		epilogue	25.8 ±5.2	25.3 ±5.2	20.9 ±3.8	28.4 ±6.5	27.6 ±4.7	26.7 ±5.5	21.8 ±4.0	**19.6 ±2.4**	22.7 ±5.4
		fresh	20.4 ±4.2	22.2 ±4.9	23.6 ±5.0	22.2 ±4.9	21.3 ±2.1	26.7 ±4.2	27.6 ±5.9	**20.0 ±4.2**	22.2 ±5.1

Table 12: ARC-AGI-1, `arc1-small`: budget used (percent, lower is better), as Table 1, but with an epilogue after every order of the same 25 held-out tasks. Fresh is the reference for all three epilogues, and the tables that follow use this layout.

			floors		hermes-agent		openclaw		prime-agent		program
GPT-6 Luna			stateless	demo-cache	stateless	persistent	stateless	persistent	stateless	persistent	library
effort	low	workstreams	55.5 ±6.5	54.5 ±7.3	62.0 ±6.0	55.0 ±6.2	60.0 ±5.7	65.0 ±6.3	59.5 ±5.6	63.0 ±6.7	**71.5 ±7.7**
		epilogue	48.0 ±9.6	56.0 ±9.7	40.0 ±9.6	56.0 ±9.4	52.0 ±10.1	56.0 ±9.8	56.0 ±9.8	52.0 ±10.2	**72.0 ±9.5**
		sequential	60.0 ±6.3	57.0 ±6.5	60.5 ±5.8	58.5 ±6.0	69.5 ±6.0	64.5 ±5.8	63.5 ±5.5	65.0 ±6.1	**72.5 ±7.3**
		epilogue	60.0 ±9.8	52.0 ±9.9	56.0 ±10.2	44.0 ±9.4	48.0 ±9.9	48.0 ±9.6	44.0 ±9.5	44.0 ±9.8	**68.0 ±8.8**
		random	56.5 ±7.1	63.5 ±7.2	61.5 ±6.5	65.0 ±6.1	64.0 ±6.6	68.0 ±6.3	59.5 ±5.7	63.0 ±5.8	**79.5 ±6.6**
		epilogue	**60.0 ±9.8**	52.0 ±9.9	48.0 ±9.9	**60.0 ±10.1**	56.0 ±9.3	56.0 ±10.3	48.0 ±9.6	56.0 ±9.7	56.0 ±9.7
		fresh	56.0 ±9.7	52.0 ±9.9	52.0 ±9.6	56.0 ±10.1	52.0 ±10.1	48.0 ±9.9	52.0 ±9.4	56.0 ±10.0	**68.0 ±9.4**
	medium	workstreams	80.0 ±4.1	80.5 ±5.1	86.5 ±3.7	89.5 ±3.3	89.0 ±3.6	85.0 ±4.0	87.5 ±3.1	87.0 ±4.0	**90.0 ±4.3**
