Generative Simulators: The Foundations for Self-Adapting Intelligence

Patronus AI

Abstract

As machine learning systems become increasingly agentic, static datasets, hand-written environments, and fixed benchmarks are proving insufficient for training, evaluation, and governance. Agents deployed in real-world settings must reason under uncertainty, interact over long horizons, and continuously remember and adapt as their understanding of tasks, tools, and constraints evolves. We introduce *Generative Simulators*, adaptive environments that jointly co-generate tasks, world dynamics, and reward functions. We believe that generative simulators constitute the foundational infrastructure for self-adaptive world modeling, extending beyond reinforcement learning algorithms and beyond human-curated datasets.

1 Introduction

Modern machine learning systems are most commonly evaluated on static benchmarks, but these are susceptible to contamination, leakage, and saturation [4]. The paradigm of training and evaluating on reinforcement learning environments, each containing a set of tasks, world tools through which the model interacts with the environment and reward functions to evaluate and provide feedback on performance, has given rise to a new era of post-training. Most commonly found reinforcement learning post-training environments are designed for domain specific tasks like software engineering [2] and customer service [5, 1] and help assess and improve basic tool use, environment exploration and instruction following capabilities of models. However, with the recent improvements in models, these benchmark environments have come to saturate as well, such as τ^2-Bench getting nearly saturated by GPT-5.2 [3]. This saturation is due to lack of plasticity of these environments. We believe that the future of environments is dynamic and self-adaptive to model capabilities. This report provides a position and showcases a preview of our work on environments built to auto-scale with model capabilities. Additionally, we show that careful orchestration and design of task generation processes, tooling infrastructure and reward processes can remove the domain-specific restriction of environments and turn them into high plasticity worlds.

2 Generative Simulators

To demonstrate that automatic increases in plasticity are possible, we propose a novel class of adaptive environment generators called *Generative Simulators*. One such example of a *Generative Simulator* is shown in Figure 1. Once a task's difficulty and simple configurations have been specified, our Task Generation module would sequentially create tasks that satisfy these constraints, coupled with task timelines that are reflective of difficulty (e.g. difficult tasks should take longer to execute). A set of tools is also selected based on the task difficulty, with more challenging tasks requiring more extensive tool sets. Given the generated tasks, we would perform curriculum-based task filtering based on current agent's capabilities and required difficulty levels and couple these

tasks with tool stacks of appropriate complexity. These task-tool tuples would then be used to evaluate and train an agent. The agent then executes the tasks, producing a series of output trajectories. These trajectories are finally scored using a reward function to populate the final evaluation report. In an ideal scenario, this reward function is co-generated with the task definition but is ensured to be verifiable. In such a system, the three synthetic components of task generation, world tooling and reward modeling can be independently or jointly made more difficult which helps scale difficulty for problematic areas of the model specifically. This provides the required plasticity in the world. Furthermore, with this design, the domain specificity of an RL environment can be naturally altered by adding, removing or swapping out toolsets specific to a domain. For example, adding a browser use toolset to an existing SWE-Bench like task can help extend the domain-set to frontend development situations where the agent is required to debug visually using the browser tools.

