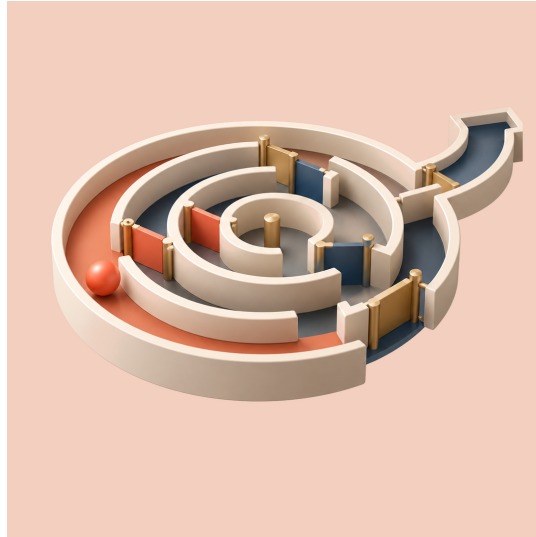




When the Curriculum Cheats: Designing RL Environments That Teach Transferable Skills

How adaptive difficulty, procedural task generation, and hidden shortcuts shape what AI agents actually learn.

Ahsan Raza | Deaimer Research

Abstract

Reinforcement learning (RL) agents do not learn “general ability” directly. They optimize behavior on experiences produced by a particular distribution of environments. A curriculum can make those experiences progressively harder and improve training reward while still teaching a narrow shortcut: recognizing a template, exploiting a simulator convention, or solving one familiar mechanism behind many cosmetic variations. This article examines that failure mode through research on procedural generation, generalization benchmarks, unsupervised environment design, prioritized level replay, adaptive curricula, and tool-use environments. Procgen and CoinRun established the importance of unseen levels; PAIRED, prioritized replay, and ACCEL demonstrated ways to select learning experiences; XLand explored open-ended training; RandomWorld introduced compositional synthetic tool environments; and RLVE reported results from adapting the difficulty distributions of 400 verifiable environments. These are distinct contributions with different assumptions and demonstrated scopes. Building on them, we propose a **mechanism-centered curriculum audit**: separate surface features from causal task structure, measure how much a generator truly varies, and evaluate on increasingly distant held-out mechanisms. A controlled experiment is outlined, not claimed to have been performed. The central argument is that the objective of curriculum design should be transferable competence under specified changes—not simply a rising success curve on an increasingly familiar training world.

1. The suspiciously successful student

Suppose a coding agent trains on thousands of generated repository-repair tasks. Its reward increases steadily. It solves new tasks sampled from the same generator and seems to need

fewer tool calls. The generator varies repository names, filenames, error strings, package versions, and the number of source files. On a dashboard, the tasks look diverse.

Then the agent encounters a repository built by a different team. The failure still concerns dependency resolution, but the root cause is no longer signaled by the same warning format; the fix must preserve an interface that the training tasks never modified. The agent repeatedly applies a familiar edit, sees no improvement, and runs out of budget. It learned a powerful regularity of its training distribution, not necessarily the underlying capability the task author intended.

This is an **illustrative scenario**, not a reported experiment. Its point is that the curriculum itself can create an illusion of progress. The learner is responding rationally to the evidence it receives. If superficial cues predict reward in the training world, exploiting those cues is an efficient policy. The mistake lies in concluding that the resulting policy has acquired transferable skill without evaluating it against a meaningfully different distribution.

Reinforcement learning offers a natural language for this problem. A policy is trained under some environment distribution, receives rewards for trajectories generated there, and is deployed or tested under another distribution. Even when those distributions share task descriptions, they may differ in which strategies work. Research on RL generalization has been explicit about this challenge for years. CoinRun and later Procgen used procedural levels to separate training experience from previously unseen instances [1][2]. Packer and colleagues developed controlled assessments of in-distribution and out-of-distribution generalization and found that specialized generalization techniques did not always outperform straightforward baselines in their tested settings [3].