		epilogue	80.0 ±7.9	88.0 ±6.5	88.0 ±6.1	84.0 ±7.5	**100.0 ±0.0**	92.0 ±5.3	88.0 ±6.4	92.0 ±4.9	88.0 ±6.5
		sequential	85.5 ±3.8	82.0 ±4.4	89.0 ±3.4	88.5 ±3.3	87.5 ±3.8	89.0 ±3.5	90.0 ±3.0	93.0 ±2.8	**93.5 ±2.9**
		epilogue	80.0 ±7.9	88.0 ±6.9	88.0 ±6.9	**92.0 ±5.6**	**92.0 ±5.5**	88.0 ±6.4	**92.0 ±5.3**	84.0 ±6.8	**92.0 ±5.7**
		random	83.0 ±3.8	85.5 ±3.8	86.5 ±3.7	90.5 ±3.5	86.0 ±4.0	88.0 ±3.0	85.5 ±3.5	90.0 ±3.0	**94.0 ±3.5**
		epilogue	84.0 ±7.2	**96.0 ±4.2**	92.0 ±5.5	88.0 ±6.8	88.0 ±6.5	92.0 ±5.1	88.0 ±6.3	88.0 ±6.7	92.0 ±5.7
		fresh	88.0 ±6.7	92.0 ±5.4	88.0 ±6.3	84.0 ±7.4	**96.0 ±4.1**	92.0 ±5.5	84.0 ±7.6	84.0 ±7.4	88.0 ±6.5
	high	workstreams	90.5 ±2.6	87.5 ±3.5	90.0 ±2.9	93.5 ±3.0	91.0 ±2.7	90.0 ±2.7	90.5 ±3.0	93.5 ±2.3	**96.5 ±1.8**
		epilogue	96.0 ±4.2	92.0 ±5.7	**100.0 ±0.0**	92.0 ±5.5	92.0 ±5.5	92.0 ±5.5	96.0 ±4.0	**100.0 ±0.0**	92.0 ±5.7
		sequential	93.5 ±2.9	93.0 ±2.3	93.5 ±2.0	95.5 ±1.8	95.0 ±1.9	94.5 ±2.3	91.0 ±2.8	**97.0 ±1.0**	95.5 ±2.4
		epilogue	92.0 ±5.4	96.0 ±4.2	96.0 ±3.7	96.0 ±4.2	**100.0 ±0.0**	92.0 ±5.2	96.0 ±4.2	96.0 ±4.0	92.0 ±5.7
		random	90.0 ±2.5	92.0 ±2.8	91.5 ±2.7	88.5 ±3.2	93.0 ±2.2	91.0 ±2.5	91.0 ±2.3	93.0 ±2.5	**98.0 ±1.4**
		epilogue	92.0 ±5.4	92.0 ±5.7	96.0 ±3.7	84.0 ±7.4	96.0 ±4.1	88.0 ±6.5	96.0 ±4.2	**100.0 ±0.0**	92.0 ±5.7
		fresh	96.0 ±4.2	92.0 ±5.6	92.0 ±5.5	92.0 ±5.5	**100.0 ±0.0**	96.0 ±3.7	88.0 ±6.3	96.0 ±4.0	92.0 ±5.7

