AutoPILOT: A Visual Agent Harness for Real-Time Gameplay

Mark Zhao[1,2*†], Pengcheng Wang[1*], Rui Yang[1*], Jiarui Yao[1], Xirui Jiang[1], Yuxi Chen[1], Bingxuan Li[1]
Hanyang Chen[1], Ke Yang[1], Manling Li[3], Tong Zhang[1], Huan Zhang[1]
[1]University of Illinois Urbana-Champaign, [2]Georgia Institute of Technology, [3]Northwestern University
https://autopilot-harness.github.io/

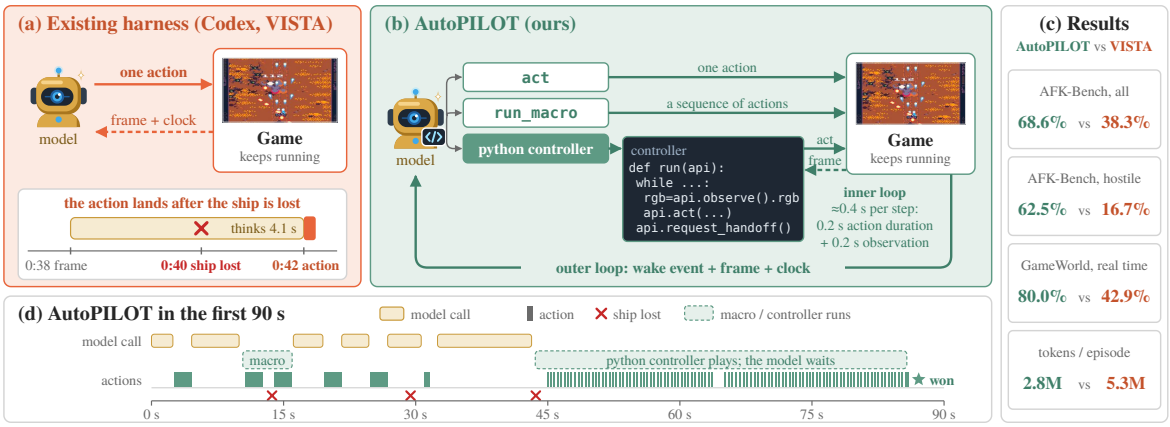

Figure 1 Existing harnesses vs. AutoPILOT on Praxis Fighter X in real time. (a) Existing harness acts once per model call, so the action lands after the ship is lost. Given the same restarts as AutoPILOT, it keeps losing. (b) AutoPILOT can also start a macro or a model-written controller that keeps playing until a wake event. (c) Real-time results with GPT-6.1 Sol on AFK-Bench and GameWorld [28]. (d) The first 90 s of AutoPILOT: after three losses, a controller takes over and wins at 1:26.

Abstract

Multimodal foundation models can play video games well when the game waits for them, but often fail when it keeps running. We find that much of this gap lies in the harness, which decides what the model sees, what it remembers, and how its outputs become actions over time. Existing harnesses such as VISTA improve the first two, yet turn each model call into at most one action, so the agent acts only as often as its model thinks. We introduce AutoPILOT, which changes the third: in each model call, the model can send a primitive action, start an open-loop macro, or write a closed-loop controller that keeps playing from the screen without further model calls until it hands control back. To evaluate agents in real-time games, we build AFK-Bench, 120 diverse browser games grouped by how they respond to inaction and playable in real time or with the clock paused. With GPT-6.1 Sol under the same budgets, AutoPILOT reaches 68.6% real-time success against 38.3% for VISTA, and on GameWorld it keeps 80.0% in real time while VISTA drops from 84.1% to 42.9%. Its gains hold across closed and open-source models, and it reaches VISTA's final success rate at a fifth of its API cost. Our results suggest that how a harness turns model outputs into actions over time is a central yet underexplored part of agent design, and we hope AutoPILOT and AFK-Bench will serve as a foundation for agents in a world that does not wait.

1 Introduction

Multimodal foundation models can now play video games [34, 36, 37, 44], operate graphical interfaces [7, 24, 30, 47, 52], and solve visual puzzles [3, 18] from pixels. Their success depends heavily on the *harness*,

*Equal contribution. †Work done as an intern at UIUC. Emails: zzhao661@gatech.edu, ry21@illinois.edu

Method	Observation	No game-specific training	One harness for many games	Real-time evaluation	Macro actions	Model-written controller	Env. type	# Envs
Voyager [41]	game state	✓	✗	–	✓	✓	Minecraft	1
Code as Policies [20]	perception API	✓	✗	–	✓	✓	robotics	3
Cradle [37]	screen	✓	✓	✗	✓	✗	PC games, apps	10
Game-TARS [44]	screen	✗	✓	✗	✗	✗	video games	7
HLA [22]	text	✓	✗	✓	✓	✗	Overcooked	1
DPT-Agent [57]	text	✓	✗	✓	✓	∼	Overcooked	1
AgileThinker [45]	text	✓	✓	✓	✗	✗	grid games	3
Lumine [36]	screen	✗	✓	✓	✓	✗	3D action games	4
TORSI [29]	text	✓	✗	✓	∼	✗	games	6
Realtime-Bench [38]	screen	✓	✓	✓	✗	✗	web, games	50
AsyncLLM [48]	frames, text	✓	✓	✓	✗	✗	QA, video, ViZDoom	6
AutoHarness [23]	text	✓	✗	✗	✗	✓	text games	145
VISTA [11]	screen	✓	✓	✗	✗	✗	games, puzzles	70
AutoPILOT (ours)	screen	✓	✓	✓	✓	✓	browser games	154

Table 1 Comparison with game agents and harnesses. ∼ partly (DPT-Agent uses rules for a hand-built state machine, and TORSI makes multi-step plans in 2 of 6 games). – not reported. Real-time: the game runs during model calls. Macro actions: several actions per model call. # Envs: distinct games, apps or benchmarks (ours: 120 in AFK-Bench, 34 in GameWorld).

the system around the model [46]: with the model unchanged, a better harness can raise success substantially [49], and VISTA [11] took an off-the-shelf model to a perfect score on ARC-AGI-3 [3]. A harness decides what the model sees and remembers, and how its outputs become actions over time. Prior work has mostly refined the first two [11, 12, 37, 49, 59]. The third is usually left implicit, following the standard *single-step* agent loop [53] in which one model call yields at most one action and the environment is assumed to wait for it, as in ARC-AGI-3 or game benchmarks with settings that pause during inference [54, 56].

When the world keeps running, this default becomes a bottleneck: the agent can act only as often as its model thinks, and on outdated frames (Fig. 1a). Letting the game clock run lowers VISTA's success rate on GameWorld [28] from 84.1% to 42.9%, a gap also reported by Realtime-Bench [38] and for text-state agents [29]. Improving what the model sees and remembers does not remove it, as VISTA performs on par with the plain Codex harness in real time despite its visual memory and inspection tools (38.3% *vs.* 38.9%). Nor does more reasoning, which adds 4–7 points to VISTA at high effort. Much of the bottleneck lies in how the harness turns model outputs into actions.

We introduce **AutoPILOT**, a visual agent harness that changes the third aspect by letting one model call cover many environment steps. Like a pilot with an autopilot, the model delegates routine control and takes over when called back. Each call chooses one of three execution modes (Fig. 1b). (1) A *primitive action* is sent directly to the game. (2) An *open-loop macro* is a sequence of actions written by the model. (3) A *closed-loop controller* is a short Python program that the model writes at run time. It reads the screen and keeps acting without further model calls. Control returns to the model on a *wake event*: the macro or controller finishes, requests a handoff, or meets a screen it was not written for. Both the model and its controllers see screenshots, not the game state.

To evaluate agents in real time, we introduce **AFK-Bench** (*Away From Keyboard*), 120 browser games evenly divided into three types by what happens while the agent is idle. In *hostile real-time environments*, inaction risks failure. *Benign real-time environments* keep changing but do not punish inaction. *Turn-based environments* advance only when the agent acts. A lost game restarts within fixed budgets of model calls, game time and environment steps. Every game can be played in real time or paused, and the difference measures what real time costs the agent.

With the same model (GPT-6.1 Sol) and budgets, AutoPILOT succeeds in 68.6% of real-time episodes on AFK-Bench, against 38.3% for VISTA, and in 62.5% *vs.* 16.7% of hostile real-time ones. Running the clock costs AutoPILOT 4.5 points on AFK-Bench and 2.9 on GameWorld, against 17.8 and 41.2 for VISTA, and AutoPILOT leads even with the clock paused (73.1% *vs.* 56.1%), because one model call can execute a whole plan. The gains hold with Claude Opus 5.5, three open-source models and one more closed model, and AutoPILOT uses 47% fewer tokens and reaches VISTA's final success rate at 0.2× its API cost (Fig. 4d).

(a) Open-loop macro Trichromat

```
# model, 1:36: "I'll send
# the nine moves directly."
# 9 keys, each held 0.15 s
run_macro([
  'ArrowDown', 'ArrowLeft',
  'ArrowLeft', 'ArrowDown',
  'ArrowRight','ArrowRight',
  'ArrowLeft', 'ArrowDown',
  'ArrowRight'])
# worker starts one every
# 0.36 s without reading the
# screen, then wakes the
# model: "macro done"
```

(b) Closed-loop controller Praxis Fighter X

```
def run(api):
 import numpy as np
 for i in range(240):
  o=api.observe(); a=o.rgb
  # Blue hull pixels near the bottom identify current horizontal position.
  m=(a[560:680,:,2]>100)&(a[560:680,:,0]<130)&(a[560:680,:,1]>70)
  yy,xx=np.where(m); x=float(np.median(xx)) if len(xx)>10 else 640
  # Bright magenta/red projectiles, excluding the side HUD.
  b=(a[:,:,0]>235)&(a[:,:,1]<115)&(a[:,:,2]>35)
  b[:300]=False; b[:, :100]=False; b[:,1180:]=False
  by,bx=np.where(b)
  near=(abs(bx-x)<150)&(by>390)
  if np.any(near):
   threat=float(np.mean(bx[near]))
   direction='ArrowRight' if threat<x else 'ArrowLeft'
  else:
   direction='ArrowRight' if (i//22)%2==0 else 'ArrowLeft'
  if x<220: direction='ArrowRight'
  if x>1060: direction='ArrowLeft'
  api.act({'action':'press_keys','keys':['x',direction],'duration':0.18})
 api.request_handoff('Short firing/dodging trial finished; inspect score and lives.')
```

Figure 2 Examples of a macro and a controller in real-time episodes of GPT-6.1 Sol. (a) A macro in the turn-based game *Trichromat*, compacted from the action log. Key presses start about every 0.36 s: each key is held for 0.15 s, a frame is captured after release (about 0.14 s), and the next key follows (about 0.06 s). (b) A controller in the hostile real-time game *Praxis Fighter X*, verbatim: it locates the ship and nearby bullets by color, dodges while firing, and hands control back when its loop ends. It played 103 steps until the win in Fig. 1d.

Our contributions are:

- **Acting over time as harness design.** We treat how model outputs become actions over time as a design choice of the harness, and propose AutoPILOT, in which one model call can start a model-written macro or controller (Sec. 3).
- **AFK-Bench.** We introduce AFK-Bench, 120 browser games measuring what real time costs an agent (Sec. 4).
- **Detailed analysis.** AutoPILOT outperforms VISTA in real time at lower cost and, on AFK-Bench, even with the clock paused. We analyze how its components, budgets, reasoning effort and generation speed matter (Sec. 5).

2 Related Work

Tab. 1 compares AutoPILOT with representative agents and harnesses, and Sec. A gives an extended discussion.

Harnesses for visual and game agents. Most agent harnesses follow ReAct [53], taking one action per reasoning step. Later work changes what the model sees or which tools it may call [21, 49]. On ARC-AGI-3 [3], VISTA [11] lifts the same models to perfect scores through visual memory and re-inspection of past frames. OpenVLHarness [59] adds visual tools and multimodal memory for static visual reasoning. Embodied agent frameworks and harnesses let general vision-language models act in simulated environments and control robots [6, 19, 50]. Computer-use agents [30, 47] and game agents [13, 36, 37, 44] also act on screenshots. Most game agents, however, pause the game while the model generates, as do many game benchmarks [54] and the Lite setting of VideoGameBench [56]. These harnesses improve what the model sees or remembers, but assume that the world waits for it.

Evaluation in real time. Latency is a major reason why vision-language models fail at real-time games [56], and recent benchmarks keep the environment running during inference [16, 38, 40, 45]. Closest to ours, The World Won't Wait [29] compares paused and live clocks and finds that a strong but slow model wins only when the game is paused. Its agents, however, read structured text states.

Its TORSI method still calls a model for every decision, with a slow teacher guiding a fast student. AutoPILOT acts on pixels and instead changes what one model call can do.

Acting between model calls. Temporally extended actions date back to options [35]. Language models also write programs that serve as policies or skills [17, 20, 41, 42], but these programs mostly run on structured state in environments that wait. AutoHarness [23] refines a code policy with environment feedback and resets offline, per game and for turn-based text games. Real-time agents pair a slow model with a fast layer: a reactive thread [45], concurrent inference coroutines [48], action chunks [4], options held until a timer or repetition check fires [15], a scripted executor over fixed macro actions [22], rule-based reflexes with latency charged to the simulation clock [58], or reflexes and policy trees compiled in advance [5, 9]. Asynchronous function calling lets a returning call interrupt the model [10]. The most similar mechanism is DPT-Agent [57], in which the model writes condition rules at run time that steer a hand-built state machine over the symbolic state of one game. In these systems, the fast path is mostly another model inference, code fixed in advance, or rules for a game-specific executor. AsyncLLM can rewrite its coroutines, but only between evaluation rounds. AutoPILOT instead *delegates* acting to controllers that the model writes from pixels during the episode, including their perception, and *interrupts* them through wake events.

