SentinelBench: A Benchmark for Long-Running Monitoring Agents

Matheus Kunzler Maldaner[1] **Adam Fourney**[2] **Amanda Swearngin**[2] **Hussein Mozannar**[2]
Gagan Bansal[2] **Maya Murad**[2] **Rafah Hosn**[2] **Saleema Amershi**[2]

[1]University of Florida [2]Microsoft Research, AI Frontiers

ABSTRACT

AI agents are increasingly asked to carry out work that spans minutes, hours, or longer. Yet the default model of agent behavior is continuous action: issuing tool calls, refreshing pages, searching for alternatives, or otherwise trying to force progress. This is the wrong approach for many long-running tasks, which are better served by a strategy of sustained attention. Instead, agents should monitor an environment, notice when an external event makes progress possible, then respond promptly without wasting resources while waiting. To measure progress on this class of tasks, we introduce `SentinelBench`, an open-source benchmark for time-evolving monitoring tasks.

`SentinelBench` contains 100 tasks across 10 synthetic web environments, including email, calendars, finance, professional networking, and entertainment. Each environment exposes a live web interface and replays a scripted sequence of events, requiring agents to navigate and reason about web pages whose state shifts underfoot. `SentinelBench` measures task completion, reaction time, and resource use, exposing the tradeoff between responsiveness and cost. We report results across three models and two browser-agent harnesses, establishing performance baselines for future comparison and demonstrating how agent design choices can dramatically impact key metrics. Together, these results show that `SentinelBench` distinguishes meaningful differences in agent behavior.

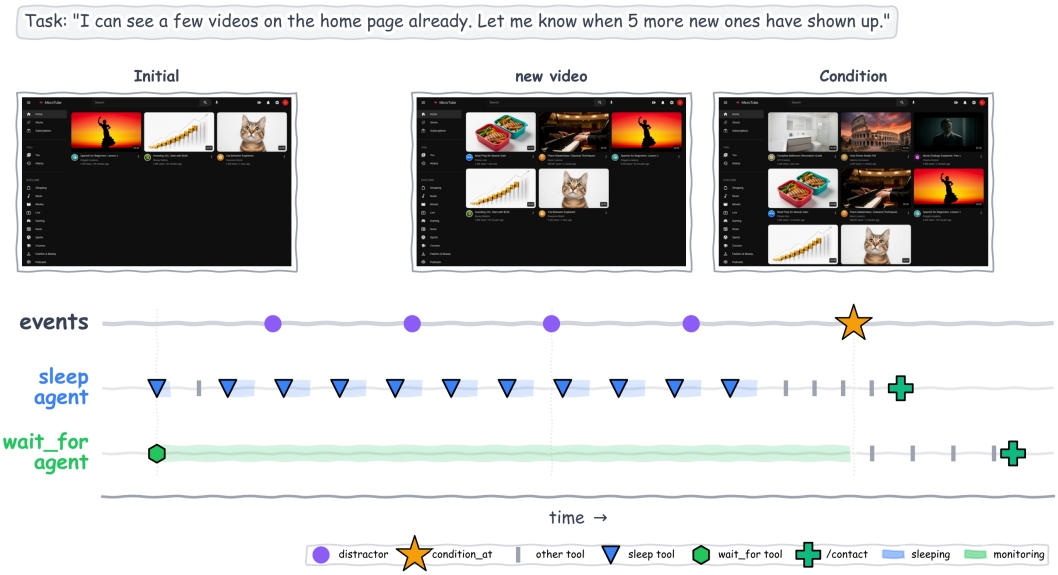


Figure 1: Agent execution timelines in `SentinelBench` for a representative scenario. Each row visualizes one agent strategy over simulated time as events are played back in the simulation (screenshots show the state of the app after selected events). The `sleep` agent spends most of its time executing fixed interval polling while the `wait_for` agent waits for a condition for an extended period and resumes once the environment changes to meet the condition. `SentinelBench` enables systematic investigation of agent implementation choices for measuring progress on time-evolving monitoring tasks.

1 Introduction

Modern AI agents are increasingly capable of long and complex work. METR, for example, has tracked the 50% task-completion time horizon of AI models and found that it has doubled approximately every seven months, increasing from 4 seconds in 2019 to more than 16 hours in 2026 METR (2025); Kwa et al. (2025). In practice, developers already use coding agents for long-running work such as large codebase migrations and comprehensive research projects that take hours to complete Anthropic (2025b).

While impressive, these workflows often embed an implicit assumption: that meaningful state changes in the environment result from the agent's direct actions, such as executing tools or communicating with other agents. In other words, if the environment is not in a state enabling forward progress, the agent needs to move the environment toward such a state. This, of course, is not always a correct model of the world. Many tasks require an agent not to act continuously, but instead to watch and wait until conditions become conducive to task completion. For example, no amount of reloading a webpage will make concert tickets go on sale faster, nor will broadening a web search help when ticket sales are exclusive to a particular site. When agents are compelled to always take action, they waste resources at best, and fail the task outright at worst, especially since task success often drops precipitously as trajectory length increases Sinha et al. (2026).

In previous work, we presented designs for agents that are better able to handle these monitoring-like tasks Mozannar et al. (2025a). We referred to such tasks as *Sentinel Tasks* and previewed a benchmark for measuring agent performance on these workloads Mozannar et al. (2025b). In *this* paper, we formally present and release that benchmark, which we call `SentinelBench` (Figure 2). The full benchmark (code, environments, task scenarios, synthetic catalogs, and data-generation pipeline) is publicly made available at `https://github.com/microsoft/sentinel_environments`.

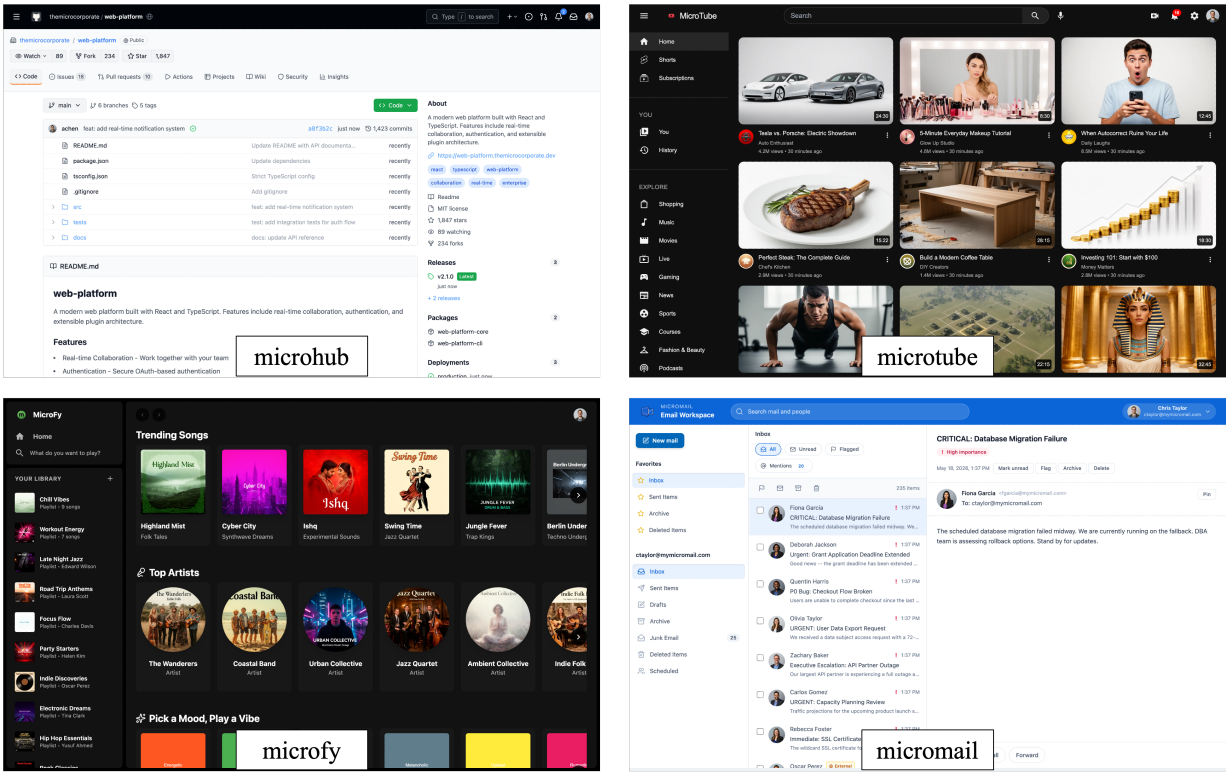


Figure 2: Four `SentinelBench` environments. Each consists of a synthetic environment with a live web interface and a set of monitoring tasks with scripted sequences of events that can be played back to simulate the environment evolving over time.

SentinelBench is a collection of 100 tasks divided across 10 synthetic web environments, including email, calendars, finance, professional networking, entertainment, and others. Each task scenario replays a sequence of events over time and asks the agent to complete work in this live, self-evolving setting. For example, one task asks the agent to "*Keep an eye on the jobs page. If a new role mentions Kubernetes in the requirements, apply to it and let me know.*" The agent succeeds if it notices and applies to the job posting, and then takes steps to contact the user. The benchmark also measures *reaction time* (how soon after the condition was met the agent completed the task) and resource utilization[1] (the number of tokens consumed and generated). For many strategies, such as polling, this sets up a natural tension: poll too frequently and costs soar; poll too slowly, and reaction time rises. By default, each of the 100 tasks is designed to be achievable within 10 minutes, but a *speed_factor* can be applied to stretch tasks to much longer durations making the reaction-cost tradeoff even more stark.

To demonstrate that SentinelBench can meaningfully measure a broad range of outcomes across model scales and agent capabilties, we evaluate three models and two agent harness configurations against SentinelBench. Specifically, we consider GPT-5.4 (low reasoning) to represent a newer agentic frontier model, Qwen 3.5:9b to represent a local agentic model, and GPT-4o to represent an older frontier chat model. We pair each model with a web browsing harness that either includes a classic `sleep(time)` tool, which allows an agent to unconditionally yield for a specified amount of time, or a purpose-built `wait_for(condition, timeout)` tool, which allows agents to heuristically detect and respond to changes in page content. Across these data points, with the default speed_factor, we observe task completion rates ranging from 46% to 75%, while token usage ranges from 70,000 to over 500,000 per task (Table 4). When tasks are extended to 40 minutes, we show that GPT-5.4 agents using `sleep(time)` can cost 10x more ($4.65 vs $0.48 USD), while completing fewer total tasks (56% vs 69%). This demonstrates both that SentinelBench can measure these important metrics and that design choices can dramatically affect the viability of an agent in performing these tasks.

In the remainder of this paper we detail the 10 environments, and characterize the tasks and key elements of their design. We then review the task life cycle, the protocols driving environment events, and the metrics computed by the benchmark. Next, we present detailed findings from the evaluations described above, demonstrating both the need to measure agents on these tasks and that SentinelBench is fit for this purpose. Finally, we compare SentinelBench to related work and discuss future evolutions of this work.

2 SentinelBench Environments & Tasks

In this section, we provide an overview of the SentinelBench environments, their design principles, and the process by which they were populated with data. We also present the tasks that make up the benchmark, and review key elements of their design.

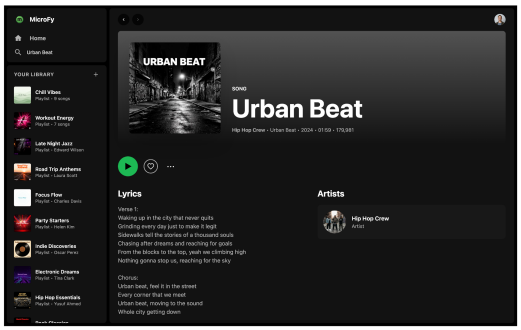
2.1 Overview

SentinelBench includes ten high-fidelity replicas of popular web applications as listed in Table 1. Each environment is rich enough to include common interaction surfaces, and to respond in realistic manners to client actions. For example, the email application has screens for the inbox, other folders, email composition, account settings, and others. Emails are marked as read when viewed, are moved to Trash when deleted, and appear in Sent when composed and mailed. Under the hood, environments remain lightweight, modular and easy to deploy. Each is implemented as a React application served by a Vite development server, backed by a FastAPI backend and a SQLite database. These applications are built with the assumption that they are

[1] While SentinelBench is designed to work with any web browsing agent, token-based metrics require a cooperative agent that self-reports token use at the end of the task.

id: microfy-lyric-subway-absolute-active
environment: microfy
start_page: `HOST_ADDRESS/microfy`
prompt: "Watch the trending feed. When a song drops whose lyrics mention 'subway', like it for me."
condition_at: 329.11 s
kill_at: 630.0 s
timeline_end: 720.0 s
events: t = 0.0, `preload_tracks` (24 tracks)
t = 119.7, `new_track`: track-061
t = 224.4, `new_track`: track-062
t = 329.1, `new_track`: track-013 ★
t = 539.2, `new_track`: track-063
eval_sql:

```
SELECT COUNT(*) >= 1
FROM track_states ts
JOIN tracks t ON t.id = ts.track_id
WHERE LOWER(t.lyrics) LIKE '%subway%'
AND ts.isLiked = 1
```



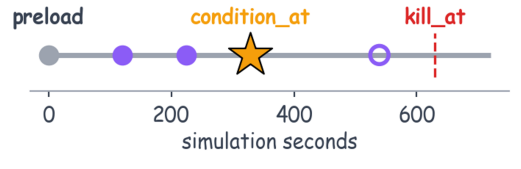


Figure 3: Anatomy of a `SentinelBench` monitoring task: prompt, environment UI at the trigger moment, event timeline, and the underlying scenario JSON that drives the harness (from the MicroFy environment).

serving a single client who is already authenticated. In other words, while they present a convincing facade, they should never be deployed as, or mistaken for, production web applications.

Environment	UI Surfaces
MicroMail	Inbox, folders, attachments, search, flag and move actions
MicroChat	Teams, channels, DMs, reactions, calls, calendars
MicroDin	Feed, connections, jobs, messaging, notifications
MicroFy	Tracks, playlists, artists, lyrics, follow and like actions
MicroGram	Feed, stories, DMs, profiles, activity
MicroHood	Portfolio, market orders, watchlist, news
MicroHub	Repo, issues, PRs, commits, workflows, releases
MicroLendar	Month, week, day views, event CRUD, tasks
MicroScholar	Search, papers, authors, citations, alerts
MicroTube	Feed, player, comments, channels, subscriptions

Table 1: The ten environments in `SentinelBench`. Each is a thin clone of a real consumer product with a multi-screen UI, a typed REST API, and a catalog of synthetic data from which events are sampled.

2.2 Synthetic Data

An email client with no emails or contacts would make for a poor simulation, as would a video-sharing website with no videos. We therefore developed a synthetic data generation pipeline to populate our environments.