Table 13: ARC-AGI-1, `arc1-small`: the share of each measure's instances solved within the cap (percent, higher is better), in the layout of Table 12. A failed instance is charged the full cap, so the budget alone cannot tell whether a system solves more tasks or solves them more cheaply, and this table separates the two.

against its stateless row at medium effort, 10.8 [5.2, 16.7] at high and 9.6 [4.0, 15.8] at xhigh, with 95 percent intervals by bootstrap over tasks. Its stateless row at max effort had not finished at the time of submission, and its cells are marked –.

			floors		hermes-agent		openclaw		prime-agent		program
GPT-6 Luna			stateless	demo-cache	stateless	persistent	stateless	persistent	stateless	persistent	library
effort	low	workstreams	–	**+8.7** ±2.9	–	-4.8 ±2.2	–	+5.4 ±2.8	–	+3.6 ±3.6	–
		epilogue	–	+5.8 ±5.3	–	**+17.3** ±9.2	–	-5.3 ±6.5	–	-4.9 ±7.9	–
		sequential	–	**+5.4** ±2.3	–	-2.7 ±2.0	–	-3.4 ±2.2	–	+4.1 ±3.5	–
		epilogue	–	-4.4 ±3.9	–	-2.7 ±8.4	–	**+7.6** ±6.2	–	-1.8 ±8.2	–
		random	–	**+14.3** ±2.2	–	+1.1 ±2.3	–	+3.8 ±1.6	–	+4.0 ±3.0	–
		epilogue	–	-3.1 ±5.3	–	+5.8 ±7.9	–	+3.6 ±3.6	–	**+11.6** ±7.1	–
		fresh	–	-9.3 ±5.7	–	-2.7 ±6.2	–	-2.2 ±6.3	–	**+9.8** ±4.7	–
	medium	workstreams	–	**+8.4** ±3.2	–	+6.1 ±1.4	–	-3.0 ±1.6	–	+4.7 ±2.4	–
		epilogue	–	**+3.1** ±4.2	–	-0.4 ±4.8	–	-8.4 ±5.9	–	+2.7 ±6.1	–
		sequential	–	+6.3 ±2.1	–	+1.6 ±1.9	–	+3.6 ±1.5	–	**+6.6** ±2.0	–
		epilogue	–	**+5.3** ±4.8	–	-1.3 ±7.2	–	-8.9 ±4.4	–	-5.8 ±5.5	–
		random	–	**+12.0** ±2.0	–	+3.3 ±1.7	–	+0.3 ±1.5	–	+5.4 ±2.0	–
		epilogue	–	**+6.7** ±3.1	–	-3.1 ±3.4	–	-2.7 ±5.8	–	-3.6 ±6.7	–
		fresh	–	+1.8 ±3.4	–	-0.4 ±4.4	–	**+5.3** ±5.4	–	+4.9 ±7.4	–
	high	workstreams	–	**+7.4** ±2.7	–	+3.8 ±2.0	–	-2.3 ±1.4	–	+4.8 ±1.3	–
		epilogue	–	-0.0 ±4.8	–	-3.1 ±4.5	–	**+5.8** ±3.9	–	+3.1 ±4.8	–
		sequential	–	**+9.2** ±2.6	–	+2.8 ±1.6	–	-2.2 ±2.3	–	+7.6 ±2.3	–
		epilogue	–	-1.3 ±2.9	–	+0.9 ±3.6	–	-4.9 ±3.4	–	**+2.2** ±5.7	–
		random	–	**+10.0** ±2.2	–	-1.2 ±1.9	–	-2.3 ±2.1	–	+5.5 ±2.0	–
		epilogue	–	+0.4 ±3.4	–	-7.6 ±4.5	–	+0.9 ±7.1	–	**+2.2** ±3.9	–
		fresh	–	-1.8 ±1.6	–	+1.3 ±2.3	–	-5.3 ±3.0	–	**+7.6** ±5.9	–

Table 14: ARC-AGI-1, `arc1-small`: budget saved by memory, in percent of the cap (higher is better). Each entry is a system's budget subtracted from that of its stateless variant, on the same slots of the same measure. For a harness the stateless variant is row H_0, the same harness wiped before every instance, and for the demo-cache floor it is the stateless floor. Systems without a stateless variant are empty. On the fresh measure both variants are reset before every instance, so they are two runs of one protocol, and that row indicates the noise in every other row of the table.