3 AutoPILOT

AutoPILOT is developed on VISTA [11], which is built on the Codex harness, with additional tools for running macros and controllers in the background and wake events that return control to the model. As in VISTA, the model sees rendered frames together with the game clock and its remaining budgets, but never the game state or the DOM, from which only restart and decision-screen events are derived (Sec. B). AutoPILOT keeps VISTA's `act` tool, read-only tools (`inspect`, `read_pixels`, `history`) and note files, so the two harnesses expose the same tools apart from delegation. All games share the same harness instructions and only the game description, task and controls differ. Sec. C shows the full prompt, which adds budget rules, play tips and delegation rules to VISTA's instructions. AutoPILOT also reports the remaining budget after each tool result and notifies the model of restarts.

3.1 Acting in real time

At time t the agent can read a frame $o(t)$ and send an *action*: one keyboard or mouse event or an explicit wait, which counts as one *environment step*. The model is invoked in discrete *model calls*: call k reads the frame at time t_k and takes ℓ_k seconds, a median of 6.1 s for GPT-6.1 Sol at low effort. Existing visual harnesses turn each model call into one action, computed from $o(t_k)$ but executed at $t_k + \ell_k$. In real time this causes two problems: every action is ℓ_k seconds stale, so a projectile that was safe in $o(t_k)$ may have arrived by then, and the agent acts at most once per model call. Pausing the clock removes both but changes the task.

AutoPILOT instead lets a single model call act over many steps, by starting a macro or a controller that keeps acting without further model calls. The model thus chooses how many steps each model call covers, and its actions no longer wait for the next call.

3.2 Three execution modes

As shown in Fig. 1b, each model call may invoke one of three tools that act on the game. They differ in how long they act without the model and whether they read the screen.

Primitive action (`act`). The model sends one action. `act` also stops any running macro or controller.

Open-loop macro (`run_macro`). The model writes a sequence of actions, *e.g.* a menu route learned from earlier interaction, and the worker sends them without looking.

Closed-loop controller (`python`). The model writes a short Python function `run(api)` that the worker executes for up to 120 s in a sandbox with one CPU, 1 GB of memory and no network access. Sec. B gives the full setup. It reads frames with `api.observe()`, decides with ordinary code, and acts with

`api.act()`. A controller re-reads the frame and acts with no model call. Each step takes about 0.4 s. The model decides in its code how long each action lasts. This averages about 0.2 s. Taking the next screenshot and reading the state take the other 0.2 s. It can also ask the model for help through a *handoff*, one of the wake events described in Sec. 3.3. Fig. 2 shows a macro and a controller from two recorded episodes.

3.3 Handing control back

For delegation to be safe, control must return to the model once the running program no longer fits the situation. While a macro or controller runs, the model's tool call blocks: the model waits instead of producing tokens, and the game is played by the worker. The call returns with a fresh frame on one of six *wake events*:

- **From the program.** *Handoff*: the controller requests one with a reason, *e.g.* on reaching a goal or seeing something it was not written for. It may keep playing while the model reads the reason. *Return*: `run` ends at the stopping condition the model wrote or with an error. *Macro done*: all macro actions are sent.
- **From the harness.** *Decision screen*: the game waits for a choice, such as in a shop. *Scene change*: most of the screen differs from a second earlier and then holds still. The harness then stops the program, since its plan may no longer fit. *Timeout*: the program has run for 120 s.

A game restart after a loss affects the two modes differently. It stops a running macro, which then returns and wakes the model. It does not wake the model while a controller runs, since the controller observes an incremented `reset_epoch` and can keep playing. Restarts do not pause the game: the next attempt starts at once, whether the model is thinking or a program is running. A running controller keeps playing it, and the model learns of the failure at its next wake event, where it can reflect and write a revised controller. After waking, the model can keep, replace or stop (with `act`) the running controller. Thinking and acting overlap only after a handoff, while the controller keeps playing and the model reads the reason. This overlap is small in practice (2.3–3.5% of inference time, Sec. F.3).

4 AFK-Bench

AFK-Bench contains three environment types that vary the cost of inaction, and two clock protocols that play the same games in real time and paused, to test whether an agent acts well in real time.

Games and tasks. AFK-Bench consists of 120 browser games, each with one task and an official success check, played in a real browser. The games span shooters, platformers, arcade and action games, puzzles and board-like games. Sec. E describes how we collected them from public game sites and validated each with 13 scripted checks. Success and progress are computed from the game state by the benchmark's evaluator. This state is never exposed to the agent, which sees only rendered frames plus restart and decision-screen events (Sec. B).

Environment types. We divide the games into three types of 40 by what happens when the agent does nothing. To assign each game, a scripted probe stops all actions for 60 s and checks whether the attempt fails, the world keeps changing, or nothing happens. Fig. 3 shows three example games of each type, together with how often VISTA and AutoPILOT win them in real time.

• *Hostile real-time environments*, where inaction risks failure: an idle agent loses the attempt, health or a life. In the shooter *Praxis Fighter X*, a ship that does not dodge is destroyed within seconds.

• *Benign real-time environments*, where the world keeps moving but inaction causes no direct harm: an idle agent loses time, not the attempt, as in *Tiny Truck Racing*.

• *Turn-based environments*, where the game changes only when the agent acts and no countdown runs, as in the box-pushing puzzle *Trichromat*.

The types thus separate two costs of slow thinking: in hostile real-time environments it causes failure, while in the other two it only wastes time and budget. Sec. E details the probe, including a second probe for games whose danger appears only after some play.



Figure 3 The three environment types of AFK-Bench. Each row shows three of the 40 games of one type. Below each game are the real-time wins of VISTA and AutoPILOT out of three episodes with GPT-6.1 Sol. Tab. 8 lists all 120 games.

Clock protocols. Every game can be played under two clock protocols. In *real time*, the game clock never stops: the game keeps running during each model call, as for a human player who looks away from the keyboard. Under the *paused clock*, the harness freezes the game from the start of each model call until it ends, together with any running macro or controller. Game time therefore passes only while no model call is in progress. Every game becomes turn-based for the agent, with no time pressure while the model thinks. Under both protocols, the games' own pause and restart controls are blocked, so the agent cannot stop the clock itself. The drop from the paused clock to real time measures how much an agent loses to its own latency.

Budgets. An episode ends with a win, the game's own end, or when one of its budgets is used up: **100 model calls**, **2000 s of game time** or **2000 environment steps**. A model call is one invocation of the model, which issues at most one tool call (a reply without one also counts). An environment step is one action sent to the game, whether by `act`, a macro or a controller. A single-step harness thus spends one model call per step and sends at most 100 steps, so only the call and time budgets bind it, whereas one AutoPILOT model call can launch a macro or controller that sends many steps, and the step budget caps these. In real time, thinking time counts as game time, while loading and automatic restarts do not. After a loss, the game restarts within the same budgets, so the agent can try again.

Metrics. Our main metric is the success rate (SR): the fraction of episodes in which the game's official success check fires before any budget is used up. Because an episode can span several attempts, SR measures whether an agent wins within its budgets, possibly after earlier losses. Sec. 5.3 also reports success on the first attempt. The mean progress (PG) gives a finer signal. It averages over episodes the highest progress reached in any attempt, which the evaluator normalizes to $[0, 1]$ for each game. We also report the number of model calls, tokens and API cost per episode.

5 Experiments

We aim to answer four questions: how much a single-step harness loses when the world keeps running and whether AutoPILOT recovers it (Sec. 5.2), where the gains come from (Sec. 5.3), how they scale with budget, effort and speed (Sec. 5.4), and whether they generalize (Sec. 5.5).

Harness	Model	Hostile real-time	Benign real-time	Turn-based	Overall
		Paused Clock			
Codex only	GPT-6.1 Sol	52.5 .737	45.0 .693	57.5 .780	51.7 .737
Claude Code only	Claude Opus 5.5	47.5 .735	40.0 .594	40.0 .664	42.5 .665
VISTA	GPT-6.1 Sol	56.7 .783	55.8 .746	55.8 .796	56.1 .775
	Claude Opus 5.5	37.5 .629	40.0 .644	40.0 .670	39.2 .648
	DeepSeek-V4.1-Flash	25.0 .537	22.5 .468	25.0 .496	24.2 .500
	Qwen4-Max	32.5 .552	27.5 .513	15.0 .445	25.0 .503
	GLM-5.3-Flash	2.5 .240	0.0 .233	2.5 .130	1.7 .201
	Hy4-flash	12.5 .298	2.5 .272	2.5 .131	5.8 .233
AutoPILOT	GPT-6.1 Sol	73.3 .857	63.3 .795	82.5 .915	**73.1** .856
	Claude Opus 5.5	65.0 .795	40.0 .663	77.5 .892	**60.8** .784
	DeepSeek-V4.1-Flash	50.0 .663	32.5 .531	50.0 .692	**44.2** .629
	Qwen4-Max	50.0 .653	30.0 .515	45.0 .718	**41.7** .628
	GLM-5.3-Flash	35.0 .588	20.0 .453	25.0 .513	**26.7** .518
	Hy4-flash	45.0 .658	30.0 .538	40.0 .688	**38.3** .628
		Real-time Clock			
—	No-op	0.0 .079	0.0 .001	0.0 .002	0.0 .027
	Random	10.0 .360	2.5 .214	5.0 .199	5.8 .258
Codex only	GPT-6.1 Sol	17.5 .469	36.7 .594	62.5 .811	38.9 .625
Claude Code only	Claude Opus 5.5	20.0 .486	25.0 .544	52.5 .729	32.5 .586
VISTA	GPT-6.1 Sol	16.7 .442	37.5 .595	60.8 .801	38.3 .613
	Claude Opus 5.5	17.5 .453	27.5 .502	45.0 .733	30.0 .563
	DeepSeek-V4.1-Flash	10.0 .313	25.0 .506	25.0 .576	20.0 .465
	Qwen4-Max	7.5 .298	10.0 .377	25.0 .528	14.2 .401
	GLM-5.3-Flash	5.0 .226	2.5 .188	2.5 .072	3.3 .162
	Hy4-flash	7.5 .220	0.0 .186	2.5 .157	3.3 .188
AutoPILOT	GPT-6.1 Sol	62.5 .784	60.0 .777	83.3 .912	**68.6** .824
	Claude Opus 5.5	47.5 .671	42.5 .657	70.0 .882	**53.3** .737
	DeepSeek-V4.1-Flash	40.0 .642	30.0 .539	47.5 .667	**39.2** .616
	Qwen4-Max	25.0 .540	25.0 .484	32.5 .606	**27.5** .543
	GLM-5.3-Flash	12.5 .421	10.0 .428	32.5 .516	**18.3** .455
	Hy4-flash	30.0 .532	25.0 .477	42.5 .716	**32.5** .575

Table 2 Main results on AFK-Bench. Success rate (%) with mean progress in small type. Bold: best overall success rate for each model and clock. DeepSeek-V4.1-Flash, GLM-5.3-Flash and Hy4-flash are open-source, and Qwen4-Max is closed. All models use low reasoning effort. Hy4-flash has a single thinking mode. Tab. 11 gives 95% intervals and statistics over the three seeds.

5.1 Setup

Models. We compare all harnesses with two closed models. We access GPT-6.1 Sol through the Codex app server [26] and Claude Opus 5.5 through the Claude Code CLI [2]. VISTA and AutoPILOT also run with three open-source models, DeepSeek-V4.1-Flash [8], GLM-5.3-Flash [55] and Hy4-flash [39]. They also run with one more closed model, Qwen4-Max [31]. We access these four models through OpenAI-compatible APIs. Sec. F.2 adds paired tests for these four models and GameWorld results for Qwen4-Max and Hy4-flash. It also runs AutoPILOT with GPT-6 Astra [27] and Claude Fable 5.2 [1]. All main runs use low reasoning effort. Hy4-flash has a single thinking mode. Sec. 5.4 compares low and high effort for GPT-6.1 Sol and Claude Opus 5.5.

Baselines. We compare against three visual agent harnesses. *VISTA* [11] is the previous state-of-the-art visual harness on ARC-AGI-3 [3] and GameWorld. We run its official implementation through the same Codex app server, with its own prompt and tools: lossless visual memory, visual inspection, notes, and one action per `act` call. *Codex* and *Claude Code* are the plain harnesses of the two models, with only `act` and `screenshot` as game tools, and represent a standard single-step computer-use loop. We

tell every harness that the game runs in real time. Sec. D gives the exact instructions. Methods that read text states [23, 29, 45] or require training [36, 44] do not apply to this screen-only, training-free setting. We additionally include two simple policies without a model to validate the benchmark: a *no-op agent* that sends no step, and a *random agent* that sends random valid steps as fast as possible.

Protocol. Unless noted otherwise, results are in real time on all 120 tasks. For GPT-6.1 Sol, we run three episodes per task with different random seeds (360 in total). To limit cost, we run one episode per task for the other models.

5.2 Main results

AutoPILOT outperforms existing harnesses. As Tab. 2 shows, AutoPILOT nearly doubles the real-time success rate of existing harnesses: 68.6% against 38.3% for VISTA and 38.9% for Codex with GPT-6.1 Sol, and 53.3% against 30.0% and 32.5% for VISTA and Claude Code with Claude Opus 5.5, while costing \$0.89 per episode against \$1.50 for VISTA with GPT-6.1 Sol (Tab. 4). The margin is largest in hostile real-time environments (62.5% *vs.* 16.7% with GPT-6.1 Sol) and remains large in the other two types. The no-op and random agents win at most 5.8%. Of the 360 paired GPT-6.1 Sol episodes, 111 are won only by AutoPILOT and 2 only by VISTA.

Open-source models and Qwen4-Max. The three open-source models and the closed Qwen4-Max show the same pattern at lower success rates. With each of them, AutoPILOT outperforms VISTA under both clocks, by 13.3 (Qwen4-Max) to 29.2 points (Hy4-flash) in real time, and every difference is significant (Sec. F.2). AutoPILOT with DeepSeek-V4.1-Flash (39.2%) surpasses VISTA with GPT-6.1 Sol (38.3%), and AutoPILOT with Hy4-flash (32.5%) surpasses VISTA with Claude Opus 5.5 (30.0%), so **a better harness can make up for a weaker model**. The reason VISTA falls behind is not the same for the open-source models as for GPT-6.1 Sol and Claude Opus 5.5. With GPT-6.1 Sol and Claude Opus 5.5, it loses much of its success to the clock (17.8 and 9.2 points). With the three open-source models, it is already weak when the game waits (1.7–24.2%) and loses at most 4.2 points in real time. With Qwen4-Max, it reaches 25.0% when the game waits and loses 10.8 points in real time. With GLM-5.3-Flash and Hy4-flash, VISTA wins almost no tasks under either clock, while AutoPILOT wins 18.3–38.3%. In relative terms, the gain is larger with each of these four models: AutoPILOT reaches 1.9–9.8× the real-time success rate of VISTA, against 1.8× with GPT-6.1 Sol and Claude Opus 5.5. When a model plays poorly one action at a time, letting it write a whole plan helps all the more.

