RELIVEGYM: EVALUATING LONG-LIVED AGENTS OVER WEEKS OF REPLAYED REALITY

Anonymous authors
Paper under double-blind review

ABSTRACT

As large language model (LLM) agents become widely adopted, they are increasingly deployed for tasks that require persistent monitoring or recurring actions (e.g., market analysis). These agents are expected to operate unattended for days or weeks, act at the right timing, and adapt to the dynamic environment over time. These challenges are not fully captured in the existing long-horizon agent work, as they often consider a static environment that is not temporally changing. We introduce ReLiveGym, a diagnostic evaluation environment of *long-lived* tasks in which agents act sparsely over simulated weeks of chronologically replayed real-world news, market, and social-media streams. The tasks span diverse levels of time sensitivity, reasoning intensity, and recurrence. Across eight base language models, we investigate how model choice and harness design affect agent performance on such long-lived tasks. Our results show that how agents determine when to act arises as an important harness-design axis for long-lived tasks; and that the optimal design varies across tasks and sometimes model choices as well. We also evaluate how continuous learning from hindsight feedback affects performance and addresses failure modes observed in these long-lived tasks. These findings indicate model choice, action timing mechanism, and use of feedback as important considerations in the design of long-lived agents. Code: `https://anonymous.4open.science/r/Long-Lived-Agent-428C`.

1 INTRODUCTION

Large language model (LLM)-powered AI agents are increasingly deployed to operate unattended on tasks that unfold over days to months: from daily automation (OpenAI, 2026; n8n, 2026) to monitoring news for market-moving events (Qian et al., 2025). We refer to these systems as *long-lived* agents: agents that act intermittently over days to months in an evolving world in pursuit of an instruction or goal. They must act at the right moment, adapt to naturally occurring changes in their environment, and ideally continuously improve over time as the task recurs.

Existing works often capture only parts of the challenges in long-lived agents. Long-horizon or long-running agents (Chan et al., 2025; Novikov et al., 2025; Merrill et al., 2026; Zhu et al., 2026) act non-stop towards solving complicated programming, math, frontier science and engineering problems; however, unlike long-lived agents, these works typically do not assume the environment evolves chronologically without agents' actions. Conversely, some recent works study an independently evolving environment as an important challenge in practical agent use cases. In this setup, GAIA2 (Froger et al., 2026) evaluates agents in mobile app use and demonstrates that agents struggle with making timing decisions, and SentinelBench (Maldaner et al., 2026) further compares different wait mechanism designs. However, their tasks span only minutes to hours; challenges and opportunities such as continuous improvement can emerge in a longer time span. Evaluations in live environments (*e.g.*, real-stock-market trading (Qian et al., 2025; Ma et al., 2025)) or over replayed streams (Zhou et al., 2021; Goel et al., 2026; Chen et al., 2025) may span weeks to months, but often invoke agents on a fixed hourly or daily schedule; the choice of timing policy is rarely evaluated.

To study the challenges faced by long-lived agents, we introduce ReLiveGym, a diagnostic evaluation environment comprising 8 tasks that span weeks of chronologically replayed real-world streams. ReLiveGym replays news, social media, and market data on a simulated clock following Andrews et al. (2025); Goel et al. (2026). Its tasks vary along several axes: some admit fixed-interval schedules,

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.

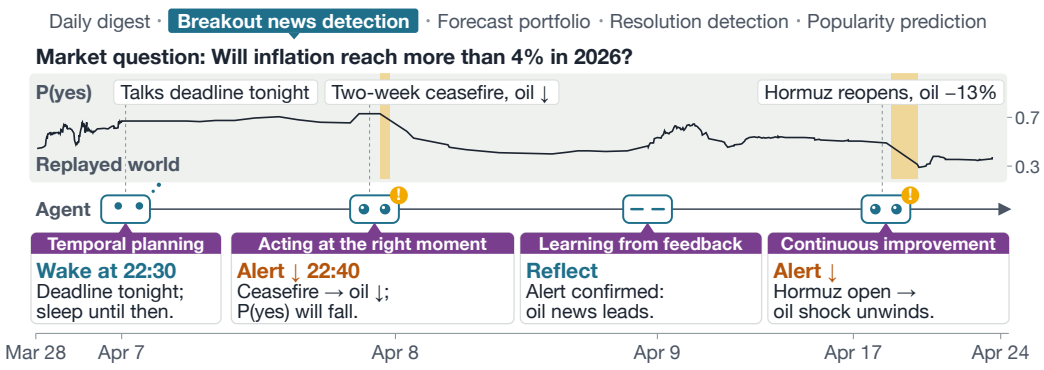


Figure 1: **A long-lived task in ReLiveGym.** The agent watches one replayed prediction market and news for four weeks and must alert before the price moves (shaded windows). It wakes sparsely, plans a wake for a known deadline, alerts before the market reacts, continously improve by reflecting on the confirmed outcome, and applies the lesson ten days later.

whereas others require timely, event-driven reactions. Some tasks require strong forecasting capabilities, whereas others emphasize the reliable execution of routine actions. Some tasks provide delayed feedback, creating opportunities for continuous test-time improvement. Each task is evaluated using deterministic metrics, such as accuracy and alert delay. Figure 1 illustrates an example breakout news detection task in ReLiveGym and the agent capabilities it requires.

Our key research questions are which factors affect the performance of long-lived agents and which failure modes arise empirically. We evaluate the effects of base LLM choice and multiple axes of harness design. Across tasks, base-model rankings are fairly consistent: GPT-family models perform best on reasoning-intensive tasks, whereas Qwen models perform best on routine tasks. We find that how an agent determines when to act arises as an important dimension of harness design; the effectiveness of each trigger mechanism depends on the task and, in some cases, the base model. We further evaluate continuous improvement methods on tasks that provide hindsight feedback. We find that simple methods, such as parallel sampling of agent trajectories (Koh et al., 2025; Gonzalez-Pumariega et al., 2025a), mitigate empirically observed failure modes, including persistence with a poor policy chosen at the outset. The performance of a given base LLM can vary substantially across repeated runs, but trigger mechanisms and continuous-improvement methods can improve its stability. We expect the artifact and these findings to contribute to the development of long-lived agents.

In summary, our contributions are: (1) ReLiveGym, an evaluation environment for long-lived agents comprising 8 tasks over weeks of chronologically replayed data; and (2) an analysis of how base-model choice and multiple axes of harness design affect long-lived agent performance, together with an empirical study of failure modes.

2 Problem Formulation

We formalize long-lived agents as programs that act intermittently in an exogenously evolving environment. The formulation specifies the agent program, its timed interactions with the environment, and the task constraints under which it operates.

Agents. An agent is a *program* $\mathcal{A} = (H, M)$ comprising a harness H and a base LLM M. The harness is the code that defines prompts and tools, initiates new turns, and manages context and memory. Together, H and M induce a policy that maps the interaction history to the next action. Given an initial user instruction x, the agent acts at a discrete sequence of timestamps over the horizon $[0, T]$, taking its i-th action a_i at time t_i. These actions include querying the current environment state $E(t_i)$ and committing user-facing results y_j at times t_j. Agents incur costs when invoking LLMs and operate under a cost budget B. We calculate these costs using provider rates listed in Table 3 in the Appendix and hold the rates fixed throughout the study.

Long-lived tasks and environments. A long-lived task is defined by an environment E, an initial user instruction x, a cost budget B, a time horizon T, and an evaluation metric f. The environment E evolves continuously over a horizon of world time $[0, T]$ (days to weeks). Crucially, this evolution is *exogenous*: the world state $E(t)$ evolves on the world's clock t whether or not the agent is watching. The metric function scores each commitment $\langle y_j, t_j \rangle$, so the value of a commitment depends on its timing t_j. We assume that the cost budget, time horizon, and evaluation metric are fixed and known to the agent at the outset and therefore need not be inferred; in practice, we include them in the agent's instruction. However, the agent cannot observe the future evolution of $E(t)$, which is grounded in changes recorded in the real world.

A solution to a ReLiveGym task is a *program*. It may be a ReAct agent (Yao et al., 2023), a ReAct agent invoked on a fixed cadence, a fixed LLM-based workflow, or a deterministic program that does not invoke an LLM. We focus our evaluation on the first two because a general ReAct loop can be transferred across tasks by changing its instruction and available tools.

3 RELIVEGYM: EVALUATING LONG-LIVED AGENTS OVER REPLAYED REALITY

ReLiveGym consists of 8 tasks built over chronological replay of real-world data for weeks of world time. The tasks are chosen to span the capability requirements across different axes. We describe the worlds and tasks, their curation process, and the evaluated capability axes. We then introduce how the agents are evaluated in a simulated environment.

3.1 CURATING TASKS AND SIMULATED WORLDS FOR RELIVEGYM TASKS

ReLiveGym tasks are created by first identifying replayable streaming data in the real world. We use timestamped news, market, financial-report, and social media data.

Simulated worlds. We build a news world from Common Crawl's news crawl with full-text BM25 search APIs available for the agent. The publish time of an article is inferred from both the publisher's claimed time and the crawler's crawl time. We build the prediction-market world from the non-sports Polymarket markets with $\geq \$100k$ volume that overlap the window, replaying each market's question, open date, minute-resolution prices and resolution. For market prices, we replay three exchanges' 5-minute candles. For financial filings, we snapshot the SEC's public filing data[1] for every held company and serve it as of the clock. For social media, we replay Reddit posts with their content and timestamped comment trees.

We sample data only after March 1, 2026, which is beyond the known knowledge cutoff dates of the evaluated LLMs, which minimizes the impact of data contamination. We release the data curation pipeline so that future LLMs can be evaluated on a newer environment.

3.2 RELIVEGYM TASKS AND CAPABILITY AXES

After the identification of replayable data, the authors proposed task templates that are realistic and cover diverse axes of the agent capability challenges. The task templates are instantiated as concrete tasks (*e.g.,* tracking concrete markets for a period) when grounded in a time window. We list ReLiveGym tasks across diverse axes of the agent capability in Figure 2, with details included in Table 2 in the Appendix. Below we introduce the key challenges and the associated tasks.

Action timing and schedule. ReLiveGym tasks diverge in how sharply the value of a commitment decays with earliness or delay, and whether a task implies a fixed schedule or is driven by world events. At one end, the task *daily digest* implies a fixed schedule: a once-a-day wake-up is the right policy. In the middle, *forecasting portfolio* is gradually time-sensitive: a forecast may be reviewed at any time before a question resolves, while earlier forecasts earn more credit. At the far end, tasks are sharply sensitive in one direction: *breakout news detection* requires early alerts, and is worth credit only before a market price move; *resolution detection*, in contrast, penalizes early claims before a

[1] `https://data.sec.gov`

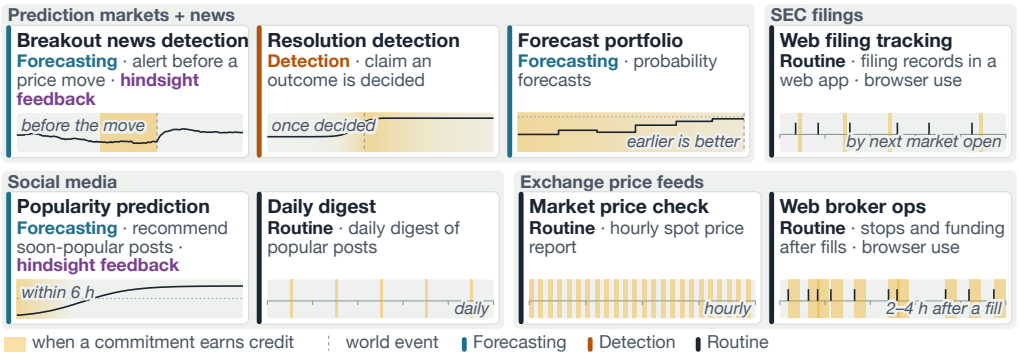


Figure 2: **The eight ReLiveGym tasks** curated over real-world replayed data streams. Agents are evaluated for their commitments such as breakout news alerts in the simulated weeks. The tasks diverge in challenge axes such as their timing requirements and reasoning depth.

controversy is settled in the world. *Web broker operations* requires agents to act before a deadline that is several hours after an event happens.