For language agents, the concern becomes sharper. Environments include codebases, tool protocols, simulated people, databases, and complex sequential tasks. The generator can vary these components independently, but not every change introduces a new reasoning problem. A change in error wording may be cosmetic; a change in transaction semantics can require a new strategy. Curriculum design must distinguish the two.

2. The curriculum is a distribution, not a staircase

The familiar metaphor of a curriculum is a staircase: solve easy tasks first, then move to harder ones. A more useful formalization is a **time-varying distribution of environment instances**. Let e denote a sampled environment and $D_t(e)$ the distribution used at training step t . The policy π_t interacts with environments from D_t ; observed returns, failures, or auxiliary scores may then update both the policy and the future sampling distribution.

Two optimizations are happening at once. The agent optimizes its behavior within the sampled worlds. The curriculum mechanism decides which worlds the agent is likely to encounter. When the curriculum adapts to recent performance, the training data is **endogenous**: the current learner helps determine its future evidence.

That can be beneficial. Tasks that are already solved reliably may contribute little new information; tasks that no policy can solve may supply almost no positive learning signal. A curriculum can concentrate interaction on informative cases. But it can also narrow the learner’s experience around a small region of the generator, particularly if difficulty estimation responds to superficial factors rather than the intended skill.

Define $R(\pi, e)$ as the expected return of policy π in environment e . Training optimization roughly targets a sequence of expectations under D_t . Deployment, however, is judged against an external target distribution D_{target} . High return under the adaptive training distributions does not imply high return under D_{target} without additional assumptions about coverage, causal similarity, and the policy’s inductive biases.

The formal distinction explains why **difficulty and transfer are separate axes**. A task may be hard because it demands a valuable compositional skill, or hard because its random layout obscures a familiar pattern. A task may be easy but introduce a rare, foundational mechanism that the learner has never seen. Sampling solely by failure rate or loss can overemphasize pathological cases and underemphasize strategically important coverage gaps.

The right question is not simply, “Is the next task harder?” It is, “What previously unavailable evidence about the target capability does the next task provide?”

3. Four meanings of diversity

The word *diversity* is often used as if it has one operational definition. For environment generation, it helps to distinguish at least four kinds.

Surface diversity changes names, visual layouts, natural-language wording, and other features that should not affect the optimal strategy. Surface variation helps reduce memorization of incidental cues, but it may create almost no new decision structure.

Parameter diversity changes quantities within a familiar mechanism: timeout thresholds, inventory levels, dependency versions, or graph size. Parameter changes can be useful when a strategy should be robust across numerical regimes. They can also be insufficient when all tasks preserve the same underlying causal topology.

Structural diversity changes relationships among components: one API now depends on another, a failure propagates through a different service, or a repository moves from a monolithic interface to a plug-in architecture. Structural variation can require a different plan even when the same tools remain available.

Mechanism diversity changes the rule by which actions affect future state: a transaction becomes eventually consistent, retries become non-idempotent, a permission model changes, or a tool begins returning stale information. Mechanism changes can expose whether the agent learned a transferable method of investigation instead of a fixed recipe.

These categories are an analytical proposal, not a taxonomy that all prior papers use. Their purpose is to stop authors from treating ten thousand rewritten prompts as ten thousand independent learning problems.

Variation dimension	Illustrative generated change	Transfer question
Surface	Rename services and rewrite error text	Can the agent ignore irrelevant labels?
Parameter	Vary retry limits and queue sizes	Does the policy adapt to changed thresholds?

Variation dimension	Illustrative generated change	Transfer question
Structural	Insert a cache between worker and database	Can it reason across a new dependency?
Mechanism	Change acknowledgments from synchronous to delayed	Can it revise assumptions about completion?

A robust evaluation should not collapse all four into one “unseen task” bucket. A test instance can be unseen in a random-seed sense while remaining highly familiar in its causal structure.

4. What procedural benchmarks established

Early benchmark design made the generalization problem visible in compact, reproducible worlds. CoinRun, introduced by OpenAI in 2018, uses procedurally generated levels to examine whether a policy trained on one collection of levels succeeds on unseen ones [1]. Progen, published at ICML 2020, expands this idea into 16 procedurally generated game-like environments and formal evaluation protocols addressing sample efficiency and generalization [2]. Its authors argue, on the basis of their experiments, that diverse environment distributions are important for training and evaluation.