Memory and perception tools bring no overall gain. In real time, VISTA performs no better than the plain Codex and Claude Code harnesses (38.3% *vs.* 38.9% with GPT-6.1 Sol, 30.0% *vs.* 32.5% with Claude Opus 5.5), despite its lossless visual memory and inspection tools. Each VISTA action is computed from a frame about one model call old, so a better view of it cannot help. Even with the clock paused, these tools add 4.4 points over Codex (56.1% *vs.* 51.7%) and none over Claude Code (39.2% *vs.* 42.5%).

AutoPILOT is less sensitive to the game clock. Moving from the paused clock to real time lowers the success rate of VISTA and Codex by 17.8 and 12.8 points with GPT-6.1 Sol, but that of AutoPILOT by 4.5. The drops are closer with Claude Opus 5.5 (9.2 and 10.0 for VISTA and Claude Code, 7.5 for AutoPILOT), while on GameWorld VISTA loses 41.2 points and AutoPILOT only 2.9 (Sec. 5.5).

5.3 Why AutoPILOT wins

We analyze the gap in three steps: gains beyond latency, the responsible component, and recovery from failures.

Gains beyond latency. Even with the clock paused, where no action is stale, AutoPILOT beats VISTA by 17.0 points with GPT-6.1 Sol and 21.7 with Claude Opus 5.5, so *latency explains only part of the gap*. The rest comes from **executing a whole plan per model call instead of a single action**, which makes

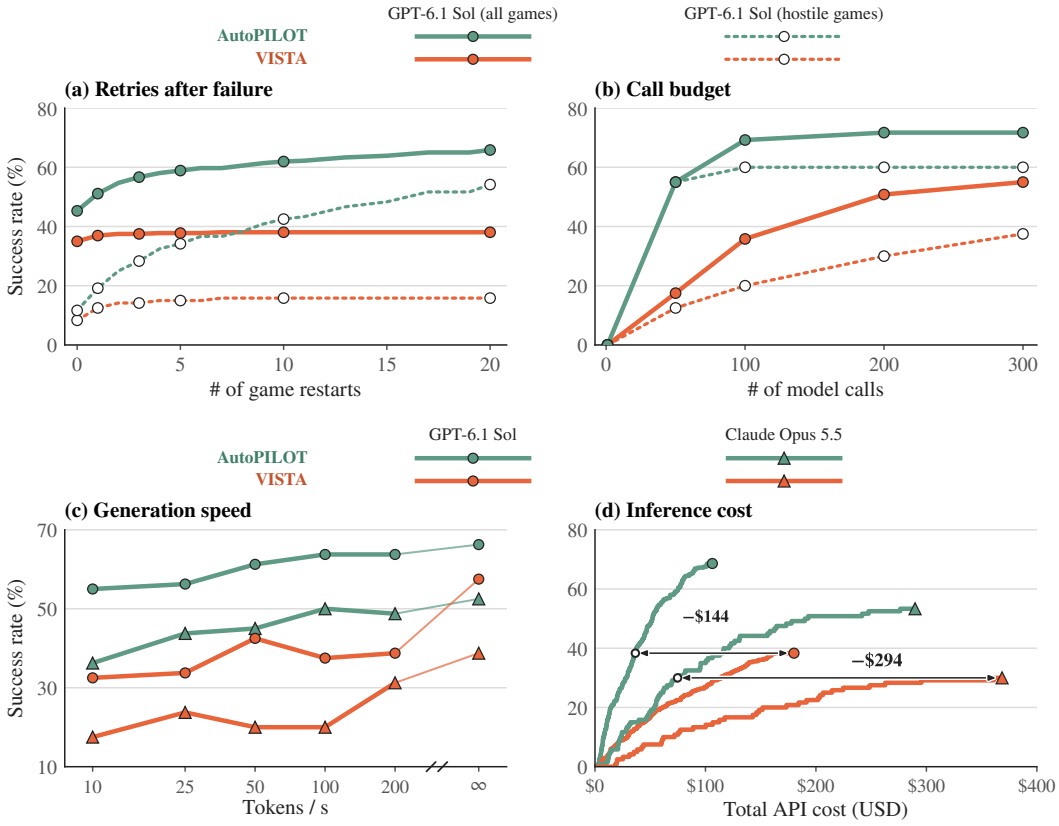

Figure 4 Recovery, model-call budget, generation speed and cost. (a) Success within k restarts and **(b)** within the first b model calls (300-call budget), on all 120 games (solid) and the 40 hostile real-time games (dotted), with GPT-6.1 Sol. **(c)** Success of AutoPILOT and VISTA on the 80 hostile real-time and benign real-time tasks against simulated generation speed (in tokens/s, where ∞ means a paused clock), with GPT-6.1 Sol and Claude Opus 5.5. **(d)** Success against the total list-price API cost over all 120 tasks as a per-episode spending cap is raised. Arrows run from where AutoPILOT first matches VISTA's final success (white ring) to VISTA's total cost.

far better use of the model-call budget. VISTA spends one model call on each action, so long action sequences use up its budget: 208 of its 222 real-time failures end by exhausting its 100 calls, after only 78 steps per episode. AutoPILOT instead sends 681 steps from 43 model calls, about 16 per call, because one macro or controller carries out many steps. *The model also chooses the mode that fits the game* (Tab. 3): controllers send 98% of the steps in hostile real-time and 89% in benign real-time environments, whereas in turn-based environments, which allow planning ahead, macros account for 36% of calls and 26% of steps.

Which component is responsible. Tab. 4 removes the two delegation tools, `python` and `run_macro`, one at a time (seeds in Tab. 10). Without controllers, which reduces AutoPILOT to batching actions into macros, success in hostile real-time environments drops from 62.5% to 50.8%, with fewer steps and more model calls. Without macros, overall success drops slightly, mainly in turn-based environments, where macros are used most (83.3% to 78.3%, Tab. 16). With either tool alone, AutoPILOT still clearly outperforms VISTA, so **the two modes can largely stand in for each other**, while combining them further improves overall success with the fewest tokens (2.8M). Controllers that the model writes from its own view but that cannot read the screen while running win slightly fewer tasks than closed-loop ones (78 *vs*. 83, Sec. F.3). Removing both drops AutoPILOT to 45.8% overall and 22.5% in hostile real-time environments. This variant keeps VISTA's tools and adds only our prompt, budget line and restart notices. It still outperforms VISTA on the same seed, but *these additions matter far less than the two*

Env.	% of tool calls				% of env steps			Steps per game
	`act`	macro	ctrl.	other	`act`	macro	ctrl.	
Hostile	37	10	46	7	1	1	98	1043
Benign	46	25	25	4	4	7	89	693
Turn-based	43	36	17	4	6	26	68	307
All	43	25	27	5	3	7	90	681

Table 3 Tool calls and steps of AutoPILOT (GPT-6.1 Sol, real time). Share of tool calls and environment steps by tool (each sums to 100%), and steps per game. Macros and controllers (ctrl.) send many steps per call. "Other" covers note and inspection tools.

	Success (%)				
Harness	Hostile	Overall	Calls	Tokens	$/ep.
Codex	17.5	38.9	81	5.1M	1.35
VISTA	16.7	38.3	80	5.3M	1.50
AutoPILOT	**62.5**	**68.6**	**43**	**2.8M**	**0.89**
No controller	50.8	64.7	58	4.6M	1.31
No macro	**62.5**	67.5	46	3.1M	0.91
No controller & macro	22.5	45.8	81	5.5M	1.45

Table 4 Ablations and cost (GPT-6.1 Sol, real time). Success rate (%) and model calls, tokens and cost per episode.

Budget	model calls / steps / time	Success	Cost / ep.
AutoPILOT			
Standard	100 / 2000 / 2000 s	69.2	$0.83
3× model calls	300 / 2000 / 2000 s	71.7 (+2.5)	$1.18
2× steps	100 / 4000 / 2000 s	70.8 (+1.7)	$0.90
2× game time	100 / 2000 / 4000 s	70.8 (+1.7)	$0.94
All relaxed	300 / 4000 / 4000 s	**78.3** (+9.2)	$1.92
VISTA			
Standard	100 / – / 2000 s	37.5	$1.48
3× model calls	300 / – / 2000 s	55.0 (+17.5)	$5.28
All relaxed	300 / – / 4000 s	51.7 (+14.2)	$6.53

Table 5 Scaling the budget. Success rate (%) and API cost per episode with GPT-6.1 Sol in real time on 120 episodes. Parentheses give the change from the standard budget. VISTA sends one step per model call, so the step budget does not apply (–).

tools, which account for 28 of the 38 extra wins overall and 17 of 19 in hostile real-time environments.

Recovering from failures. Restarts are where AutoPILOT gains most (Fig. 4a). Without any restart, AutoPILOT wins 11.7% of hostile real-time episodes and VISTA 8.3%. With up to 20 restarts, AutoPILOT climbs to 54.2% and is still rising, whereas VISTA stalls at 15.8% after about seven restarts. VISTA must *still play every retry one action per model call*, so each attempt meets the same latency, whereas AutoPILOT can play it at game speed in a macro or controller. An attempt costs AutoPILOT 2 model calls against 8 for VISTA with GPT-6.1 Sol (1 *vs.* 12 with Claude Opus 5.5, Fig. 7). After a failure, AutoPILOT therefore still has budget to retry. It wins 58% of paired episodes against 6% for VISTA, which had used up its calls in 110 of 123 (Sec. F.4).

5.4 Scaling with budget, effort and speed

Scaling the budget. Since AutoPILOT sends far more steps per episode, we test whether VISTA simply runs out of budget too early (Tabs. 5 and 19). Relaxing the budgets does help VISTA: with three times

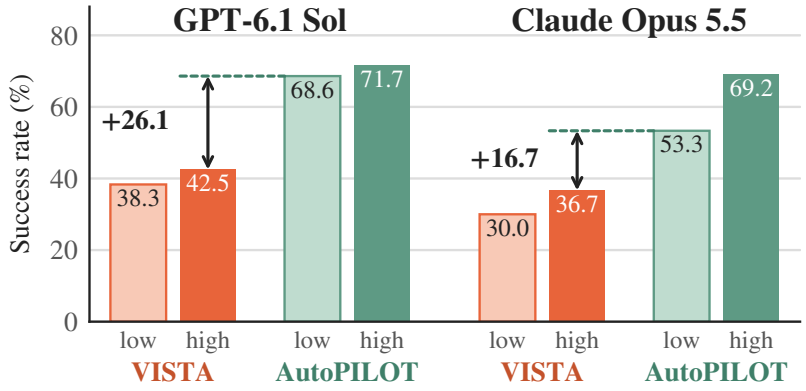

Figure 5 Low vs. high reasoning effort (AutoPILOT and VISTA, real time). Overall success rate. Arrows mark the gap from VISTA at high effort to AutoPILOT at low effort, in points.

the model calls, it rises from 37.5% to 55.0%. Yet a large gap remains, as AutoPILOT reaches 69.2% at the standard budget and 78.3% with all budgets relaxed, and in hostile real-time environments VISTA stays at or below 37.5% under every budget. VISTA's gain is costly: \$5.28 per episode, 6× that of AutoPILOT, which reaches 55% within 50 calls (Fig. 4b). Capping the spending per episode (Fig. 4d), AutoPILOT matches VISTA's final success rate at 0.2× its total cost with both models (\$37 *vs.* \$180 with GPT-6.1 Sol, \$75 *vs.* \$369 with Claude Opus 5.5).

Scaling reasoning effort. Prior work finds that slower, reasoning-heavy agents degrade in real time, as the world moves on while the model thinks [29, 45]. In our runs, high effort hurts neither harness, but only AutoPILOT turns it into a large gain (Fig. 5). With Claude Opus 5.5, AutoPILOT rises from 53.3% to 69.2% and VISTA from 30.0% to 36.7%. With GPT-6.1 Sol, both gain a few points (68.6% to 71.7% and 38.3% to 42.5%). At either effort, VISTA stays far below AutoPILOT at low effort, by 26.1 and 16.7 points at high effort. For AutoPILOT, a longer call yields a better program that then acts for many steps.

Scaling generation speed. We simulate faster or slower models with a decoding speed of v tokens/s: the clock stops during generation and then advances by N/v s for N output tokens. On the 80 hostile real-time and benign real-time tasks (Fig. 4c), AutoPILOT with GPT-6.1 Sol at low effort rises from 55.0% at 10 tokens/s to 63.8% at 200 tokens/s, close to the paused clock. VISTA, by contrast, reaches 38.8% even at 200 tokens/s, against 57.5% when paused. AutoPILOT stays strong even at low speed because **speed only delays when a working controller starts**: once written, a controller plays at full speed however slowly the model thinks. Slow generation hurts most when outputs are long: Claude Opus 5.5 at high effort, which writes about twice as many tokens per call as GPT-6.1 Sol, gains the most from speed (31.3% to 62.5%, Tab. 20).

5.5 Generalization to GameWorld

We also evaluate both harnesses on GameWorld [28], whose 170 tasks in 34 browser games do not overlap with AFK-Bench, using the same budgets (Tab. 6). With the clock paused, the two harnesses are close (84.1% *vs.* 82.9% with GPT-6.1 Sol), as GameWorld tasks are short enough that VISTA rarely runs out of calls (3 of its 27 failures). In real time, however, VISTA loses 41.2 points with

Model	Harness	Paused	Real time
GPT-6.1 Sol	VISTA	**84.1** .917	42.9 .653
	AutoPILOT	82.9 .911	**80.0** .900
Claude Opus 5.5	VISTA	67.1 .830	41.2 .623
	AutoPILOT	**72.4** .855	**67.6** .845

Table 6 Results on GameWorld. Success rate (%) on its 170 tasks, with mean progress in small type. Bold marks the best per model and clock.