Reasoning depth and complexity. ReLiveGym tasks diverge in their reasoning depth and complexity. Some tasks, such as *price check* and *daily digest*, primarily require applying deterministic rules; this also applies to tasks like *web filing* and *web broker operations*, which add complexity by requiring browser use. Conversely, some tasks require synthesizing evidence across sources and reasoning under uncertainty. For example, *breakout news detection* requires agents to forecast future events; *resolution detection* requires agents to detect in a noisy world whether a controversy is settled.

Hindsight feedback opportunity. Some tasks in ReLiveGym enable hindsight feedback about an agent's past actions without human intervention. This applies to *breakout news detection* and *popularity prediction*, for which each alerted market's subsequent price path and each recommended post's popularity are visible within hours.

3.3 Evaluating Agents on ReLiveGym

Agents interact with each task through a simulated clock that controls which information is available. At time t, the environment returns only information that was visible by that point in the replayed stream. The clock advances only when the agent waits, using one of the mechanisms introduced in Sec. 4. Unlike the setup of Froger et al. (2026), our simulation does not model the LLM's own generation latency. We make this simplification because agents in our tasks act sparsely over simulated periods of several weeks.

At the start of a run, the agent receives an instruction specifying the task, its evaluation metric, and the end of the evaluation window. It must maximize its score over this window while operating under a hard cost budget, and the run terminates if the budget is exhausted. To support budget-aware decisions, the agent receives an updated summary of its spending and remaining budget on each simulated day. We score the agent deterministically by comparing its timed commitments with the corresponding ground truth. Table 2 in the Appendix provides detailed task descriptions and defines the task-specific evaluation metrics.

4 Experiment Setup

Our experiments on ReLiveGym answer the following research questions. **RQ1:** How do base LLM and harness designs affect model performance across ReLiveGym tasks? For harness designs, we focus on the mechanism by which agents make timing decisions, and continuous improvement methods. **RQ2:** What are notable failure modes of long-lived agents, and how do harness design and continuous improvement address them?

4.1 Trigger Mechanisms

We vary the trigger mechanisms by providing different tools to the agent for scheduling their next triggers.

LLM-owned sleeping tool (sleep harness). The ReAct agent is provided a *sleep* tool. The agent can name a duration, and the agent loop continues after the duration expires.

LLM-authored watcher programs (watcher harness). The agent is provided tools to read, author, and execute a Python watcher program as in the programmatic tool calling or CodeAct (Wang et al., 2024a;b) paradigm. The harness further allows the program to invoke tools that the agent has access to. The program can exit and hand over control back to the LLM with optional returned results.

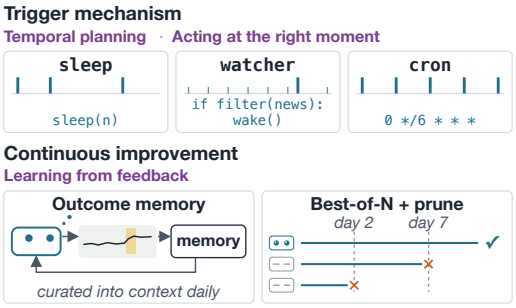


Figure 3: **Evaluated harness design axes**. Trigger mechanisms affect how agents plan over time and act at the right moment. Continous improvement methods affect how agents use feedback to improve continuously.

Agent-managed scheduled jobs (cron harness). A fixed schedule fires the program, runs a bounded ReAct episode at the current timestamp, and exits when the ReAct episode ends. The agent can list, create, update, and delete its own one-time and recurring triggers through environment endpoints with an optional note describing the intent of the trigger.

We further note that all three triggering mechanisms are expressive enough to allow the agent to schedule wakes at any timestamp $t \in [0, T]$. Details of the triggering mechanisms and corresponding examples are included in Appendix A.

4.2 Learning from Hindsight Feedback

We examine a memory-based approach and a parallel-sampling-based approach to learn from hindsight feedback as the agent acts online.

Outcome memory. Following Goel et al. (2026), after each simulated day, a process curates a memory entry that includes the agent's past commitments, the settled outcomes in the world, and numeric statistics (*e.g.*, the number of false positive alarms), added to the context of the agent.

Best-of-N + Prune. (Koh et al., 2025) As a simple and cost-efficient approach to test-time scaling with feedback, we simultaneously launch $N = 3$ agents at the start. For each day $[t_D, t_{D+1}]$ where t_D is the timestamp of the start of a day, only the commitments of the best-performing agent over $[0, t_D]$ are accepted. We prune agents by performance early on day 2 and day 7 to keep the run under budget.

Besides memory, we also test periodic harness optimization following recent works (Lee et al., 2026) in the code space. However, we notice that performance improves only with strong optimizer models (*e.g.*, GPT-5.6-sol), which exhaust the budget. We leave the discussion in Appendix B.

4.3 Model and Harness Details

We evaluate eight LLMs under our harness design axes. We only evaluate base LLMs that are cost-efficient when running over simulated weeks (*e.g.,* GPT-5.6-luna, GPT-5.6-terra). We skip more powerful models from the same family (*e.g.,* GPT-5.6-sol) if cheaper models already exhaust $20 in a single run over simulated weeks.

Our experiments control the harness within each ReLiveGym task. For the two tasks requiring a web user interface (*web filing* and *web broker ops*), we use the pinned-version OpenCode harness with Playwright MCP for browser use. For the other tasks, we use a customized agent harness and ReAct agent loop. The agent accumulates context over each wake and discards the earliest interaction history (other than the instruction) when the context exceeds a token limit of $60,000$. We believe more

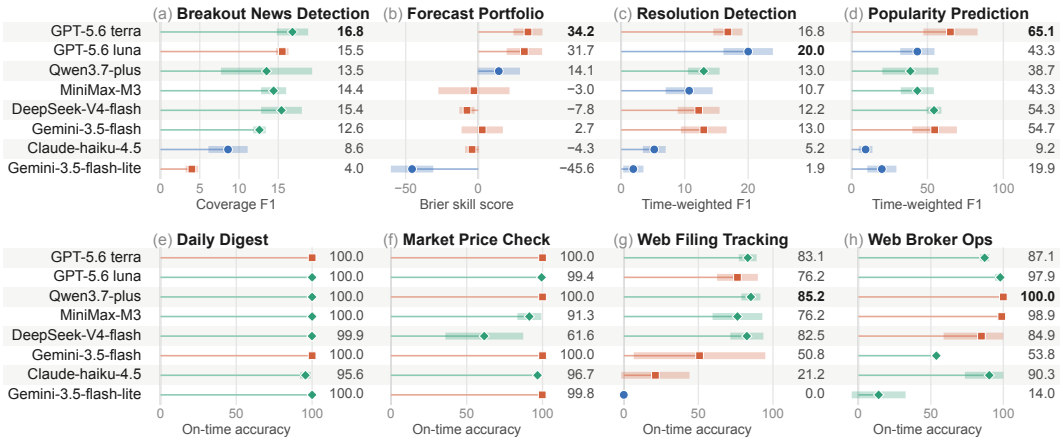


Figure 4: **Every model under its best triggering mechanism.** Each marker is the harness that scored highest for that model on that task (● sleep, ■ watcher, ◆ cron): mean score ($\times 100$) over seeds, band one standard deviation. Full results are in Table 4.

harness axes such as context length can impact cost and performance, and we leave such evaluation as future work.

5 Experiment Results

We start by analyzing the effects of models and trigger mechanisms on agent performance. We then analyze noticeable failure modes of long-lived agents and evaluate how continuous improvement methods address known failure modes of long-lived agents and improve their performance.

5.1 Model and Triggering Mechanism Effects on Performance

Performance of base LLMs. Figure 4 shows the performance of each base LLM under their best triggering mechanism. We notice that the best-performing model is fairly consistent: GPT-5.6-luna or terra performs best on forecasting and detection tasks that require extensive reasoning; Qwen-3.7-plus obtains the best (and even perfect) on-time accuracy in repeated runs on routine tasks such as daily digest, market price check, and web broker ops.

Trigger mechanisms affect the performance of agents. We analyze the effects of trigger mechanisms on the performance of frontier models. Figure 5 compares trigger mechanisms across two reasoning-intensive tasks and two routine tasks made more complex by browser use. On (a) breakout news detection, the watcher and cron triggers outperform the sleep trigger by approximately five coverage-F1 points for GPT models, whereas the triggers are indistinguishable for Qwen. On (b) forecast portfolio, the cron trigger performs worse than the sleep trigger. Under cron, agents wake most frequently and repeatedly shift their predictions toward the extremes of yes and no, resulting in many overconfident, incorrect forecasts. On the two browser-use tasks (c) and (d), the sleep trigger can perform catastrophically relative to the cron trigger: agents sometimes skip entire days without performing meaningful work. Figure 7 in the Appendix further visualizes trigger effects on weaker models. On daily digest, Gemini-3.5-flash-lite achieves 100% accuracy with the cron harness. It also achieves 100% accuracy with the watcher harness on price check by using programmatic tool calls in the agent-authored program to check prices and issue notifications, while performing poorly under other trigger mechanisms. These results show that the trigger mechanism can significantly affect agent performance on both routine and complex tasks, for frontier and weaker models alike.

Variance decomposition of model and trigger mechanism effects. We quantitatively evaluate how repeated runs, model choice, trigger mechanism, and their interactions affect agent performance. Figure 6 decomposes per-seed scores into components attributable to the model, trigger, model$\times$trigger interaction, and seed, restricting the analysis to the five strongest models per task.

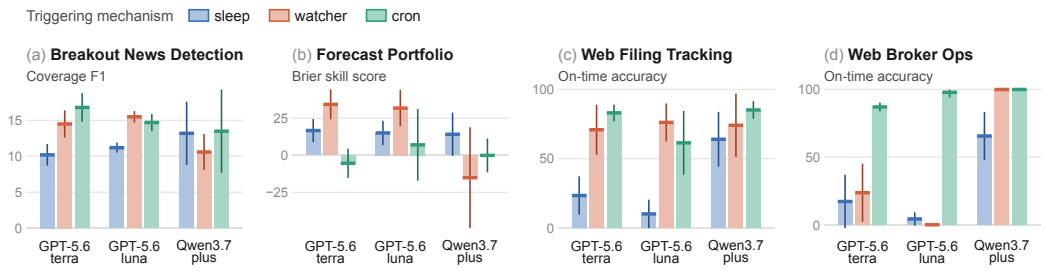


Figure 5: **Trigger mechanism effect in the models at the frontier performance.** Mean score ($\times 100$) over seeds under each harness (● sleep, ■ watcher, ♦ cron), whiskers one standard deviation.

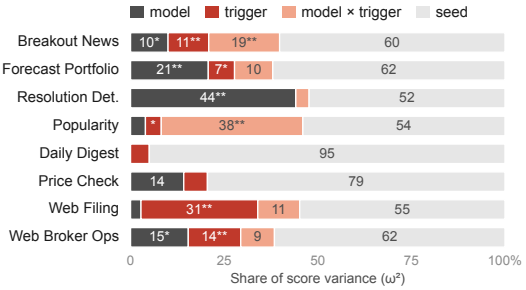


Figure 6: **Where the score variance goes among the five strongest models per task.** Two-way ANOVA share (ω^2) of per-seed scores attributed to the model, the harness, their interaction, and seed-to-seed variation; $^{*}p < 0.05$, $^{**}p < 0.01$ (permutation test).