These are valuable baselines precisely because they separate experience on training levels from performance on withheld levels. They do not, however, imply that fresh random seeds always represent unfamiliar mechanisms. A generator can create infinitely many arrangements from a fixed set of rules. Generalization to new arrangements is meaningful, but it is not identical to transfer when the underlying rules change.

Packer and colleagues’ 2018 study is relevant here because it compared generalization methods under controlled in-distribution and out-of-distribution settings [3]. One of its findings was that ordinary RL algorithms could outperform specialized methods in some tested generalization conditions. The lesson is methodological rather than a blanket verdict about all algorithms: a new training method deserves evaluation against strong simple baselines and well-described shifts.

For agent developers, procedural tasks are attractive because they are inexpensive to vary once the generator exists. They also make train/test contamination difficult to detect by inspection. Two repositories may contain different text yet instantiate the same dependency graph and repair procedure. Two synthetic workflows may use different API names yet present the same tool ordering problem. Counting examples is not enough; the generator’s underlying design space needs to be documented.

5. Adaptive curricula: genuine advances and new failure modes

Curriculum-learning research provides methods more principled than increasing a fixed difficulty label. Unsupervised environment design allows a system to select or generate task parameters during learning. In PAIRED, introduced by Dennis and colleagues at NeurIPS 2020, an environment-generating adversary is guided by regret based on the performance

difference between a protagonist and an antagonist. The design aims to create challenging but solvable environments rather than merely the worst conceivable worlds [4].

Prioritized Level Replay (PLR), introduced at ICML 2021, revisits previously seen procedurally generated levels according to estimates of their future learning value. Jiang and colleagues investigate temporal-difference error as a useful priority signal and report improvements in sample efficiency and generalization on Procgen in their study [5]. The contribution is not just “make tasks harder”; it is to allocate experience toward levels where additional learning appears valuable.

ACCEL, introduced at ICML 2022, combines regret-based selection with editing of existing levels so that environmental complexity can evolve near an agent’s capability frontier [6]. Its authors report empirical gains in the settings they study. Each of these methods offers a different answer to the **teacher’s optimization problem**: how should the distribution change when the student changes?

The methods also reveal a general hazard. A teacher that optimizes a proxy for usefulness can be exploited by the student or by its own generator. A particularly noisy level may trigger high temporal-difference error without teaching a transferable concept. A highly adversarial level may expose a valid weakness or may be nearly unsolvable because of a generator defect. A model may improve at the teacher’s preferred family of problems while leaving another capability untouched.

None of these cautions invalidates the published methods. They identify conditions under which a training result should be interpreted carefully when moving from controlled game environments to tool-using language agents. The criterion for a good teacher ultimately depends on external transfer, not only on its internal priority score.

6. Open-ended task spaces and agent adaptation

Open-ended environment research broadens the objective beyond mastery of one task family. DeepMind’s XLand work, reported in *Open-Ended Learning Leads to Generally Capable Agents* in 2021, trains agents across dynamically constructed multiplayer challenges within a common 3D world. Its authors study generalization to held-out tasks and observe behaviors such as experimentation and cooperation within the designed task universe [7]. Those results are significant within their experimental domain; they should not be treated as an unrestricted demonstration of general intelligence across real-world settings.

Subsequent work on *Human-Timescale Adaptation in an Open-Ended Task Space* studies fast adaptation to novel embodied 3D tasks following large-scale training [8]. It shifts the evaluation question from “Did the agent already know the solution?” to “How quickly can it learn a new task with limited experience?” That distinction is useful for software agents too. An agent that rapidly diagnoses an unfamiliar API may be more valuable than one with high one-shot success on a familiar template.

Open-endedness adds new measurement problems. Tasks can differ in attainable return, required interactions, and the number of relevant decisions. A single scalar average over an evolving world may not reveal which capabilities improved. Held-out probes should therefore include **specific changes in the environment’s causal structure**, not just uniformly sampled variants from the same world universe.