GPT-6.1 Sol and 25.9 with Claude Opus 5.5, while AutoPILOT loses only 2.9 and 4.7. The **robustness of AutoPILOT to the game clock carries over to a different set of games**.

6 Conclusion

We presented AutoPILOT, a visual agent harness in which the model delegates acting to macros and controllers that it writes at run time. Because each model call can launch a whole plan rather than a single action, AutoPILOT substantially outperforms existing harnesses in real time with closed and open-source models and on two benchmarks, without any training. With GPT-6.1 Sol, it also costs less. Our results suggest that how model outputs become actions over time deserves as much attention in harness design as what the model sees and remembers, for game agents and agents operating live interfaces or robots.

References

[1] Anthropic. Introducing Claude Fable 5.2 and Claude Mythos 5.2. https://www.anthropic.com/claude-fable-and-mythos-5-2, November 2026.

[2] Anthropic. Claude Opus 5.5. https://www.anthropic.com/claude-opus-5-5, September 2026.

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

[4] Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In *NeurIPS*, 2025.

[5] Guojun Chen, Alex Schott, and Lin Zhong. TypeGo: An OS runtime for embodied agents. *arXiv preprint arXiv:2607.05482*, 2026.

[6] Yanzhe Chen, Zechen Bai, Zhijun Cao, Wenzheng Zeng, Kevin Qinghong Lin, Yiqi Lin, Guoqiang Liang, Kevin Yuchen Ma, Qiming Huang, and Mike Zheng Shou. Show-Harness: Just a VLM agent can play robots. *arXiv preprint arXiv:2609.10522*, 2026.

[7] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. In *ACL*, 2024.

[8] DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. *arXiv preprint arXiv:2609.19969*, 2026.

[9] Zihan Dong, Rui Qian, Qishi Zhan, Dongshen Peng, Kaixin Li, and Yu Li. Why are GUI agents correct but late? Decode on the decision-time critical path, tested with pre-compiled policy trees. *arXiv preprint arXiv:2607.28399*, 2026.

[10] In Gim, Seung-seob Lee, and Lin Zhong. Asynchronous LLM function calling. *arXiv preprint arXiv:2412.07017*, 2024.

[11] Qiushi Han, Keya Hu, Linlu Qiu, Cathy Wu, and Kaiming He. VISTA: A visual harness for reasoning in an interactive world. *arXiv preprint arXiv:2610.02200*, 2026.

[12] Lanxiang Hu, Mingjia Huo, Yuxuan Zhang, Haoyang Yu, Eric P. Xing, Ion Stoica, Tajana Rosing, Haojian Jin, and Hao Zhang. lmgame-Bench: How good are LLMs at playing games? In *ICLR*, 2026.

[13] Xirui Jiang, Yuxi Chen, Rui Yang, Feng Luo, Pengcheng Wang, Bingxuan Li, Zihao Zhao, Yushan Li, Han Wang, Manling Li, Yang Zhang, Shiyu Chang, Tong Zhang, and Huan Zhang. Game2Player: Scaling gameplay experience for visual agents through environment-player co-development. *arXiv preprint arXiv:2611.24154*, 2026.

[14] Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, 2011.

[15] Zhao Kaiya, Michelangelo Naim, Jovana Kondic, Manuel Cortes, Jiaxin Ge, Shuying Luo, Guangyu Robert Yang, and Andrew Ahn. Lyfe Agents: Generative agents for low-cost real-time social interactions. *arXiv preprint arXiv:2310.02172*, 2023.

[16] Hao Kang, Qingru Zhang, Han Cai, Weiyuan Xu, Tushar Krishna, Yilun Du, and Tsachy Weissman. Win fast or lose slow: Balancing speed and accuracy in latency-sensitive decisions of LLMs. In *NeurIPS*, 2025.

[17] Seth Karten, Joel Zhang, Tersoo Upaa Jr, Ruirong Feng, Wenzhe Li, Chengshuai Shi, Chi Jin, and Kiran Vodrahalli. Continual Harness: Online adaptation for self-improving foundation agents. *arXiv preprint arXiv:2605.09998*, 2026.

[18] Jens Lehmann, Andrei Aioanei, and Sahar Vahdati. Tycho: Active abstraction with programmatic world models for ARC-AGI-3. *arXiv preprint arXiv:2607.28287*, 2026.

[19] Bingxuan Li, Siqi Song, Yizhuo Wu, Jiarui Yao, Tong Zhang, and Huan Zhang. MotorMind: Scaffolding general vision language models for zero-shot robot manipulation. *arXiv preprint arXiv:2609.38078*, 2026.

[20] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as Policies: Language model programs for embodied control. In *ICRA*, 2023.

[21] Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. SwiftSage: A generative agent with fast and slow thinking for complex interactive tasks. In *NeurIPS*, 2023.

[22] Jijia Liu, Chao Yu, Jiaxuan Gao, Yuqing Xie, Qingmin Liao, Yi Wu, and Yu Wang. LLM-powered hierarchical language agent for real-time human-AI coordination. In *AAMAS*, 2024.

[23] Xinghua Lou, Miguel Lázaro-Gredilla, Antoine Dedieu, Carter Wendelken, Wolfgang Lehrach, and Kevin P. Murphy. AutoHarness: Improving LLM agents by automatically synthesizing a code harness. *arXiv preprint arXiv:2603.03329*, 2026.

[24] Dunjie Lu, Shuai Bai, Tianyi Bai, et al. Qwen-CUA: Native computer use for (almost) everything. *arXiv preprint arXiv:2608.02352*, 2026.

[25] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[26] OpenAI. Introducing GPT-6.1 Sol. https://openai.com/index/introducing-gpt-6-1-sol/, September 2026.

[27] OpenAI. GPT-6 Astra: A new generation of intelligence. https://openai.com/index/gpt-6-astra/, 2026.

[28] Mingyu Ouyang, Siyuan Hu, Kevin Qinghong Lin, Hwee Tou Ng, and Mike Zheng Shou. GameWorld: Towards standardized and verifiable evaluation of multimodal game agents. In *ECCV*, 2026.

[29] Ningxin Pan, Hanyu Li, Yeqing Shen, Ning Ding, Yehui Tang, and Wentao Zhang. The world won't wait: Evaluating and teaching agents under a live clock. *arXiv preprint arXiv:2610.20665*, 2026.

[30] Yujia Qin, Yining Ye, Junjie Fang, et al. UI-TARS: Pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

[31] Qwen Team. Qwen4-Max: Your new teammate has arrived. https://qwen.ai/blog?id=qwen4-max, November 2026.

[32] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

[33] SIMA Team. Scaling instructable agents across many simulated worlds. *arXiv preprint arXiv:2404.10179*, 2024.

[34] SIMA Team. SIMA 2: A generalist embodied agent for virtual worlds. *arXiv preprint arXiv:2512.04797*, 2025.

[35] Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2): 181–211, 1999.

[36] Weihao Tan, Xiangyang Li, Yunhao Fang, et al. Lumine: An open recipe for building generalist agents in 3D open worlds. *arXiv preprint arXiv:2511.08892*, 2025.

[37] Weihao Tan, Wentao Zhang, Xinrun Xu, et al. Cradle: Empowering foundation agents towards general computer control. In *ICML*, 2025.

[38] Jingyu Tang, Xu Yang, Zhehao Zhang, Zheng Wang, Qi Cheng, Haozhe Zhao, Manling Li, Wenxuan Li, Xide Xia, and Jiaxin Pei. Realtime-Bench: Can frontier models solve real-time interactive problems? *arXiv preprint arXiv:2610.22155*, 2026.

[39] Tencent Hy Team. Hy4 flash. https://github.com/Tencent-Hunyuan/Hy4-flash, November 2026.

[40] Dawei Wang, Di Zhao, Jianran Yu, Chengming Zhou, Marci Chi, Xiaoyang Liu, Beining Zhang, Shuliang Chu, Yun Sing Koh, Gary Ushaw, and Richard Davison. ROAR: Benchmarking LMM agents under real-time constraints in multiplayer action-shooter games. In *NeurIPS*, 2026.

[41] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *TMLR*, 2024.

[42] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *ICML*, 2024.

[43] Zihao Wang, Shaofei Cai, Anji Liu, Yonggang Jin, Jinbing Hou, Bowei Zhang, Haowei Lin, Zhaofeng He, Zilong Zheng, Yaodong Yang, Xiaojian Ma, and Yitao Liang. JARVIS-1: Open-world multi-task agents with memory-augmented multimodal language models. *IEEE TPAMI*, 2025.

[44] Zihao Wang, Xujing Li, Yining Ye, et al. Game-TARS: Pretrained foundation models for scalable generalist multimodal game agents. *arXiv preprint arXiv:2510.23691*, 2025.

[45] Yule Wen, Yixin Ye, Yanzhe Zhang, Diyi Yang, and Hao Zhu. Real-time reasoning agents in evolving environments. In *ICLR*, 2026.

[46] Lilian Weng. Harness engineering for self-improvement. https://lilianweng.github.io/posts/2026-07-04-harness/, 2026.

[47] Tianbao Xie, Danyang Zhang, Jixuan Chen, et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *NeurIPS*, 2024.

[48] George Yakushev, Denis Mazur, Vladimir Bartenev, Vyacheslav Zhdanovskiy, Timofey Byzov, Vladimir Kaurkin, and Vadim Pastushenko. LLMs are general asynchronous agents. *arXiv preprint arXiv:2609.35427*, 2026.

[49] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *NeurIPS*, 2024.

[50] Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, Heng Ji, Huan Zhang, and Tong Zhang. EmbodiedBench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. In *ICML*, 2025.

[51] Rui Yang, Qianhui Wu, Yuxi Chen, Hao Bai, Wenlin Yao, Hao Cheng, Baolin Peng, Huan Zhang, Tong Zhang, and Jianfeng Gao. OpenWebRL: Demystifying online multi-turn reinforcement learning for visual web agents. In *NeurIPS*, 2026.

[52] Rui Yang, Qianhui Wu, Zhaoyang Wang, Hanyang Chen, Ke Yang, Hao Cheng, Huaxiu Yao, Baolin Peng, Huan Zhang, Jianfeng Gao, and Tong Zhang. GUI-Libra: Training native GUI agents to reason and act with action-aware supervision and partially verifiable RL. In *NeurIPS*, 2026.

[53] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *ICLR*, 2023.

[54] Lance Ying, Ryan Truong, Prafull Sharma, Kaiya Ivy Zhao, Nathan Cloos, Kelsey R. Allen, Thomas L. Griffiths, Katherine M. Collins, José Hernández-Orallo, Phillip Isola, Samuel J. Gershman, and Joshua B. Tenenbaum. AI Gamestore: Scalable, open-ended evaluation of machine general intelligence with human games. *arXiv preprint arXiv:2602.17594*, 2026.

[55] Z.ai. GLM-5.3-Flash. https://huggingface.co/zai-org/GLM-5.3-Flash, August 2026.

[56] Alex L. Zhang, Thomas L. Griffiths, Karthik R. Narasimhan, and Ofir Press. VideoGameBench: Can vision-language models complete popular video games? *arXiv preprint arXiv:2505.18134*, 2025.

[57] Shao Zhang, Xihuai Wang, Wenhao Zhang, Chaoran Li, Junru Song, Tingyu Li, Lin Qiu, Xuezhi Cao, Xunliang Cai, Wen Yao, Weinan Zhang, Xinbing Wang, and Ying Wen. Leveraging dual process theory in language agent framework for real-time simultaneous human-AI collaboration. In *ACL*, 2025.

[58] Yangqing Zheng, Shunqi Mao, Dingxin Zhang, and Weidong Cai. Reflex first, reflect later: Latency-aware embodied LLM agents for dynamic response. *arXiv preprint arXiv:2506.07223*, 2025.

[59] Yu Zhou, Cheng-Fu Yang, Rui Sun, Di Wu, Sibo Peng, Mingyang Zhang, Zhen Yang, Ruiyi Zhang, Nanyun Peng, Zhe Gan, and Kai-Wei Chang. OpenVLHarness: Real world visual agentic tool-use with multimodal memory. *arXiv preprint arXiv:2610.26860*, 2026.

Appendix

A Extended related work

Agent harnesses. A harness is the code around a model that turns the environment into the model's observations and the model's outputs into actions. ReAct [53] interleaves reasoning with one action per turn. SWE-agent [49] shows that the interface given to the model matters as much as the model. CodeAct [42] lets a single action be a program that calls many tools. VISTA [11] perceives the game as images, keeps every frame in a lossless visual memory and lets the model re-inspect frames as it reasons, raising the ARC-AGI-3 score of the same model from 40.68 to 100. Our episodes restart after a loss, so the model can reflect on failed attempts as in Reflexion [32].

Computer-use and game agents. Vision-based computer-use agents ground actions in screenshots [7, 24, 30, 47, 51]. Game agents range from reinforcement-learning policies trained per game [25] to foundation-model agents that play many games from pixels or structured state [33, 34, 37, 41, 43, 44]. Lumine [36] trains a vision-language model to play 3D open-world games in real time. It reads frames at 5 Hz, emits actions at 30 Hz and reasons only when needed. It reaches real-time play through training, whereas AutoPILOT uses a general model without training.

Real-time evaluation. VideoGameBench [56] reports latency as a major failure cause for vision-language models. Real-Time Reasoning Gym [45] and ROAR [40] advance their environments during inference. Realtime-Bench [38] evaluates 30 web and 20 game tasks from screenshots under wall, paused and token clocks with a one-action-per-model-call harness. Win Fast or Lose Slow [16] adapts model size and quantization to the time pressure. The World Won't Wait [29] evaluates 180 tasks from six game types. It uses structured text observations and game-specific action interfaces. Several of these plan ahead, such as button plans in Mario and flap schedules in Bird. While a model call is pending, the previous control stays active.