The process begins by using generative models[2] to populate a list of 100 user personas and 201 entities, shared across all 10 environments. User personas represent fictitious people. Their profiles include names, biographical and demographic information, interests, social media presence, and other settings. User profiles can also contain multimedia assets such as avatars and profile photos. Figure 4 presents the profile of Chris

[2]See Appendix A for more details about data generation.

Chris Taylor

Product Associate, 29
San Francisco, CA

name:	Chris Taylor
username:	ctaylor
email:	ctaylor@mymicromail.com
age:	29
location:	San Francisco, CA
job_title:	Product Associate
bio:	A product-focused employee trying to build useful things. Juggling features, bugs, and way too many browser tabs.
personality:	adaptable, inquisitive, team-player
interests:	tech, product development, agile methodologies, rock climbing

Per-environment profiles

MicroGram:	847 followers, 312 following, 24 posts
MicroScholar:	Redwood University, PhD Candidate, 1,569 citations
MicroFy:	156 followers, 12 playlists, 247 liked tracks
MicroHub:	pinned repos, achievements, contribution history
MicroDin:	TheMicroCorporate, Enterprise Software, 897 connections

Figure 4: `SentinelBench` includes 100 synthetic user personas. Each persona consists of core demographic and biographical attributes plus per-environment subprofiles that ground the same identity coherently across all 10 simulated applications (MicroGram, MicroScholar, MicroFy, MicroHub, MicroDin shown). Chris Taylor, shown here, is the principal user. I.e., the user whose profile is accessed by the agent.

Taylor, the persona whose accounts are accessed by the agent in the benchmark, i.e., the agent's owner or principal user.

Conversely, entities describe companies, music bands, content creator channels, institutions, news organizations, or other outlets whose content may appear in `SentinelBench` environments. Similar to user profiles, entity profiles can include multimedia assets, such as company logos and banner images.

Having established our universe of personas and entities, we again leverage generative models to produce environment-appropriate content. For example, with a user's profile loaded as context, we produce emails, instant messages, social media posts, playlists, and other records to populate the applications. These elements then enter a content catalog from which tasks are later constructed: some of these items pre-populate the environments, while others become scheduled events that unfold over time. In total, MicroMail ships 260 emails with realistic subjects, bodies, and attachments; MicroChat ships 200 messages across 30 conversations and 10 teams; MicroDin ships 50 posts, 35 connection requests, 25 jobs, and 12 notifications; And so forth. Additionally, catalogs can also include waypoints that allow values to vary continuously over time. This is especially important for environments like MicroHood, whose simulated stock prices are constantly changing. Pre-generating this content upfront, together with personas and entities, ensures that the environments and scenarios exhibit a degree of in-world consistency. Per-environment catalog inventories appear in Appendix B.

2.3 Benchmark Tasks

With applications, personas, entities, and content in hand, we next generate the tasks that form the basis of the `SentinelBench` benchmark. Because the usefulness of the benchmark depends directly on the quality of its tasks, we describe this step in detail. We first introduce the task dimensions that guide benchmark construction, then detail the mechanical process of generation. Finally, we describe how we validated tasks prior to inclusion in the benchmark. We summarize the tasks in Table 3, and we include a full listing of task prompts in Appendix C.

2.3.1 Task Dimensions

We consider two independent axes of task design: *action requirement* and *criterion type*, described below.

Action Requirement: Some meaningful tasks are achievable by passively watching for a target condition to occur, possibly followed by some minimal additional action such as contacting the user. These *passive* tasks are arguably the quintessential monitoring workflow. An example of such a task is "*Watch Lithium Energy (LITH) for me and let me know when it reaches at least $500*." The task is considered complete if the agent affirmatively concludes its monitoring *after* the requisite event has occurred. The task fails if the agent concludes too soon (a false positive) or has not concluded by the time the simulation reaches its end. By default, all tasks are designed to be achievable within 10 minutes. Once 10 minutes is reached, agents are granted an additional 30-second grace period if they are still running, and are then terminated. As we will discuss later, a `speed_factor` parameter (default: 1.0) can be used to scale tasks to arbitrary lengths, but the grace period is not scaled – it remains 30 seconds regardless of whether the task is meant to take 10 minutes, 40 minutes, or something else.

In contrast, *active tasks* require agents to periodically perform an action to gather new information or to respond to in-world events. An example of an active task is "*Let me know the moment Charles Davis sends me a file in any of my chats. You'll need to open each conversation to check—attachments aren't visible from the sidebar.*" This task is considered complete if the agent affirmatively concludes its work and leaves the application database in the correct state. In this example, an attachment-bearing message from Charles Davis must exist and be marked as read. Any other outcome is considered a task failure.

Finally, we recognize that late-occurring false-positive detections may be scored as successful for passive tasks. Likewise, a malicious agent with basic knowledge of the benchmark might unconditionally conclude its monitoring just before the simulation ends, then act as if the event has occurred. For example, with 10-minute simulations, an agent might always declare that the condition has occurred after 9 minutes and 30 seconds. While such behavior would score poorly on reaction time, passive tasks would still be considered successful. To discourage such strategies and to re-balance the benchmark, we include 20 *no-operation (no-op)* tasks. On the surface, these tasks resemble passive or active tasks, but we designed them such that the requisite conditions never occur. These tasks are considered successful only if the simulation ends while the agent is still working or monitoring the environment.

Criterion Type: For some agents and monitoring strategies, small changes in criterion wording can dramatically affect performance. For example, consider the task of alerting a user when a social media post crosses "100 likes." In this case, a single screenshot is sufficient to determine whether the condition has been met. A memoryless agent, or an agent with aggressive context management, can achieve this task without much trouble. However, if the task is instead to alert the user when the post "gets 100 *more* likes," the agent must reason over an initial and an intermediate state. An agent with a long trajectory might easily forget how many likes the post started with. We refer to the first type of task as an *absolute task*, and the latter as a *relative task*. We include both in the benchmark.

	Absolute	**Relative**
Passive	18	20
Active	23	19
No-operation	20	

Table 2: Task counts by action requirement and criterion type.

2.3.2 Task Generation

We first define the overall composition of `SentinelBench`, then generate tasks to fit these parameters. `SentinelBench` consists of 10 tasks per environment, for 100 tasks overall. Of these, 20 are no-operation tasks, with 2 per environment. We divide the remaining 80 tasks approximately evenly between active tasks

Environment	Task prompt	Categorization
MicroHub	"Watch the star count on this repo and let me know when it hits at least 2,000."	passive · absolute
MicroHood	"Watch ChipForge Industries (CHIP) for me and let me know when it reaches at least $520."	passive · absolute
MicroMail	"Keep an eye on the junk folder for me, tell me when 3 more emails end up there."	passive · relative
MicroChat	"Let me know when unread messages show up in 3 more of my chats than right now. Just watch the sidebar, don't open any of the conversations, I don't want people seeing 'seen' on their messages before I've had a chance to read them."	passive · relative
MicroScholar	"I've got a couple of unread alerts already, let me know when 3 more unread alerts come in."	passive · relative
MicroFy	"Watch the trending feed. When a song drops whose lyrics mention 'subway', like it for me."	active · absolute
MicroGram	"Watch my DMs for a new message about a mockup, the word might not show in the preview, so you'll need to open conversations to check."	active · absolute
MicroLendar	"I'm expecting a calendar invite about a trip to Hawaii, the details should be in the event description, so you'll need to open each event to check."	active · absolute
MicroTube	"I already watched a couple videos from my subscriptions earlier. Keep watching that feed and watch 3 more new uploads for me as they appear."	active · relative
MicroDin	"Two new Python-heavy roles should come through today. Apply to both when you see them, you'll need to open the job details to check the requirements."	active · relative

Table 3: One example task from each `SentinelBench` environment, with categorizations for each across *criterion type* and *action requirement*. Appendix C includes the full prompts for the remaining scenarios.

(42) and passive tasks (38), and separately between absolute (41) and relative tasks (39). A full breakdown is presented in table 2, while figure 3 provides an example task for each application.

For each task, we specify an environment, action requirement, criterion type, and condition time sampled uniformly from $[10, 600]$ seconds. We then use an AI coding agent (Claude Code with Opus 4.6), to generate a plausible task scenario. This involves generating a prompt, sampling enough events from the data catalog to cover a 10-minute simulation, scheduling those events so that they respect the chosen condition time, and composing a SQL query used to evaluate agent success at the end of the task, e.g., to ensure the application is in the correct state. Figure 3 shows an example task.

Once generated, we subject each task to a series of deterministic automated checks to identify obvious errors. These checks resemble unit tests and ensure, for example, that passive tasks fail if concluded before the condition time but pass afterward. Likewise, we pair active tasks with their requisite actions (generated by the coding agent alongside the task), then ensure that these actions are actually necessary to pass the tests. As tasks are generated, we use rejection sampling, and regenerate any tasks that fail these basic tests. The resultant tasks are then subjected to additional manual and LLM scrutiny, as described next.

2.3.3 Task Validation

Even when tasks pass automated checks, they may still contain ambiguities or implementation issues. For this reason, we manually inspected every task prompt for clarity. We also attempted most tasks ourselves, in a browser, to assess their feasibility. Additionally we repeatedly ran the full benchmark with our baseline web agents, discussed later, and inspected logs for failures attributable to the tasks or environments rather than to the agents themselves. This log analysis was again performed with the help of coding agents, namely: Claude Code with Opus 4.6 and 4.7.

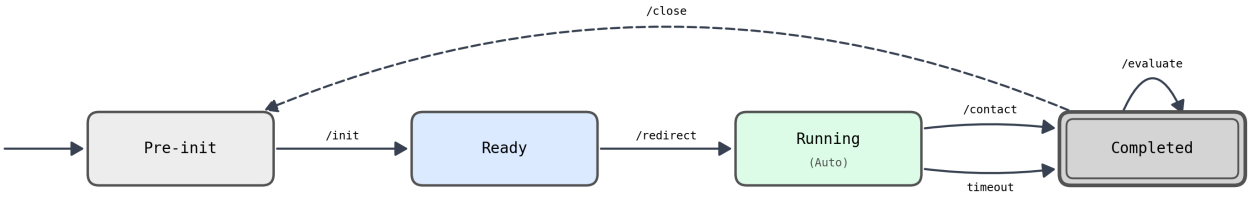


Figure 5: The simulation life cycle for `SentinelBench` environments. The evaluation harness interacts with the server to transition between life cycle states.

For example, one task asked the agent to alert the user if any instant messages mentioned failures with a site's payment-processing system. However, the scenario also included messages discussing problems on the site's checkout page. These messages were meant to serve as unrelated background events, but they created understandable ambiguity: Was the checkout page throwing an HTTP 500 error because the payment-processing system was down? We considered agent resolution of such ambiguity to be out of scope for this benchmark, and replaced the offending messages with others sampled from the content catalog.

In a separate instance, we found that the MicroScholar application did not render correctly when searches yielded zero results. We fixed this error in the environment code.

This process of manual and AI-assisted task validation continued until successive runs surfaced few or no novel issues. While we cannot guarantee that the benchmark is now free of errors, we are confident that the tasks and environments are generally fit for purpose and provide useful signal about how agents perform on time-evolving tasks.

3 Evaluation Protocol and Metrics

`SentinelBench` is designed to present agents with a natural and unopinionated interface to benchmark tasks and environments. From a benchmark user's perspective, the evaluation loop is simple: the user provides a shell command template with placeholders for the task prompt and starting webpage URL; see Figure 3. For each task, the evaluation harness fills in these values, invokes the command, runs the simulation, and computes metrics.

This design is flexible: any web-capable agent that can be invoked from the terminal can be evaluated. The only additional requirement is that the shell process exit when the agent concludes its work. Unlike other related benchmarks or gyms Froger et al. (2026); Zhou et al. (2023); Le Sellier De Chezelles et al. (2024); Drouin et al. (2024); Xie et al. (2024), we do not require a particular agent or browser framework, such as Playwright, nor do we assume any particular observation format or modality, such as screenshots, accessibility trees, or the document object model. However, enabling this level of flexibility places additional requirements on the evaluation harness and metrics, which we describe in the remainder of this section.

3.1 Simulation Life Cycle

As noted earlier, each `SentinelBench` environment includes a FastAPI backend server that serves two purposes: First, it powers the web applications, for example by serving search results and populating social feeds. Second, it works with the evaluation harness to move each scenario through its life cycle. Figure 5 illustrates these state transitions, and we provide additional details below:

Pre-initialization: The `SentinelBench` environment server begins in the pre-initialization state, in which all environment databases are in their initial conditions and no events are queued for playback. The evaluation harness loads a scenario from disk, then calls the server's `/init` endpoint (e.g., `http://localhost:8000/init`) and provides the parameters of the scenario. Parameters include: up-

dates to the initial database state, the agent's starting page, a list of events to play back, a `speed_factor` parameter to control the playback speed, and SQL query to evaluate task success. The server then transitions to the *Ready* state.

Ready: Upon receiving confirmation that the server has entered the Ready state, the evaluation harness invokes the agent with the scenario task prompt. The prompt is augmented with instructions to facilitate evaluation. Specifically, agents are instructed to complete a web form at the end of the task to notify the principal user, thus signaling task completion.

We recognize that agents can take time to start up, especially when browsers or sandboxes are involved. To avoid counting this startup time against the agent, the evaluation harness initially directs the agent to a `/redirect` endpoint (e.g., `http://localhost:8000/redirect`) as the starting webpage URL. When the agent accesses this URL, the server responds with an HTTP 301 redirect to the true task starting page, previously loaded during pre-initialization. This triggers two concurrent actions: the agent's browser follows the redirect, and the server enters the *Running* state.

Running: In the Running state, the server plays back all scheduled events at their corresponding times. These times are computed by dividing each event's default time by an optional `speed_factor` parameter. For example, if an event was originally meant to occur after 30 seconds, then a `speed_factor` of 2.0 schedules it to occur 15 seconds into the simulation (i.e., the simulation runs twice as fast). The `speed_factor` defaults to 1.0, giving a 10 minute task, but can be scaled to any desired interval.

When events occur, they update the application database and are reflected in the interface. If the simulation reaches its natural end, the server initiates a fixed 30-second grace period (i.e., independent of the `speed_factor`) before transitioning to the *Completed* state.

The server also transitions to Completed if the agent accesses the `/contact` form (e.g., `http://localhost:8000/contact`), signaling that the agent believes the task is done. This cancels all remaining scheduled events, even if the transition occurs well before the scenario end time.

Completed: Upon detecting that the server has entered the Completed state, the evaluation harness accesses the `/evaluate` endpoint (e.g., `http://localhost:8000/evaluate`). This endpoint executes the previously saved SQL query against the database to determine whether the task was successful. It also returns timing metadata, such as the number of seconds the simulation spent in the Running state. These values are recorded and used to compute benchmark metrics after the final task.