			floors		hermes-agent		openclaw		prime-agent		program
GPT-6 Luna			stateless	demo-cache	stateless	persistent	stateless	persistent	stateless	persistent	library
effort	low	workstreams	–	+8.7 ±2.9	+6.3 ±3.1	+1.5 ±2.9	+3.2 ±2.4	+8.7 ±2.9	+6.2 ±3.9	+9.7 ±4.3	**+29.3** ±5.6
		epilogue	–	+5.8 ±5.3	-8.4 ±9.5	+8.9 ±6.0	-1.8 ±5.4	-7.1 ±7.9	+3.6 ±9.0	-1.3 ±6.2	**+16.9** ±7.4
		sequential	–	+5.4 ±2.3	-0.3 ±2.8	-3.0 ±3.5	+5.4 ±3.3	+2.0 ±3.0	+2.2 ±2.5	+6.3 ±4.6	**+24.1** ±4.4
		epilogue	–	-4.4 ±3.9	-7.1 ±7.9	-9.8 ±7.0	-12.4 ±5.3	-4.9 ±6.0	-9.3 ±4.9	-11.1 ±9.9	**+11.1** ±5.7
		random	–	+14.3 ±2.2	+4.9 ±2.5	+6.0 ±3.5	+3.7 ±2.2	+7.6 ±2.4	+2.8 ±3.1	+6.8 ±3.7	**+34.4** ±4.6
		epilogue	–	-3.1 ±5.3	-9.8 ±6.0	-4.0 ±5.9	-9.3 ±6.4	-5.8 ±6.3	-10.7 ±5.7	+0.9 ±7.3	**+2.2** ±6.2
		fresh	–	-9.3 ±5.7	-4.4 ±7.6	-7.1 ±7.8	-6.7 ±8.1	-8.9 ±6.1	-9.3 ±7.0	+0.4 ±6.1	**+6.2** ±6.2
	medium	workstreams	–	+8.4 ±3.2	+4.9 ±3.0	+11.0 ±2.8	+6.6 ±2.2	+3.6 ±2.1	+7.6 ±3.0	+12.3 ±3.9	**+22.3** ±3.8
		epilogue	–	+3.1 ±4.2	+1.3 ±6.8	+0.9 ±6.9	**+9.3** ±6.5	+0.9 ±4.7	+4.9 ±6.5	+7.6 ±8.6	+8.9 ±5.3
		sequential	–	+6.3 ±2.1	+3.9 ±2.7	+5.6 ±2.9	-0.5 ±3.2	+3.1 ±2.8	+4.3 ±2.7	+10.8 ±3.5	**+21.2** ±3.8
		epilogue	–	+5.3 ±4.8	+9.3 ±6.8	+8.0 ±4.8	+6.2 ±3.6	-2.7 ±5.7	+5.3 ±6.0	-0.4 ±5.4	**+11.1** ±5.2
		random	–	+12.0 ±2.0	+3.3 ±2.9	+6.6 ±3.2	+1.4 ±2.0	+1.7 ±1.9	+2.1 ±3.3	+7.6 ±3.0	**+23.4** ±4.2
		epilogue	–	+6.7 ±3.1	+9.8 ±4.5	+6.7 ±4.3	+8.9 ±4.5	+6.2 ±6.1	+9.8 ±4.7	+6.2 ±6.7	**+14.2** ±5.2
		fresh	–	**+1.8** ±3.4	-0.4 ±5.3	-0.9 ±4.5	-4.0 ±5.3	+1.3 ±4.4	-4.0 ±6.5	+0.9 ±5.4	+0.4 ±6.6
	high	workstreams	–	+7.4 ±2.7	+0.7 ±2.4	+4.6 ±2.3	+1.3 ±2.3	-1.0 ±1.9	+1.9 ±2.1	+6.7 ±2.1	**+19.0** ±3.1
		epilogue	–	-0.0 ±4.8	+4.4 ±4.7	+1.3 ±3.7	-4.4 ±5.2	+1.3 ±5.9	+5.3 ±2.5	**+8.4** ±5.0	+5.8 ±3.8
		sequential	–	+9.2 ±2.6	+1.6 ±3.2	+4.4 ±2.8	+2.0 ±3.3	-0.2 ±3.1	+2.4 ±3.3	+10.0 ±2.9	**+16.3** ±3.2
		epilogue	–	-1.3 ±2.9	-3.6 ±5.4	-2.7 ±4.4	-0.0 ±4.6	-4.9 ±6.1	-1.8 ±6.4	**+0.4** ±6.5	-0.4 ±5.1
		random	–	+10.0 ±2.2	+1.3 ±2.6	+0.1 ±3.3	+2.7 ±2.3	+0.3 ±2.0	+1.1 ±2.5	+6.6 ±3.3	**+20.4** ±3.4
		epilogue	–	+0.4 ±3.4	+4.9 ±5.1	-2.7 ±5.6	-1.8 ±6.4	-0.9 ±4.8	+4.0 ±5.1	**+6.2** ±5.9	+3.1 ±4.5
		fresh	–	-1.8 ±1.6	-3.1 ±3.8	-1.8 ±3.3	-0.9 ±3.8	-6.2 ±4.8	-7.1 ±5.0	**+0.4** ±5.6	-1.8 ±4.2

Table 15: ARC-AGI-1, `arc1-small`: budget saved against the stateless floor, in percent of the cap (higher is better). Each entry is the stateless floor's budget minus the system's, on the same slots of the same measure. It measures what a system adds over one model call per attempt, tools and memory together. For a harness, the persistent column minus the stateless column is the saving by memory of Table 14.