Acting while thinking. AgileThinker [45] runs a reactive and a planning thread in parallel, and SwiftSage [21] pairs a small fine-tuned model with a large one, in the spirit of dual-process theories [14]. AsyncLLM [48] is a general framework for asynchronous inference rather than a game agent, and it is evaluated mainly on streaming video understanding. In two ViZDoom scenarios, a background thinker sets a course of action that a fast coroutine turns into per-frame actions from the two latest frames. An appendix experiment lets the agent rewrite its own coroutines between evaluation rounds. Real-time chunking [4] executes one action chunk of a robot policy while the next is generated. Dong *et al.* [9] show that GUI agents often choose correctly but act too late. They pre-compile guarded policy trees sized to the decoding latency, which raises success within the decision window from 0.50 to 0.79. Their branches are enumerated in advance for discrete GUI states, whereas AutoPILOT's model writes closed-loop programs for continuous games at run time. TypeGo [5] runs planning loops at several timescales and has an LLM compile reflexes offline into Python condition and action functions that run at 100 Hz. AutoPILOT synthesizes its controllers at run time from the current screen.

B Harness details

Tools. Tab. 7 lists the tools available to the model in each harness. AutoPILOT keeps VISTA's `act` and read-only tools and adds `run_macro` and `python`, which together give the three execution modes of Sec. 3.2.

Actions. Every action, whether sent by `act`, a macro or a controller, uses one schema. Its types are `wait`, `press_key`, `press_keys`, `type`, `click`, `click_hold`, `mouse_move`, `drag` and `scroll`. The action `press_key` holds one key, and `press_keys` holds two keys at once but rejects opposing directions. The action `type` enters up to five characters. Each action is one environment step. Coordinates refer to the 1280×720 screenshot, durations are at most 2 s, and the default hold or wait is 0.4 s. Controllers usually set shorter durations. Only the keys of the game's declared controls are allowed.

Tool	Codex	VISTA	AutoPILOT
`act` (one action per call)	✓	✓	✓
`run_macro` (open-loop)			✓
`python` (closed-loop controller)			✓
`inspect`, `read_pixels`, `history`		✓	✓
notes (`GUIDE.md`, `WORKING.md`)		✓	✓

Table 7 Game tools per harness in the GPT-6.1 Sol runs. Every harness sees a screenshot after each action, and AutoPILOT also on every wake. The Codex harness also has `screenshot` and keeps Codex's code execution without shell, file or network access.

Macros. `run_macro` takes a list of at most 200 actions and sends them in order without reading the screen. With the default `on_reset: stop`, a game restart stops the macro. With `continue`, the list restarts on the new attempt.

Controller API. A controller is a function `run(api)` that a background worker executes for at most 120 s. It runs in a Docker sandbox with one CPU, 1 GB of memory, 60 s of CPU time, no network and no access to the game or host files. Python, NumPy and a `run_macro` helper are available. `api.observe()` returns the latest frame as a $720 \times 1280 \times 3$ RGB array together with the public state: the game clock, the remaining budgets and `reset_epoch`. `api.act(action)` sends one action in the format above, which counts as one step toward the step budget. `api.request_handoff(reason)` wakes the model while the controller keeps running. Returning from `run` ends the controller and also wakes the model. `api.notify_reset(reason)` records a game restart, signalled by an incremented `reset_epoch`, without waking the model. A new controller replaces the running one, and `act` stops it. There is no fixed control rate. A controller observes and acts as fast as its own code and action durations allow. Sec. F.3 reports the measured step time.

Waiting and waking. After a `run_macro` or `python` call, the harness holds the tool result while the macro or controller runs, so the model produces no tokens in the meantime. It polls every 0.25 s for one of the wake events of Sec. 3.3. The result is released with a fresh frame on a handoff, when the program ends, stops or times out, or when the episode ends.

Screen watch. Each macro and controller has a screen watch, which raises the scene-change wake event and which the model can tune or turn off per call. A *scene change* fires when at least half of the screen differs from the frame about one second earlier and the new screen then holds for three frames, so a flash or continuous motion does not trigger it. An optional *static* check fires when the screen stays unchanged for 10 s while actions are sent. On a trigger the watch stops the program by default. It can instead only wake the model.

Signals derived from the game state. No harness reads the game state, but two kinds of events derived from it reach the agent. *Automatic restarts.* After a loss, AutoPILOT's controllers see an incremented `reset_epoch`, and the model receives a one-line notice that the attempt failed and the game restarted. Codex receives only this notice, and VISTA receives the last frame before the restart. *Decision screens.* Four of the 120 tasks, all hostile real-time, declare game states in which the game waits for a choice. In real time, AutoPILOT and Codex poll the game state every 0.25 s. When such a screen appears, the running controller is stopped and the model receives a neutral notice. VISTA receives no such notice. No harness receives it with the clock paused. Removing the four tasks changes the real-time gap between AutoPILOT and VISTA from +30.3 to +30.2 points with GPT-6.1 Sol and from +23.3 to +21.6 with Claude Opus 5.5.

C Prompt

Each prompt starts with the game's description, task and controls, which are shared by all harnesses. VISTA then receives its official instructions, with one added sentence in real time: *"Real time continues*

while you think and while tools run; the game is not paused between your actions." AutoPILOT receives the same text followed by the rules block below, shown in its real-time version for the GPT-6.1 Sol runs. The prompts are reproduced verbatim and call environment steps "game inputs". Under the paused clock, the first paragraph instead states that the game is paused while the model thinks, and the last rule that a running controller is paused while the model thinks.

```
# Real-time play

The game keeps running while you think, call tools, and run Python.

## Budget and ending
- You have 2000 seconds of game time. Loading and automatic restarts are not counted.
- You can make at most 100 tool calls; a response without a tool call also counts as one. Actions sent by your running
      controller do not count toward this limit. All game inputs together are limited to 2000.
- A response without a tool call is submitted as a final answer and the episode continues. Only the game decides
     success.

## Choosing a tool
- Use `act` for single inputs: starting, quick reactions and recovery.
- Calling `act` stops a running controller.
- Use `run_macro` for a short sequence whose effect you have already seen.
- Use `python` for a controller that keeps playing from the screen while you think. Prefer `run_macro` when it is
     enough.
- Test an uncertain control briefly before you rely on it for longer.
- Decide what visible change counts as progress. Motion on screen and passing time are not progress.
- Let a controller keep running while you think only if it is already working. Otherwise have it hand control back and
      stop.
```

The Codex harness receives no instructions of ours beyond the game text and the following rules:

```
- The game runs in real time. It does not pause while you think or use tools.
- You have 2000 seconds of game time. Loading and automatic restarts are not counted.
- You can send at most 100 responses. Each `screenshot` or `act` call counts as one response, and so does a reply with
      no tool call.
- You can send at most 2000 game inputs. Each `act` is one input, including waits.
- A reply with no tool call is submitted as a final answer and the episode continues; it also uses one game input. The
      episode ends only on success, game end or budget end.
```

Later runs of AutoPILOT add three rules and end every tool result with a budget line such as `Budget left: 1999 of 2000 game inputs, 99 of 100 responses, 1989 of 2000 s game time.`

```
- Each `act`, each `run_macro` action and each `api.act` call uses one game input, including waits. A response without
      a tool call can also use one game input. Other tools and `api.observe()` use none.
- If any budget remains, keep playing: try a different approach instead of giving a final answer.
- Combine several waits, clicks or key presses into one `run_macro` call: each tool call uses one of your 100
     responses.
```

Later runs of Codex and VISTA add the keep-playing rule and a similar budget line.

D Baselines

Sec. C gives the real-time instructions of VISTA and the Codex harness. VISTA runs its official GameWorld profile and tools through the Codex app server of Codex CLI 0.159.2, like AutoPILOT. Its budget is 100 tool calls plus 2000 s of game time. Every tool call counts, including `inspect`, `read_pixels`, `history` and the note tools. The Codex harness runs Codex's own base instructions, agent loop and context compaction in a single thread per episode. Web search, browser and computer use, skills, plugins and sub-agents are disabled. Its code execution has no shell, file system or network access. The Claude Code harness runs Claude Code's default system prompt and agent loop with `act` and `screenshot` as its only tools. All built-in tools are disabled, and context-compaction turns do not count as model calls. The no-op agent sends no step from the start of an episode to its end, with the same automatic restarts and the same evaluator. Each episode runs for 600 s of game time. It wins no task. Its mean progress is .079 in hostile real-time environments and at most .002 in the other two, consistent with the type assignment of Sec. 4. The random agent sends a uniformly random valid step as fast as the harness allows, at a median of 2.9 steps per second. It runs one episode per task, within 2000 steps and 2000 s. It wins 5.8% of tasks, mostly in hostile real-time environments, where it wins 10.0%. Its mean progress is .258, against .027 for the no-op agent, so progress is best read against this floor.

E Benchmark details

Games. The 120 games run offline in a pinned Chromium image. Of these, 37 are built with HTML5 canvas, 21 with PICO-8, 22 with Unity WebGL, 17 with Flash through the Ruffle emulator, 13 with Godot, 9 with other WebAssembly engines and 1 with Phaser. Each game exposes its state to the evaluator through a small per-game adapter, which the agent cannot access. None of the 34 games of GameWorld is among them. We collected and wrapped all 120 games ourselves from freely playable web games. The manifest records the source of 112 of them and pins every game file by its hash. Of these, 94 are on itch.io, 10 on Armor Games, 3 on Kongregate, 3 on Newgrounds, 1 on Not Doppler and 1 on GitHub. Tab. 9 lists each game's author, source and licence. Eight games carry a licence stated by their author. The other 112 games state no licence, so we treat their terms as unknown.

Selection. From a pool of 568 collected games, each game was assigned an environment type from its description and idle probes. A task was admitted only after a scripted acceptance test that uses no model and runs with the same code and browser image as the reported runs. Its 13 checks test whether a scripted policy can win in real time. They also require that pausing through either harness stops the game clock and that resuming restores normal speed. Score, prompt and screen must agree, the game must show no undeclared freeze, and failures and restarts must be detected. Memory must stay within its limit, and 60 s of idling must cause no penalty in turn-based games but must cause one in hostile real-time games. The game's own pause must be blocked, and phases the agent cannot control must be recognized. The budget must be feasible, the prompt must name only allowed keys, and the game state must be sampled at a regular rate. The 120 tasks were frozen before any run reported here. For games whose danger appears only after some play, a second probe first plays a scripted policy for up to 120 s and then idles for 60 s.

Episode rules. All runs block the games' own pause and restart controls. From the moment a game reports a failure until it is playable again, all actions are discarded but still counted as steps, including those of a running controller. This time does not count as game time, and neither does loading or an automatic restart. Phases the agent cannot control, such as automatic cutscenes, do not count as game time either. A wall-clock safety cap of 4000 s ends any paused episode. Every real-time episode is audited against the game's own clock for pauses. An episode is scored by the benchmark's evaluator only. A run that fails for infrastructure reasons is retried automatically and not scored. A used-up budget is a failure. A model response that takes longer than the per-call deadline of 300 s, or a request that the model provider rejects as too large, counts as a failure of the episode. Tab. 8 lists all 120 games with their environment type and per-game results.

Hostile real-time (40) Game	VISTA	AutoPILOT
A Dance of Fire and Ice	0/3	**3/3**
Backstreet Warriors	0/3	0/3
Bad Monday Simulator	0/3	**2/3**
Cave Buster	0/3	**1/3**
Dead Estate CLASSIC	0/3	**3/3**
Deadbeats Encore	0/3	0/3
driftr	1/3	1/3
Eat not, Want not	0/3	**3/3**
Enigmata 2: Genu's Revenge	3/3	3/3
Epic Coaster	0/3	0/3
Fair Squares	1/3	**3/3**
Flooded Caves	3/3	3/3
Galactic Wars	2/3	**3/3**
Generic Fighter Maybe	0/3	**1/3**
Grave	0/3	**3/3**
Gun Mayhem	0/3	0/3
HIT8OX	2/3	**3/3**
Just One Boss	0/3	**1/3**
Lumberjack Run	3/3	3/3
MOBS, INC	0/3	**3/3**
Monolith	0/3	0/3
My Rusty Submarine	0/3	0/3
O (⊙)	0/3	**3/3**
Packed to the Gills	0/3	0/3
PortaBoy	1/3	**3/3**
PRAXIS FIGHTER X	0/3	**3/3**
Project: Intellisphere	0/3	**1/3**
Radius Raid	0/3	**3/3**
RetroFab	1/3	**3/3**
return to sender	0/3	0/3
SCARLET SHIFT	0/3	0/3
Seraph's Last Stand	0/3	**3/3**
Sharpest Spear	0/3	**3/3**
Sir Truck	0/3	**1/3**
Smash 'n Dab	0/3	**1/3**
Strike Force Heroes	3/3	3/3
Superfighters	0/3	**3/3**
Tenjutsu 48h	0/3	**2/3**
UZG	0/3	**1/3**
XENOPUNCH	0/3	**3/3**
Total	20/120	75/120

Benign real-time (40) Game	VISTA	AutoPILOT
Achievement Unlocked	3/3	3/3
APOLLO	0/3	0/3
CELESTE Classic	0/3	0/3
Chronotron	3/3	3/3
Crabjuice	0/3	**2/3**
Dank Tomb	0/3	0/3
Debbie's Diner Derby	0/3	**3/3**
Dusk Child	0/3	0/3
Egg	0/3	0/3
Eggdog Extended	0/3	0/3
Exit Path	0/3	**1/3**
Frogfall	0/3	**3/3**
Give Up	0/3	0/3
Golf Sunday	0/3	**2/3**
Interdimensional Spike	3/3	3/3
King of Drift	3/3	3/3
Lo-Fi Room	0/3	0/3
Looper	3/3	3/3
Moss Moss	3/3	3/3
Mr. Platformer	0/3	0/3
Offroaders 2	3/3	3/3
Öoo (demo)	3/3	3/3
pico sonic	0/3	**2/3**
PolyTrack	0/3	0/3
PUP	2/3	**3/3**
Ray Ardent Science Ninja	3/3	3/3
Reversal Rooms	2/3	**3/3**
SELF	0/3	**3/3**
SHIFT 2	3/3	3/3
SHRUBNAUT	0/3	0/3
Sky Run	2/3	**3/3**
Smol Ame	0/3	0/3
speedloop	0/3	**3/3**
Sulka	1/3	**3/3**
TEMPRES	0/3	**3/3**
The Sun And Moon	3/3	3/3
This Is The Only Level	3/3	3/3
Tiny Truck Racing	2/3	2/3
UFO Swamp Odyssey	0/3	0/3
William and Sly	0/3	0/3
Total	45/120	72/120

Table 8 The 120 games of AFK-Bench, by environment type, with real-time wins out of three episodes with GPT-6.1 Sol for VISTA and AutoPILOT (bold: higher). Titles follow the games' source pages. Tab. 9 lists sources and licences.