Training distributions also need a concept of coverage that is independent of the current model. If a policy happens to solve one category easily, its teacher might deprioritize that category even though it is underrepresented in the target population. Coverage constraints—by domain, mechanism, and required skill—can act as guardrails against a curriculum that overfits to the learner’s present failures.

7. The new opportunity: synthetic tool worlds and verifiable RL

Recent language-agent work makes curriculum design a practical concern rather than an analogy to game-playing. RandomWorld, introduced by Sullivan, Hartmann, and Koller at EMNLP 2025, procedurally generates interactive tools and compositional tool-use data. The authors report improvements on several tool-use benchmarks after supervised fine-tuning and RL with this synthetic data, including results on NESTFUL [9]. Its contribution is a mechanism for creating interactive, compositional training experiences rather than merely rewriting tool-call transcripts.

RLVE, published at ICML 2026, provides an especially direct example of adaptive problem distributions for language-model RL. Zeng and colleagues introduce RLVE-Gym, a suite of **400 verifiable environments**, and adapt the difficulty of generated problems as training progresses [10]. They report that joint training across these environments produced a **3.37-percentage-point average gain** across six reasoning benchmarks for their specified 1.5B model baseline, compared with **0.49 points** from the reported continuation baseline despite its greater compute use [10]. These are the paper authors’ findings under that study’s design, not results produced by Deaimer or by the present article.

SWE-Gym contributes a related but distinct strand by using real-world software-engineering task instances in executable repositories for training and verifier research [11]. The combined literature suggests that selecting and creating suitable environments is an increasingly central part of post-training. It does not establish that one selection heuristic or dataset design dominates across models, optimization methods, tool settings, and target workflows.

In particular, verifiability is not identical to transferability. A generated math problem can have an exact answer while remaining narrowly distributed. A tool-use task can have a rigorous outcome checker but still teach a shortcut tied to its API generator. Good reward signals are essential; they do not guarantee coverage of the target mechanisms.

8. Five ways a curriculum can cheat without breaking its rules

The phrase “curriculum cheats” is a metaphor for **distributional shortcuts**, not an allegation that a particular published system intentionally manipulates its results. The following are hypothesized failure modes that environment authors can test.

1. Template leakage. The task generator changes variable names and prose but keeps a diagnostic sentence that uniquely predicts the required fix. A learner may build a reliable lexical classifier instead of learning debugging. This can remain invisible if withheld tasks use the same generator templates.

2. Difficulty by friction. Harder tasks contain larger logs, more irrelevant files, or slower tools but no new causal dependencies. Agents fail more often because they run out of context or time. The training signal then encourages superficial search compression rather than

the intended domain skill.

3. Teacher–student collusion. An adaptive generator discovers that one narrow task family reliably creates large learning signals, so it oversamples that family. The student’s measured progress rises there while the rest of the target capability space receives little attention. This is an optimization pathology, not necessarily deliberate coordination.

4. Generator-specific affordances. All tasks share the same command conventions, error codes, or recovery sequences. A policy can exploit these stable conventions even when filenames and descriptions differ dramatically. Removing the generator or changing tool semantics reveals the shortcut.

5. Survivorship through invalid filtering. A generator silently discards tasks that are difficult to verify, have inconvenient state transitions, or require unusual tool operations. The resulting corpus is clean and high-scoring but systematically excludes important parts of the target domain. Validation improves sample quality while unintentionally narrowing capability coverage.

Each failure mode has an opposite risk. Excessive surface randomization may obscure useful cues. Requiring dramatic mechanism change in every task can eliminate gradual learning. Overly strict novelty filters can produce bizarre tasks that are not representative of real work. The goal is not maximal randomness; it is **intentional variation with a defensible link to target capabilities**.

9. Mechanism-centered environment generation

We propose treating a task generator as a structured program with named factors of variation. Rather than sampling one opaque seed and hoping the result is diverse, define separate generators for domain context, dependency topology, tool behavior, observable evidence, resource constraints, and target objective.

For a repository-repair curriculum, these factors might include: package manager; number of cooperating services; location of the underlying fault; whether the fault is deterministic; whether the correct repair changes a public interface; and whether a restart is required to expose success. Some factors are merely aesthetic, while others change the strategy needed to complete the task.