Finally, the evaluation harness accesses a `/close` endpoint (e.g., `http://localhost:8000/close`), which resets the database states and transitions the server back to the pre-initialization state, ready for the next task.

3.2 Evaluation Metrics

When `SentinelBench` is run, it creates a `results` folder to store execution artifacts. Within this folder, each task receives its own subdirectory, containing the agent's terminal output and the results data returned by the server's `/evaluate` endpoint at the end of the simulation. Because the benchmark is permissive about agents, it cannot assume privileged knowledge of token usage or cost. Agents may therefore produce an optional `costs.json` sidecar file declaring the number of input tokens, output tokens, and tool calls used during the task. When present, this file is used to compute token-related metrics. We describe these and other metrics below.

Success. A task's success criterion depends on its action type, as enumerated earlier. An agent succeeds at a passive task if it accesses the `/contact` form at any point after the target event has occurred. Similarly, an active task succeeds if the database is in the correct state when evaluated, as determined by the task-specific SQL query. Finally, no-operation tasks succeed if the simulation ends without the agent ever opening the `/contact` form.

Task Completion Time and Reaction Time. We define a task's completion time as the total number of seconds the simulation spends in the Running state, prior to the agent accessing the `/contact` endpoint or timing out. Likewise, a scenario's target time is the number of seconds into a simulation at which the target condition (Figure 3, `condition_at`) is met. Reaction time is then the difference between the completion time and the target time. For example, a reaction time of 30 seconds indicates that the agent accessed the contact form 30 seconds after the target condition was first met. Reaction time cannot be meaningfully computed for no-operation tasks, since no event ever meets the requisite task condition.

Input Token Utilization, Output Token Utilization, and Monetary Costs. Another important category of metrics is inference cost. When agents self-report their token utilization, we track the cost of each task in terms of input tokens, output tokens (including reasoning tokens), and the total monetary cost charged by the inference provider. Monetary cost is an especially useful measure because it meaningfully combines input and output tokens into a single value, accounting for the fact that output tokens are often charged at a higher rate than input tokens. It also allows direct comparisons across models (e.g., a weaker model may use more tokens, but is cheaper overall).

Having described task lifecycle, and metrics, we now present an initial set of evaluations.

4 Baseline Evaluations

We used `SentinelBench` to compare three models, and two agent configurations. The purpose of these evaluations is twofold: First, we demonstrate that `SentinelBench` can meaningfully distinguish between models and agents. Second, the evaluations establish baselines from which to compare future benchmarks scores. We detail these experiment conditions below, then present results, and provide some additional error analysis of common agent failures, as detected by the benchmark.

4.1 Experiment Conditions

For our baselines, we pair three multimodal models: GPT-5.4 with 'low' reasoning, Qwen 3.5:9B, and GPT-4o, with a web browsing agent adapted from our work on Magentic-UI Mozannar et al. (2025a). Here, the agent follows a simple tool-calling loop: At each iteration, the model receives a screenshot of the current browser state along with a brief description, then selects from a set of browser tools such as *click*, *type*, *scroll*, and *visit URL*. The loop continues until the model stops making tool calls or invokes a *terminate* tool.

For this evaluation, we compare two variations of the agent toolset. In the baseline condition, the toolset includes a `sleep(time)` tool. When invoked, this tool unconditionally blocks the execution thread for the specified number of seconds before returning control to the agent loop.

In the second condition, agents are equipped with a purpose-built `wait_for(condition, timeout)` tool. This tool takes a natural-language condition and a timeout specifying the maximum amount of time it may block before returning. For example, `wait_for("a new email to arrive", 300)` blocks until a new email is detected or 300 seconds have elapsed.

The implementation of `wait_for` is intentionally simple: When invoked, the tool first captures a textual snapshot of the page to use as a baseline. It then enters a fast loop, capturing a new textual snapshot once per second. On each iteration, the Python `difflib` library computes a unified diff between the baseline and current page state. The resulting diff consists of blocks, each representing a contiguous section of the page that has changed. Blocks seen in previous iterations are removed, on the assumption that these changes have already been evaluated. The remaining blocks are passed to the LLM with a prompt asking whether the new changes satisfy the target condition. The loop breaks if the answer is 'yes', and continues if the answer is 'no'. The `wait_for` tool also reloads the webpage periodically to capture changes on static pages, and

Condition	Overall Success Rate	Action Requirement			Criterion Type	
		No-op	Passive	Active	Absolute	Relative
GPT-5.4, `wait_for`	0.75	0.95	0.92	0.50	0.64	0.77
GPT-5.4, `sleep`	0.68	0.70	0.76	0.60	0.64	0.72
GPT-4o, `wait_for`	0.48	0.95	0.63	0.12	0.50	0.23
GPT-4o, `sleep`	0.46	1.00	0.53	0.14	0.33	0.33
Qwen 3.5:9b, `wait_for`	0.48	0.95	0.50	0.24	0.45	0.28
Qwen 3.5:9b, `sleep`	0.49	0.95	0.39	0.36	0.48	0.28

Table 4: Success rates for three models and two agent configurations (six conditions total). As expected, GPT-5.4 performs better overall than GPT-4o and Qwen 3.5:9B. Likewise, agents configured to use `wait_for` perform about as well as, or better than, agents configured to use `sleep`. We also observe that no-operation tasks are generally easier than passive tasks, and that active tasks are harder still. The one exception is GPT-5.4 with `sleep`, which performs unexpectedly poorly on no-operation tasks.

enforces moderate rate limits to prevent thrashing on fast-changing websites. A full pseudo code listing is provided in Appendix D

4.2 Results

Across all six conditions (three models, each paired with either `wait_for` or `sleep`), we report three complementary measures of performance: how often agents complete tasks (*success rate*), how much they spend doing so (*per-task cost*), and how quickly they respond once a condition is met (*reaction time*). Reporting these together is deliberate: an agent can look strong on success rate alone while being far too expensive or too slow to be practical, and it is precisely this tension between responsiveness and cost that `SentinelBench` is built to expose. We examine each measure in turn below, then show in Section 4.3 how the gaps between conditions widen as tasks are stretched to require longer waits.

4.2.1 Success Rate

With three models and two agent configurations, we evaluate a total of six conditions. We first consider the success rates of each condition, as presented in Table 4. As expected, GPT-5.4, in the 'low' reasoning setting, performs consistently and substantially better (0.75 with `wait_for` and 0.68 with `sleep`) than the much smaller Qwen 3.5:9B model (0.48 and 0.49) and the much older GPT-4o model (0.48 and 0.46).

Also as expected, we find that passive tasks have higher success rates than active tasks in all conditions, suggesting that they are generally easier to accomplish. We also find that both Qwen 3.5:9B and GPT-4o perform substantially better when tasks use absolute prompt phrasing. Somewhat unexpectedly, GPT-5.4 performs slightly, but consistently, better on tasks that use relative phrasing.

Finally, the choice between `sleep` and `wait_for` matters little for Qwen 3.5:9B and GPT-4o. However, when GPT-5.4 uses the `sleep` tool, it fails an additional five no-operation tasks, accounting for most of the 7-point performance difference between these conditions. Inspection of the logs shows that when GPT-5.4 uses the `sleep` tool, it occasionally concludes monitoring early, even while recognizing that the condition has not been met. This is evident from the agent trajectories, with GPT-5.4 concluding tasks with messages such as "*I checked the chats and did not find any conversation where Diana Miller @mentioned you.*"

4.2.2 Per-task Cost

The differences between models, and between `wait_for` and `sleep`, become even clearer when we examine per-task cost. Figure 6 presents these results, computed from API prices posted on May 15, 2026. For GPT

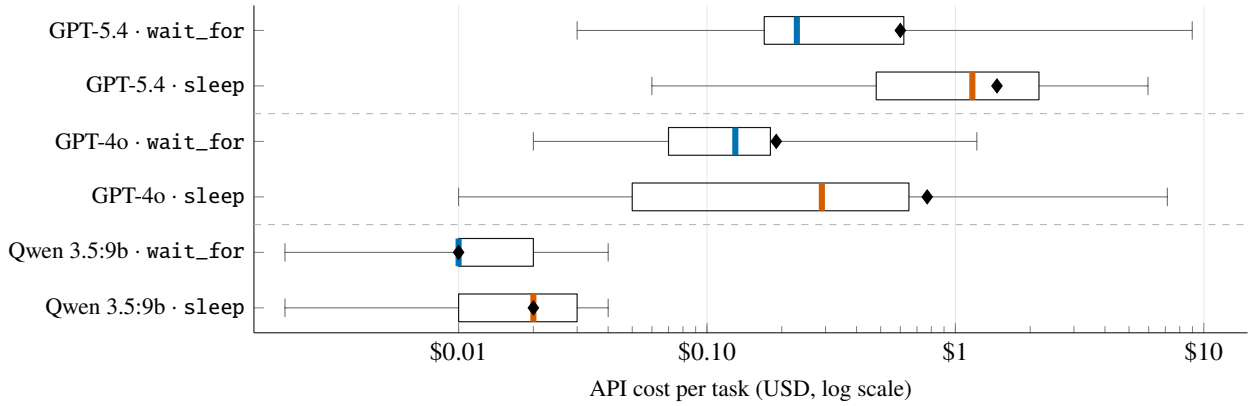


Figure 6: Per-task API cost in USD for each model and tool configuration. Box outlines show the interquartile range, the colored bar inside each box marks the median (blue for `wait_for`, orange for `sleep`), whiskers extend to the observed minimum and maximum, and the diamond marks the mean. Dashed horizontal lines separate model groups. The horizontal axis is logarithmic; Qwen 3.5:9B is roughly two orders of magnitude cheaper than the GPT models, and within every model `sleep` is consistently more expensive than `wait_for`. Means landing outside Q3 (GPT-5.4 `wait_for`, GPT-4o `sleep`) reflect right-skewed distributions with high-cost outliers visible at the upper whisker.

	Number of Tool Calls					
Condition	**Mean**	**Median**	**Q1**	**Q3**	**Min**	**Max**
GPT-5.4, `wait_for`	9.6	6	5	12	1	42
GPT-5.4, `sleep`	20.9	19.5	12.25	28.75	3	47
GPT-4o, `wait_for`	4.9	5	2	6	1	21
GPT-4o, `sleep`	11.9	9.5	3	14	0	50
Qwen 3.5:9b, `wait_for`	13.8	11.5	6	21	1	32
Qwen 3.5:9b, `sleep`	17.0	16	8.25	26	1	33

Table 5: Descriptive statistics of the number of tools calls made in each of the six conditions. `sleep` uses more tool calls, and leads to longer trajectories, in both the mean and median cases.

models, we use prices from `https://developers.openai.com`; for Qwen 3.5:9B, we use prices from `https://openrouter.ai`.

Across all conditions, `sleep` is substantially more expensive than `wait_for`. With GPT-5.4, the median task cost is 5.1× higher with `sleep` than with `wait_for` ($1.17 vs. $0.23). Similarly, `sleep` is about 2.2× and 2.0× as costly as `wait_for` for GPT-4o and Qwen 3.5:9B, respectively ($0.29 vs. $0.13; and $0.02 vs. $0.01). Inspecting the logs, we find that agents configured with `sleep` either sleep for very brief intervals, such as 5–10 seconds at a time, or do not sleep at all. This leads to longer agent trajectories, as reflected in Table 5. For example, GPT-5.4 makes a median of 6 tool calls with `wait_for`, compared with 19.5 with `sleep`. Given that `wait_for` performs as well as or better than `sleep` on task completion, these cost differences suggest little downside to using `wait_for`. More importantly, these findings highlight the importance of including tokens costs as a metric in this benchmark.

4.2.3 Reaction Time

In the previous section, we reported that `wait_for` is substantially cheaper than `sleep`, but these savings could conceivably come at the cost of slower agent reaction times. To address this concern, Table 6 reports

Condition	Reaction Time in Seconds	
	Mean	**Median**
GPT-5.4, `wait_for`	81.4 s	51.7 s
GPT-5.4, `sleep`	73.0 s	42.3 s
GPT-4o, `wait_for`	35.1 s	22.8 s
GPT-4o, `sleep`	59.7 s	48.6 s
Qwen 3.5:9b, `wait_for`	100.6 s	60.1 s
Qwen 3.5:9b, `sleep`	140.9 s	123.8 s

Table 6: Mean and median reaction times for successful tasks across all six conditions. Unsuccessful tasks and no-operation tasks are excluded because reaction time is not well-defined in these cases. Comparisons within each model family are more informative than comparisons across model families, since each inference endpoint may have different token throughput or may be subject to different load. With GPT-5.4, `wait_for` responds 8.4 seconds more slowly on average than `sleep`. In the other conditions, `wait_for` is the faster of the two tools.

the mean and median agent reaction times for successful tasks. Here, we exclude no-operation tasks and failed tasks, because reaction time is poorly defined under these conditions.

Overall, we find a mixed signal. For GPT-5.4, the median reaction time is 9.4 seconds slower when using `wait_for` than when using `sleep` (51.7 seconds vs. 42.3 seconds). However, `wait_for` is nearly twice as fast for GPT-4o and Qwen 3.5:9B (22.8 seconds vs. 48.6 seconds, and 60.1 seconds vs. 123.8 seconds, respectively).

Here, we note that direct comparisons across models are less meaningful because inference times depend on the characteristics of their respective endpoints. For example, Qwen 3.5:9B's reaction times are longest because we hosted this model locally and achieved lower token throughput than the production GPT endpoints.

4.3 Impact of Task Duration (speed factor)

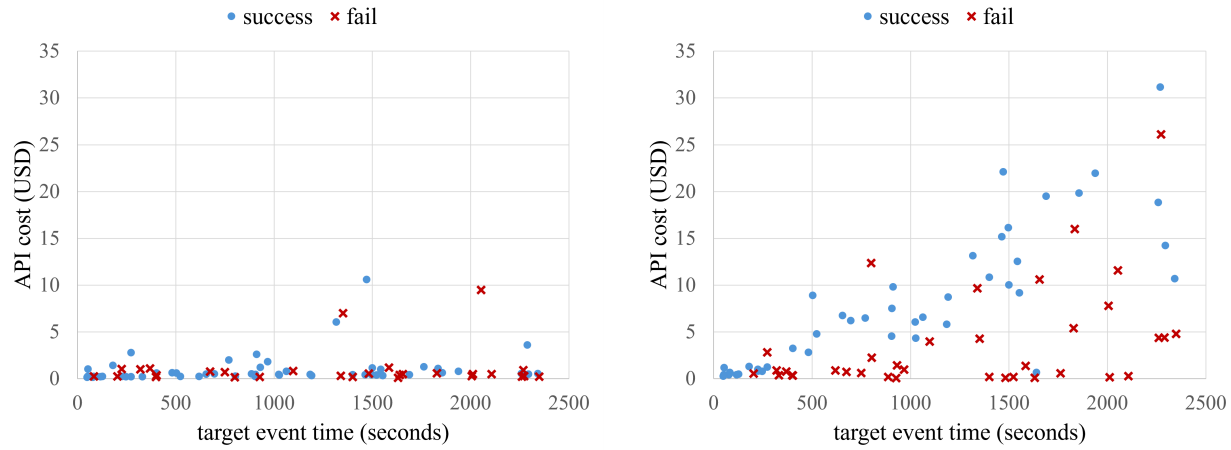
Finally, we are interested in how the `speed_factor` impacts baseline performance. Recall that, by default, `SentinelBench` tasks are designed to be completable within 10 minutes. If an agent has not completed a task within this time, the process is terminated and the results are evaluated. The choice to use a 10-minute period was a compromise: it is long enough to differentiate agents on monitoring tasks, yet short enough that the benchmark can be run in a reasonable amount of time.