Turn-based (40) Game	VISTA	AutoPILOT
AGLOBO	0/3	**1/3**
awake	3/3	3/3
Blackout	0/3	0/3
Bloody Fun Day	1/3	**2/3**
Bob Velseb the Cannibal the Dating Simulator	3/3	3/3
Cavern Sweeper	3/3	3/3
Chessformer	0/3	**1/3**
Claustrowordia	3/3	3/3
Combo Pool	0/3	**3/3**
Coming Out Simulator 2014	3/3	3/3
Covemount	0/3	0/3
Curious Fishing	2/3	**3/3**
Dungeon Raid	0/3	**3/3**
FARA	**3/3**	2/3
Fossil Quest	3/3	3/3
Gnomitaire	3/3	3/3
Karawan	3/3	3/3
Katti	3/3	3/3
Kitsu	**2/3**	1/3
Last Chess Standing	3/3	3/3
Lowland Keepers	0/3	**3/3**
Mai-Chan's Sweet Buns	0/3	**3/3**
Mole Mole	0/3	**2/3**
Nip For Speed	3/3	3/3
Nono 1k	3/3	3/3
PancitoMerge	3/3	3/3
Patient Rogue	3/3	3/3
Peggy's Post	0/3	0/3
Pegote	3/3	3/3
Roguewood	1/3	**3/3**
Roulette Knight	0/3	**2/3**
Sebastian's Quest	0/3	**2/3**
Six-Sided Streets	2/3	**3/3**
Snug	2/3	**3/3**
Streamline	3/3	3/3
The Pigeon Post Principle	3/3	3/3
Tiny Tower Defense	2/3	**3/3**
Trichromat	2/3	**3/3**
Vanguard Monsters	3/3	3/3
Woodworm Classic	2/3	**3/3**
Total	73/120	100/120

Table 8 The 120 games of AFK-Bench (continued).

Game	Author	Src.	Lic.
Achievement Unlocked	John	A	–
A Dance of Fire and Ice	fizzd et al.	I	–
AGLOBO	Daniel Moreno	I	–
APOLLO	Daniel Moreno et al.	I	–
awake	droqen	I	–
Backstreet Warriors	SebaGamesDev	I	–
Bad Monday Simulator	Lumpy et al.	I	–
Blackout	FRESH	I	–
Bloody Fun Day	urbansquall	K	–
Bob Velseb the Cannibal the Dating Simulator	numijulie	I	–
Cave Buster	Jon Topielski et al.	I	–
Cavern Sweeper	Hempuli	I	–
CELESTE Classic	Maddy Makes Games et al.	I	–
Chessformer	rob1221	I	–
Chronotron	–	N	–
Claustrowordia	anttihaavikko	I	–
Combo Pool	NuSan	I	NC-SA
Coming Out Simulator 2014	Nicky Case	I	PD
Covemount	Hempuli	I	–
Crabjuice	dev_dwarf et al.	I	–
Curious Fishing	–	–	–
Dank Tomb	Jakub Wasilewski	I	–
Deadbeats Encore	fie	I	–
Dead Estate CLASSIC	Milkbar Lads	I	–
Debbie's Diner	–	–	–
Derby driftr	kindanice	I	–
Dungeon Raid	4Cats	I	–
Dusk Child	Sophie Houlden	I	–
Eat not, Want not	Sitrex et al.	I	–
Egg	Terry Cavanagh	I	–
Eggdog Extended	zamsire et al.	I	–
Enigmata 2: Genu's Revenge	kidgamez	K	–
Epic Coaster	knigg	K	–
Exit Path	John	A	–
Fair Squares	kindanice	I	–
FARA	BrianIsCreative	I	–
Flooded Caves	NuSan	I	NC-SA
Fossil Quest	Unsigned Double	I	–
Frogfall	Kultisti	I	–
Galactic Wars	Volcano Bytes	I	NC-ND
Generic Fighter Maybe	Astrobard Games et al.	I	–
Give Up	John	A	–
Gnomitaire	Arnold Rauers	I	–
Golf Sunday	Johan Peitz	I	–
Grave	Heartbeast	I	–
Gun Mayhem	KevinGu	A	–
HIT8OX	vladcomarlau	I	–
Interdimensional Spike	yaotu	I	–
Just One Boss	ayla˜nonsense	I	NC-SA
Karawan	–	–	–
Katti	Kultisti	I	–
King of Drift	–	A	–
Kitsu	Fáyer et al.	I	–
Last Chess Standing	FeatureKreep	I	–
Lo-Fi Room	Bearmask	I	–
Looper	FeatureKreep	I	–
Lowland Keepers	Twotinydice	I	–
Lumberjack Run	terrasima	I	MIT
Mai-Chan's Sweet Buns	Krystman	I	–
MOBS, INC	overboy	I	–
Mole Mole	Adam Saltsman	I	–
Monolith	samloeschen	I	–
Moss Moss	Noel Cody	I	–
Mr. Platformer	–	–	–
My Rusty Submarine	Sørb et al.	I	–
Nip For Speed	Knackelibang et al.	I	–
Nono 1k	NuSan	I	–
O (⊙)	sirmilkman	I	–
Offroaders 2	ArmorGames	A	–
Öoo (demo)	NamaTakahashi	I	–
Packed to the Gills	moonlightjammers et al.	I	–
PancitoMerge	Fáyer	I	–
Patient Rogue	watabou	I	–
Peggy's Post	digitarium	I	–
Pegote	jvolonte et al.	I	–
pico sonic	komehara	I	MIT/NC
PolyTrack	Kodub	I	–
PortaBoy	–	–	–
PRAXIS	SPARSE// GameDev	I	–
FIGHTER X	stylemistake	I	–
Project: Intellisphere			
PUP	mmatt_ugh	I	–
Radius Raid	Jack Rugile	G	MIT
Ray Ardent		A	–
Science Ninja	shaneneville		
RetroFab	Itizso	I	–
return to sender	kindanice	I	–
Reversal	Taelsdafoox	I	–
Rooms			
Roguewood	watabou	I	–
Roulette	Four Quarters team	I	–
Knight			
SCARLET SHIFT	sirmilkman et al.	I	–
Sebastian's Quest	Adam Saltsman	I	–
SELF	dev_dwarf et al.	I	–
Seraph's Last Stand	–	–	–
Sharpest Spear	NPJarcade	I	–
SHIFT 2	Tony	A	–
SHRUBNAUT	gleeson	I	–
Sir Truck	–	–	–
Six-Sided Streets	Chris Klimowski	I	–
Sky Run	bikas	A	–
Smash 'n Dab	Will Herring	I	–
Smol Ame	MooCow Games	I	–
Snug	Cookiecrayon	I	–
speedloop	kindanice et al.	I	–
Streamline	Francois van Niekerk	I	–
Strike Force Heroes	Sky9 Games	D	–
Sulka	Kultisti	I	–
Superfighters	–	N	–
TEMPRES	tak	I	–
Tenjutsu 48h	Deepnight Games	I	–
The Pigeon Post Principle	cassowary	I	–
The Sun And Moon	Daniel Linssen	I	–
This Is The Only Level	John	A	–
Tiny Tower Defense	BuilderBot	I	–
Tiny Truck Racing	–	–	–
Trichromat	spratt	I	–
UFO Swamp Odyssey	Paranoid Cactus	I	–
UZG	nex	I	–
Vanguard Monsters	Jay	I	–
William and Sly	–	N	–
Woodworm Classic	spratt	I	–
XENOPUNCH	overboy	I	–

Table 9 Sources and licences of the 120 games. Author: the first author or uploader named on the source page. Src.: I itch.io, A Armor Games, K Kongregate, N Newgrounds, D Not Doppler, G GitHub, – no source page recorded. Lic.: the licence the author states on the source page, in the source repository or in a licence file shipped with the game. MIT is the MIT licence (code), NC, NC-SA and NC-ND are CC BY-NC 4.0, CC BY-NC-SA 4.0 and CC BY-NC-ND 4.0 (assets or cartridge), MIT/NC is MIT code with CC BY-NC 4.0 assets, PD is a public-domain dedication, and – means that no licence is stated (unknown).

F Additional results

F.1 Main results and their reliability

Runs. The configurations with GPT-6.1 Sol in Tab. 10 and VISTA with GPT-6.1 Sol under the paused clock consist of three runs over all 120 tasks. All other configurations are run once per task.

Harness (real time, GPT-6.1 Sol)	run 1	run 2	run 3	mean
Codex	39.2	37.5	40.0	38.9
VISTA	37.5	39.2	38.3	38.3
AutoPILOT	69.2	68.3	68.3	68.6
No controller	66.7	65.0	62.5	64.7
No macro	65.0	70.0	67.5	67.5
AutoPILOT, high effort	71.7	72.5	70.8	71.7
AutoPILOT, direct API	68.3	65.8	69.2	67.8
AutoPILOT, paused clock	71.7	75.8	71.7	73.1

Table 10 Success rate (%) in each of the three runs, with random seeds 42, 43 and 44. Each run covers 120 tasks.

Intervals and paired tests. Tab. 11 gives a 95% bootstrap interval over tasks for the overall success rate of each harness with GPT-6.1 Sol and Claude Opus 5.5, and exact McNemar tests on paired episodes of the same task for the comparisons in Sec. 5.

By game genre. Tab. 12 breaks the environment types of Tab. 2 down into game genres.

Model, clock	Harness	SR (%)	95% interval	Run std
GPT-6.1 Sol, paused	VISTA	56.1	[48.3, 63.9]	1.0
	AutoPILOT	73.1	[65.8, 80.0]	2.4
GPT-6.1 Sol, real time	Codex	38.9	[30.8, 47.2]	1.3
	VISTA	38.3	[30.3, 46.4]	0.8
	AutoPILOT	68.6	[61.1, 76.1]	0.5
Claude Opus 5.5, paused	VISTA	39.2	[30.8, 48.3]	–
	AutoPILOT	60.8	[51.7, 69.2]	–
Claude Opus 5.5, real time	VISTA	30.0	[21.7, 38.3]	–
	AutoPILOT	53.3	[44.2, 62.5]	–

Comparison (A *vs.* B)	Pairs	A only	B only	p
GPT-6.1 Sol, AFK-Bench				
AutoPILOT *vs.* VISTA, real time	360	111	2	10^{-30}
Codex *vs.* VISTA, real time	360	18	16	0.86
AutoPILOT *vs.* VISTA, paused	360	71	10	2×10^{-12}
VISTA paused *vs.* real time	360	82	18	6×10^{-11}
AutoPILOT paused *vs.* real time	360	34	18	0.036
AutoPILOT *vs.* No controller	360	28	14	0.044
AutoPILOT *vs.* No macro	360	23	19	0.64
High *vs.* low effort	360	27	16	0.13
Direct API *vs.* Codex app server	360	20	23	0.76
No controller & macro *vs.* VISTA	120	12	2	0.013
hostile real-time only	40	3	1	0.62
AutoPILOT *vs.* No controller & macro	120	30	2	2.5×10^{-7}
hostile real-time only	40	18	1	8×10^{-5}
Relaxed *vs.* standard budget	120	13	2	0.007
4000 s *vs.* 2000 s	120	7	5	0.77
4000 *vs.* 2000 steps	120	9	7	0.80
Claude Opus 5.5, AFK-Bench				
AutoPILOT *vs.* VISTA, real time	120	30	2	2.5×10^{-7}
AutoPILOT *vs.* VISTA, paused	120	30	4	6×10^{-6}
High *vs.* low effort, real time	120	24	5	5×10^{-4}
GPT-6.1 Sol, GameWorld				
AutoPILOT *vs.* VISTA, real time	170	63	0	2×10^{-19}
AutoPILOT paused *vs.* real time	170	9	4	0.27
AutoPILOT *vs.* VISTA, paused	170	7	9	0.80
VISTA paused *vs.* real time	170	72	2	3×10^{-19}

Table 11 Intervals and paired tests. Top: success rate with a 95% bootstrap interval over tasks and the std over the three runs. The bootstrap uses 4,000 resamples and averages the episodes of a task. A dash marks a single run. Bottom: pairs won by only one side, and the two-sided exact McNemar p.

		GPT-6.1 Sol		Claude Opus 5.5	
Genre	Tasks	VISTA	AutoPILOT	VISTA	AutoPILOT
Hostile real-time					
Shooter	18	24.1	63.0	11.1	55.6
Fighting	11	6.1	72.7	18.2	36.4
Action (other)	11	15.2	51.5	27.3	45.5
Benign real-time					
Platformer	29	42.5	62.1	37.9	48.3
Racing & other	11	24.2	54.5	0.0	27.3
Turn-based					
Puzzle	20	63.3	83.3	45.0	70.0
Card & strategy	9	55.6	92.6	44.4	77.8
Story, sim & other	11	60.6	75.8	45.5	63.6

Table 12 Success by game genre in % in real time at low effort, averaged over the episodes of a task and then over tasks. Genres come from a keyword rule on each game's description, and genres with fewer than six tasks join "other".