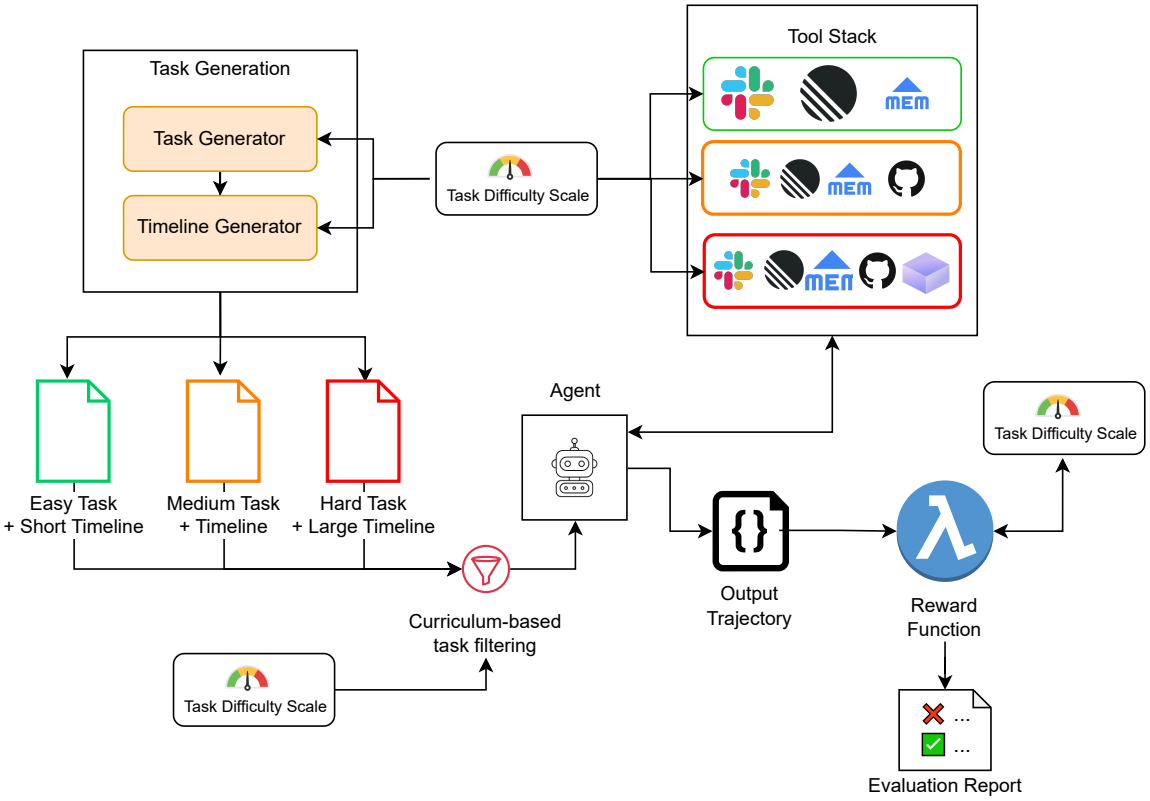

Figure 1: An illustration of Generative Simulators. Our adaptive environment generators are capable of producing a diverse set of tasks and tool sets with minimal specification.

2.1 Formal Definition of Generative Simulators

Let $\mathcal{L}$ denote a generative LLM. A Generative Simulator is a tuple $\mathcal{G} = (\mathcal{T}, \mathcal{W}, \mathcal{R}, \mathcal{C})$ where each component is an LLM-parameterized generator:

$\mathcal{T} : \Theta \times \mathcal{D} \xrightarrow{\mathcal{L}} \mathcal{M}$ — generates task instances given specifications and directives

$\mathcal{W} : \mathcal{M} \times \mathcal{D} \xrightarrow{\mathcal{L}} 2^{\mathcal{O}}$ — generates tool configurations conditioned on task and directives

$\mathcal{R} : \mathcal{M} \times \mathcal{D} \xrightarrow{\mathcal{L}} (\mathcal{H} \to \mathbb{R})$ — generates verifiable reward functions

$\mathcal{C} : \mathcal{M} \times \Pi \times \mathcal{D} \xrightarrow{\mathcal{L}} 0, 1$ — decides task inclusion based on agent capability

where $\mathcal{D}$ is the space of natural language directives (e.g., "increase task complexity", "add more tools", "make rewards sparser").

Algorithm 1 Generative Simulator Training with LLM-based Adaptation

Require: Generator LLM $\mathcal{L}_G$, Oversight LLM $\mathcal{L}_O$
Require: Initial directive $d_0 \in \mathcal{D}$, batch size K, threshold η
Initialize policy π_θ
Initialize directive $d \leftarrow d_0$
Initialize oversight memory $\mathcal{B} \leftarrow \emptyset$
for epoch $e = 1, \ldots, N$ **do**
  $\mathcal{D}_e \leftarrow \emptyset$ {Epoch buffer}
  for episode $k = 1, \ldots, K$ **do**
    {**Phase 1: LLM-based Environment Generation**}
    Sample base specification $\theta_k \sim p(\Theta)$
    $m_k \leftarrow \mathcal{L}_G\big(\texttt{GenTask}(\theta_k, d)\big)$
    $O_k \leftarrow \mathcal{L}_G\big(\texttt{GenTools}(m_k, d)\big)$
    $r_{m_k} \leftarrow \mathcal{L}_G\big(\texttt{GenReward}(m_k, d)\big)$
    {**Phase 2: LLM-based Curriculum Filtering**}
    $accept \leftarrow \mathcal{L}_O\big(\texttt{Filter}(m_k, \pi_\theta, d)\big)$
    if not *accept* **then**
      continue
    end if
    {**Phase 3: Agent Rollout**}
    $s_0 \sim \rho_0(m_k), \quad \tau_k \leftarrow \langle\rangle$
    for $t = 0, \ldots, T_{\max} - 1$ **do**
      $a_t \sim \pi_\theta(\cdot \mid s_t, O_k)$
      $s_{t+1} \sim P(\cdot \mid s_t, a_t, m_k, O_k)$
      $\tau_k \leftarrow \tau_k \oplus (s_t, a_t)$
      if $\textsc{Terminal}(s_{t+1}, m_k)$ **then**
        break
      end if
    end for
    $r_k \leftarrow r_{m_k}(\tau_k)$
    $\mathcal{D}_e \leftarrow \mathcal{D}_e \cup \{(\tau_k, r_k, m_k, O_k)\}$
  end for
  {**Phase 4: Policy Update**}
  $\theta \leftarrow \textsc{RLUpdate}(\theta, \mathcal{D}_e)$
  {**Phase 5: Oversight Analysis & Memory**}
  $\mathcal{F}_e \leftarrow \{(\tau, m, O) : (\tau, r, m, O) \in \mathcal{D}_e,\ r < \eta\}$
  $\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{F}_e$
  $\bar{r}_e \leftarrow \textsc{MeanReward}(\mathcal{D}_e)$
  {**Phase 6: LLM-based Directive Update**}
  $d \leftarrow \mathcal{L}_O\big(\texttt{Adapt}(d, \bar{r}_e, \mathcal{B})\big)$
end for
return π_θ