A **mechanism inventory** records what the generator can and cannot express. For example, an environment that always models synchronous writes cannot teach reasoning about eventual consistency, regardless of how many tasks it emits. A generator that never creates conflicting requirements cannot test prioritization under constraints. Writing this inventory allows an honest claim: the collection covers specified families of mechanisms, not every imaginable operational problem.

The inventory also supports validity checks. For each factor, the author can identify at least one *witness pair*: two valid generated instances that differ in that factor, where an expert’s reasoning or the observable consequences should change in a specified way. If changing the supposed “retry semantics” factor has no effect on any trajectory, the factor may be dead configuration rather than genuine task diversity.

When factors interact, one-at-a-time variation is not enough. A permission change may matter only when combined with cross-service data movement. A resource limit may matter

only with a large input. The generator should include planned combinations, particularly those where locally reasonable actions produce unexpected downstream consequences.

A useful generator should also produce **negative controls**: pairs that differ cosmetically but should admit equivalent solutions. Together, witness pairs and negative controls let reviewers ask both whether genuine mechanisms differ and whether irrelevant variation is being mistaken for difficulty.

10. A four-tier evaluation of transfer

Evaluating solely on unseen random seeds is insufficient for a curriculum whose intended outcome is broad agent capability. We propose a nested evaluation design with four levels of shift.

Evaluation tier	What changes	What it primarily tests
Tier A: unseen instances	New seeds and data under familiar rules	Interpolation within the generator
Tier B: surface shift	New names, phrasing, layouts, benign tool wrappers	Robustness to incidental variation
Tier C: recombination	Familiar mechanisms in unseen combinations	Compositional generalization
Tier D: mechanism shift	Withheld transition rule, dependency, or tool semantic	Diagnosis and adaptation beyond trained rules

The tiers are not a claim that every application admits an absolute ordering of difficulty. A Tier B instance may be harder for a particular model than a Tier D instance; tiers describe **what changed**, not a guaranteed monotonic ranking. Report them separately rather than averaging away their differences.

Tier D should be chosen with care. If an agent never had a chance to infer a withheld rule, poor performance reveals an information barrier, not necessarily an inability to generalize. Give the learner realistic observations and bounded exploration opportunities, then measure whether it updates its plan. This connects generalization to adaptation, not clairvoyance.

An evaluation must also be independent of the training generator where possible. A separate implementation, authoring team, or collection of naturally occurring tasks can expose conventions shared by all generated examples. At minimum, hold out entire combinations of causal factors, not just instance identifiers. Public benchmark overlap and accidental contamination should be documented transparently.

The result is a transfer profile, not one headline number. A policy may be strong at new-instance interpolation and weak at mechanism shift; another may be slower on familiar tasks but better at investigating new dependencies. Those are different capabilities that matter to different deployments.

11. Illustrative case: two agents, the same rising training curve

Consider a hypothetical curriculum for agents that must migrate an API integration. Every training task contains two services, a client, and a failing compatibility test. The generator chooses file layouts, language versions, error messages, and test inputs. An adaptive teacher raises difficulty by adding services and extending logs.

Agent A learns that most tasks can be solved by replacing one import path and modifying an expected-response fixture. Agent B learns to trace the request path, identify the contract between client and service, update relevant code, and test compatibility under both old and new response variants. Within the generator, both agents can achieve high scores; Agent A may even be faster.

Now introduce a held-out mechanism: the provider changes from an immediate API response to an asynchronous job with a status endpoint. The new task looks similar on the surface, but an import replacement no longer addresses the contract. Agent A repeatedly applies the familiar patch. Agent B inspects tool behavior, discovers the deferred state transition, and revises the integration flow. Such an outcome is plausible, but not established here by an experiment.

The example gives the evaluation designer three concrete levers. First, training tasks should vary the causal structure of tool responses when that skill is desired. Second, generated cases should include compatible-looking decoys whose successful solution requires inspection, not pattern matching. Third, held-out tests should ask whether the policy recognizes unfamiliar semantics and gathers the evidence needed to adapt.