However, trends observed in earlier experiments suggest that differences between agents become even more stark as tasks grow longer and agents are required to wait more. To evaluate this, we lower the benchmark `speed_factor` parameter from 1.0 to 0.25, so that tasks may require as long as 40 minutes to complete. We then re-evaluate the best-performing model, GPT-5.4, with both `wait_for` and `sleep`. The results of this experiment are summarized in Table 7.

When tasks require 40 minutes of waiting, the median per-task API cost of `sleep` ($4.65) is 9.7× greater than `wait_for` ($0.48), and 13 fewer tasks are completed successfully overall (56 out of 100, versus 69 out of 100). This story grows even more clear in Figures 7 and 8, which plot API cost, and task completion time, versus target condition time, respectively. Specifically, Figure 7b shows that the cost for a successful task grows steadily when using `sleep`, but remains relatively constant when using `wait_for` (Figure 7a).

Measurement	**`wait_for`**	**`sleep`**
Success Rate	0.69	0.56
Median API Cost (USD)	$0.48	$4.65
Median Reaction Time	54.8 s	38.9 s

Table 7: GPT-5.4 with `wait_for` vs. `sleep` at `speed_factor` = 0.25 (tasks stretched to as long as 40 minutes). In this setting, the agent solves 13 more tasks when using `wait_for` than when using `sleep`, despite the former being 9.7× cheaper.



(a) API cost vs. target event time (GPT-5.4; `wait_for`)

(b) API cost vs. target event time (GPT-5.4; `sleep`)

Figure 7: Per-task API cost as a function of target event time for GPT-5.4 under the `wait_for` and `sleep` tool configurations. Successful tasks appear as blue dots. Failed tasks appear as red 'X's. Here, tasks are scaled to take up to 40 minutes to complete (2400 seconds). When using `wait_for`, costs remain relatively stable and low, with some outliers as high as $10.59. When using `sleep`, costs trend upward with time, especially for successful tasks, with costs as high as $31.15.

Similarly, Figure 8 plots the agent's task completion time against the time at which the necessary event occurred, i.e., the target time. The closer a dot falls to the line $y = x$, the better the agent's reaction time. From this graph, we can clearly observe that most failures in the `sleep` condition (Figure 8b) correspond to negative reaction time, meaning the agent gave up on the task too early.

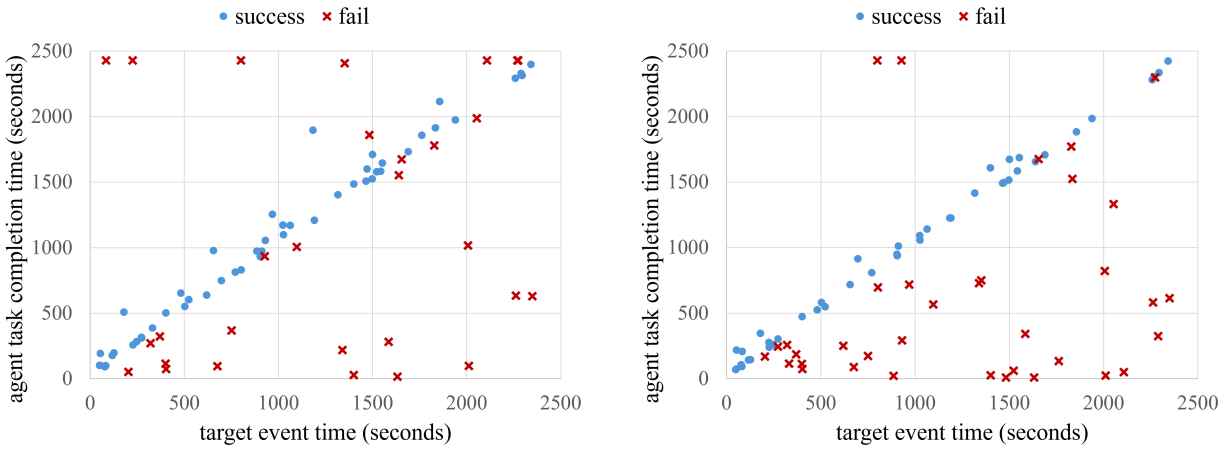
From this we conclude that, when when agents use the same model, their tooling can dramatically impact performance and costs. `SentinelBench` clearly exposes these differences.

5 Discussion and Limitations

In this paper, we introduced `SentinelBench` and demonstrated that it can measure meaningful differences between models or agents. Like any benchmark, it makes deliberate tradeoffs between realism and controlled evaluation, in our case around task duration, and scenario diversity. In this section, we outline these limitations, highlight opportunities for improvement, and discuss future work.

A first limitation of `SentinelBench` is that event timing is artificial. Tasks are built around a 10-minute window, target events are scheduled randomly, and other event timings are selected by an LLM during task generation. While this is sufficient for creating dynamic environments and challenging agents with monitoring tasks, future versions may require realistic timing distributions (e.g., achievable by sampling them from real online systems). As an example, a future model might anticipate how often different page elements update, and allocate resources accordingly. Artificial timings would interfere with such strategies.

Relatedly, `SentinelBench` environments are lightweight facsimiles of popular websites. Prolonged exploration may expose the edges of these environments. We designed tasks to focus on areas of the environments with good feature coverage, but researchers authoring new tasks may need to extend the environments with new features. We also cannot guarantee that the environments are error-free. Because our debugging process was greedy (driven by baseline runs and manual inspection), remaining errors are most likely to occur in trajectories that diverge substantially from the paths we tested.



(a) Task completion time vs. target event time (GPT-5.4; `wait_for`)

(b) Task completion time vs. target event time (GPT-5.4; `sleep`)

Figure 8: Per-task completion time as a function of target event time for GPT-5.4 under the `wait_for` and `sleep` tool configurations. Again, successful tasks appear as blue dots. Failed tasks appear as red 'X's. The closer a successful task falls to the diagonal $y = x$, the better the agent's reaction time to that event. Points below this line are failures resulting from premature termination of the task. Agents are terminated after 2430 seconds, if not already concluded. Points appearing at this y-value indicate failed event detections. In both conditions, the diagonal is prominent, indicating that reaction times are often fast when events are detected. However, `sleep` is prone to terminating too early. Conversely, `wait_for` performs better overall, but is slightly more prone to missing events and thus terminating late.

Several task dimensions also remain underexplored. Although `SentinelBench` includes both absolute and relative task phrasings, most success criteria are objective, involving specific numeric targets, entities, or topics (e.g., "when the repo hits 2,000 stars"). Future tasks could introduce more subjective criteria (e.g., "when any urgent bug reports come in"), requiring agents to make more subjective judgments about the relevance of events to specific criteria. Similarly, most current tasks monitor for persistent conditions, i.e., those that are unlikely to revert once they become true. For instance, once an email or instant message has arrived, it typically remains visible in the conversation history unless later moved or deleted by the user. In the future we may consider tasks that monitor for ephemeral conditions. Here, the target condition may hold only briefly, and a missed opportunity may make the task impossible to complete (e.g., "buy the CHIP stock when it dips below $500").

Finally, `SentinelBench` may also be useful for training agents, not just for evaluating them. However, two pieces are needed to support this use case. First, task generation must scale. This likely means eliminating any remaining manual verification and debugging steps. Second, roll outs will be to be compressed or accelerated, so agents do not actually spend 10 (or 40) minutes on each task. GAIA2 Froger et al. (2026) offers one possible time acceleration strategy, advancing simulation time to the next event whenever the agent sleeps. In `SentinelBench`, this is more complicated because agents often control full web browsers, and these clients often rely on time to verify certificates, pace animations or page updates, or render relative timestamps through JavaScript (e.g., "posted 5 minutes ago".) Compressing simulation time would therefore require all clients and components to synchronize against a common clock.

In summary, `SentinelBench` provides a controlled foundation for studying time-evolving monitoring tasks, but it is only a starting point. Future work should improve event timing realism, broaden task criteria, support ephemeral events, and make the environments more scalable for both evaluation and training.

These extensions would move SentinelBench from a focused benchmark toward a broader testbed for long-running agent behavior.

6 Related Work

Most benchmarks for agentic systems assume reactive environments, where state changes occur only in response to the agent's actions. We review four categories of such benchmarks and identify the measurement gaps that SentinelBench addresses. The closest related work is ARE (Froger et al., 2025), which departs from the reactive paradigm with an event-driven simulation environment where the world evolves independently of the agent. We situate SentinelBench relative to ARE and highlight the additional evaluation capabilities SentinelBench provides.

Long-horizon evaluations. Agents today still struggle with long-running tasks, as shown by METR's time-horizon work (Kwa et al., 2025). Defining the 50%-task-completion time horizon as the amount of human time required to complete tasks that an AI agent can solve with at least 50% reliability, frontier models reached approximately fifty minutes in early 2025, continuing a trend of doubling roughly every seven months. Importantly, this horizon varies substantially by domain, including math, software, and visual computer-use tasks (METR, 2025). Follow-up analysis also finds a per-step hazard rate, showing that task success drops exponentially with task length (Ord, 2025).

RE-Bench (Wijk et al., 2025) and HCAST (Rein et al., 2025) are two foundational benchmarks for long-horizon tasks. Complementary benchmarks, such as (Wang et al., 2026; Motwani et al., 2026; Jang et al., 2026; Garikaparthi, 2026), probe different axes of the long-horizon problem, including failure attribution, reasoning length, and duration estimation. Together, these works motivate SentinelBench, since monitoring tasks often unfold over many minutes or hours, and can require many steps. However, whereas prior work largely measures tasks in which agents act continuously, SentinelBench focuses on tasks where progress sometimes requires the agent to wait until the environment enters a state permitting forward progress.

Web and computer-use benchmarks. A second line of related work introduces benchmarks for evaluating agents that operate web browsers or windowed operating systems. WebArena (Zhou et al., 2023) and VisualWebArena (Koh et al., 2024) include realistic web applications, such as a forum, e-commerce platform, project management site, and content-editing environment, with more than 800 templated tasks. Similarly, WebVoyager (He et al., 2024) evaluates agents on fifteen consumer websites, Mind2Web (Deng et al., 2023) includes tasks across 137 sites, and AssistantBench (Yoran et al., 2024) introduces 214 open-web research tasks that take humans a meaningful amount of time to complete. Building on this foundation, BrowserGym (Le Sellier De Chezelles et al., 2024) combines several of these benchmarks into a single gym-like interface, while AgentBench (Liu et al., 2023) aggregates related tasks into a multi-domain suite. WebGames (Thomas et al., 2025) adds interactive UI puzzles, and ST-WebAgentBench (Levy et al., 2026) evaluates safety and trust dimensions. Finally, looking beyond web applications, OSWorld (Xie et al., 2024) introduces OS-level tasks involving desktop applications. Like the long-horizon benchmarks discussed above, these benchmarks generally evaluate agents on tasks that run to completion in a single uninterrupted execution loop, with little or no waiting required.

Workplace and multi-application simulators. A third line of work introduces benchmarks for multi-application workflows that more closely resemble digital-worker activity. AppWorld (Trivedi et al., 2024) simulates nine applications and roughly one hundred users through API-style tool calls. TheAgentCompany (Xu et al., 2024) simulates a small software company with internal websites and data; on this benchmark, the most competitive agent at release achieved approximately 30% autonomous task completion. OdysseyBench

(Wang et al., 2025) targets office-suite workflows spanning multiple documents and multi-session dialogues. AMA-Bench (Zhao et al., 2026) measures long-horizon *memory* for agentic applications. More recent multi-application benchmarks and simulators (Li et al., 2026; Xiu et al., 2026; Fu et al., 2026; Lu et al., 2026) broaden the scope to more applications, services, and longer tasks. Like `SentinelBench`, these benchmarks either self-host realistic application interfaces or move toward production-grounded tasks (Lu et al., 2026); however, they still score agents on contiguous execution. `SentinelBench` complements this work by evaluating agents not only on how they act, but also on when they act.

Monitoring and scheduled agents. Several products and systems have begun to address the need to run agents periodically, on a schedule, or to monitor an environment until specified conditions are met. In one early instance, Thacker (2024) argued for a "snooze button" that would let an agent go dormant until an optimal time, similar to the `sleep` evaluated earlier. More recently, OpenAI's scheduled-task feature (OpenAI, 2025b) in ChatGPT (OpenAI, 2025a), along with a similar feature in Claude Cowork Anthropic (2026), lets users schedule one-shot or recurring tasks. In `SentinelBench`, however, events can happen at any time, so fixed scheduling approaches naturally degenerate into slow polling. `SentinelBench` can be used directly to measure the effectiveness of such strategies, and you might expect to see slow reaction times.

More closely aligned with the agents considered in this paper, Yutori's Scouts (Yutori, 2025) is a product marketed as "always-on AI agents that monitor the Web for anything you care about." As with `SentinelBench`, users describe scout tasks in natural language, and the service monitors web content on their behalf. Google (Google, 2026) also recently introduced both scheduled-task and monitoring capabilities in web-agent offerings, although the implementation details are not public.

Beyond web agents, Claude Code's `monitor` tool (Anthropic, 2025a) lets Claude react to changes in terminal environments. This tool is conceptually similar to the `wait_for` tool we evaluate in `SentinelBench`, with the key difference being that `monitor` watches files, logs, and scripts for events, whereas `wait_for` watches web application state. More importantly, these systems provide agents, while `SentinelBench` provides a benchmark for measuring their effectiveness.

From scheduled events to environmental monitoring. Across these categories, no benchmark makes waiting the primary task, although ARE Froger et al. (2025) comes closest. ARE introduces a simulation platform in which time advances independently of the agent's actions, allowing scheduled events to fire in much the same way as in `SentinelBench`. Within ARE, the most relevant benchmark is GAIA2 Froger et al. (2026), which consists of 1,120 scenarios across 12 smartphone app settings, including Messages, Contacts, and Shopping. A small subset of these tasks are temporal, testing an agent's ability to reason about and respond to time and events. Although frontier models reach a pass@1 of 42.1% overall (GPT-5, High), they perform poorly on temporal tasks: GPT-5 (High) scores 0.0%, while the best-performing model, Claude 4-Sonnet (thinking), reaches only 8.5%. `SentinelBench` shares ARE's premise that environments can evolve independently of agent actions, but differs in two key ways. First, GAIA2 is built around API access to applications, with a clear notification queue and event pipeline, whereas `SentinelBench` requires agents to monitor natural, messy web pages. Second, GAIA2's temporal tasks often specify temporal constraints or expectations in the task prompt, whereas `SentinelBench` tasks require agents to infer the triggering condition and waiting strategy.