The decomposition shows that seed-to-seed variation within the same model and trigger configuration is the largest component for every task. The differences across repeated runs can be further divided into two parts. Some differences concern individual scorable events, such as whether to issue an alert; expanding episodes (*e.g.*, time windows) per task can reduce the variance. Other differences affect many decisions over the course of a run. For example, an agent may repeatedly sleep too long, forget its instruction, or exhaust its budget before the episode ends. These persistent behaviors can produce large differences between runs, even when a task contains many scorable events; we consider this source of variance as a challenge of long-lived tasks, rather than evaluation noise. Table 6 in the Appendix further reports the overall spread across repeated runs, which differs across trigger mechanisms on several tasks. The case studies in Sec. 5.2 further illustrate persistent behaviors within runs.

Among the remaining components, the base LLM dominates for resolution detection (44%), forecast portfolio (21%), and web broker ops (15%). Models sometimes have their own best trigger mechanism: the contribution of the trigger mechanisms can arise as an interaction effect between the model and the triggering mechanism. This model$\times$trigger interaction is significant for breakout news detection (19%) and popularity prediction (38%). On some tasks we also see a trigger main effect, *i.e.,* certain triggers are uniformly better across all base models. This applies to breakout news detection (11%, with cron ahead of sleep and watcher with Wilcoxon $p = 0.039, 0.039$ over 8 models), forecast portfolio (7%, with cron behind sleep and watcher with $p = 0.016, 0.078$), web filing (31%, cron ahead of sleep with $p = 0.031$), and web broker ops (31%, cron ahead of sleep with $p = 0.023$).

Triggering mechanisms affect LLM cost, but cost alone does not explain performance. The harness accounts for 16–28% of the variance in LLM cost on breakout news detection, resolution detection, and popularity prediction. However, across all model and harness configurations, LLM cost and average scores are only weakly correlated ($-0.22 \leq \rho \leq 0.09$). This suggests that the effects of harness design on performance cannot be explained simply by cost.

5.2 Analysis of Failure Modes

Our earlier analysis identifies notable failures associated with particular models or trigger mechanisms. We now examine agent execution traces to identify recurring failure modes.

Agents can deviate from instructions over long deployments. On daily digest, an agent may fail after a period of normal operation. In one Claude-Haiku run with the sleep harness, the agent operated normally for the first week, then slept for three weeks and attempted to deliver the remaining

Table 1: Continous improvement from hindsight feedback: Mean$_{\pm\text{std}}$ over seeds; below each value, mean LLM spend ($) or, if a seed ran out of budget, the mean share of the window seen (%). Best method per column shaded.

Model	Method	**Breakout News Detection** `coverage F1`			**Popularity Prediction** `time-weighted F1`		
		sleep	watcher	cron	sleep	watcher	cron
GPT-5.6 luna	No learning	11.2$_{\pm0.7}$ ($4.84)	15.5$_{\pm0.8}$ ($7.16)	14.7$_{\pm1.2}$ ($10.81)	43.3$_{\pm11.3}$ ($4.15)	38.2$_{\pm15.6}$ (87%)	41.9$_{\pm5.3}$ ($7.84)
	Memory	13.0$_{\pm0.7}$ ($6.82)	15.5$_{\pm0.8}$ (99%)	13.8$_{\pm0.7}$ ($11.25)	37.0$_{\pm12.9}$ ($3.65)	51.1$_{\pm18.9}$ (87%)	52.1$_{\pm18.0}$ ($8.39)
	Best-of-N+Prune	11.6$_{\pm0.8}$ ($8.04)	16.0$_{\pm0.3}$ ($9.65)	14.0$_{\pm0.6}$ ($16.19)	49.2$_{\pm8.2}$ ($6.85)	45.2$_{\pm15.8}$ (88%)	42.6$_{\pm1.3}$ ($14.99)
Qwen3.7-plus	No learning	13.2$_{\pm4.4}$ ($7.56)	10.6$_{\pm2.5}$ (33%)	13.5$_{\pm5.8}$ (71%)	23.9$_{\pm13.4}$ ($4.61)	26.3$_{\pm21.1}$ (86%)	38.7$_{\pm18.5}$ ($10.11)
	Memory	13.8$_{\pm2.2}$ ($11.50)	13.6$_{\pm2.4}$ (26%)	12.1$_{\pm4.5}$ (64%)	37.9$_{\pm13.1}$ ($5.22)	28.0$_{\pm17.3}$ (56%)	55.2$_{\pm7.9}$ ($13.89)
	Best-of-N+Prune	14.4$_{\pm2.9}$ (89%)	8.5$_{\pm2.3}$ (17%)	13.1$_{\pm2.5}$ (42%)	32.1$_{\pm5.4}$ ($8.42)	37.7$_{\pm15.2}$ (82%)	45.7$_{\pm6.7}$ (90%)

outputs on the final day, contrary to the task instructions in the system prompt. Similarly, in a run with the cron harness, the agent independently decided to "wait until more comments are settled before notification," resulting in missed notifications.

Strategies chosen early can persist. In contrast to late-stage deviations, Reddit popularity prediction exhibits an early, detectable failure mode across models. With the sleep harness, agents often choose their sleep intervals at the beginning of the task and retain them throughout the run; the weekly median sleep interval changes by less than one hour. On Qwen-3.7-plus, a run triggered only every 12 hours performed catastrophically (F1 score $= 5.5$) compared with runs triggered every six hours (F1 score $= 38$). This issue is not limited to the sleep or watcher harnesses. Even under the cron harness, an agent may persist with a poor selection strategy adopted at the outset. For example, one Qwen run with the cron harness consistently performed a no-op at every trigger.

In the next section, we show that test-time feedback can address some of these failures and improve the stability of agent performance.

5.3 Utilizing Test-Time Feedback

Best-of-N + Prune helps mitigate early failures. On popularity prediction, hindsight feedback identifies better policies among seeds within the first few days. Best-of-N selection based on accumulated feedback consistently improves every evaluated cell (*e.g.,* GPT-5.6-luna with sleep, $43.3 \rightarrow 49.2$; Qwen-3.7-plus with sleep, $23.9 \rightarrow 32.1$), halves the variance across seeds under the sleep trigger, and incurs on average $1.61\times$ increase in cost. However, this benefit appears specific to the early-failure mode in popularity prediction: the approach does not consistently improve breakout news detection, which does not exhibit the same failure mode.

Effect of memory curated from hindsight feedback. On breakout news detection, memory yields directionally positive changes under the sleep and watcher triggers but a negative change under the cron trigger. On popularity prediction, the change is directionally positive in five of six cells, with GPT-5.6-luna under the sleep harness as the exception. This overall positive tendency suggests that agent memory can improve the performance of long-lived agents. However, its weaker consistency relative to best-of-N sampling indicates that memory alone may be insufficient to redirect an agent after it adopts a poor strategy. Future work can explore cost-efficient methods that help agents use past experience both to accumulate task knowledge and to revise earlier strategies.

We also experiment with continuous harness updates following prior work (Lee et al., 2026; Karten et al., 2026). The algorithm is effective only with strong reflector models, whose cost causes the approach to exhaust its budget early in our setup. We provide further discussion in the Appendix B.

6 Related Works

Long-horizon agents and temporal awareness. Agent benchmarks have expanded from bounded software and tool-use tasks (Jimenez et al., 2023; Yao et al., 2024) to extended coding and research

tasks (Yang et al., 2026a; Zhu et al., 2026). A complementary line studies agents in environments that change over time: TimeArena and Robotouille model action durations and asynchronous execution (Zhang et al., 2024; Gonzalez-Pumariega et al., 2025b), while real-time reasoning evaluates the tradeoff between reasoning depth and response latency (Wen et al., 2025). GAIA2 evaluates dynamic, asynchronous interactions (Froger et al., 2026), and SentinelBench directly compares waiting mechanisms for monitoring tasks using success, reaction time, and resource use (Maldaner et al., 2026). Closely related work studies lightweight triggers for LLM invocation (Liu et al., 2026a) and the execution of deferred intentions across a simulated week (Liu & Gabriel, 2026). Together with evidence of temporal misalignment in tool use (Cheng et al., 2025), these works motivate evaluating timing alongside task completion. ReLiveGym extends this evaluation to recurring obligations and sparse actions over weeks of replayed real-world streams under a cumulative budget. Related problems also arise in stream monitoring (Lin et al., 2016), event- and self-triggered control (Heemels et al., 2012), and change detection with costly observations (Banerjee & Veeravalli, 2012).

Agent evaluation on live and replayed worlds. Forecasting benchmarks evaluate predictions against subsequently resolved events (Zeng et al., 2025; Bianchi et al., 2025; Karger et al., 2025; Yang et al., 2025), while financial agents are evaluated on live markets and historical data (Qian et al., 2025; Yu et al., 2024; Li et al., 2024; Chen et al., 2025). Chai et al. (2026) simulates the state of the web at any specific past date through post-cutoff agentic content filtering. Particularly close to our setting, FutureSim replays news chronologically over 90 days and lets agents revise forecasts and adapt from resolved outcomes; its reported environment advances at a daily cadence (Goel et al., 2026). ReLiveGym shares the use of replay, while evaluating finer-grained trigger decisions across forecasting, detection, and routine automation tasks.

Test-time continuous improvement. Agents can improve through reflection on execution feedback (Shinn et al., 2023; Zhao et al., 2023) and reuse experience through memories, workflows, or skills (Wang et al., 2024c; Ouyang et al., 2025; Zheng et al., 2025). Live-Evo studies memory evolution from continuous forecasting feedback (Zhang et al., 2026), and EvoArena evaluates memory evolution in dynamic environments (Xu et al., 2026). Other methods optimize prompts (Agrawal et al., 2025), skills (Yang et al., 2026b), harnesses (Lee et al., 2026; Wei et al., 2026), or entire agent programs (Zhang et al., 2025; Yin et al., 2025); Adaptive Auto-Harness extends harness adaptation to open-ended task streams (Liu et al., 2026b). Complementary work scales agent inference through search and multiple candidate trajectories (Koh et al., 2025; Gonzalez-Pumariega et al., 2025a).

7 Conclusion

We introduced ReLiveGym, an evaluation environment that places agents in a chronologically replayed world for weeks. We separated two harness decisions: when the agent wakes and how it uses hindsight feedback, and evaluated three trigger mechanisms and two continuous improvement methods across eight LLMs with repeated runs per configuration. Our experiments demonstrate that base models and trigger mechanisms have an impact on the performance of agents via standalone effects or their interactions. Best-of-N selection with pruning mitigates the early failures that reflection leaves in place, while memories curated from hindsight feedback improve performance less consistently. We hope our paper motivates future research on more stable, performant, and cost-effective models and harnesses for long-lived agents deployed in real-world, online, dynamic environments.

Limitations. This paper is an evaluation study rather than a leaderboard. The replay environment and data pipeline support arbitrarily many models, harnesses, and repeated runs; we report three to six runs per configuration, run 1 to 3 episodes per task, and draw conclusions based on comparisons aggregated across models or harness design axes. We do not rank individual model-and-trigger pairs in this work. Future work that needs to perform benchmarking and separate methods at the level of individual cells can run more seeds per configuration and replay additional episodes to add scorable events; the environment supports both without any change to the tasks or metrics.

In addition, we evaluated only two axes of harness design: triggering mechanisms and test-time learning methods. Evaluation along additional axes, such as context management, would be valuable. Furthermore, within these two axes of harness design, we evaluated only three triggering mechanisms and two test-time learning approaches. Expanding the set of methods is left for future work.

AI USE STATEMENT

In this work, we used AI assistance to implement the experimental framework and methods. Human authors used AI coding agents to implement an agent-interactive environment with simulated time, pipelines for batch processing raw data (such as Common Crawl news), the agent loop used by task-solving agents, triggering mechanisms, and continuous-improvement algorithms. All code was reviewed by the human authors.