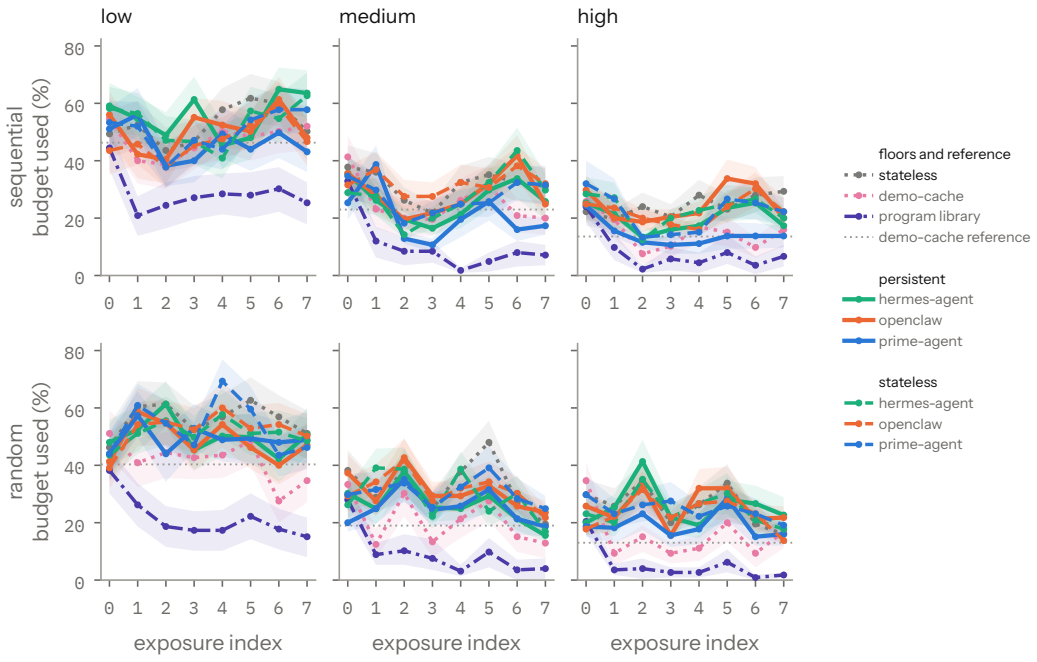

Figure 11: ARC-AGI-1, budget used by exposure index in the sequential order (top), each task's instances served together, and the random order (bottom), uniform over the stream, one column per effort, as Figure 2 for the workstreams order. Every number of both orders, on every measure, is in the tables above.

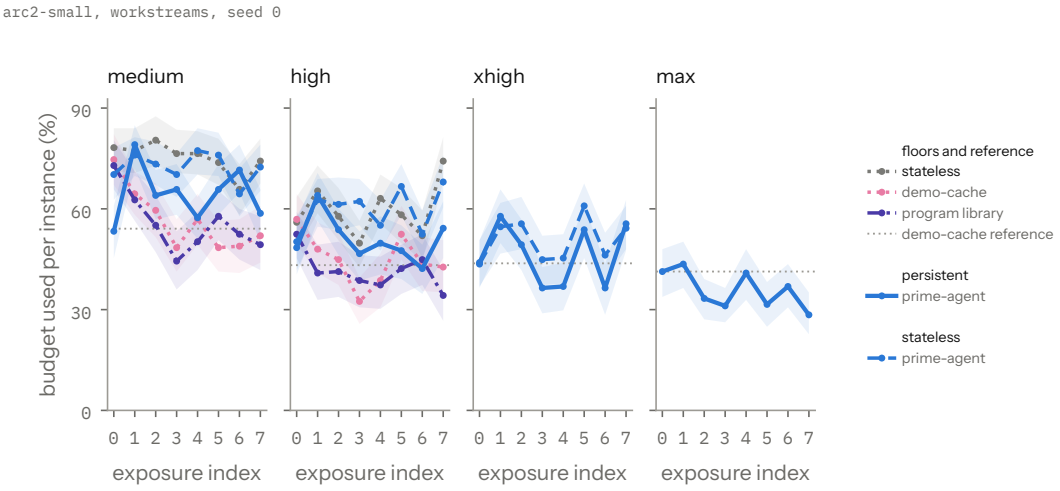

Figure 12: ARC-AGI-2, budget used by exposure index, one panel per effort of GPT-6 Luna, workstreams mode, as Figure 2.