Another close peer to `SentinelBench` is Pare-Bench (Nathani et al., 2026), which evaluates proactive assistance across 143 scenarios. In Pare-Bench, agents are scored on whether they intervene at the right moment for a simulated user. Both Pare-Bench and `SentinelBench` evaluate wait-then-act scenarios, but they differ in their trigger sources and evolution mechanisms: Pare-Bench centers on user-simulator events, while `SentinelBench` uses a broader range of environment events, such as a new song being released or a new paper being published.

In summary, SentinelBench fills an important gap by benchmarking agents on their ability to monitor and respond to changes in web applications without relying on direct API access or dedicated notification channels. Table 8 summarizes this contrast.

Benchmark	Scale	Long-horizon framing
GAIA (Mialon et al., 2024)	466 questions	Single-turn QA, no waiting
WebArena (Zhou et al., 2023)	812 tasks, 4 sites	Multi-step navigation, single session.
WebVoyager (He et al., 2024)	643 tasks, 15 sites	Live-web navigation, single session.
Mind2Web (Deng et al., 2023)	2350 tasks, 137 sites	Multi-step navigation, single session.
AssistantBench (Yoran et al., 2024)	214 tasks, 200+ sites	Auto-verifiable info-seeking, single session.
OSWorld (Xie et al., 2024)	369 tasks	OS-level control, single session.
AppWorld (Trivedi et al., 2024)	750 tasks, 9 apps	Multi-app API chains, single session.
TheAgentCompany (Xu et al., 2024)	175 tasks	Multi-app workflows, single session.
WebGames (Thomas et al., 2025)	50+ tasks	Interactive UI puzzles, single session.
OdysseyBench (Wang et al., 2025)	≈602 tasks	Office workflows, multi-session dialogue.
Pare-Bench (Nathani et al., 2026)	143 tasks, 4 application domains	Asynchronous events as notifications, user simulator events
GAIA2 (Froger et al., 2026)	1,120 tasks, 12 apps	Asynchronous events as notifications, timing for triggers supplied in prompts
SentinelBench (ours)	100 scenarios, 10 envs	Externally-scheduled triggers discovered by polling, configurable-duration waits, plus no-op scenarios.

Table 8: Public benchmarks for evaluating LLM-based agents on web and workplace tasks. SentinelBench is the only benchmark whose tasks require the agent to wait for an externally-scheduled condition while performing periodic state checks.

7 Conclusion

As AI agents take on longer-running tasks, they will increasingly encounter situations in which progress depends not on immediate action, but on recognizing when to wait, monitor, and respond at the right moment. In this paper, we introduced SentinelBench, a benchmark for evaluating agents on time-evolving monitoring tasks across 10 synthetic web environments and 100 task scenarios. SentinelBench measures not only whether agents complete these tasks, but also how quickly they react and how many resources they consume. Our baseline evaluations show that model choice matters, but that harness and tool design can matter just as much: a simple `wait_for` tool substantially reduces cost relative to `sleep`, especially as task duration increases, while maintaining or improving task success in most conditions. To support reproducible evaluation, we release the code, task scenarios, synthetic catalogs, data-generation pipeline, and evaluation protocol at https://github.com/microsoft/sentinel_environments, providing a shared benchmark for studying how agents perform on monitoring tasks. More generally, these results and artifacts highlight the need to evaluate agents under conditions where the world changes independently of their actions, and to build agents that are more patient, efficient, and appropriately responsive.

References

Anthropic. Claude Code tools reference: Monitor tool. Claude Code documentation, 2025a. URL https://code.claude.com/docs/en/tools-reference#monitor-tool. Accessed: 2026-05-27.

Anthropic. Effective context engineering for ai agents. Anthropic Engineering Blog, September 2025b. URL https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents. Accessed: 2026-05-17.

Anthropic. Schedule recurring tasks in Claude Cowork. Anthropic Help Center, April 2026. URL https://support.claude.com/en/articles/13854387-schedule-recurring-tasks-in-claude-cowork. Accessed: 2026-05-15.

Black Forest Labs. FLUX.2 [dev]: Open-weights text-to-image diffusion model. Model card on Hugging Face, 2025. URL https://huggingface.co/black-forest-labs/FLUX.2-dev.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2306.06070.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks? In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 11642–11662. PMLR, 21–27 Jul 2024. URL https://dl.acm.org/doi/abs/10.5555/3692070.3692532.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, et al. Are: Scaling up agent environments and evaluations. *arXiv preprint arXiv:2509.17158*, 2025. URL https://arxiv.org/abs/2509.17158.

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 Ménard, 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 (ICLR)*, 2026. URL https://arxiv.org/abs/2602.11964.

Daocheng Fu, Jianbiao Mei, Rong Wu, Xuemeng Yang, Jia Xu, Ding Wang, Pinlong Cai, Yong Liu, Licheng Wen, and Botian Shi. The agent's first day: Benchmarking learning, exploration, and scheduling in the workplace scenarios. *arXiv preprint arXiv:2601.08173*, 2026. URL https://arxiv.org/abs/2601.08173.

Aniketh Garikaparthi. Can llms perceive time? an empirical investigation. *arXiv preprint arXiv:2604.00010*, 2026. URL https://arxiv.org/abs/2604.00010.

Junmin Gong, Wenxiao Zhao, Sen Wang, Shengyuan Xu, and Jing Guo. ACE-Step: A step towards music generation foundation model. Model card on Hugging Face, 2025. URL https://huggingface.co/ACE-Step/ACE-Step-v1-3.5B.

Google. Gemini Spark: Your 24/7 personal AI agent for productivity. Product page, 2026. URL https://gemini.google/overview/agent/spark/. Accessed: 2026-05-27.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*, 2024. URL https://arxiv.org/abs/2401.13919.

Lawrence Keunho Jang, Jing Yu Koh, Daniel Fried, and Ruslan Salakhutdinov. Odysseys: Benchmarking web agents on realistic long horizon tasks. *arXiv preprint arXiv:2604.24964*, 2026. URL https://arxiv.org/abs/2604.24964.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024. URL https://arxiv.org/abs/2401.13649.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, Ryan Bloom, Thomas Broadley, Haoxing Du, Brian Goodrich, Nikola Jurkovic, Luke Harold Miles, Seraphina Nix, Tao Lin, Neev Parikh, David Rein, Lucas Jun Koba Sato, Hjalmar Wijk, Daniel M. Ziegler, Elizabeth Barnes, and Lawrence Chan. Measuring AI ability to complete long software tasks. *arXiv preprint arXiv:2503.14499*, 2025. URL https://arxiv.org/abs/2503.14499.

Thibault Le Sellier De Chezelles, Maxime Gasse, Alexandre Drouin, Massimo Caccia, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Quentin Cappart, Graham Neubig, Ruslan Salakhutdinov, Nicolas Chapados, and Alexandre Lacoste. The BrowserGym ecosystem for web agent research. *arXiv preprint arXiv:2412.05467*, 2024. URL https://arxiv.org/abs/2412.05467.

Ido Levy, Ben Wiesel, Sami Marreed, Alon Oved, Avi Yaeli, and Segev Shlomov. ST-WebAgentBench: A benchmark for evaluating safety and trustworthiness in web agents. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2410.06703.

Xiangyi Li, Kyoung Whan Choe, Yimin Liu, Xiaokun Chen, Chujun Tao, Bingran You, Wenbo Chen, Zonglin Di, Jiankai Sun, Shenghan Zheng, et al. Clawsbench: Evaluating capability and safety of llm productivity agents in simulated workspaces. *arXiv preprint arXiv:2604.05172*, 2026. URL https://arxiv.org/abs/2604.05172.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. *arXiv preprint arXiv:2308.03688*, 2023. URL https://arxiv.org/abs/2308.03688.

Pengrui Lu, Bingyu Xu, Wenjun Zhang, Shengjia Hua, Xuanjian Gao, Ranxiang Ge, Lyumanshan Ye, Linxuan Wu, Yiran Li2 Junfei Fish Yu, Yibo Zhang11, et al. Alphaeval: Evaluating agents in production. *arXiv preprint arXiv:2604.12162*, 2026. URL https://arxiv.org/abs/2604.12162.

METR. How does time horizon vary across domains? METR blog post, July 2025. URL https://metr.org/blog/2025-07-14-how-does-time-horizon-vary-across-domains/.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2311.12983.

Sumeet Ramesh Motwani, Daniel Nichols, Charles London, Peggy Li, Fabio Pizzati, Acer Blake, Hasan Hammoud, Tavish McDonald, Akshat Naik, Alesia Ivanova, et al. Longcot: Benchmarking long-horizon chain-of-thought reasoning. *arXiv preprint arXiv:2604.14140*, 2026. URL https://arxiv.org/abs/2604.14140.

Hussein Mozannar, Gagan Bansal, Cheng Tan, Adam Fourney, Victor Dibia, Jingya Chen, Jack Gerrits, Tyler Payne, Matheus Kunzler Maldaner, Madeleine Grunde-McLaughlin, Eric Zhu, Griffin Bassman, Jacob Alber, Peter Chang, Ricky Loynd, Friederike Niedtner, Ece Kamar, Maya Murad, Rafah Hosn, and Saleema Amershi. Magentic-UI: Towards human-in-the-loop agentic systems. *arXiv preprint arXiv:2507.22358*, 2025a. URL https://arxiv.org/abs/2507.22358.

Hussein Mozannar, Matheus Kunzler Maldaner, Maya Murad, Jingya Chen, Gagan Bansal, Rafah Hosn, and Adam Fourney. Tell me when: Building agents that can wait, monitor, and act. Microsoft Research blog, October 2025b. URL https://www.microsoft.com/en-us/research/blog/tell-me-when-building-agents-that-can-wait-monitor-and-act/.

Deepak Nathani, Cheng Zhang, Chang Huan, Jiaming Shan, Yinfei Yang, Alkesh Patel, Zhe Gan, William Yang Wang, Michael Saxon, and Xin Eric Wang. Proactive agent research environment: Simulating active users to evaluate proactive assistants. *arXiv preprint arXiv:2604.00842*, 2026. URL https://arxiv.org/abs/2604.00842.

OpenAI. Introducing ChatGPT agent: Bridging research and action. OpenAI blog post, July 2025a. URL https://openai.com/index/introducing-chatgpt-agent/.

OpenAI. Tasks in ChatGPT. OpenAI Help Center, 2025b. URL https://help.openai.com/en/articles/10291617-tasks-in-chatgpt.

Toby Ord. Is there a half-life for the success rates of AI agents?, 2025. URL https://arxiv.org/abs/2505.05115.

David Rein, Joel Becker, Amy Deng, Seraphina Nix, Chris Canal, Daniel O'Connel, Pip Arnott, Ryan Bloom, Thomas Broadley, Katharyn Garcia, Brian Goodrich, Max Hasin, Sami Jawhar, Megan Kinniment, Thomas Kwa, Aron Lajko, Nate Rush, Lucas Jun Koba Sato, Sydney Von Arx, Ben West, Lawrence Chan, and Elizabeth Barnes. HCAST: Human-calibrated autonomy software tasks. *arXiv preprint arXiv:2503.17354*, 2025. URL https://arxiv.org/abs/2503.17354.

Akshit Sinha, Arvindh Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping. The illusion of diminishing returns: Measuring long horizon execution in LLMs. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2509.09677.

Joseph Thacker. Empowering long-running AI agents with timers. Personal blog, May 2024. URL https://josephthacker.com/ai/2024/05/16/empowering-ai-with-timed-tasks.html.

George Thomas, Alex J. Chan, Jikun Kang, Wenqi Wu, Filippos Christianos, Fraser Greenlee, Andy Toulis, and Marvin Purtorab. WebGames: Challenging general-purpose web-browsing AI agents. *arXiv preprint arXiv:2502.18356*, 2025. URL https://arxiv.org/abs/2502.18356.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. *arXiv preprint arXiv:2407.18901*, 2024. URL https://arxiv.org/abs/2407.18901.

Wan-AI Team. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025. URL https://arxiv.org/abs/2503.20314.

Weixuan Wang, Dongge Han, Daniel Madrigal Diaz, Jin Xu, Victor Rühle, and Saravan Rajmohan. OdysseyBench: Evaluating LLM agents on long-horizon complex office application workflows. *arXiv preprint arXiv:2508.09124*, 2025. URL https://arxiv.org/abs/2508.09124.

Xinyu Jessica Wang, Haoyue Bai, Yiyou Sun, Haorui Wang, Shuibai Zhang, Wenjie Hu, Mya Schroder, Bilge Mutlu, Dawn Song, and Robert D Nowak. The long-horizon task mirage? diagnosing where and why agentic systems break. *arXiv preprint arXiv:2604.11978*, 2026. URL https://arxiv.org/abs/2604.11978.

Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Holden Karnofsky, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. RE-Bench: Evaluating frontier AI R&D capabilities of language model agents against human experts. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2411.15114.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024. URL https://arxiv.org/abs/2404.07972.

Zidi Xiu, David Q Sun, Kevin Cheng, Maitrik Patel, Yizhe Zhang, Jiarui Lu, Omar Attia, Raviteja Vemulapalli, Oncel Tuzel, Meng Cao, et al. Astra-bench: Evaluating tool-use agent reasoning and action planning with personal user context. *arXiv preprint arXiv:2603.01357*, 2026. URL https://arxiv.org/abs/2603.01357.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. TheAgentCompany: Benchmarking LLM agents on consequential real-world tasks. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2412.14161.

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. AssistantBench: Can web agents solve realistic and time-consuming tasks? *arXiv preprint arXiv:2407.15711*, 2024. URL https://arxiv.org/abs/2407.15711.

Yutori. Building the proactive, multi-agent architecture powering scouts. Yutori blog post, 2025. URL https://yutori.com/blog/building-the-proactive-multi-agent-architecture-powering-scouts.

Yujie Zhao, Boqin Yuan, Junbo Huang, Haocheng Yuan, Zhongming Yu, Haozhou Xu, Lanxiang Hu, Abhilash Shankarampeta, Zimeng Huang, Wentao Ni, Yuandong Tian, and Jishen Zhao. AMA-Bench: Evaluating long-horizon memory for agentic applications. *arXiv preprint arXiv:2602.22769*, 2026. URL https://arxiv.org/abs/2602.22769.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. URL https://arxiv.org/abs/2307.13854.