The experiments involve three non-code artifacts created with AI assistance: (1) the initial task instructions provided to the evaluated agents and (2) the synthetic trades placed against replayed real-world market data in the *web broker ops* task, (3) the web portals that *web filing tracking* and *web broker ops* task operates on. All artifacts were reviewed and revised by the human authors.

During the early stages of the project, we also used AI assistance for brainstorming, defining the scope of the paper, and identifying related work. Later, AI assisted with statistical and qualitative analyses of raw experimental records, including agents' final scores and action trajectories. AI also assisted in polishing writing and generating vector graphics for this paper.

REPRODUCIBILITY STATEMENT

We release code and data curation scripts at `https://anonymous.4open.science/r/Long-Lived-Agent-428C`. All required data is publicly accessible and permissively licensed; details in Appendix D.

REFERENCES

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei A. Zaharia, and O. Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning. *ArXiv*, abs/2507.19457, 2025.

Pierre Andrews, Amine Benhalloum, Gerard Moreno-Torres Bertran, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Romain Froger, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurenccon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Ménard, Grégoire Mialon, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Thomas Scialom, Vladislav Vorotilov, Mengjue Wang, and Ian Yu. Are: Scaling up agent environments and evaluations. *ArXiv*, abs/2509.17158, 2025.

Taposh Banerjee and Venugopal V. Veeravalli. Data-efficient quickest change detection with on–off observation control. *Sequential Analysis*, 31(1):40–77, 2012. doi: 10.1080/07474946.2012.651981. URL `https://doi.org/10.1080/07474946.2012.651981`.

Federico Bianchi, Junlin Wang, Zain Hasan, Shang Zhu, Roy Yuan, Clementine Fourrier, and James Zou. Back to the future: Evaluating ai agents on predicting future events. Together AI blogpost, July 2025. URL `https://www.together.ai/blog/futurebench`.

Jingyi Chai, Bingyang Zheng, Xiangrui Liu, Haoye Lu, Zihan Zhou, Tianchen Wang, Kemeng Zhang, and Si-Heng Chen. Agentic time machine as an infrastructure for future-event forecasting. *ArXiv*, abs/2606.21013, 2026. URL `https://api.semanticscholar.org/CorpusID:289621445`.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=6s5uXNWGIh`.

Yanxu Chen, Zijun Yao, Yantao Liu, Jin Ye, Jianing Yu, Lei Hou, and Juanzi Li. Stockbench: Can llm agents trade stocks profitably in real-world markets? *ArXiv*, abs/2510.02209, 2025.

Yize Cheng, Arshia Soltani Moakhar, Chenrui Fan, Kazem Faghih, Parsa Hosseini, Wenxiao Wang, and Soheil Feizi. Your llm agents are temporally blind: The misalignment between tool use decisions and human time perception. In *Annual Meeting of the Association for Computational Linguistics*, 2025.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, Kunal Malkan, Dheeraj Mekala, Pierre Menard, Gerard Moreno-Torres Bertran, Ulyana Piterbarg, Mikhail Plekhanov, Mathieu Rita, Andrey Rusakov, Vladislav Vorotilov, Mengjue Wang, Ian Yu, Amine Benhalloum, Grégoire Mialon, and Thomas Scialom. Gaia2: Benchmarking LLM agents on dynamic and asynchronous environments. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=9gw03JpKK4.

Shashwat Goel, Nikhil Chandak, Arvindh Arun, Ameya Prabhu, Steffen Staab, Moritz Hardt, Maksym Andriushchenko, and Jonas Geiping. Futuresim: Replaying world events to evaluate adaptive agents. *ArXiv*, abs/2605.15188, 2026.

Gonzalo Gonzalez-Pumariega, Vincent Tu, Chih-Lun Lee, Jiachen Yang, Ang Li, and Xin Eric Wang. Scaling agents for computer use. *Trans. Mach. Learn. Res.*, 2026, 2025a.

Gonzalo Gonzalez-Pumariega, Leong Su Yean, Neha Sunkara, and Sanjiban Choudhury. Robotouille: An asynchronous planning benchmark for llm agents. *ArXiv*, abs/2502.05227, 2025b.

W. P. M. H. Heemels, Karl H. Johansson, and Paulo Tabuada. An introduction to event-triggered and self-triggered control. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, 2012. doi: 10.1109/CDC.2012.6425820. URL https://doi.org/10.1109/CDC.2012.6425820.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *ArXiv*, abs/2310.06770, 2023.

Ezra Karger, Houtan Bastani, Chen Yueh-Han, Zachary Jacobs, Danny Halawi, Fred Zhang, and Philip Tetlock. Forecastbench: A dynamic benchmark of AI forecasting capabilities. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=lfPkGWXLLf.

Seth Karten, Joel Zhang, Tersoo Upaa, Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual harness: Online adaptation for self-improving foundation agents. *ArXiv*, abs/2605.09998, 2026. URL https://api.semanticscholar.org/CorpusID:288257013.

Jing Yu Koh, Stephen Marcus McAleer, Daniel Fried, and Ruslan Salakhutdinov. Tree search for language model agents. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=QF0N3x2XVm.

Yoonho Lee, R Lakshmi Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *ArXiv*, abs/2603.28052, 2026.

Haohang Li, Yupeng Cao, Yangyang Yu, Shashidhar Reddy Javaji, Zhiyang Deng, Yueru He, Yuechen Jiang, Zining Zhu, Koduvayur Subbalakshmi, Guojun Xiong, Jimin Huang, Lingfei Qian, Xueqing Peng, Qianqian Xie, and Jordan W. Suchow. Investorbench: A benchmark for financial decision-making tasks with llm-based agent. *ArXiv*, abs/2412.18174, 2024.

Jimmy Lin, Adam Roegiest, Luchen Tan, Richard McCreadie, Ellen Voorhees, and Fernando Diaz. Overview of the TREC 2016 real-time summarization track. In *Proceedings of the Twenty-Fifth Text REtrieval Conference (TREC 2016)*, 2016. URL https://trec.nist.gov/pubs/trec25/papers/Overview-RT.pdf.

Genglin Liu and Saadia Gabriel. Pm-bench: Evaluating prospective memory in llm agents. In *Conference on Language Modeling (COLM)*, 2026.

Xiaoze Liu, Ruowang Zhang, Amir H. Abdi, Michel Galley, Zhikai Chen, Siheng Xiong, Xiaoqian Wang, and Jing Gao. Do proactive agents really need an llm to decide when to wake and what to anchor? *ArXiv*, abs/2605.30152, 2026a.

Zewen Liu, Zhan Shi, Yisi Sang, Bingxiang He, Minhua Lin, Tianxin Wei, Dakuo Wang, Benoît Dumoulin, Wei Jin, and Hanqing Lu. Adaptive auto-harness: Sustained self-improvement for agentic system deployment on open-ended task streams. *ArXiv*, abs/2606.01770, 2026b.

Tianmi Ma, Jiawei Du, Wenxin Huang, Wenjie Wang, Liang Xie, Xian Zhong, and Joey Tianyi Zhou. Agent trading arena: A study on numerical understanding in llm-based agents. In *Conference on Empirical Methods in Natural Language Processing*, 2025.

Matheus Kunzler Maldaner, Adam Fourney, Amanda Swearngin, Hussein Mozannar, Gagan Bansal, Maya Murad, Rafah Hosn, and Saleema Amershi. Sentinelbench: A benchmark for long-running monitoring agents. *ArXiv*, abs/2606.05342, 2026.

Mike A. Merrill, Alexander G Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Hyeob Shin, Thomas Walshe, Estefany Kelly Buchanan, Junhong Shen, Guanghao Ye, Hao Lin, Jason Poulos, Maoyu Wang, Marianna Nezhurina, Jenia Jitsev, Di Lu, Orfeas Menis-Mastromichalakis, Zhiwei Xu, Zizhao Chen, Yue Liu, Robert Zhang, Leon Liangyu Chen, Anurag Kashyap, Jan-Lucas Uslu, Jeffrey Li, Jianbo Wu, Minghao Yan, Song Bian, Vedang Sharma, Ke Sun, Steven Dillmann, Akshay Anand, Andrew Lanpouthakoun, Bardia Koopah, Changran Hu, Etash Kumar Guha, Gabriel H. S. Dreiman, Jiacheng Zhu, Karl Krauth, Li Zhong, Niklas Muennighoff, Robert K. Amanfu, Shangyin Tan, Shreyas Pimpalgaonkar, Tushar Aggarwal, Xia Lin, Xin Lan, Xuandong Zhao, Yiqing Liang, Yuanli Wang, Zilong Wang, Changzhi Zhou, David Heineman, Hange Liu, H. Trivedi, John Yang, Junhong Lin, Manish Shetty, Michael Yang, Nabil Omi, Negin Raoof, Shanda Li, Terry Yue Zhuo, Wu Lin, Yiwei Dai, Yuxin Wang, Wenhao Chai, Shang Zhou, Dariush Wahdany, Ziyu She, Jiaming Hu, Zhikang Dong, Yuxuan Zhu, Sasha Cui, Ahson Saiyed, Arinbjörn Kolbeinsson, Jesse Hu, Christopher Rytting, Ryan Marten, Yixin Wang, Alexandros G. Dimakis, Andy Konwinski, and Ludwig Schmidt. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. *ArXiv*, abs/2601.11868, 2026.

n8n. n8n – the platform for ai agents and workflow automation. Github, 2026. URL `https://github.com/n8n-io/n8n`.

Alexander Novikov, Ngân V˜u, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav M. Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, Matej Balog, and Google DeepMind. Alphaevolve: A coding agent for scientific and algorithmic discovery. *ArXiv*, abs/2506.13131, 2025.

OpenAI. Scheduled tasks: Run tasks on a schedule or from supported app events in chatgpt. OpenAI Documentation, 2026. URL `https://learn.chatgpt.com/docs/automations`.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. Reasoningbank: Scaling agent self-evolving with reasoning memory. *ArXiv*, abs/2509.25140, 2025.

Lingfei Qian, Xueqing Peng, Yan Wang, Vincent Jim Zhang, Huan He, Hanley Smith, Yi Han, Yueru He, Haohang Li, Yupeng Cao, Yangyang Yu, Alejandro Lopez-Lira, Peng Lu, Jian-Yun Nie, Guojun Xiong, Jimin Huang, and Sophia Ananiadou. When agents trade: Live multi-market trading benchmark for llm agents. *ArXiv*, abs/2510.11695, 2025.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems 36*, 2023.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*, 2024a.

Zhiruo Wang, Daniel Fried, and Graham Neubig. Trove: Inducing verifiable and efficient toolboxes for solving programmatic tasks. In *International Conference on Machine Learning*, 2024b.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *ArXiv*, abs/2409.07429, 2024c.

Tian-Xin Wei, Zhan Shi, Min hua Lin, Bing He, Zewen Liu, Yi-Si Sang, Yuanchen Bei, Xu-Ying Ning, Jiaru Zou, Ting-Wei Li, Xiao Lin, Yan-Jun Zhao, Chi Wang, Benoît Dumoulin, Dakuo Wang, Jing-Rui He, and Hanqing Lu. Evo-harness: Context-to-harness skill compilation for self-evolving agents. 2026. URL https://api.semanticscholar.org/CorpusID:291172284.

Yule Wen, Yixin Ye, Yanzhe Zhang, Diyi Yang, and Hao Zhu. Real-time reasoning agents in evolving environments. *ArXiv*, abs/2511.04898, 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*, abs/2606.13681, 2026.

John Yang, Kilian Lieret, Jeffrey Jian Ma, Parth Thakkar, Dmitrii Pedchenko, Sten Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, Diyi Yang, and Ofir Press. Programbench: Can language models rebuild programs from scratch? *ArXiv*, abs/2605.03546, 2026a.