effort		GPT-6 Luna	floors stateless	floors demo-cache	prime-agent stateless	prime-agent persistent	program library
effort	low	random	97.2 ±1.4	**81.4 ±2.7**	–	–	–
		epilogue	**96.4 ±3.5**	98.7 ±1.3	–	–	–
		fresh	**97.3 ±2.6**	**97.3 ±2.5**	–	–	–
	medium	workstreams	75.3 ±4.7	56.7 ±4.9	72.5 ±4.7	64.4 ±4.8	**55.6 ±5.7**
		epilogue	73.8 ±6.8	73.3 ±7.4	67.1 ±7.9	**64.9 ±7.5**	72.4 ±7.9
		sequential	–	–	75.2 ±4.9	**69.2 ±5.6**	–
		epilogue	–	–	**65.8 ±7.6**	77.3 ±6.6	–
		random	77.2 ±4.4	**60.8 ±5.0**	–	–	–
		epilogue	79.1 ±7.2	**76.4 ±7.0**	–	–	–
		fresh	75.1 ±6.7	82.7 ±6.0	–	**67.6 ±7.5**	76.9 ±7.0
	high	workstreams	59.6 ±5.1	44.9 ±4.9	59.8 ±5.1	50.8 ±5.4	**41.5 ±5.2**
		epilogue	59.1 ±7.4	59.6 ±7.6	62.7 ±8.3	**57.3 ±7.6**	**57.3 ±8.2**
		sequential	–	–	58.7 ±5.0	**46.1 ±4.4**	–
		epilogue	–	–	**55.6 ±8.1**	59.1 ±7.9	–
		random	62.4 ±5.4	**50.6 ±5.4**	–	–	–
		epilogue	**53.3 ±7.7**	57.3 ±7.5	–	–	–
		fresh	59.1 ±8.0	54.7 ±7.7	–	**52.0 ±7.6**	53.8 ±7.6
	xhigh	workstreams	–	–	50.7 ±4.7	**46.3 ±5.1**	–
		epilogue	–	–	**41.8 ±7.1**	47.6 ±8.4	–
		sequential	–	–	56.8 ±5.0	**42.1 ±5.6**	–
		epilogue	–	–	47.1 ±7.7	**43.1 ±7.3**	–
		fresh	47.1 ±8.0	47.6 ±7.2	–	**44.9 ±7.2**	–
	max	workstreams	–	–	–	35.9 ±3.9	–
		epilogue	–	–	–	37.8 ±7.8	–
		sequential	–	–	–	32.3 ±4.5	–
		epilogue	–	–	–	33.3 ±6.2	–
		fresh	**32.4 ±6.7**	43.6 ±8.0	–	42.2 ±8.2	–

Table 16: ARC-AGI-2 track, `arc2-small`: budget used, in the layout and with the markers of Table 12. Instances are augmentations of the public pairs of evaluation tasks new to the second edition (Appendix A). Table 5 gives the rows that run at each effort, and rows that no system runs are left out, here and in the other appendix tables.

G.3 ARC-AGI-3

On ARC-AGI-3, we run prime-agent on Claude Opus 5.5 at low effort, in rows H and H_0, on the workstreams stream of six games over 48 instances with a four-game epilogue (Table 18). Since the budget on this stream mixes level difficulty with retention (Appendix A), we look separately at the levels met twice. The persistent row used 32.1 percent of the budget on the stream and completed 41 of its 48 levels, with six of its seven failures on levels of one game, sp80. Three games wrapped, and on the 15 levels met a second time, the persistent row used 21 percent of the budget against 29 percent at the first meeting, using less on nine levels, the same on three and more on three. A level is the same problem at its second meeting, and so this is the track's most direct view of retention. Row H_0, on the same schedule, used 43.5 percent of the budget and completed 37 of its 48 levels. On the same 15 levels met a second time, it used 40 percent against 35 percent at the first meeting, using less on five levels, the same on one and more on nine. The persistent row's saving at the second meeting is therefore not an effect of the order of levels, although with one run of each row and 15 levels, this reading is tentative.