F.2 Other models

Both harnesses with four more models. Tab. 2 includes VISTA and AutoPILOT with three open-source models, DeepSeek-V4.1-Flash, GLM-5.3-Flash and Hy4-flash. It also includes one more closed model, Qwen4-Max. We access them through OpenAI-compatible APIs and run them through the same Codex app server as GPT-6.1 Sol. DeepSeek-V4.1-Flash, GLM-5.3-Flash and Qwen4-Max think at low effort. Hy4-flash has a single thinking mode. Budgets, tasks and rules are those of the main experiments, with one episode per task. AutoPILOT wins more tasks than VISTA with every model under both clocks, and every comparison in Tab. 13 is significant. Pausing the clock helps these models much less than it helps VISTA with GPT-6.1 Sol, and only Qwen4-Max gains clearly with both harnesses. The gap between the harnesses thus remains when the game waits, so with these models it does not come mainly from the running clock.

Turn-based environments. AutoPILOT also leads with these models in turn-based environments, where latency costs nothing. Much of the gain thus comes from executing a whole plan per model call rather than from avoiding latency.

GameWorld. On GameWorld, AutoPILOT also wins far more tasks than VISTA with Qwen4-Max and Hy4-flash, as Tab. 14 shows.

Provider limits and cost. These providers reject requests above a size limit. VISTA sends its visual memory and AutoPILOT its recent frames with each request, so long episodes can exceed it, and such episodes count as failures. With Qwen4-Max and DeepSeek-V4.1-Flash, a real-time episode costs well under a dollar with either harness.

Model	Clock	VISTA	AutoPILOT	Only one wins	p
DeepSeek-V4.1-Flash	real time	20.0	39.2	1 / 24	2×10^{-6}
	paused	24.2	44.2	1 / 25	8×10^{-7}
Qwen4-Max	real time	14.2	27.5	1 / 17	1.5×10^{-4}
	paused	25.0	41.7	2 / 22	4×10^{-5}
GLM-5.3-Flash	real time	3.3	18.3	0 / 18	8×10^{-6}
	paused	1.7	26.7	1 / 31	2×10^{-8}
Hy4-flash	real time	3.3	32.5	0 / 35	6×10^{-11}
	paused	5.8	38.3	1 / 40	4×10^{-11}

Table 13 Paired tests for the four further models on AFK-Bench. Each model plays one episode on each of the 120 tasks. Success rates by environment type are in Tab. 2. Only one wins: tasks won only by VISTA/ only by AutoPILOT. p: two-sided exact McNemar test.

Model	VISTA	AutoPILOT	Only one wins	p
GPT-6.1 Sol	42.9 .653	80.0 .900	0 / 63	2×10^{-19}
Claude Opus 5.5	41.2 .623	67.6 .845	6 / 51	6×10^{-10}
Qwen4-Max	27.6 .515	57.1 .758	3 / 53	8×10^{-13}
Hy4-flash	20.0 .399	51.8 .713	0 / 54	1×10^{-16}

Table 14 Other models on GameWorld. Success rate (%) with mean progress in small type, in real time with one episode on each of the 170 tasks. Only one wins: tasks won only by VISTA/ only by AutoPILOT.

Further models with AutoPILOT only. Tab. 15 runs only AutoPILOT in real time with two more models, GPT-6 Astra [27] and Claude Fable 5.2 [1], on all 120 tasks with one episode per task. Both do at least as well as GPT-6.1 Sol (Tab. 2).

Model	Effort	Hostile real-time	Benign real-time	Turn-based	Overall
GPT-6 Astra	low	70.0 .822	62.5 .781	87.5 .934	73.3 .846
	high	70.0 .841	67.5 .820	87.5 .941	75.0 .867
Claude Fable 5.2	low	70.0 .810	60.0 .791	85.0 .945	71.7 .849
	high	75.0 .862	55.0 .777	85.0 .926	71.7 .855

Table 15 Further models with AutoPILOT only. All runs are in real time with one episode per task. Small type gives mean progress.

Harness	Hostile real-time	Benign real-time	Turn-based	Overall	Model calls	Tokens	Cost / ep.
Codex	17.5	36.7	62.5	38.9	81	5.1M	$1.35
VISTA	16.7	37.5	60.8	38.3	80	5.3M	$1.50
AutoPILOT	**62.5**	60.0	**83.3**	**68.6**	**43**	**2.8M**	**$0.89**
No controller	50.8	60.0	**83.3**	64.7	58	4.6M	$1.31
No macro	**62.5**	**61.7**	78.3	67.5	46	3.1M	$0.91
No controller & macro	22.5	45.0	70.0	45.8	81	5.5M	$1.45

Table 16 Ablations and cost for all environment types. All runs use GPT-6.1 Sol in real time. Tokens include input, output, cached and reasoning tokens. Cost is at list price. VISTA's model calls are its tool calls.

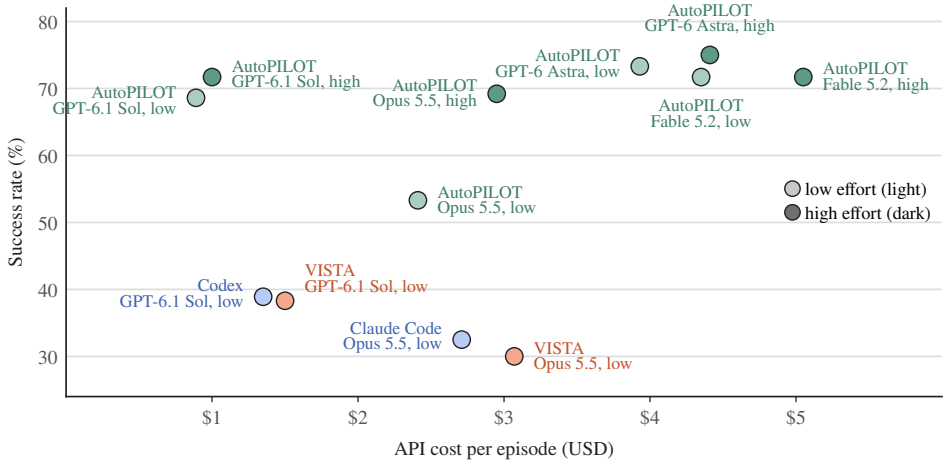

Figure 6 Success against cost on AFK-Bench in real time. Color marks the harness, light and dark shades low and high effort. Cost per episode at list price. Claude Fable 5.2 cache prices assume Fable 5.1 ratios.

F.3 Tool use, ablations and cost

Ablations, all environment types. Tab. 16 extends Tab. 4 to all environment types. In the No controller & macro variant, every failed episode runs out of its 100 model calls, and the model takes 77 steps per episode, against 659 for the full AutoPILOT on the paired episodes.

Cost against success. Fig. 6 plots success against API cost per episode for AutoPILOT with four models at low and high effort and for the baselines at low effort. It omits the models of Sec. F.2 and VISTA at high effort.

Cost with Claude Opus 5.5. With Claude Opus 5.5 at Claude list prices, a real-time episode costs 21% less with AutoPILOT than with VISTA, as Tab. 17 shows. A success costs about $4.53 against $10.25. With the clock paused it costs $4.49 against $7.22, and AutoPILOT at high effort pays $4.27 per success. Part of the difference comes from outcomes, since lost episodes use more of the budget.

Successful and failed episodes. Of the 360 real-time AutoPILOT episodes with GPT-6.1 Sol in Tab. 2, 247 succeed and 113 fail (Tab. 18). Successful AutoPILOT episodes end early, with 32 model calls and 335 steps on average, against 66 model calls and 1436 steps in failed ones. Failed episodes rely more

Setting	Model	AutoPILOT	VISTA	Both win	Both lose
AFK-Bench, real time	GPT-6.1 Sol	0.89	1.50	0.42 / 0.68	1.65 / 2.01
	Claude Opus 5.5	2.41	3.07	0.94 / 1.27	3.89 / 3.80
AFK-Bench, paused	Claude Opus 5.5	2.73	2.83	0.86 / 1.37	4.88 / 3.72
GameWorld, real time	GPT-6.1 Sol	0.56	1.35	0.20 / 0.39	1.35 / 2.06
	Claude Opus 5.5	2.02	2.53	0.41 / 0.67	4.37 / 3.91

Table 17 Cost per episode in $ at list price. Both win and both lose: paired episodes that both harnesses win or both lose, given as AutoPILOT/ VISTA.

on `python`, with 35% of calls against 20%, and less on `act`, with 34% against 51%. Their controllers send 93% of their steps against 84%. When a game goes badly, the model keeps rewriting its controller. This also holds within tasks: among the 23 tasks with both successes and failures across their three episodes, the failed episodes have the higher `python` share in 19. The controller share thus reflects how the model reacts to trouble, rather than which tool wins.

	Tool calls (%)				Steps (%)		
Episodes	`act`	macro	`python`	other	model	macro	ctrl.
Won (247)	51	24	20	5	5	11	84
Lost (113)	34	27	35	5	2	5	93

Table 18 Tool use in successful and failed episodes of AutoPILOT with GPT-6.1 Sol in real time, pooled over episodes. Here "other" means read-only tools and "ctrl." means controllers.

Who is playing when. In the direct-API runs, a controller plays alone for 55.5% of game time, the model thinks while no controller runs for 40.3%, and both happen at once for 1.5%, which is 3.5% of inference time. With the Codex app server, a controller runs during 2.3% of inference time. In hostile real-time environments the first two shares are 70.1% and 27.4%, and the overlap is 6.2% of inference time. Of the 3,239 automatic restarts, 74% happen while a controller plays alone, 24% while the model thinks, and 1% while both are active. A model call takes 6.1 s at the median, 10.7 s at the 75th percentile and 18.1 s at the 90th. A controller step takes 0.41 s at the median. The action itself takes about half of it and the model decides its duration. The screenshot and state read that follow take the rest. Handing off while the controller keeps playing is supported but rarely used: in 83% of handoffs the controller stops within 1 s. In a variant that kept the controller running and woke the model every 120 s, tested on 36 tasks, the share of inference time during which a controller kept playing rose from 7.8% on these tasks to 26% without more wins. It won 7 of the 36 tasks, against 8 in a standard run on the same tasks.

Open-loop controllers. To test whether controllers need to read the screen while they run, we run AutoPILOT with GPT-6.1 Sol with controllers that cannot read the screen. The run plays one episode per task. In it, `api.observe()` is refused, and `api.act()` reports only success and game end. The model still sees the frames when it writes a controller, so such a controller is a timed program planned in advance, like a macro with loops and timing. Without frames, the controller has no screen watch, and its tool description is adapted accordingly. Macros, the model's own frames, restart notices and budgets are unchanged. Programs that cannot read the screen at run time still win 78 of 120 tasks, against 83 for the full harness and 80 without controllers. Exact McNemar tests give $p = 0.30$ and $p = 0.79$. Against the full harness, 5 tasks are won only by the open-loop version and 10 only by the full one. By environment type, they win 23, 25 and 30 in hostile real-time, benign real-time and turn-based environments, against 26, 23 and 34 for the full harness. Most of the gain over VISTA thus comes from delegating timed action sequences, which macros already provide. Reading the screen at run time adds five wins, a difference that is not significant.

Shorter context. To test whether AutoPILOT needs the full context history, we cap its context window at 128k tokens, so that Codex's context compaction triggers at 56k tokens. The run uses GPT-6.1 Sol with one episode per task on the same tasks as the first main-table run. It wins 76 of 120 tasks against

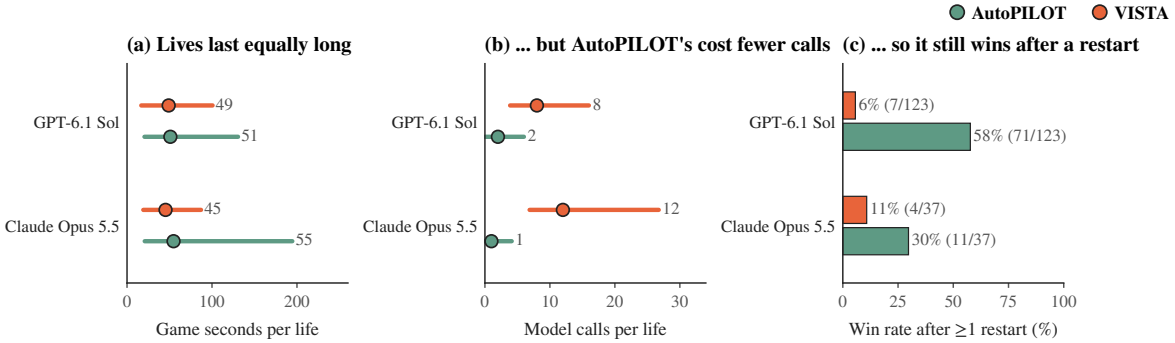

Figure 7 Lives and restarts in real time with GPT-6.1 Sol and Claude Opus 5.5. A *life* is the playable game time between consecutive restarts. The last, censored segment of each episode is excluded from (a) and (b). (a) Game seconds and (b) model calls per life: median over episodes of each episode's median life, with interquartile range, over episodes with at least one restart. (c) Success on paired episodes in which both harnesses restarted: GPT-6.1 Sol 71/123 *vs.* 7/123 ($p \approx 10^{-19}$) and Claude Opus 5.5 11/37 *vs.* 4/37 ($p = 0.039$).

83 for the uncapped run ($p = 0.118$), with 4 tasks won only with the cap and 11 only without. Among the 72 tasks whose context exceeded 56k tokens in the uncapped run, it wins 35 against 39 ($p = 0.34$). Neither difference is significant.

F.4 Failures and restarts

How episodes end. In real time, all 113 failed episodes of AutoPILOT end with a used-up budget: game time in 44, steps in 42 and model calls in 27. Of VISTA's 222 failed episodes, 208 end by exhausting its 100 model calls and 14 end earlier, before any budget is used up. VISTA spends 544 s of game time per episode on average and AutoPILOT 829 s. With 4000 s of game time, only 3 of AutoPILOT's successes come after 2000 s, and its remaining failures run out of steps in 19 episodes and of model calls in 16, rather than of time.