Qingchuan Yang, Simon Mahns, Sida Li, Anri Gu, Jibang Wu, and Haifeng Xu. Llm-as-a-prophet: Understanding predictive intelligence with prophet arena. *ArXiv*, abs/2510.17638, 2025.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, Kai Qiu, Yuqing Yang, Dongdong Chen, Xue-Ting Yang, and Chong Luo. Skillopt: Executive strategy for self-evolving agent skills. *ArXiv*, abs/2605.23904, 2026b.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *ArXiv*, abs/2406.12045, 2024.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel agent: A self-referential agent framework for recursively self-improvement. In *Annual Meeting of the Association for Computational Linguistics*, 2025.

Haofei Yu, Yining Zhao, Guanyu Lin, and Jiaxuan You. Building social world models with large language models. *ArXiv*, abs/2606.11482, 2026.

Yangyang Yu, Zhiyuan Yao, Haohang Li, Zhiyang Deng, Yuechen Jiang, Yupeng Cao, Zhi Chen, Jordan W. Suchow, Zhenyu Cui, Rong Liu, Zhaozhuo Xu, Denghui Zhang, Koduvayur Subbalakshmi, Guojun Xiong, Yueru He, Jimin Huang, Dong Li, and Qianqian Xie. Fincon: A synthesized llm multi-agent system with conceptual verbal reinforcement for enhanced financial decision making. In *38th Conference on Neural Information Processing Systems (NeurIPS 2024)*, 2024.

Zhiyuan Zeng, Jiashuo Liu, Siyuan Chen, Tianci He, Yali Liao, Jinpeng Wang, Zaiyuan Wang, Yang Yang, Lingyue Yin, Mingren Yin, Zhenwei Zhu, Tianle Cai, Zehui Chen, Jiecao Chen, Yantao Du, Xiang Gao, Jiacheng Guo, Liang-shu Hu, Jianpeng Jiao, Xiangsheng Li, Jingkai Liu, Shuang Ni, Zhoufutu Wen, Ge Zhang, Kaiyuan Zhang, Xin Zhou, José H. Blanchet, Xipeng Qiu, Mengdi Wang, and Wenhao Huang. Futurex: An advanced live benchmark for llm agents in future prediction. *ArXiv*, abs/2508.11987, 2025.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin godel machine: Open-ended evolution of self-improving agents. *ArXiv*, abs/2505.22954, 2025.

Yao Zhang, Yiran Wu, Yi Yu, Qingyun Wu, and Huazheng Wang. Live-evo: Online evolution of agentic memory from continuous feedback. *ArXiv*, abs/2602.02369, 2026.

Yikai Zhang, Siyu Yuan, Caiyu Hu, Kyle Richardson, Yanghua Xiao, and Jiangjie Chen. Timearena: Shaping efficient multitasking language agents in a time-aware simulation. In *Annual Meeting of the Association for Computational Linguistics*, 2024.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Y. Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *AAAI Conference on Artificial Intelligence*, 2023.

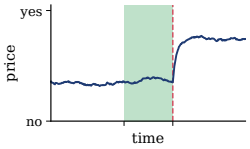
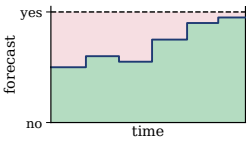
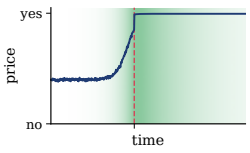
Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. Skillweaver: Web agents can self-improve by discovering and honing skills. *ArXiv*, abs/2504.07079, 2025.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

Zhihan Zhou, Liqian Ma, and Han Liu. Trade the event: Corporate events detection for news-based event-driven trading. In *Findings*, 2021.

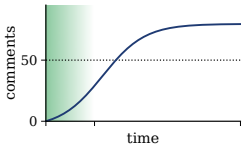
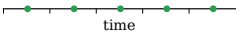
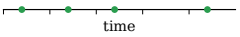
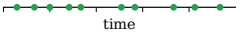
Deyao Zhu, Xin Zhou, Shengling Qin, Xuekai Zhu, Hangliang Ding, Shu Zhong, Zixin Wen, Zhonglin Xie, Chenhui Gou, Linxuan Ren, Yueyang Wang, Jun-Dong Zhong, Rui Liu, Tian Gao, Yangguang Lin, Jingyuan Zhang, Maojia Song, Xuan Qi, Jinhong Wu, Chenyang Zhang, Yinzhu Piao, Ziru Niu, Hongbin Lin, Lingxiang Meng, Peng Tang, Chen Tang, Shan-Shui Wu, Hua Zheng, Yu Liu, Liya Zhu, Hesong Wang, Mingguang Ding, Ziyu Wan, Hao Liu, Sibo Wang, Hao Zhu, Xintian Zhang, Nan Chai, Yipeng Liu, Panhao Lai, Sihang Yuan, Zi-Lu Su, Ge Zhang, Wangchunshu Zhou, Yantao Du, Wenhao Huang, and Guang Shi. Edgebench: Unveiling scaling laws of learning from real-world environments. 2026.

Table 2: The eight ReLiveGym tasks: what the agent must do (world, commitment, horizon) and the primary metric. The schematic sketches show the signal the agent watches over time, with the shade (or dots) marking when a commitment earns credit, fading as the credit decays over time.

Task	Description and primary metric
Breakout News Detection 	**Task description.** The agent monitors 8 Polymarket markets over a four-week episode, with the BM25 news search API and each market's minute-resolution price history up to the current time. A *breakout* is an abrupt price move following the definition of Yu et al. (2026): a day-over-day change that is statistically large against the market's recent volatility, localized to the minute it starts. A commitment is a notification ⟨market, article, direction⟩ claiming that the market will start a breakout in that direction within the next 24 h, citing the news article as evidence. A notification stands until a breakout in the named market starts or 24 h pass, and only one notification per market may stand at a time. A notification placed at or after the move starts earns nothing. The task is run over three episodes with disjoint markets and days, March 1–28, March 28–April 24 and April 24–May 21, 2026, holding 25, 30 and 33 ground-truth breakpoints (88 in total); scores pool the three episodes per seed. **Metric: coverage F1.** A breakpoint is *covered* if a notification with the right market and direction was standing in the 24 h *before* it started. Recall = covered breakpoints / all breakpoints; precision = covering notifications / all resolved notifications.
Forecast Portfolio 	**Task description.** The agent maintains standing probability forecasts on a portfolio of 60 real-world Polymarket questions (elections, conflicts, economic events) over 31 days (March 2026), with the news API; market prices are **not provided**. Questions enter the world at their real open date and may appear at any time. A commitment is a probability distribution over `yes` or `no`. Setup adapted from FutureSim Goel et al. (2026). **Metric: Brier skill score.** At every instant of a question's life its standing forecast p incurs the Brier score $\sum_o (p_o - \mathbf{1}[o \text{ resolved}])^2$; the per-question score is the exact time integral of this step function over the question's visible window divided by its length, and the run's Brier score BS is the mean over questions that resolved in-window (35 here). The skill score is $1 - \mathrm{BS}/\mathrm{BS}_{\text{base}}$, where $\mathrm{BS}_{\text{base}}$ is the Brier score of always forecasting the base rate of `yes` among the resolved questions (0.2 here). A perfect forecast scores 1, the base-rate forecast 0, and the uniform forecast over `yes` and `no` −0.56.
Resolution Detection 	**Task description.** The agent watches a portfolio of 60 standing Polymarket questions (elections, conflicts, corporate and economic events) over a 45-day episode with the news API. Market prices and resolutions are **not provided**: the agent must judge from news alone whether the world has already settled a question. Questions enter the portfolio at their real open date. In each episode 27 questions resolve inside the window and 33 do not; the latter stay open throughout, so any claim on them is a false alarm. A commitment is a claim, with the outcome, that a question is *already decided* in the world, as opposed to a forecast of how it will resolve. The agent gets one claim per question, ever; it cannot be revised or withdrawn, and questions decided during the run but never claimed count as misses. The task is run over three episodes with disjoint questions, March 1–April 15, April 10–May 25 and May 16–June 30, 2026 (81 in-window resolutions in total); scores pool the three episodes per seed. **Metric: time-weighted F1.** Ground truth is the earliest instant t^* at which the outcome was decided according to Polymarket (with a grace window). A correct claim at time t_i earns credit $e^{-(t^* - t_i)/12\text{h}}$ if early and $e^{-(t_i - t^*)/48\text{h}}$ if late. Recall = total credit / decided questions, precision = total credit / claims issued.

continued on next page

Table 2: (continued)

Task	Description and primary metric
Popularity Prediction 	**Task description.** The agent watches a live replay of ten AI-focused subreddits for 30 days (April 2026); new posts arrive continuously and each post's comment cascade is served as its causal prefix. A commitment is a recommendation of one post predicted to become *popular*, defined as ≥ 50 comments 24 h after posting; at most $K = 10$ recommendations are accepted per rolling 24 h. **Metric: time-weighted F1.** Each popular post that was recommended earns weight $\max(0,\ 1 - \text{latency}/6\text{h})$ for its first recommendation, where latency is measured from the post's creation—full credit at posting, none after six hours. Recall = total weight / the cap-limited oracle total ($\sum_{\text{days}} \min(K, \#\text{popular posts that day})$); precision = popular recommendations / all settled recommendations.
Daily Digest 	**Task description.** Same subreddit replay and 30-day window (April 2026). Once a day the agent delivers one digest of 10 popular (≥ 50 comments) posts from yesterday. **Metric: on-time accuracy.** For each scored day, the first 10 distinct ids of the accepted digest are checked: the day's score is the fraction of them that lie in the candidate window and are popular. A day without a digest scores 0.
Market Price Check 	**Task description.** Every hour for 14 days (March 1–15, 2026) the agent must report the current spot price of three given assets. Prices come from three exchanges' public 5-minute candle endpoints, replayed through a proxy that returns raw bodies and injects an incident stream fitted to public exchange status-page archives. The window holds 9 provider incidents (5 maintenance windows of 14–201 min, one 30 min outage, three degraded stretches of 47–318 min on one venue). **Metric: on-time accuracy.** The reference price for a report is the cross-venue median at the moment of submission. A symbol-hour scores 1 if the report is within 5 basis points of the reference and 0 otherwise.
Web Filing Tracking 	**Task description.** The agent keeps a fund's internal portfolio web app current for 33 holdings over April 1–June 1, 2026, reading a replay of SEC EDGAR's JSON endpoints served as of the current time. A commitment is a filing record entered through the app's web form in a browser: for each held company's earnings press release (8-K, item 2.02) and periodic report (10-Q/10-K), the accession number, form and acceptance time, plus, for periodic reports, the fiscal period, revenue and diluted EPS from the filing's XBRL facts. 63 filings count in this window. Following WebArena (Zhou et al., 2024), we created synthetic web form app. **Metric: on-time accuracy.** Each filing is checked once, at the first NYSE open after its acceptance. It scores 1 if its record exists, matches the filing's XBRL values within tolerance, and was saved after the filing was accepted and after the agent fetched that company's data from the replay; otherwise 0. The score is the mean over filings.
Web Broker Ops 	**Task description.** The agent keeps a levered crypto account (BTC, ETH; replayed 5-minute exchange candles) in good standing on a broker web portal for 14 days (April 1–15, 2026). The manager's book trades on a fixed schedule of day-orders; the agent does not trade. All actions go through the portal's forms in a browser. A commitment is a duty done in time: an opposite-side protective stop within 2 h of each fill, and a funding instruction within 4 h of each margin call (issued when the margin ratio falls below the broker's requirement, which the broker raises twice by notice). Every trade event is announced via synthetic event API accessible by the agent. The window holds 31 duty instances (29 fills, 2 margin calls) at instants spread across the day. **Metric: on-time accuracy.** Each duty instance is checked once, at its deadline, and scores 1 if it was met. The score is met instances / all instances.