			floors		prime-agent		program
GPT-6 Luna			stateless	demo-cache	stateless	persistent	library
effort	low	random	6.0 ±2.7	**11.0 ±3.5**	–	–	–
		epilogue	**4.0 ±3.9**	**4.0 ±3.9**	–	–	–
		fresh	**4.0 ±3.9**	**4.0 ±3.7**	–	–	–
	medium	workstreams	39.5 ±6.8	46.5 ±6.4	43.5 ±6.2	**51.0 ±6.1**	48.0 ±7.0
		epilogue	40.0 ±9.7	36.0 ±9.8	48.0 ±10.5	**52.0 ±10.2**	40.0 ±10.2
		sequential	–	–	37.0 ±6.7	**44.0 ±7.3**	–
		epilogue	–	–	**56.0 ±10.3**	36.0 ±9.8	–
		random	35.0 ±6.3	**43.5 ±6.3**	–	–	–
		epilogue	32.0 ±9.6	**40.0 ±10.5**	–	–	–
		fresh	40.0 ±10.1	28.0 ±9.3	–	**48.0 ±9.8**	32.0 ±9.2
	high	workstreams	58.0 ±6.9	61.0 ±6.0	58.0 ±6.1	**65.5 ±6.3**	64.0 ±6.3
		epilogue	**68.0 ±9.8**	60.0 ±10.4	52.0 ±10.6	64.0 ±9.8	52.0 ±9.9
		sequential	–	–	58.5 ±6.2	**71.5 ±5.4**	–
		epilogue	–	–	**64.0 ±9.8**	60.0 ±10.0	–
		random	**56.0 ±6.8**	55.5 ±7.1	–	–	–
		epilogue	**68.0 ±9.6**	**68.0 ±9.6**	–	–	–
		fresh	60.0 ±10.2	64.0 ±10.1	–	**68.0 ±9.5**	**68.0 ±9.8**
	xhigh	workstreams	–	–	**69.0 ±5.8**	**69.0 ±6.0**	–
		epilogue	–	–	**80.0 ±8.1**	68.0 ±9.8	–
		sequential	–	–	61.0 ±5.6	**74.0 ±6.3**	–
		epilogue	–	–	68.0 ±9.7	**76.0 ±8.8**	–
		fresh	68.0 ±9.8	72.0 ±9.0	–	**76.0 ±9.0**	–
	max	workstreams	–	–	–	82.0 ±4.3	–
		epilogue	–	–	–	76.0 ±8.8	–
		sequential	–	–	–	82.0 ±5.1	–
		epilogue	–	–	–	88.0 ±6.8	–
		fresh	**88.0 ±6.8**	72.0 ±9.5	–	68.0 ±9.6	–

Table 17: ARC-AGI-2, `arc2-small`: the share of each measure's instances solved within the cap (percent, higher is better), in the layout of Table 16, read as Table 13.

			prime-agent	
Claude Opus 5.5			stateless	persistent
effort	low	workstreams	43.5 ±8.6	**32.1 ±9.2**
		epilogue	**10.6 ±0.6**	12.8 ±2.7
		fresh	–	11.5 ±1.8

			prime-agent	
Claude Opus 5.5			stateless	persistent
effort	low	workstreams	77.1 ±11.6	**85.4 ±11.6**
		epilogue	**100.0 ±0.0**	**100.0 ±0.0**
		fresh	–	100.0 ±0.0

Table 18: ARC-AGI-3 track, `arc3-small`: budget used (left, percent of the action cap, lower is better) and share of levels completed within the cap (right, higher is better), in the layout and with the markers of Table 12. The stream serves successive levels of each game, the epilogue and the fresh reference first levels only, so the stream row is not comparable with the two below it.