A Data generation pipeline

In this section, we detail the pipeline used in `SentinelBench` to generate the synthetic data that populate the environments. We begin by motivating the need for such a process, then describe the procedures used to generate both multimedia and text content.

A.1 Why Synthetic Environments?

Synthetic web environments are useful for evaluating agents for several important reasons, and they are especially important for evaluating agents on monitoring tasks. First, benchmark tasks do not arise from organic user needs. They are synthetically generated to resemble real user requests, but do not correspond to any specific person's immediate need. Directing benchmark traffic to live user-facing websites could place undue strain or costs on those sites, and might conflict with their posted terms of service. Operating in synthetic environments avoids these problems.

More importantly, synthetic environments are necessary for benchmark reproducibility, especially for monitoring tasks where timing is a critical factor. `SentinelBench` needs events to unfold on a controlled schedule. For example, an email might arrive at minute 12, a stock might cross a threshold at minute 40, or a connection request might appear partway through a run and require a response. Live platforms such as Instagram or YouTube cannot provide this level of control, nor can facsimiles that track live data, such as a mock trading platform that follows market movements. In both cases, repeatable evaluation would be impossible. Self-contained clones backed by synthetic data give us a world that we can pause, script, and replay deterministically.

Finally, this same control makes these environments valuable beyond evaluation. The machinery that scripts a test scenario can also generate large volumes of labeled trajectories for *training* monitoring agents, or environments for reinforcement learning.

A.2 Synthetic multimedia content

Within this controlled world, a few design choices shape the data. Every asset is generated rather than scraped from the web, so the benchmark contains no real photos, names, or PII and can be released openly. To avoid introducing brand marks, we also try to keep generated media mostly free of rendered text. All media assets are generated media using open-weight models (Table 9), which keeps the pipeline reproducible on common hardware. Finally, each synthetic persona remains coherent across environments. The same likeness, biography, and interests follow a user from MicroDin to MicroChat to MicroGram, carried by a stable slug that threads through every stage of generation.

Modality	Model	Settings
Images	FLUX.2-dev (Black Forest Labs, 2025)	28 steps, guidance 4.0, bfloat16
Videos	Wan2.2-T2V-14B (Wan-AI Team, 2025)	50 steps, 81 frames, 720p
Audio	ACE-Step-HQ (Gong et al., 2025)	150 steps, 2 min, 44.1 kHz

Table 9: Three open-weights models cover the three modalities (Table 9). Generation runs on an NVIDIA B200 (183 GB VRAM) node with CUDA 12.8. Approximate generation time: 100 images ≈ 15 min; 50 videos ≈ 22 hr; 100 music tracks ≈ 5 hr.

Figure 9 shows the flow for the visual and audio assets (the structured text catalogs are generated separately; see Section A.3). Generation begins with an environment-independent *core*: a shared pool of LLM-generated personas and entities (companies, bands, channels, institutions) that is created once and reused across all ten environments to help with world coherence. For each environment we then write a *prompt* for every image,

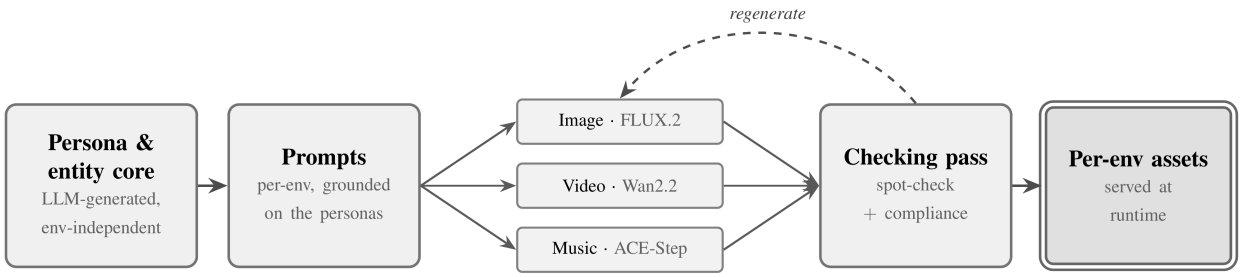


Figure 9: The media-generation pipeline. An environment-independent core of LLM-generated personas and entities is shared across all ten environments. For each environment, a prompt is written for every image, video, and audio asset, grounded on the relevant personas. The prompts fan out to three generative models (image, video, audio). Generated media passes a checking pass (spot-check and compliance), which loops failures back for regeneration, before the per-environment media assets are assembled. Structured text catalogs are generated separately (Section A.3).

video, and audio asset, grounded on the relevant personas and entities so the asset fits that environment and that identity; each prompt is tagged with a slug recording which persona, post, or team it belongs to, and prompts that could plausibly render text also carry anti-text guardrails. The prompts *fan out to the appropriate generative model*, (images with FLUX.2-dev, video with Wan2.2, audio with ACE-Step). Generated media then pass an additional check that routes unusable assets (garbled text, accidental brand marks) back for regeneration. The result is the set of *per-environment media assets* the apps serve at runtime. Table 10 details the number of assets, by type, included in `SentinelBench`.

A.3 Synthetic textual content

All catalog text for the synthetic personas, including chat messages, emails, captions, post bodies, calendar event descriptions, and paper abstracts, is generated separately using ChatGPT 5.4. Avatars and banners are seeded from structured personas drawn from a 100-user synthetic identity table, with fields for name, age, gender, race, location, job title, biography, personality, and interests. Each persona's slug is reused across environment catalogs, so a single user retains a coherent identity across environments, including the same face, biography, and interests. A representative avatar prompt is shown below.

> *"create a professional headshot for this person: name: Chris Taylor, age: 29, gender: male, race: White, location: San Francisco, CA, jobTitle: Product Associate, bio: A product-focused employee trying to build useful things. Juggling features, bugs, and way too many browser tabs., personality: adaptable, inquisitive, team-player, interests: tech, product development, agile methodologies, rock climbing"*

Environment	Description	Generated assets
Users	Synthetic profiles	100 avatars, 100 banners
MicroDin	Professional network	91 company logos, 91 banners, 20 posts
MicroMail	Email	25 document attachments
MicroTube	Video platform	19 channels, 50 videos, 50 thumbnails
MicroFy	Music streaming	100 tracks with covers
MicroGram	Photo sharing	300 posts, 50 stories
MicroChat	Team messaging	10 team icons

Table 10: Asset counts per environment. The four text-only environments (MicroHub, MicroHood, MicroLendar, MicroScholar) use structured JSONL data with no AI-generated media.

Every prompt that could plausibly produce readable text includes explicit anti-text directives. Video prompts repeat "no text no words no letters" verbatim on each line. Audio prompts describe genre and mood without referencing real artists. Representative prompts follow:

MicroTube video (Wan2.2-T2V-14B):

> *"Sleek electric car racing on a futuristic track, dynamic camera angles, cinematic lighting, high-speed motion blur, professional automotive footage, no text no words no letters. 16:9 landscape."*

MicroGram post (FLUX.2-dev):

> *"Data science workshop presentation, whiteboard with ML diagrams, engaged audience, tech conference setting, square format 1:1, natural lighting, social media aesthetic."*

MicroFy track (ACE-Step-HQ):

> *"indie folk, acoustic guitar, warm vocals, gentle strumming, summer vibes, mellow and peaceful"*

MicroChat team icon (FLUX.2-dev, abstract logo):

> *"Minimal vector icon for Engineering (Software Development): simplified circuit trace + code brackets "<>" motif, flat geometric monoline shapes with rounded corners, lots of negative space, pure white background, single accent color #5B5FC7 (no gradients), centered icon, no text, no shadows, crisp edges."*

B Per-environment details

This appendix gives an overview of each of the 10 environments (Figures 10-19). For each environment we show the two most representative populated views side by side, with the real-world analog, the catalog scale, and the supported end-to-end actions in the caption.

B.1 MicroMail — Email

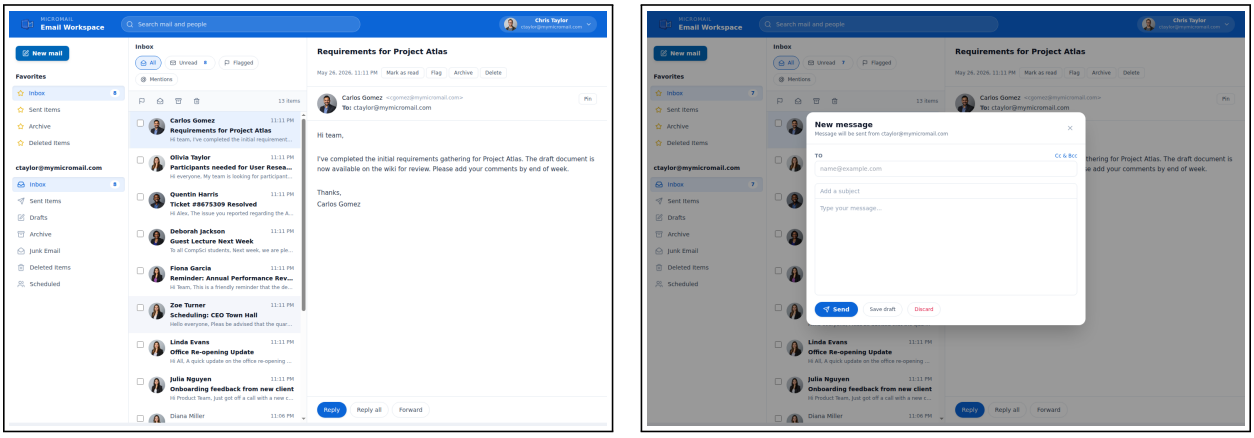


Figure 10: MicroMail, modeled on Gmail/Outlook. Left: the populated inbox with an email open in the reading pane, one of seven folders (Inbox, Sent, Drafts, Archive, Junk, Deleted, Scheduled) over a catalog of 260 emails with realistic subjects and bodies plus 25 attachments. Right: the compose surface (a new-message modal over the inbox). Supported end-to-end actions include mark-read, flag, move-to-folder, delete, restore, and client-side search; scenario events inject new emails over the run.

B.2 MicroChat — Team messaging

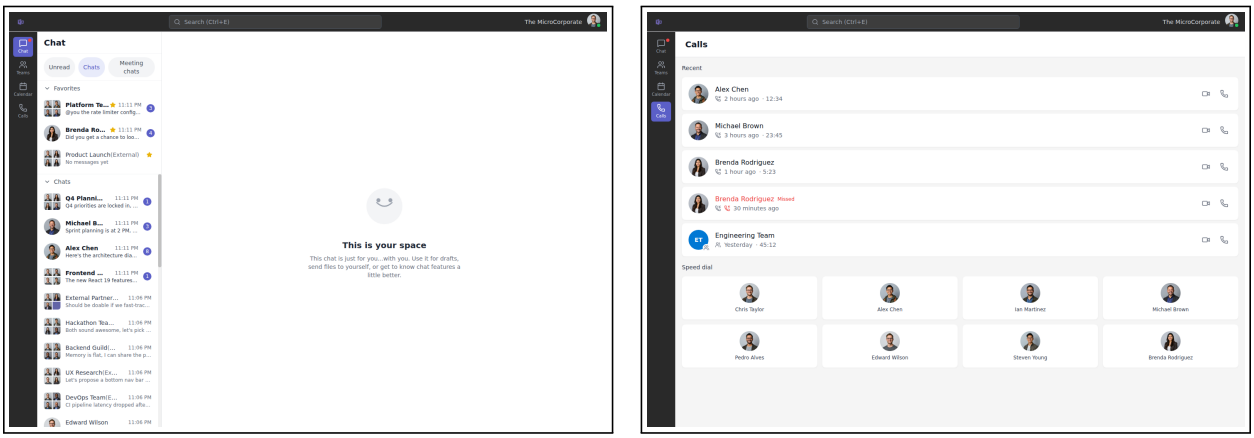


Figure 11: MicroChat, modeled on Slack/Teams. Left: the conversation list, with collapsible favorites and channels, unread-count badges, and presence indicators, backed by 200 messages across 30 conversations and 10 teams. Right: the calls tab, showing recent and missed calls and a speed-dial grid (20 calls in the catalog). Supported actions include mark-read, react, mute, and pin; events inject new messages and calls.

B.3 MicroDin — Professional networking

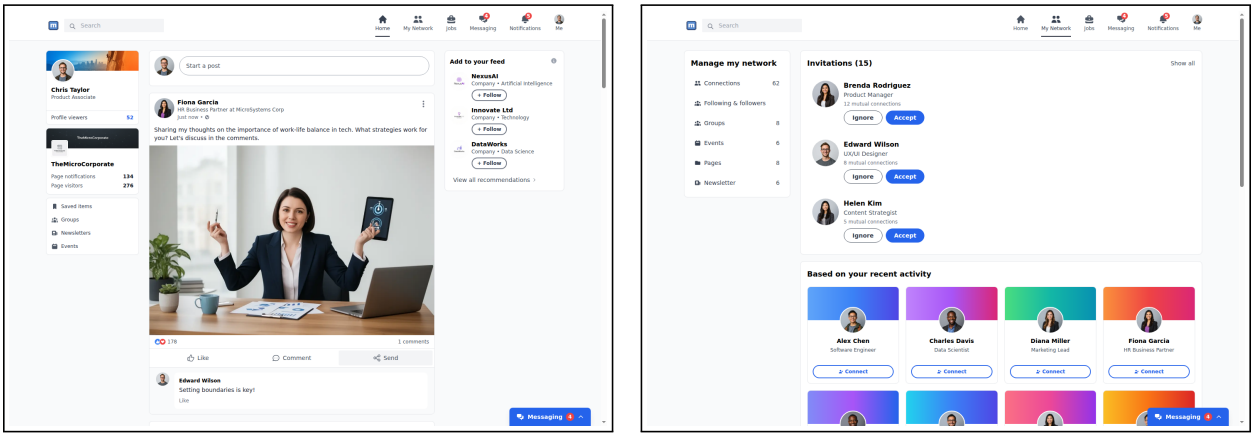


Figure 12: MicroDin, modeled on LinkedIn. Left: the home feed. Right: the My Network page, with pending invitations and connection suggestions. Catalog: 50 posts, 35 connections (some pending), 25 job listings, and 12 notifications. Supported end-to-end actions include liking a post, accepting or rejecting a connection, applying to a job, and marking a notification read; events inject new posts, jobs, connection requests, and notifications.