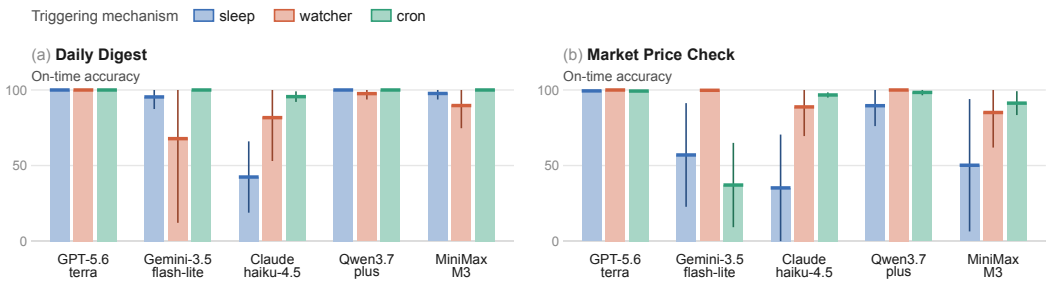


Figure 7: **The trigger lifts and steadies models below the frontier.** Mean score ($\times 100$) over seeds under each triggering mechanism, whiskers one standard deviation. GPT-5.6 terra is at the ceiling under any trigger; the other models reach it only under the right one.

Table 3: LLM cost rates in $ per million tokens. Every run's LLM spend, including the budget it is charged against, is computed from these fixed rates; cached input is the prompt-cache read rate.

Model	Provider	Input	Cached input	Output
GPT-5.6 luna	OpenAI	0.20	0.02	1.20
GPT-5.6 terra	OpenAI	2.00	0.20	12.00
Gemini-3.5-flash-lite	Google	0.30	0.03	2.50
Gemini-3.5-flash	Google	1.50	0.15	9.00
Claude-Haiku-4.5	Anthropic	1.00	0.10	5.00
Qwen-3.7-plus	Alibaba	0.32	0.064	1.28
MiniMax-M3	MiniMax	0.30	0.06	1.20
DeepSeek-V4-flash	DeepSeek	0.22	0.022	0.66

A Triggering Mechanism Details

Sleep harness. In the sleep harness, the agent is provided with a `sleep` tool. The environment fast-forwards its time to the specified time when the agent invokes the tool. Table 7 exemplifies an agent trace in the first simulated week on the breakout news detection task.

Watcher harness. Table 8 illustrates an agent trace with the watcher harness. In this harness, the agent authors a watcher program in Python and is provided with the `list_dir`, `read_file`, `write_file`, `edit_file`, and `run_program` tools. The agent can access its workspace directory, which is initialized with `envkit`, a library that allows agent-authored Python programs to invoke the same tools available to the agent. The workspace also includes an agent skill that instructs LLMs on how to write watcher programs using `envkit`. The skill comprises a Markdown instruction file and two example scripts: `sleep.py`, which sleeps until a specified timestamp, and `gatekeeper.py`, a basic task-specific condition watcher with placeholder entity names. Figure 8 shows an example of the default watcher program, and Figure 9 shows a program authored by GPT-5.6-luna.

In a ReAct loop, the agent authors the watcher program and invokes the `run_program` tool to execute the program. This tool call blocks until the program exits, or until `envkit.handover` (L22, Figure 9) is called in the program, which adds its argument to the tool call output. The simulated time of the environment passes only via `envkit.wait` called in the program (L23, Figure 9). A loop that never waits is terminated after $200,000$ executed lines of the agent-authored code, with library calls counting as a single line.

Cron harness. In the cron harness, the agent is triggered according to a cron schedule. The schedule table is initialized with a task-specific baseline schedule whose minimum trigger frequency is determined by the authors (*e.g.,* every six hours for breakout news detection and daily for the daily digest task). This baseline schedule is read-only for the agent, but the agent can create and update additional recurring or one-time trigger schedules. Table 9 illustrates an agent trace.

Table 4: Primary metric per model and timing mode (mean$_{\pm\text{std}}$ over 3 to 6 seeds). Below each value: ($) mean LLM spend when no seed ran out of budget, or (%) otherwise: the mean share of the simulated window survived across all seeds.

Model	**Breakout News Detection** `coverage F1`			**Forecast Portfolio** `brier skill score`			**Resolution Detection** `time-weighted F1`			**Popularity Prediction** `time-weighted F1`		
	sleep	watcher	cron	sleep	watcher	cron	sleep	watcher	cron	sleep	watcher	cron
GPT-5.6 luna	$11.2_{\pm 0.7}$ ($4.84)	$15.5_{\pm 0.8}$ ($7.16)	$14.7_{\pm 1.2}$ ($10.81)	$15.0_{\pm 8.2}$ ($0.26)	$31.7_{\pm 12.2}$ ($0.20)	$7.0_{\pm 24.1}$ ($0.41)	$20.0_{\pm 3.9}$ ($1.20)	$17.6_{\pm 5.8}$ ($2.78)	$17.1_{\pm 4.6}$ ($3.27)	$43.3_{\pm 11.3}$ ($4.15)	$38.2_{\pm 15.6}$ (87%)	$41.9_{\pm 5.3}$ ($7.84)
GPT-5.6 terra	$10.2_{\pm 1.5}$ (66%)	$14.5_{\pm 1.9}$ (59%)	$16.8_{\pm 2.0}$ (41%)	$16.6_{\pm 7.8}$ ($4.23)	$34.2_{\pm 10.0}$ (91%)	$-5.4_{\pm 9.8}$ ($6.79)	$11.5_{\pm 3.4}$ ($11.41)	$16.8_{\pm 2.3}$ (93%)	$15.3_{\pm 6.8}$ (89%)	$37.6_{\pm 13.6}$ (69%)	$65.1_{\pm 17.9}$ (90%)	$18.8_{\pm 7.2}$ (25%)
Gemini-3.5-flash-lite	$1.2_{\pm 2.2}$ ($0.86)	$4.0_{\pm 0.8}$ ($9.58)	$3.6_{\pm 0.1}$ ($14.28)	$-45.6_{\pm 14.6}$ ($0.11)	$-62.0_{\pm 5.6}$ ($0.70)	$-58.8_{\pm 4.4}$ ($0.20)	$1.9_{\pm 1.6}$ ($0.14)	$0.1_{\pm 0.1}$ (94%)	$0.0_{\pm 0.0}$ ($0.08)	$19.9_{\pm 9.6}$ ($2.39)	$18.8_{\pm 0.7}$ (45%)	$1.2_{\pm 1.2}$ ($7.20)
Gemini-3.5-flash	$10.4_{\pm 3.6}$ (38%)	$7.1_{\pm 1.0}$ (8%)	$12.6_{\pm 0.8}$ (27%)	$-20.8_{\pm 8.5}$ (88%)	$2.7_{\pm 14.2}$ (34%)	$-50.4_{\pm 8.4}$ (97%)	$7.3_{\pm 3.6}$ ($6.06)	$13.0_{\pm 3.6}$ (90%)	$7.0_{\pm 1.6}$ ($7.36)	$35.4_{\pm 4.8}$ (74%)	$54.7_{\pm 14.7}$ (91%)	$16.9_{\pm 1.0}$ (18%)
Claude-haiku-4.5	$8.6_{\pm 2.5}$ (62%)	$5.2_{\pm 2.3}$ (14%)	$6.8_{\pm 0.9}$ (10%)	$-14.1_{\pm 19.1}$ ($12.18)	$-4.3_{\pm 4.8}$ (99%)	$-9.4_{\pm 10.3}$ (78%)	$5.2_{\pm 1.8}$ ($6.23)	$4.7_{\pm 1.8}$ (95%)	$2.6_{\pm 0.6}$ (99%)	$9.2_{\pm 4.5}$ (79%)	$6.4_{\pm 3.6}$ (78%)	$4.2_{\pm 3.0}$ (23%)
Qwen3.7-plus	$13.2_{\pm 4.4}$ ($7.56)	$10.6_{\pm 2.5}$ (33%)	$13.5_{\pm 5.8}$ (71%)	$14.1_{\pm 14.6}$ ($1.02)	$-15.1_{\pm 33.9}$ ($1.34)	$-0.1_{\pm 11.4}$ ($1.07)	$3.9_{\pm 1.4}$ ($0.96)	$7.1_{\pm 6.2}$ (83%)	$13.0_{\pm 2.5}$ ($2.21)	$23.9_{\pm 13.4}$ ($4.61)	$26.3_{\pm 21.1}$ (86%)	$38.7_{\pm 18.5}$ ($10.11)
MiniMax-M3	$12.1_{\pm 2.5}$ ($7.83)	$4.4_{\pm 3.8}$ (74%)	$14.4_{\pm 1.6}$ (79%)	$-9.2_{\pm 20.7}$ ($2.30)	$-3.0_{\pm 24.5}$ ($3.32)	$-14.8_{\pm 18.1}$ ($2.31)	$10.7_{\pm 3.7}$ ($3.49)	$6.4_{\pm 3.0}$ ($7.68)	$9.7_{\pm 1.6}$ ($2.57)	$41.3_{\pm 19.1}$ ($5.90)	$26.8_{\pm 3.5}$ (92%)	$43.3_{\pm 10.9}$ (99.9%)
DeepSeek-V4-flash	$15.1_{\pm 2.5}$ ($7.32)	$13.5_{\pm 2.1}$ (88%)	$15.4_{\pm 2.6}$ ($16.59)	$-11.3_{\pm 17.5}$ ($1.65)	$-7.8_{\pm 5.3}$ ($3.14)	$-26.7_{\pm 6.9}$ ($1.28)	$10.7_{\pm 2.0}$ ($1.33)	$12.2_{\pm 3.3}$ ($3.59)	$11.7_{\pm 1.4}$ ($1.15)	$53.1_{\pm 4.3}$ ($5.45)	$36.3_{\pm 22.0}$ ($4.31)	$54.3_{\pm 4.9}$ ($12.24)