Crucially, adding the asynchronous case to training changes the definition of what is held out. Once the learner has repeatedly encountered deferred responses, a new deferred-response task may become recombination or interpolation. Evaluation tiers must be tracked relative to the **actual training history**, not the author’s memory of what the generator was intended to produce.

12. Proposed experiment: does mechanism coverage beat cosmetic volume?

We propose a controlled study that varies two independent factors: **the amount of surface variation** and **the breadth of causal mechanisms**. Each factor has low and high settings, producing four training conditions. In the high-surface condition, prompts, names, layouts, and logs are diversified. In the high-mechanism condition, the generator changes dependency structures, action effects, or observation delays within a documented, solvable task universe.

The main hypothesis is that mechanism breadth, when accompanied by adequate feedback and learnable progression, should improve performance on Tier C and Tier D held-out tasks more consistently than surface variation alone. Surface variation may still contribute by reducing incidental memorization. This is a **proposed hypothesis**, not a result.

The study should match policies, initialization, training algorithms, reward criteria, and compute budgets. Where task lengths differ, report both environment interactions and token usage; equal task counts are not equal exposure. Repeat experiments across multiple seeds and, when feasible, different policy scales, because a curriculum that benefits one

model may overwhelm another.

At evaluation time, freeze the policy and test on all four tiers. Record success, safe-state preservation, intervention cost, number of informative tool calls, and adaptation after the first failed attempt. Report aggregate confidence intervals and paired comparisons where test instances can be matched. Analyze failure mechanisms rather than selecting only favorable examples.

A second phase would compare sampling policies on the same generator: uniform sampling, fixed easy-to-hard progression, prioritized replay, and a regret-inspired adaptive teacher. These are families motivated by existing literature [4–6], not claims that all can be transferred unchanged to language-agent RL. The objective would be to determine whether a sophisticated teacher creates value beyond a strong, well-covered uniform baseline.

Audit the generator while training. Record how often each structural factor appears; whether the teacher collapses onto a small subset; which environment families yield positive learning signals; and which valid tasks are rejected by quality filters. Inspecting the effective distribution matters because adaptive selection can move away from the intended curriculum.

Finally, use an independent evaluation world built by different templates or backed by authentic executable workflows. If gains vanish when generator conventions disappear, report that limitation. A negative transfer result would be scientifically valuable: it would identify which curriculum design features improved optimization without improving the capability the researchers sought.

13. Trade-offs, safety, and statistical restraint

An unrestricted environment generator can emit impossible, unsafe, or nonsensical tasks. More novelty is not always useful. Generated environments need solvability checks, safe execution boundaries, and rules for discarding states that violate the task contract. The filtering process itself must be audited so that entire categories of difficult but relevant tasks are not systematically removed.

Adaptive curricula introduce an additional issue: they are sensitive to the current policy and its stochastic trajectory. Two runs can induce different training distributions even with the same generator. Researchers should version generator code, record selection decisions, retain seeds, and report coverage over time. Otherwise, “trained for the same number of steps” may conceal exposure to different problem families.

Evaluation uncertainty also matters. Differences of a few successful tasks in a small sample should not be promoted into decisive evidence of better generalization. Use multiple independent task instances, repeated model runs where affordable, and uncertainty intervals. When a sample is small, conclusions should remain narrow.

There is also a safety distinction between teaching successful tool use and teaching authorized tool use. A generated environment should preserve access rules, allowed actions, and protected state when those are part of the desired behavior. An agent that solves tasks by bypassing simulated safeguards may score well under an incomplete objective while learning a policy unsuitable for deployment. Curriculum design cannot be separated entirely from environment validity and reward design.

14. Implications for Deaimer and environment buyers

Deaimer’s public research discusses custom executable agent environments and the value of experience portfolios spanning real workflows [12]. A mechanism-centered curriculum offers a narrower, testable specification for such collections: ask for coverage of distinct decision mechanisms, controlled variation within each, meaningful combinations, documented exclusions, and independent transfer tests.

For a team commissioning RL environments, the procurement question becomes, “What behaviors can this collection plausibly teach, and what evidence would show that they transfer?” Sample counts and task labels help characterize a delivery, but they cannot answer that question alone. This article proposes an evaluation approach; it does not claim Deaimer has run the experiment or established a specific performance advantage.