B.4 MicroFy — Music streaming

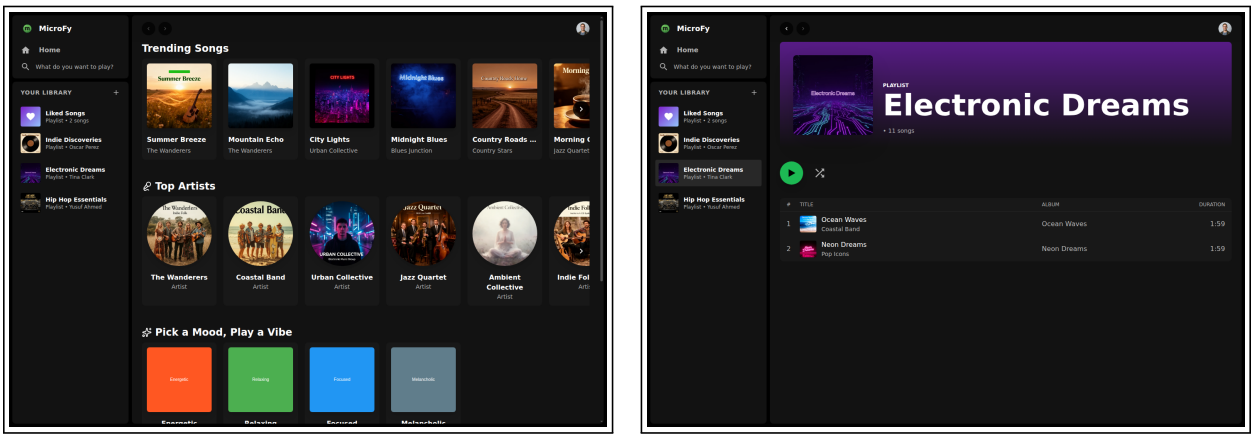


Figure 13: MicroFy, modeled on Spotify. Left: the music library track grid. Right: an artist/playlist page. Catalog: 100 tracks across 20 playlists, each with full lyrics in JSON and AI-generated cover art. Supported actions: like/unlike, create playlist, add to playlist, follow artist; events inject new releases and increment play and follower counts.

B.5 MicroGram — Photo sharing

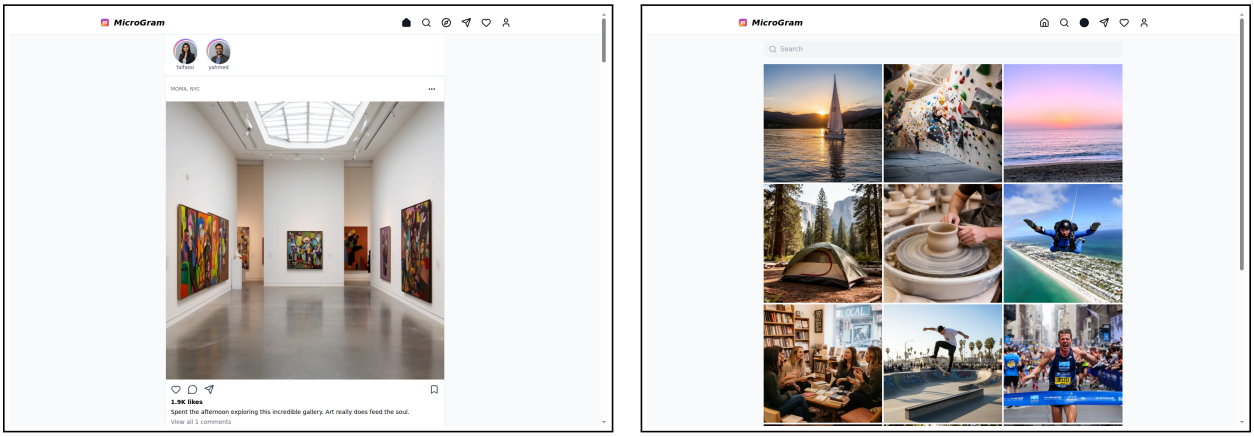


Figure 14: MicroGram, modeled on Instagram. Left: a post in the feed. Right: the photo grid on a profile. Catalog: 300 posts with hashtagged captions, 50 stories, 35 activity items, and 6 DM conversations, all images AI-generated. Supported actions: like, save, follow, mark story viewed, mark activity read; events inject posts, stories, likes, and follows.

B.6 MicroHood — Stock trading

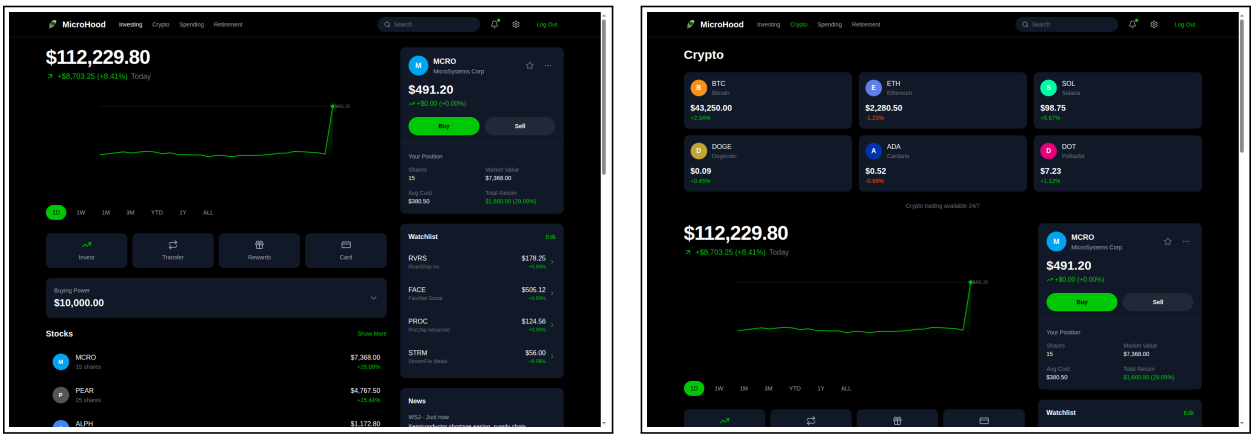


Figure 15: MicroHood, modeled on Robinhood. Left: the portfolio dashboard with its value chart. Right: the market view, with crypto and equity cards, a portfolio value chart, and a Buy/Sell order panel for the selected stock. Catalog: 50 stocks with realistic price, change, and percent fields, a 4-item watchlist, and 20 news items. Prices interpolate between authored waypoints in simulation time, so portfolio value drifts continuously; supported actions include placing market or limit buy/sell orders and editing the watchlist.

B.7 MicroHub — Code hosting

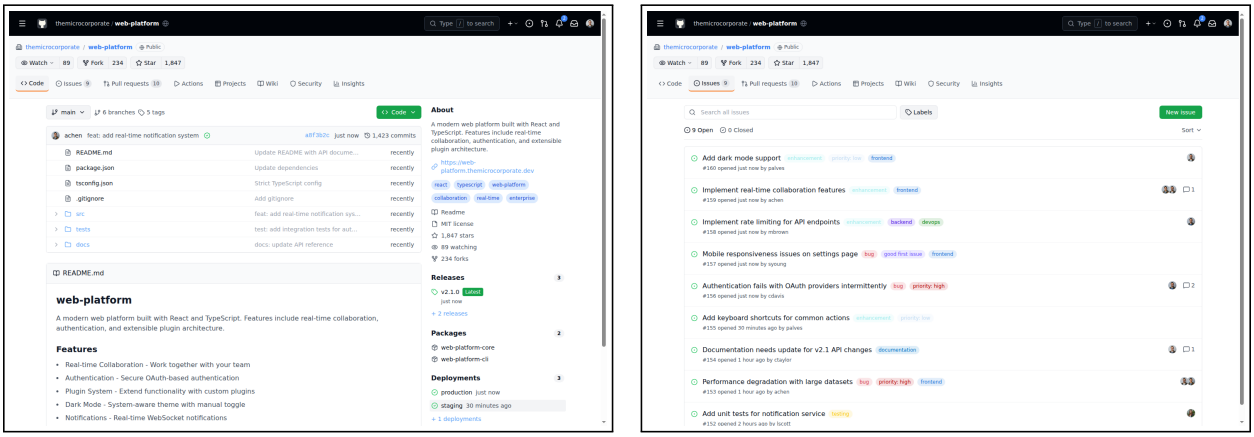


Figure 16: MicroHub, modeled on GitHub. Left: the repository overview. Right: the issues list. Catalog: one repository (with star, fork, and watch counts), 20 issues, 12 pull requests, 25 commits, 26 files, and 7 workflow runs. Supported actions: create issue, merge PR, star, fork, comment; events inject new issues, PRs, commits, and workflow runs, and the star count interpolates between waypoints.

B.8 MicroLendar — Calendar

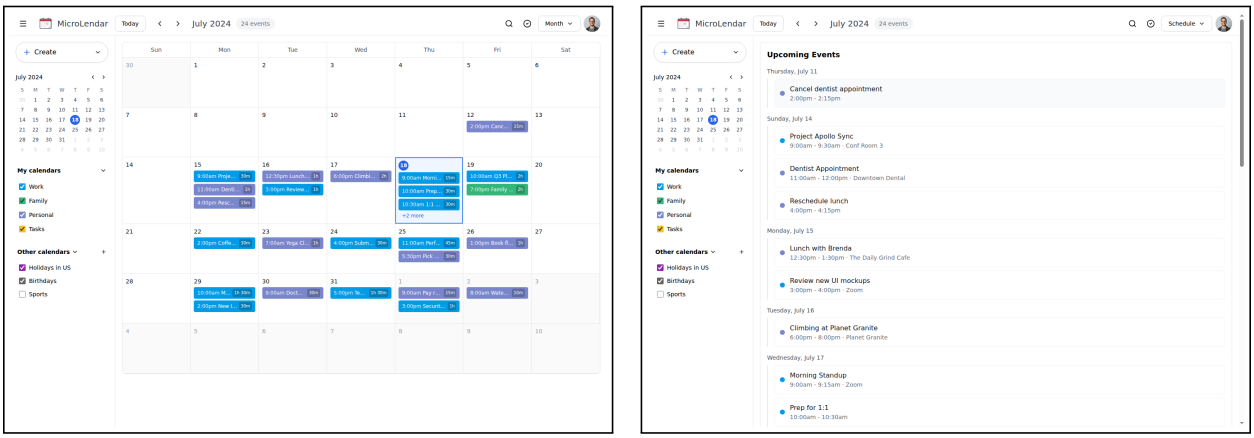


Figure 17: MicroLendar, modeled on Google Calendar. Left: the month grid. Right: the schedule/agenda view. Catalog: 57 events with realistic titles, times, locations, and attendees, plus a task list. Supported actions: create, edit, delete, and drag-to-reschedule events, and create or toggle tasks. The calendar surface is fixed after preload (no time-evolution).

B.9 MicroScholar — Academic search

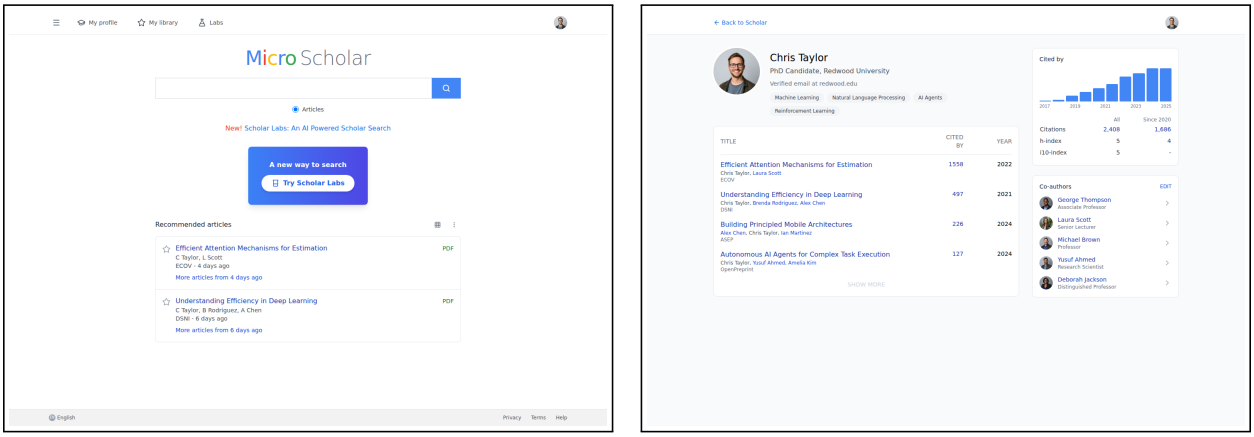


Figure 18: MicroScholar, modeled on Google Scholar. Left: a search results list. Right: an author profile with citation history. Catalog: 100 papers with realistic titles, authors, years, and citation counts, 35 coauthor relationships, and 15 alerts. Supported actions: search, save/unsave, mark alert read, and export citations in five styles; events inject new papers and alerts and can increment citation counts.

B.10 MicroTube — Video streaming

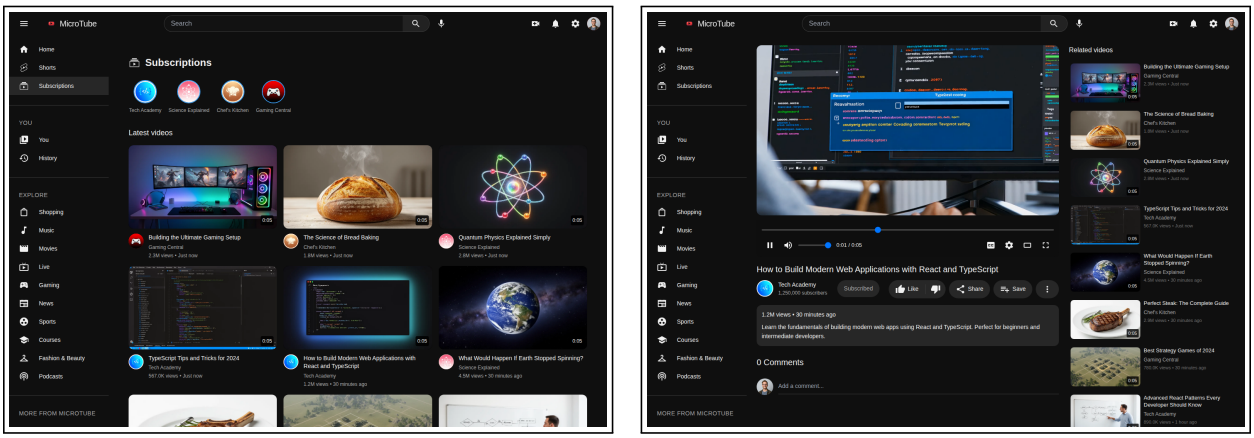


Figure 19: MicroTube, modeled on YouTube. Left: the subscriptions feed. Right: the video player with comments. Catalog: 50 videos, 20 channels, 7 comments, and 10 notifications. Supported actions: like or dislike, save to playlist, comment, subscribe, mark notification read; events inject new videos, comments, and notifications, and view and subscriber counts drift.