Model	**Daily Digest** `on-time accuracy`			**Market Price Check** `on-time accuracy`			**Web Filing Tracking** `on-time accuracy`			**Web Broker Ops** `on-time accuracy`		
	sleep	watcher	cron	sleep	watcher	cron	sleep	watcher	cron	sleep	watcher	cron
GPT-5.6 luna	$98.9_{\pm 2.0}$ ($0.43)	$98.9_{\pm 2.0}$ ($0.16)	$100.0_{\pm 0.0}$ ($0.39)	$68.2_{\pm 42.9}$ ($6.31)	$73.9_{\pm 36.9}$ ($0.01)	$99.4_{\pm 0.4}$ ($6.96)	$10.0_{\pm 10.3}$ ($0.14)	$76.2_{\pm 13.7}$ ($0.99)	$61.4_{\pm 23.1}$ ($1.10)	$76.2_{\pm 12.0}$ ($0.67)	$49.4_{\pm 40.5}$ ($0.36)	$68.3_{\pm 6.9}$ ($0.74)
GPT-5.6 terra	$100.0_{\pm 0.0}$ ($4.38)	$100.0_{\pm 0.0}$ ($0.27)	$100.0_{\pm 0.0}$ ($5.73)	$99.4_{\pm 0.1}$ (49%)	$100.0_{\pm 0.0}$ ($0.23)	$99.3_{\pm 0.0}$ (44%)	$23.3_{\pm 13.9}$ ($1.74)	$70.9_{\pm 18.1}$ ($6.99)	$83.1_{\pm 6.0}$ ($14.96)	$71.4_{\pm 9.1}$ ($5.05)	$59.5_{\pm 15.7}$ ($5.12)	$64.2_{\pm 27.9}$ (73%)
Gemini-3.5-flash-lite	$95.4_{\pm 8.0}$ ($1.82)	$67.8_{\pm 55.7}$ ($0.50)	$100.0_{\pm 0.0}$ ($0.81)	$57.0_{\pm 34.3}$ ($15.10)	$99.8_{\pm 0.1}$ ($5.15)	$37.1_{\pm 27.9}$ ($6.72)	$0.0_{\pm 0.0}$ ($0.01)	$0.0_{\pm 0.0}$ ($0.01)	$0.0_{\pm 0.0}$ ($3.07)	$0.0_{\pm 0.0}$ ($0.08)	$0.0_{\pm 0.0}$ ($0.04)	$2.5_{\pm 4.3}$ ($0.36)
Gemini-3.5-flash	$100.0_{\pm 0.0}$ ($5.61)	$100.0_{\pm 0.0}$ ($1.26)	$99.9_{\pm 0.2}$ ($5.78)	$97.6_{\pm 1.0}$ (14%)	$100.0_{\pm 0.0}$ ($0.74)	$96.5_{\pm 4.7}$ (14%)	$35.4_{\pm 56.0}$ ($8.53)	$50.8_{\pm 44.1}$ (69%)	$48.1_{\pm 24.7}$ (45%)	$65.8_{\pm 29.1}$ (71%)	$21.7_{\pm 37.5}$ (92%)	$54.4_{\pm 40.8}$ (51%)
Claude-haiku-4.5	$42.4_{\pm 23.6}$ ($6.39)	$81.7_{\pm 28.7}$ ($7.28)	$95.6_{\pm 3.5}$ ($8.07)	$35.2_{\pm 35.3}$ (72%)	$88.8_{\pm 19.3}$ (84%)	$96.7_{\pm 1.7}$ (12%)	$7.9_{\pm 7.3}$ ($2.29)	$21.2_{\pm 22.9}$ ($2.26)	$1.6_{\pm 1.6}$ ($5.91)	$28.6_{\pm 9.1}$ ($1.18)	$32.0_{\pm 7.6}$ ($1.25)	$73.8_{\pm 10.6}$ ($5.57)
Qwen3.7-plus	$100.0_{\pm 0.0}$ ($1.07)	$97.6_{\pm 3.9}$ ($0.88)	$100.0_{\pm 0.0}$ ($0.85)	$89.6_{\pm 13.5}$ ($17.73)	$100.0_{\pm 0.0}$ ($5.37)	$98.4_{\pm 1.9}$ ($18.52)	$64.0_{\pm 19.8}$ ($4.44)	$74.1_{\pm 22.9}$ ($2.64)	$85.2_{\pm 6.4}$ ($5.53)	$90.0_{\pm 3.1}$ ($5.56)	$65.5_{\pm 38.4}$ ($2.70)	$71.7_{\pm 17.2}$ (72%)
MiniMax-M3	$97.7_{\pm 4.0}$ ($1.54)	$89.7_{\pm 15.0}$ (81%)	$100.0_{\pm 0.0}$ ($1.11)	$50.2_{\pm 43.8}$ ($7.42)	$85.1_{\pm 23.2}$ ($0.51)	$91.3_{\pm 7.9}$ (99%)	$57.7_{\pm 25.5}$ ($4.41)	$33.3_{\pm 55.0}$ ($2.54)	$76.2_{\pm 16.7}$ ($8.89)	$61.1_{\pm 20.7}$ ($10.75)	$67.5_{\pm 31.5}$ ($4.16)	$83.5_{\pm 11.5}$ (87%)
DeepSeek-V4-flash	$99.9_{\pm 0.2}$ ($0.67)	$97.7_{\pm 4.0}$ ($0.23)	$99.9_{\pm 0.2}$ ($0.47)	$46.8_{\pm 10.9}$ ($6.32)	$54.7_{\pm 50.7}$ ($0.05)	$61.6_{\pm 25.7}$ ($7.61)	$25.9_{\pm 22.6}$ ($0.66)	$70.4_{\pm 20.7}$ ($0.87)	$82.5_{\pm 11.1}$ ($3.10)	$39.6_{\pm 46.3}$ ($0.35)	$77.4_{\pm 30.8}$ ($1.07)	$85.1_{\pm 7.1}$ ($3.17)

Table 5: Online harness updates on breakout news detection (GPT-5.6-luna actor, sleep harness, three seeds): coverage F1 ($\times 100$).

Harness	Coverage
No update	11.2 ± 0.7
Update, GPT-5.6-sol proposer	13.3 ± 3.8
Update, GPT-5.6-luna proposer	7.3 ± 3.0

B Online Harness Optimization

We test whether a coding agent can improve a long-lived agent's harness while the agent is running, in the spirit of end-to-end harness optimization (Lee et al., 2026). The setting is breakout news detection with a GPT-5.6-luna actor under the sleep harness on one four-week episode, cut into nine three-day spans. At every cut, a proposer (a coding agent, OpenAI Codex) reads the actor's trajectory so far together with the hindsight available at that point, *i.e.*, the price breakpoints and settled outcomes up to the cut, and edits the actor's harness code and state in place. All three seeds then resume the next span from the same world checkpoint with the edited harness. The proposer sees no future data, no held-out validation set, and no scorer; it may change anything in the actor program. We run the proposer with two backbones, GPT-5.6-sol and GPT-5.6-luna, and compare against the same harness left unchanged for the whole episode.

Table 5 summarizes the outcome. With GPT-5.6-sol as the proposer, the updated harness achieves 13.3 coverage F1 against 11.2 for the static harness; in contrast, GPT-5.6-luna proposer (same as the actor model) drops coverage F1 to 7.3. We notice that GPT-5.6-sol keeps the language model in the loop: it adds a second, stateless LLM review of every proposed notification with per-market confidence thresholds that it recalibrates at each cut from the previous span's accepted and rejected proposals, bounds how long an agent may sleep between news checks, and renders the settled outcomes so far into each agent's prompt. Its notes read like an experimenter's, attributing each missed breakpoint to cadence, direction, or reviewer strictness and adjusting a threshold. On the other hand, GPT-5.6-luna replaces judgment with fragile rules: a proposed notification must carry a catalyst term from a 45-entry list in its headline, a minimum count of direction-specific words, no

Table 6: Spread between repeated runs of the same model-and-trigger configuration, by trigger mechanism: pooled within-cell standard deviation of per-seed scores (metric $\times 100$) among the five strongest models per task, the models of Figure 6.

Task	Sleep	Watcher	Cron	All
Breakout News Detection	3.1	2.5	3.1	2.9
Forecast Portfolio	15.0	20.1	14.1	16.1
Resolution Detection	3.4	3.8	3.9	3.7
Popularity Prediction	11.8	16.0	6.1	12.3
Daily Digest	0.9	2.6	0.1	1.6
Market Price Check	27.5	17.5	4.7	19.0
Web Filing	19.3	30.0	14.3	22.2
Web Broker Ops	26.1	30.0	20.2	25.8

speculative modal, per-market identity terms, and must pass a recent-price check, unless it matches one of 47 "strong title overrides" that copy the exact headline forms of breakpoints the proposer had already seen. Hard rules distilled from a handful of past events do not transfer to the next three days, leading to performance below no update baseline.

C Seed-to-Seed Spread by Trigger Mechanism

Table 6 reports, for each task and trigger mechanism, how far repeated runs of the same model-and-trigger configuration spread apart. The spread changes with the trigger mechanism on popularity prediction, market price check, and the two web tasks, and stays within a few points under every trigger on the two detection tasks.

D Data Licenses

News articles are extracted from Common Crawl CC-NEWS (March to July 2026) under Common Crawl's Terms of Use; article text is used only inside the simulator and is not redistributed. Market metadata and hourly prices are fetched from Polymarket's public Gamma and CLOB APIs. Reddit submissions and comment trees for ten AI subreddits are obtained through the Arctic Shift public archive for non-commercial research, consistent with Reddit's Public Content Policy; no content is used for model training. We release the collection scripts, identifiers, and derived labels rather than raw third-party content.

Financial filings are taken from the SEC's EDGAR public data endpoints[2]: the ticker-to-CIK map, each company's submissions index (accession numbers, form types, and acceptance timestamps), and its XBRL company-facts records. These are U.S. government records in the public domain (17 U.S.C. § 105), which the SEC designates as public information that may be copied and distributed; we do not use the filing documents or exhibits themselves, only these structured records.

[2] https://www.sec.gov/search-filings/edgar-application-programming-interfaces

Table 7: GPT-5.6-luna w/ wait harness, in the first week on the market "Will no Fed rate cuts happen in 2026?" on breakout news detection. Each row is one `sleep` call: when the agent woke, when it chose to wake next, whether it filed a claim in that waking, and the thought it attached to the wait.

Wake (UTC)	Sleep until	Claim → outcome	Final thought
03-01 00:00	03-02 00:00	up → false alarm	Wait for the claim to resolve or for new macro news; avoid additional claims while this standing claim is active.
03-02 00:00	03-02 00:05	–	The initial claim remains active through 00:00Z; wait briefly for resolution before considering the fresh Iran-oil catalyst.
03-02 00:05	03-03 00:00	**up → covered**	Wait for the Iran-oil claim to resolve or expire; avoid locking the market with another claim.
03-02 13:55			**Breakpoint ↑ $+0.024$ (0.102 → 0.127): covered by the 00:05 claim.**
03-03 00:00	03-04 00:00	–	The March 2 upward move likely resolves the active claim; wait until it is settled before seeking another opportunity.
03-03 07:01			**Breakpoint ↑ $+0.052$ (0.127 → 0.178): missed, no claim standing.**
03-04 00:00	03-05 00:00	–	March 3 articles describe an already-priced oil-driven repricing, not a new pre-breakout catalyst; avoid a late claim and wait for fresh news.
03-05 00:00	03-05 06:00	–	No fresh March 5 Fed catalyst is visible yet, and the late March 4 drift is not an identifiable new breakout; wait for the US session/news cycle.
03-05 06:00	03-06 00:00	–	A sharp downside move began around 02:27Z, before Williams coverage published at 02:48Z, so notifying now would be late and scoreless; wait for a fresh catalyst.
03-06 00:00	03-07 00:00	up → wrong direction	Wait for the March 5 oil/inflation claim to resolve or expire before making another market claim.
03-06 13:41			**Breakpoint ↓ -0.075 (0.187 → 0.112): missed; the standing claim pointed the other way.**
03-07 00:00	03-08 00:00	–	The February payrolls shock was published and priced on March 6, while the active upward claim covered that window; a new downside notification now would be late. Wait for fresh news.

```python
"""example_gatekeeper.py | a minimal, runnable wake-condition program:
poll news on a cadence, hand over fresh keyword-matching articles.

Copy and edit it (write_file / edit_file), dry-run with
run_program(path, validate=true), then run_program(path). Wake times
live in code | wait(now() + interval); run_program's `until` is only a
hard timeout backstop. State persists in workspace files, so a trigger
interrupting the run loses nothing: just re-run the same program.

NB: `envkit` exists only inside your run_program workspace | this file is
a template, not an importable module of the repo.
"""

import json
from datetime import timedelta

from envkit import handover, now, search_news, wait, workspace

QUERY = '"rate cut" OR "FOMC"'       # BM25: quoted phrases, AND/OR
KEYWORDS = ("cut", "hike", "fed")    # cheap title filter on top of BM25
POLL_EVERY = timedelta(minutes=30)   # cadence lives in code

seen_file = workspace() / "gatekeeper_seen.json"
seen = set(json.loads(seen_file.read_text())) if seen_file.exists() else set()

while True:
    t = now()
    page = search_news(
        q=QUERY,
        date_from=(t - POLL_EVERY).strftime("%Y-%m-%dT%H:%M:%SZ"))
    hits = [h for h in page["results"]
            if h["news_id"] not in seen
            and any(k in h["title"].lower() for k in KEYWORDS)]
    seen |= {h["news_id"] for h in page["results"]}
    seen_file.write_text(json.dumps(sorted(seen)))
    if hits:
        handover({"matched": [
            {"news_id": h["news_id"], "title": h["title"],
             "published": h["published"]} for h in hits[:5]]})
    wait(t + POLL_EVERY)
```

Figure 8: Default watcher program shipped together with a ReLiveGym task for the watcher harness, exemplified with the breakout news detection task.