15. Conclusion

A curriculum can improve training reward and still leave the target capability unchanged. The failure does not require a corrupt reward or a broken simulator. It can emerge from a valid but narrow distribution of experiences, especially when a generator creates cosmetic novelty more readily than genuine changes in task structure.

Research on procedural benchmarks, unsupervised environment design, prioritized replay, open-ended learning, and adaptive verifiable environments demonstrates that **which experiences an agent receives** is a powerful design variable [1–11]. Those findings are real, but each is bounded by its experimental settings. The next question for tool-using agents is how to build curricula whose progress signals survive unfamiliar APIs, changed dependencies, and genuinely new causal mechanisms.

A practical way forward is to expose the generator’s design space, distinguish surface changes from mechanism changes, test witness pairs and negative controls, audit the effective training distribution, and evaluate transfer in tiers that make the kind of novelty explicit. Difficulty is useful when it produces informative experience. Diversity is useful when it covers relevant ways the world can change. Neither is a substitute for evidence that the learner acquired a skill it can use elsewhere.

References

- [1] OpenAI. “Quantifying generalization in reinforcement learning.” CoinRun research announcement, 6 December 2018. <https://openai.com/index/quantifying-generalization-in-reinforcement-learning/>
- [2] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. “Leveraging Procedural Generation to Benchmark Reinforcement Learning.” *ICML 2020*, PMLR 119:2048–2056. <https://proceedings.mlr.press/v119/cobbe20a.html>
- [3] C. Packer, K. Gao, J. Kos, P. Krähenbühl, V. Koltun, and D. Song. “Assessing Generalization in Deep Reinforcement Learning.” arXiv:1810.12282, 2018. <https://arxiv.org/abs/1810.12282>
- [4] M. Dennis, N. Jaques, E. Vinitzky, et al. “Emergent Complexity and Zero-shot Transfer via Unsupervised Environment Design.” *NeurIPS 2020*. <https://natashajaques.ai/publication/paired/>

- [5] M. Jiang, E. Grefenstette, and T. Rocktäschel. “Prioritized Level Replay.” *ICML 2021*, PMLR 139:4940–4950. <https://proceedings.mlr.press/v139/jiang21b.html>
- [6] J. Parker-Holder, M. Jiang, M. Dennis, et al. “Evolving Curricula with Regret-Based Environment Design.” *ICML 2022*, PMLR 162:17473–17498. <https://proceedings.mlr.press/v162/parker-holder22a.html>
- [7] Open Ended Learning Team and A. Stooke et al. “Open-Ended Learning Leads to Generally Capable Agents.” arXiv:2107.12808, 2021. <https://arxiv.org/abs/2107.12808>
- [8] J. Bauer, K. Baumli, F. Behbahani, et al. “Human-Timescale Adaptation in an Open-Ended Task Space.” *ICML 2023*, PMLR 202:1887–1935. <https://proceedings.mlr.press/v202/bauer23a.html>
- [9] M. Sullivan, M. Hartmann, and A. Koller. “Procedural Environment Generation for Tool-Use Agents.” *EMNLP 2025*, 18544–18562. <https://aclanthology.org/2025.emnlp-main.936/>
- [10] Z. Zeng, H. Ivison, Y. Wang, et al. “RLVE: Scaling Up Reinforcement Learning for Language Models with Adaptive Verifiable Environments.” *ICML 2026*, PMLR 306:153406–153428. <https://proceedings.mlr.press/v306/zeng26j.html>
- [11] J. Pan, X. Wang, G. Neubig, et al. “Training Software Engineering Agents and Verifiers with SWE-Gym.” *ICML 2025*, PMLR 267:47717–47737. <https://proceedings.mlr.press/v267/pa25g.html>
- [12] Deaimer Research. “The Experience Layer of AGI: How Real-World Tasks Shape More General AI.” Published 12 October 2026. <https://www.deaimer.com/research/the-experience-layer-of-agi-how-real-world-tasks-shape-more-general-ai>