Algorithm 2 Oversight Directive Adaptation (LLM Subroutine)

Require: Oversight LLM $\mathcal{L}_O$
Require: Current directive d, mean reward $\bar{r}$, failure memory $\mathcal{B}$, threshold η
1: {**Failure Mode Analysis**}
2: *report* $\leftarrow \mathcal{L}_O\big(\textsc{AnalyzeFailures}(\mathcal{B})\big)$
3: {**Identify Bottleneck Dimensions**}
4: *bottleneck* $\leftarrow \mathcal{L}_O\big(\textsc{ClassifyFailures}(\textit{report})\big)$
5: {Returns subset of {`task`, `tools`, `reward`}}
6: {**Generate Updated Directive**}
7: **if** $\bar{r} \geq \eta$ **then**
8: $\quad$ *context* $\leftarrow (d,$ *bottleneck*, `"increase difficulty"`$)$
9: $\quad d' \leftarrow \mathcal{L}_O\big(\textsc{UpdateDirective}(\textit{context})\big)$
10: **else**
11: $\quad$ *context* $\leftarrow (d,$ *report*, `"maintain or simplify"`$)$
12: $\quad d' \leftarrow \mathcal{L}_O\big(\textsc{UpdateDirective}(\textit{context})\big)$
13: **end if**
14: **return** d'

2.2 Algorithm Overview

We present two interconnected algorithms that formalize our approach to LLM-driven generative simulation and curriculum learning. Algorithm 1 describes the main training loop, which interleaves environment generation, policy learning, and adaptive curriculum control. Algorithm 1 details the oversight mechanism that analyzes agent performance and updates training directives accordingly.

Main Training Loop (Algorithm 1) The algorithm operates in six distinct phases per epoch. First, a generator LLM ($\mathcal{L}_G$) synthesizes novel task instances, tool configurations, and reward functions based on the current directive d (lines 8-11). Second, an oversight LLM ($\mathcal{L}_O$) filters tasks based on estimated agent capability, implementing curriculum pacing (lines 13-16). Third, the agent performs rollouts in accepted environments (lines 18-26). Fourth, policy parameters are updated via standard RL (line 29). Fifth, failed trajectories are aggregated into an oversight memory $\mathcal{B}$ for analysis (lines 31-33). Finally, the oversight LLM adapts the directive based on performance trends and failure patterns (line 36), creating a closed feedback loop that continuously calibrates task difficulty and diversity.

Directive Adaptation (Algorithm 2) The oversight subroutine implements reflective curriculum control through structured LLM reasoning. It first analyzes accumulated failures to identify systematic patterns (line 2), then classifies bottlenecks across the task, tool, or reward dimensions (lines 4-5). Based on mean performance relative to threshold η, it generates an updated directive: when performance is high, it increases difficulty along bottleneck dimensions (lines 8-9); when performance is insufficient, it maintains or simplifies task distributions (lines 11-12). This conditional branching ensures the curriculum remains in the agent's zone of proximal development while systematically expanding coverage.

3 Conclusion

Present day static datasets, hand-authored environments, and human-curated demonstrations do not automatically scale with the learning patterns of the trained model. Generative Simulators provide a principled alternative: environments that evolve, evaluate, and adapt to agent behavior over time. To this extent, we propose a formalization for Generative Simulators and attempt to outline the workflow for such dynamic environments. While extremely long-horizon tasks spanning weeks remain difficult to evaluate and many objectives involve non-verifiable rewards such as usefulness or trust, we believe that autonomous scaling as in the case of Generative Simulators is crucial. We believe that these adaptive environments will contribute significantly to the growth of the field and would minimize human effort and supervision needed for aligning models. Future challenges include noisy sensors and irreversible actions and we believe that bridging generative simulators with robotics remains an open research problem that we are determined to solve.

References

[1] Victor Barres et al. "τ^2-Bench: Evaluating Conversational Agents in a Dual-Control Environment". In: *arXiv preprint arXiv:2506.07982* (2025).

[2] Carlos E Jimenez et al. "Swe-Bench: Cand Language Models Resolve Real-World Github Issues?" In: *12th International Conference on Learning Representations, ICLR 2024*. 2024.

[3] OpenAI. *Update to GPT-5 System Card: GPT-5.2*. `https://openai.com/index/gpt-5-system-card-update-gpt-5-2/`. Accessed: 2025-12-15. Dec. 2025.

[4] Manley Roberts et al. "To the cutoff ... and beyond? a longitudinal perspective on llm data contamination". In: *The Twelfth International Conference on Learning Representations*. 2023.

[5] Shunyu Yao et al. "τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains". In: *arXiv preprint arXiv:2406.12045* (2024).