Table 8: GPT-5.6-luna w/ watcher harness in the first week on the market "Will no Fed rate cuts happen in 2026?" on breakout news detection. The agent has no `sleep` tool; it waits by running programs it writes itself. Each plain row is one waking: the program whose wait ended it (a news watcher handing over a headline, or the blind-wait program `sleep.py` reaching its wake time).

Wake (UTC)	Program run	File written	Claim → outcome	Final thought
03-01 00:00	`fed_watch.py`	write `fed_watch.py`	–	Dry-run the 30-minute Fed-news watcher before advancing simulated time.
03-01 00:00	`sleep.py`	edit `sleep.py`	up → false alarm	Advance 30 minutes while the standing claim remains unresolved.
03-01 00:30	`sleep.py`	edit `sleep.py`	–	Advance to just beyond the standing claim expiration.
03-02 00:05	`sleep.py`	edit `sleep.py`	**up → covered**	Advance beyond the 24-hour resolution window.
03-02 13:55				**Breakpoint ↑ $+0.024$ (0.102 → 0.127): covered by the 00:05 claim.**
03-03 00:10	`fed_watch2.py`	write `fed_watch2.py`	–	Continue polling for a US-specific Fed catalyst; no credible article is present now.
03-03 00:40	`fed_watch2.py`	–	–	The conflict article was published after the visible move, so it cannot support a timely claim; continue polling for a genuinely new catalyst.
03-03 01:10	`sleep.py`	edit `sleep.py`	**up → covered**	Advance to after expiration of the latest standing claim.
03-03 07:01				**Breakpoint ↑ $+0.052$ (0.127 → 0.178): covered by the 01:10 claim.**
03-04 01:15	`fed_watch2.py`	–	–	The market was already declining before the 01:02 article, so it cannot justify a timely claim; keep monitoring without notifying.
03-04 01:45	`fed_watch2.py`	–	–	The article arrived after the market's decline began and does not support a fresh upside breakout; continue monitoring without claiming.
03-04 02:15	`fed_watch2.py`	–	–	The Kashkari article was published after the upside move began at 01:50, so a notification now would be late; continue monitoring for the next breakout catalyst.
03-04 03:15	`sleep.py`	edit `sleep.py`	up → false alarm	Advance beyond the standing claim window.
03-05 03:20	`fed_watch2.py`	–	–	The ISM article supports no cuts, but the 02:27 move was sharply downward, so an upside claim would be wrong-direction and late; continue monitoring for the next breakout.
03-05 04:20	`fed_watch2.py`	–	–	No direct Fed-policy catalyst appeared after the decline; continue automated monitoring.
03-05 04:50	`sleep.py`	edit `sleep.py` (×2)	–	Advance to a later checkpoint for focused monitoring.
03-06 05:00	`sleep.py`	edit `sleep.py`	–	Advance to the jobs-data window without spending on low-relevance news.
03-06 12:00	`sleep.py`	edit `sleep.py`	**down → covered**	Advance beyond the current standing claim window.
03-06 13:41				**Breakpoint ↓ -0.075 (0.187 → 0.112): covered by the 12:00 claim on the payrolls report.**
03-07 12:05	`fed_watch2.py`	–	–	No new direct Fed-policy story is visible after the payroll-driven moves; continue automated monitoring.

Table 9: GPT-5.6-luna w/ cron harness in the first week on the market "Will no Fed rate cuts happen in 2026?" on breakout news detection. A fixed cron wakes the agent every six hours. The agent can create one-shot or recurring schedules of its own, but it never does in this run.

Trigger (UTC)	Tools called	Claim → outcome	Final thought
03-01 00:00 (bootstrap)	search ×5, article, notify	up → false alarm	No additional action is appropriate while the standing claim is unresolved.
03-01 00:05	–	–	The market is locked by the active claim, so I am monitoring without taking another action.
03-01 06:05	search	–	The market is locked by the existing claim, so the new corroborating article cannot be cited in another notification.
03-01 12:05	prices, search	–	The active claim remains unresolved and prevents replacement or additional notification.
03-01 18:05	search, prices	–	No new notification is allowed while the existing claim remains active, and no definitive breakout is visible.
03-02 00:05	search ×2, prices, article, notify ×4	**up → covered**	The new standing claim is accepted; no further notification is possible until it resolves.
03-02 06:05	prices, search	–	No new notification can be placed until the opposite breakout formally resolves the standing claim.
03-02 12:05	prices, search, notify	up (rejected)	One standing claim is still active; no further notification can be placed.
03-02 13:55			**Breakpoint ↑ $+0.024$ (0.102 → 0.127): covered by the 03-02 00:05 claim.**
03-02 18:05	search	–	The active claim already covers the upward repricing; further claims are blocked by the one-standing-claim rule.
03-03 00:05	prices, search ×2	–	Avoiding a low-confidence claim after an already-covered move and without a clearly imminent fresh breakout.
03-03 06:05	prices, search	–	Avoiding a low-confidence claim when price action is gradual rather than breakout-like.
03-03 07:01			**Breakpoint ↑ $+0.052$ (0.127 → 0.178): missed, no claim standing.**
03-03 12:05	search, prices	–	The visible move is already underway, so this is not actionable under the pre-breakout requirement.
03-03 18:05	search, prices	–	The likely move is already underway and directionally conflicted with later news, so a new claim would be late and low-confidence.
		⋮	
03-06 06:05	prices, search	–	The standing claim already covers the upward thesis and the market remains locked.
03-06 12:05	prices, search, notify	up → wrong direction	The claim is accepted and the market is locked until resolution.
03-06 13:41			**Breakpoint ↓ -0.075 (0.187 → 0.112): missed; the claim filed 96 minutes earlier pointed up.**
03-06 18:05	prices, search	–	The market is locked by the standing claim, and the latest news supports cuts rather than a new upward breakout.
03-07 00:05	prices, search	–	The current standing claim already occupies the market and latest evidence does not support a fresh upward breakout.

Table 10: GPT-5.6-terra w/ wait harness on breakout news detection for the market "Will no Fed rate cuts happen in 2026?". The agent anticipates payroll release by adaptively scheduling its wake. Out of 6 repeated runs of GPT-5.6-luna and terra, this anticipitation of the payroll release is observed on 1 seed on GPT-5.6-terra w/ wait harness, 2 seeds on GPT-5.6-terra w/ watcher harness, 1 seeds on GPT-5.6-luna w/ watcher harness, and 0 seed on the cron harness.

Wake (UTC)	Sleep until	Claim → outcome	Final thought
03-06 00:00	03-06 13:40	–	Wait for the imminent US employment release, which is more likely than overnight commentary to create a discrete Fed-path breakout.
03-06 13:40	03-06 18:00	**down → covered**	Standing down claim is active; wait while the jobs shock is incorporated.
03-06 13:41			**Breakpoint ↓ -0.075 (0.187 → 0.112): covered by the 13:40 claim on the payrolls report.**
03-06 18:00	03-07 12:00	–	No distinct post-payroll catalyst appears; preserve optionality and resume on the next news cycle.

```python
import json
from datetime import timedelta
from envkit import handover, now, search_news, wait, workspace

POLL = timedelta(minutes=30)
FILE = workspace() / 'fed_seen.json'
seen = set(json.loads(FILE.read_text())) if FILE.exists() else set()
# Broad retrieval, then strict title filter to avoid spending claims on noise.
QUERY = '("Federal Reserve" OR Fed OR FOMC OR Powell OR "interest rate" OR "rate cut" OR "rate hike" OR inflation)'
TERMS = ('fed','federal reserve','fomc','powell','rate cut','rate hike','interest rate','inflation','basis points','monetary policy')
while True:
    t = now()
    page = search_news(q=QUERY, date_from=(t-POLL).strftime('%Y-%m-%dT%H:%M:%SZ'), date_to=t.strftime('%Y-%m-%dT%H:%M:%SZ'))
    fresh=[]
    for h in page.get('results',[]):
        nid=h.get('news_id'); title=h.get('title','')
        if nid not in seen and any(x in title.lower() for x in TERMS):
            fresh.append({'news_id':nid,'title':title,'published':h.get('published','')})
        if nid: seen.add(nid)
    FILE.write_text(json.dumps(sorted(seen)))
    if fresh:
        handover({'market_id':'616902','matched':fresh[:10], 'checked_at':t.isoformat()})
    wait(t+POLL)
```

Figure 9: The watcher program authored by GPT-5.6-luna w/ watcher harness agent for the market "Will no Fed rate cuts happen in 2026?". The program polls the news index every 30 minutes, hands over to the agent only when a fresh headline matches its keyword filter, and persists the set of seen articles across runs. The program is reused and replaced by `fed_watcher2.py` in later during the run, as in the agent trace shown in Table 8.

```
# Task: recommend the AI-subreddit threads that will become popular

You are a long-running automation agent watching a set of AI subreddits (OpenAI,
Anthropic, LocalLLaMA, r/singularity, ...). New posts arrive continuously. Your
job: **recommend the posts whose discussion will turn out large --- as early as
possible**. A recommendation names one post: payload `{"root_id": ...}` to the
environment's `/notify`.

## Objective

A post's popularity is the size of its comment cascade `${horizon_hours}` hours
after posting: `descendants` = the number of comments in that window. It is
hidden until the reveal, then frozen. A post is a **tail post** when its final
`descendants` ≥ ${tail_min_desc}` --- those are the posts you are meant to catch.

Your score, computed at the end of the run, is **time-weighted tail recall**:

    TWR = Σ over tail posts of weight(your first recommendation) / (number of tail posts)
    weight = max(0, 1 − (hours between posting and your recommendation) / ${decay_hours})

- Recommend at the moment of posting: weight 1 --- full credit.
- The weight falls linearly with delay. Concretely: ${decay_examples}.
- Tail posts you never recommend earn 0. Higher TWR is better.

You get at most **${daily_cap} accepted recommendations per rolling 24 h** ---
far fewer than the tail posts that exist --- so every slot spent on a post that
fizzles is a tail post forgone. Roughly 3--4% of posts end tail-sized; the
median post ends with only a handful of comments.

## Budget and rate limit

You have **${budget_usd} total for the entire run**.${domain_caps} LLM calls bill real token
cost against it (${llm_price_table}); ...(truncated)

## Rules (violating requests are rejected free of charge, HTTP 400)

- At most ${daily_cap} accepted recommendations per rolling 24 h window.
- Each post can be recommended at most once.
- Only posts made after the run started and not yet revealed are recommendable:
  pre-run history and already-revealed posts are rejected.

## Data visibility

- A post's static metadata (subreddit, author, title, selftext, url, flair) is
  visible from the moment it is posted.
- **The comment cascade is your early signal.** `get_cascade` returns the reply
  tree *as it has grown so far* --- every comment with `created_utc` ≤ now`, its
  `parent_id` (the tree structure), `author`, and `body`. You watch reply
  velocity and branching build up and judge whether a post is taking off.
- ...(truncated)

## APIs

${APIS}

Every hour you wait before recommending shrinks the credit linearly --- but the
longer you watch a cascade, the better you can tell a trender from a dud that
wastes one of your ${daily_cap} daily slots. Spend your attention where it
changes decisions.
```

Figure 10: The popularity-prediction task instruction, as given to the agent. The instruction states the detailed evaluation metrics, the rules (such as single-recommendation-per-post), and the tool documentations. Tool documentations specific to the tasks and triggering mechanisms are appended after the instruction.