C SentinelBench Tasks

Table 11 presents a catalog of the `SentinelBench` tasks, consisting of 100 tasks (10 per environment) with 20 no-operation tasks. Each task has a categorization along the axes of *Action Requirement* (passive, active, or no-op) and *Criterion Type* (absolute, or relative).

Table 11: All 100 `SentinelBench` task prompts, grouped by environment and sorted within each group by action requirement and criterion type.

Task prompt	Categorization
MicroChat	
"Keep an eye on my calls tab and let me know when I've missed at least 3 calls."	passive · absolute
"Keep an eye on my chats and let me know when I have 10 unread messages."	passive · absolute
"I've already missed a couple of calls this morning. Let me know when 3 more missed calls come in."	passive · relative
"Let me know when unread messages show up in 3 more of my chats than right now. Just watch the sidebar—don't open any of the conversations, I don't want people seeing 'seen' on their messages before I've had a chance to read them."	passive · relative
"Let me know the moment someone sends me a new file in any of my chats. Attachments aren't visible from the sidebar, so you'll have to keep an eye on the various chats. I want to be sure I don't miss any important files that come in."	active · absolute
"Keep an eye on my chat with Alex Chen. If he reports any problems with payment processing, read the message to get the full scope of the problem, then let me know right away."	active · absolute
"I'm getting a lot of @mentions today. As new messages come in, read them in full, then let me know as soon as I have 3 more with mentions."	active · relative
"I've already got a couple of urgent pings this morning. Let me know when at least 3 more urgent messages arrive. Note: urgent messages are only marked as such inside the conversation, not in the sidebar."	active · relative
"Let me know the moment Charles Davis sends me a file in any of my chats. You'll need to open each conversation to check—attachments aren't visible from the sidebar."	no-op
"Let me know when Diana Miller @mentions me in any of my chats. Note: you'll need to open conversations to see which messages are mentions."	no-op

continued on next page

Table 11 continued from previous page

Task prompt	Categorization
MicroDin	
"Keep an eye on my network page and tell me when I have at least 15 pending connection requests."	passive · absolute
"Keep an eye on the jobs page and tell me when there are 8 or more job listings."	passive · absolute
"Ping me once at least 3 new unread notifications come in."	passive · relative
"Keep the feed open and tell me once at least 5 new posts have shown up since I left."	passive · relative
"Watch the jobs page. If a new role shows up that requires API documentation experience, apply to it for me."	active · absolute
"Keep an eye on the jobs page. If a new role mentions Kubernetes in the requirements, apply to it and let me know."	active · absolute
"Two new Python-heavy roles should come through today. Apply to both when you see them—you'll need to open the job details to check the requirements."	active · relative
"If any new jobs pop up that ask for React experience, apply to at least two of them for me. You'll need to open each job listing to check the requirements."	active · relative
"If any new jobs pop up that ask for COBOL experience, apply to at least two of them for me. You'll need to open each job listing to check the requirements."	no-op
"Keep an eye on the jobs page and tell me when there are 50 or more job listings."	no-op
MicroFy	
"MicroFy is rolling out playlist updates today. Once you've seen at least 3 new playlists appear in the sidebar, let me know."	passive · relative
"Follow the next 4 artists to appear in the trending feed. To follow an artist, you must open their Artist page."	active · absolute
"Keep checking for new trending songs. Like the next 4 new songs to appear."	active · absolute
"Watch the trending feed. When a song drops whose lyrics mention 'subway', like it for me."	active · absolute
"Keep checking for new songs in trending. Play the next 3 new songs to appear."	active · absolute
"I'm collecting songs that mention 'golden' in their lyrics. Watch for new trending songs that fit—when two more come in, play both to confirm they match."	active · relative
"I'm watching for a new blues track. When one of the new trending songs has 'whiskey' in the lyrics, like it."	active · relative
"Keep an eye on the trending feed today—once at least 5 new tracks have dropped since I logged in, save them all to my Liked Songs."	active · relative
"I've been hearing buzz that Synthwave Dreams has a release dropping today. Watch the trending feed—when one of their tracks shows up, open their Artist page and follow them."	no-op
"Watch the trending feed. When a song drops whose lyrics mention 'tangerine', like it for me."	no-op
MicroGram	
"Let me know when any post in my feed gets 4 or more comments."	passive · absolute
"Keep an eye on the story bar. Let me know when 3 new stories appear."	passive · absolute
"I've already got some notifications in my activity feed—keep an eye on it and let me know when 4 more show up."	passive · relative
"I already have a bunch of posts in my feed—let me know when 5 more new posts show up."	passive · relative
"Watch my DMs for a new message about a mockup—the word might not show in the preview, so you'll need to open conversations to check."	active · absolute
"For the next 3 new creators to appear in my feed, open their profiles and follow them."	active · absolute
"Keep an eye on my feed and like the next 3 new posts that show up."	active · absolute
"I've noticed @ahall has been commenting on posts in my feed—keep checking new posts as they come in, and let me know when 3 more posts show up that have a comment from @ahall (you'll need to open each post to see who commented)."	active · relative
"I've noticed @dmiller has been commenting on posts in my feed—keep checking new posts as they come in, and let me know when 3 more posts show up that have a comment from @dmiller (you'll need to open each post to see who commented)."	no-op
"Watch my DMs for a new message about a merger—the word might not show in the preview, so you'll need to open conversations to check."	no-op
MicroHood	
"Keep an eye on my portfolio and tell me when it reaches $108,837."	passive · absolute
"Watch ChipForge Industries (CHIP) for me and let me know when it reaches at least $520."	passive · absolute

continued on next page

Table 11 continued from previous page

Task prompt	Categorization
"Keep an eye on the news feed in MicroHood and let me know when 3 new articles have appeared since I opened the page."	passive · relative
"Watch my portfolio value and let me know when it's up 5% from where it started."	passive · relative
"Keep an eye on DRNE for me. If it hits $42.20 or higher, place a market order to buy 2 shares and let me know once it's done."	active · absolute
"Watch MCRO for me. If it goes above $480, sell 5 shares at market price and let me know."	active · absolute
"Watch Voltaic Motors (VOLT) for me. If it drops more than 10% from its current price, buy 3 shares at market price and let me know."	active · relative
"Keep an eye on Neos Pharmaceuticals (NEOS). If it goes up 15% or more from where it is now, sell all 12 of my shares to lock in the gains."	active · relative
"Keep an eye on AgriTech Innovations (AGRI) for me. If it hits $500 or higher, place a market order to buy 5 shares and let me know once it's done."	no-op
"Watch Lithium Energy (LITH) for me and let me know when it reaches at least $500."	no-op
MicroHub	
"Keep an eye on the Issues tab and tell me when there are 8 open issues."	passive · absolute
"Watch the star count on this repo and let me know when it hits at least 2,000."	passive · absolute
"I already have a few open issues. Let me know when 4 more get filed."	passive · relative
"The repo is getting some attention lately. Watch the star count and tell me when it goes up by 200 from where it is now."	passive · relative
"Watch for new issues. If one comes in that mentions compliance in its description, leave a comment on it and let me know."	active · absolute
"Keep an eye on new issues. If one shows up that mentions TOTP as a supported 2FA method, leave a comment on it and let me know."	active · absolute
"One of the existing issues has detailed steps to reproduce the bug. Watch for new issues, and let me know if another one comes in that also includes steps to reproduce—you'll need to read the issue descriptions to find it."	active · relative
"I saw a bug report earlier about the UI being unresponsive. Keep an eye on new issues and let me know if another one comes in that also mentions the UI being unresponsive—you'll need to read through the issue descriptions to check."	active · relative
"Watch for new issues. If one comes in that mentions telemetry in its description, leave a comment on it and let me know."	no-op
"Watch the star count on this repo and let me know when it hits at least 100,000."	no-op
MicroLendar	
"Keep an eye on my calendar and tell me when it reaches 21 or more events this month."	passive · absolute
"Keep an eye on my tasks and tell me when the list reaches at least 10 items."	passive · absolute
"I have a few personal events on my calendar already—keep an eye out and let me know when 2 more personal events get added."	passive · relative
"I already have a few things scheduled for today—let me know when 3 more events get added to today's schedule."	passive · relative
"I already have a few work events on my calendar. Tell me when 4 more work events get added in the next two weeks."	passive · relative
"I'm expecting a calendar invite about a trip to Hawaii—the details should be in the event description, so you'll need to open each event to check."	active · absolute
"I'm waiting for a meeting to be scheduled about the notifications feature—the specifics should be in the event description, not the title, so you'll need to open each event to check."	active · absolute
"I have an event coming up that mentions analytics in its description—let me know if another event gets added that also mentions analytics in its description, since you'll need to open each event to check."	active · relative
"I'm expecting a calendar invite about a trip to Iceland—the details should be in the event description, so you'll need to open each event to check."	no-op
"Keep an eye on my calendar and tell me when it reaches 100 or more events this month."	no-op
MicroMail	
"Let me know when I get an email from Kevin Lee."	passive · absolute
"Tell me when I have 10 or more unread emails."	passive · absolute

continued on next page

Table 11 continued from previous page

Task prompt	Categorization
"Let me know when 5 new emails appear in my inbox (not counting any unread emails already there)."	passive · relative
"Keep an eye on the junk folder for me, tell me when 3 more emails end up there."	passive · relative
"Watch for an email with the mobile app mockups attached."	active · absolute
"I'm expecting an email about platform scalability. Let me know as soon as it arrives. Note that 'platform scalability' might not be obvious from the subject, so you'll need to read through the emails."	active · absolute
"I got an email earlier mentioning a December deadline. Let me know if any new email arrives that also mentions the month of December. Note: December might only appear in the email body, so you should check there as well."	active · relative
"I'm expecting a few more emails where I'm CC'd. Let me know when 3 more come in."	active · relative
"Watch for an email with the press release attached."	no-op
"I'm expecting an important email from Lena Whitford. Let me know as soon as it arrives."	no-op
MicroScholar	
"Keep an eye on my alerts and let me know if one comes in about large language models."	passive · absolute
"I'm expecting the paper 'Autonomous AI Agents for Complex Task Execution' to be indexed soon. Please let me know as soon as it shows up in the search results."	passive · absolute
"I've got a couple of unread alerts already—let me know when 3 more unread alerts come in."	passive · relative
"I already have a bunch of papers recommended to me—keep an eye out and let me know when 5 more show up."	passive · relative
"I need to cite the paper 'Autonomous AI Agents for Complex Task Execution', but it may not have been indexed yet. Keep searching for it, and let me know as soon as you have the MLA citation."	active · absolute
"I'm looking for a paper that mentions 'downstream tasks' in its abstract—you'll need to read through the abstracts of new papers as they come in. When one does, save it to my library, then let me know."	active · absolute
"I'm tracking papers that mention 'unstructured environments'. I've already saved one to my library. Wait for, and save, 2 more as they are published."	active · relative
"Some papers by Deborah Jackson should be getting indexed soon—save each one to your library as it appears, and let me know once you've saved 3 of her papers."	active · relative
"Keep an eye on my alerts and let me know if one comes in about cold fusion."	no-op
"I'm expecting the paper 'Quantum Gravity in Discrete Spacetime' to be indexed soon. Please let me know as soon as it shows up in the search results."	no-op
MicroTube	
"Watch my notifications for me and let me know when I have at least 8."	passive · absolute
"Keep the Subscriptions feed open and tell me when there are 3 new videos from channels I already follow."	passive · absolute
"I already have a few notifications. Let me know when 4 more come in."	passive · relative
"I can see a few videos on the home page already. Let me know when 5 more new ones have shown up."	passive · relative
"A new Civilization game was just released—watch for new comments and let me know when you spot any mentioning Civilization."	active · absolute
"Keep an eye on Science Explained for me. If they post a new video, open it and like it."	active · absolute
"I saw a few comments on the gaming videos earlier. Keep checking and let me know when 3 more comments have been posted on them."	active · relative
"I already watched a couple videos from my subscriptions earlier. Keep watching that feed and watch 3 more new uploads for me as they appear."	active · relative
"I heard someone left a comment about Minecraft on one of the gaming videos—watch for new comments and let me know when you spot it."	no-op
"Keep an eye on Cinema Central for me. If they post a new video, open it and like it."	no-op

D THE `WAIT_FOR` TOOL

Section 3 describes the `wait_for(condition, timeout)` tool at a high level. Algorithm 1 fills in the details that the prose elides: the adaptive intervals that govern how often the LLM is consulted and how often the page is reloaded, the deduplication of diff blocks the LLM has already seen, and the forced final check that runs after the timeout expires.

Algorithm 1 The `wait_for(condition, timeout)` tool. The agent supplies a natural-language *condition* and a maximum blocking *timeout* (seconds); the tool returns a flag indicating whether the condition was met, along with the LLM's explanation.

function WAITFOR(*condition*, *timeout*)
  $start \leftarrow$ NOW; ▷ timestamp when WAITFOR was called
  $base \leftarrow$ PAGEMARKDOWN ▷ baseline snapshot of the page
  $reloadInt \leftarrow 180$; ▷ adaptive reload interval
  $nextReload \leftarrow start + reloadInt$ ▷ schedule the next reload
  $checkInt \leftarrow 10$; ▷ adaptive LLM-call rate limit
  $lastLLMCheck \leftarrow -\infty$; ▷ time when the LLM was last called
  $shownDiffs \leftarrow \emptyset$; ▷ Diffs already checked by the LLM
  while NOW $- start < timeout$ **do**
    if NOW $\geq nextReload$ **then** ▷ periodic reload guards against stale static pages
      RELOAD
      $reloadInt \leftarrow \min(2 \cdot reloadInt, 480)$
      $nextReload \leftarrow$ NOW $+ reloadInt$
    end if
    $cur \leftarrow$ PAGEMARKDOWN
    $blocks \leftarrow$ UNIFIEDDIFFBLOCKS($base$, cur) ▷ contiguous changed regions
    $newDiffs \leftarrow blocks \setminus shownDiffs$ ▷ drop blocks already evaluated by the LLM
    $due \leftarrow (newDiffs \neq \emptyset) \wedge$ (NOW $- lastLLMCheck \geq checkInt$)
    if due **then**
      (met, why) $\leftarrow$ ASKLLM($condition$, $newDiffs$)
      $lastLLMCheck \leftarrow$ NOW;
      $shownDiffs \leftarrow shownDiffs \cup newDiffs$
      if met **then**
        return (TRUE, why)
      end if
      $checkInt \leftarrow \min(2 \cdot checkInt, 60)$ ▷ back off on busy pages
    end if
    SLEEP(1)
  end while

  RELOAD ▷ do one final check after timeout with all diffs
  $cur \leftarrow$ PAGEMARKDOWN
  $blocks \leftarrow$ UNIFIEDDIFFBLOCKS($base$, cur)
  if $blocks = \emptyset$ **then**
    return (FALSE, "no changes detected before the timeout")
  end if
  (met, why) $\leftarrow$ ASKLLM($condition$, $blocks$)
  return (met, why)
end function