Restarts. With GPT-6.1 Sol in real time and counting every automatic restart, AutoPILOT restarts 9.1 times per episode on average and VISTA 7.9. With Claude Opus 5.5, they restart 9.3 and 5.1 times. With this count, as in Fig. 4a, AutoPILOT wins 45.3% of its 360 episodes without any restart and VISTA 35.0%. Within 20 restarts, they reach 65.8% and 38.1%, and 54.2% and 15.8% in hostile real-time environments. With GPT-6.1 Sol, one episode per task and three times the model calls, VISTA rises to 55.0% overall and 37.5% in hostile real-time environments, as Tab. 19 shows. This is still below AutoPILOT with the standard budget, at 69.2% overall and 65.0% in hostile real-time environments. AutoPILOT itself gains little from 300 model calls and reaches 71.7%.

Lives. Fig. 7 splits episodes into lives at each restart. A life lasts about as long under both harnesses. Its median is 51 *vs.* 49 s with GPT-6.1 Sol and 55 *vs.* 45 s with Claude Opus 5.5. However, an AutoPILOT life costs 2 model calls against 8 for VISTA with GPT-6.1 Sol, and 1 against 12 with Claude Opus 5.5. After a failure, VISTA thus has fewer model calls left for the next attempt. On the paired episodes of Fig. 7c, it used up its 100-call budget in 110 of 123 with GPT-6.1 Sol and in 29 of 37 with Claude Opus 5.5, against 5 and 4 for AutoPILOT. So after an automatic restart, AutoPILOT still wins 58% of paired episodes against 6% for VISTA with GPT-6.1 Sol, and 30% against 11% with Claude Opus 5.5.

Success over game time. Fig. 8 shows the share of all launched episodes that have succeeded by each moment of game time, with GPT-6.1 Sol at low effort in real time. Failed episodes stay in the denominator. No harness wins within 30 s, and the earliest win comes after 32 s. At 2 and 5 min, AutoPILOT has won 12.5% and 30.8% of all episodes, AutoPILOT without controllers, labeled no python in the figure, 11.9% and 29.7%, and VISTA 8.3% and 20.6%. In hostile real-time environments,

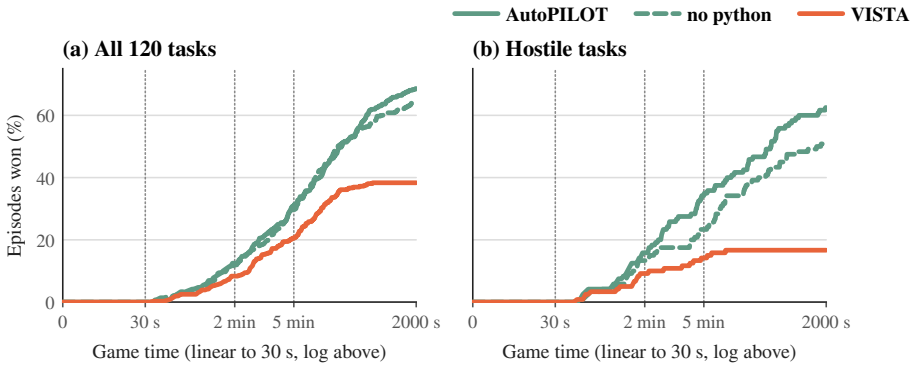

Figure 8 Success over game time with GPT-6.1 Sol at low effort in real time. Panel (b) shows hostile real-time only. Share of all episodes won by game time t. Failures stay in the denominator, so the right end equals the success rate. Dotted lines mark 30 s, 2 min and 5 min.

Budget	Limits	Hostile real-time	Benign real-time	Turn-based	Overall
Standard	100 / 2000 / 2000 s	65.0 .806	57.5 .761	85.0 .918	69.2 .828
3× model calls	300 / 2000 / 2000 s	60.0 .757	70.0 .807	85.0 .924	71.7 .829
2× steps	100 / 4000 / 2000 s	67.5 .810	62.5 .801	82.5 .907	70.8 .839
2× game time	100 / 2000 / 4000 s	65.0 .812	60.0 .772	87.5 .919	70.8 .834
Relaxed	300 / 4000 / 4000 s	77.5 .871	70.0 .846	87.5 .942	78.3 .886
VISTA					
Standard	100 / –[§] / 2000 s	17.5 .429	32.5 .576	62.5 .777	37.5 .594
3× model calls	300 / –[§] / 2000 s	37.5 .557	50.0 .668	77.5 .905	55.0 .710
Relaxed	300 / –[§] / 4000 s	30.0 .535	45.0 .673	80.0 .918	51.7 .708

Table 19 Scaling the budget, by environment type with GPT-6.1 Sol in real time on 120 tasks, with mean progress in small type. Limits are model calls / steps / game time. [§]Not applicable to VISTA, which sends one step per model call.

the numbers are 15.8% and 34.2%, 13.3% and 23.3%, and 9.2% and 14.2%. The gap opens late rather than early: AutoPILOT leads VISTA by 4.2 points at 2 min, 10.2 at 5 min and 30.3 at 2000 s. In hostile real-time environments the lead is 6.6, 20.0 and 45.8 points. This is because AutoPILOT keeps winning after VISTA has run out of model calls. VISTA's last win comes at 1014 s, and at 419 s in hostile real-time environments.

F.5 Scaling with budget, effort and speed

Budget. Tab. 19 breaks down the budget variants of Tab. 5 by environment type. Relaxing one limit at a time changes little for AutoPILOT. Relaxing all three together helps, mostly in hostile real-time environments, and the gain is significant with exact McNemar $p = 0.007$. VISTA gains from more model calls but not from more time or steps. Under the relaxed budget AutoPILOT stays far ahead, so the gap between the harnesses does not come from the budget.

Reasoning effort and model interface. Calling GPT-6.1 Sol through a direct API loop instead of the Codex app server changes success little, as Tab. 10 shows. Higher reasoning effort helps Claude Opus 5.5 clearly and GPT-6.1 Sol only a little. With Claude Opus 5.5, high effort wins 83 instead of 64 of the 120 tasks, with exact McNemar $p = 5 \times 10^{-4}$, and roughly doubles the output of each model call. Claude Fable 5.2 wins the same number of tasks at both efforts, as Tab. 15 shows. With the clock paused and high effort, GPT-6.1 Sol wins 92 of 120 tasks and Claude Opus 5.5 84.

Generation speed. The study of Fig. 4c controls latency directly, and Tab. 20 gives its numbers. The game is frozen while the model generates and then runs for $L = N/v$ seconds for N output tokens

	Generation speed (tokens/s)					
Model	10	25	50	100	200	Paused
GPT-6.1 Sol, high	55.0 .738	61.3 .762	68.8 .819	66.3 .809	66.3 .805	71.3 .846
GPT-6.1 Sol, low	55.0 .760	56.3 .762	61.3 .782	63.8 .798	63.8 .799	66.3 .821
Claude Opus 5.5, high	31.3 .522	42.5 .658	46.3 .658	58.8 .774	62.5 .770	63.8 .810
Claude Opus 5.5, low	36.3 .604	43.8 .622	45.0 .661	50.0 .703	48.8 .677	52.5 .729

Table 20 Success rate at controlled generation speed for AutoPILOT on 80 hostile real-time and benign real-time tasks, with mean progress in small type. The game is frozen during generation and then runs for N/v s for N output tokens.

before the model call takes effect, with no fixed overhead. The study uses the 80 benign real-time and hostile real-time tasks and the standard budget. Each episode is checked for timing errors and repeated if a check fails. Success rises with generation speed and levels off above about 100 tokens/s. The trend is clear within tasks, with $p = 0.005$ in a conditional logistic regression with tasks as strata, and weaker when pooled over tasks. Without synthetic latency the same tasks give about the success of the 50 tokens/s setting, and pausing the clock adds only a little more. Faster models will thus narrow but not remove the cost of thinking in real time, and AutoPILOT already recovers most of what pausing the clock would give.

Generation speed, other models. Tab. 20 also covers high effort and Claude Opus 5.5, and all four settings win more with faster generation. The latency of a model call counts all its output tokens, including reasoning, so a model that writes more per call waits longer at the same speed. Claude Opus 5.5 writes about twice as many tokens per call as GPT-6.1 Sol, yet at matched waiting time GPT-6.1 Sol still wins more. At 10 tokens/s, Claude Opus 5.5 loses almost all the tasks that only GPT-6.1 Sol wins because game time runs out.

F.6 GameWorld

With GPT-6.1 Sol and the clock paused, VISTA succeeds in 84.1% of GameWorld tasks and AutoPILOT in 82.9%. VISTA alone wins 9 tasks and AutoPILOT alone 7, as Tab. 11 shows. With Claude Opus 5.5, VISTA succeeds in 67.1% with the clock paused and 41.2% in real time, and AutoPILOT in 72.4% and 67.6%. Running the clock thus costs VISTA 25.9 points and AutoPILOT 4.7. Tab. 14 adds two more models from Sec. F.2 in real time.

GameWorld tasks are shorter than those of AFK-Bench. With the clock paused, VISTA wins them with a median of 28 model calls against 45 on AFK-Bench. Of its GameWorld wins, 55% take at most 30 model calls, against 33% on AFK-Bench. Under the same clock, AutoPILOT wins them with a median of 18 model calls and 55 steps, against 30 and 117 on AFK-Bench. Of its GameWorld wins, 66% take at most 100 steps, against 46% on AFK-Bench.

In 60 s trials, 34 of the 170 GameWorld tasks can also be completed by random actions or by doing nothing. We keep them for comparability with prior work.

G Case studies

Recorded episodes. Tab. 21 summarizes three recorded real-time episodes with GPT-6.1 Sol. Figs. 9 and 10 replay two of them in detail, one for each way a single-step harness fails. Frames, quotes, timings and code come from the recorded model outputs and action logs.

Radius Raid: enough skill, not enough time. In this hostile real-time game, VISTA strafes once per model call and reaches 88% of the goal by 3:26, as Fig. 9 shows. But enemies drain its health while it thinks, and it fails 12 times. Every restart resets progress, so its 100 model calls run out at 10:24 with a best progress of 90%. AutoPILOT turns on autofire with one `act`, and its first controller fails to send actions. The model reads the failure from the next frame and writes a second controller that

Game	VISTA	AutoPILOT
PRAXIS FIGHTER X hostile real-time	One action per model call, each after several seconds of thinking. The ship is often destroyed before the action lands. 30 failures in 100 tool calls. Best progress 25%. **Lost.**	A macro ends with the first failure at 0:14, and two more follow by 0:44. At 0:32 the model writes the screen-driven controller of Fig. 2b. It takes over at 0:44, plays 103 steps without another failure and wins at 1:26, after 6 model calls in total. **Won.**
Radius Raid hostile real-time	One strafe per model call. Health runs out 12 times. Model call 100 ends the episode at 10:24. **Lost.**	One `act` turns on autofire. Two model calls write a targeting controller that takes 58 of 59 steps. Won at 1:02 with 3 model calls, no loss. **Won.**
Trichromat turn-based	One tile per model call. Solves two of three rooms at 3:06 and 9:10. Model call 100 arrives in the last room at 14:32. **Lost.**	Writes each room as a Python model and searches it, then reports "found a 25-move solution". Tests one move, then executes the rest with controllers and one macro. 180 of 182 steps come from the worker. Won at 3:16 with 14 model calls. **Won.**

Table 21 Three recorded episodes with GPT-6.1 Sol in real time. Times are game time. Recorded separately from the main-table runs.

segments the ship and the enemies by color, aims at the nearest enemy and orbits the arena while avoiding contact. Then the model waits. The controller takes 58 of the episode's 59 steps and reaches the goal at 1:02, after three model calls and without a failure.

Trichromat: enough time, not enough model calls. Waiting costs nothing in this turn-based puzzle, so latency is not the problem. The problem is the model-call budget, as Fig. 10 shows. VISTA spends one model call per move and solves two of the three rooms, at 3:06 and 9:10, before its 100th call arrives in the last room. AutoPILOT writes the first room as a Python model and searches it before moving, tests the first move of the 25-move solution with `act`, and executes the rest with one controller. When its model of the second room has no solution, it tests the unknown mechanic, a floor button that opens a door, revises the model, and again executes the plan in bulk. It solves all three rooms by 3:16 with 14 model calls. Controllers and one macro take 180 of its 182 steps.

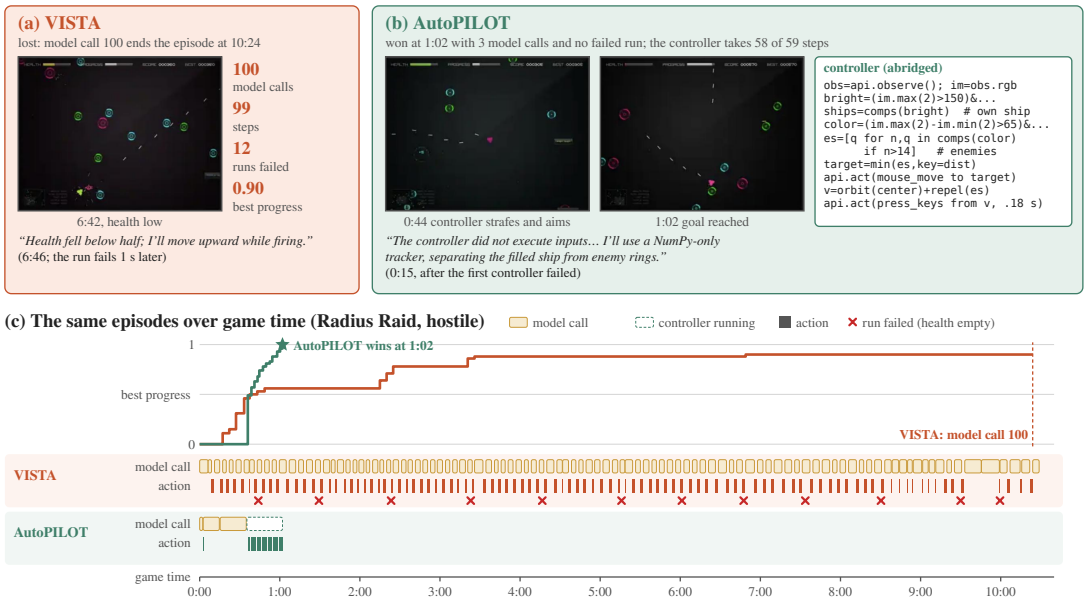

Figure 9 Case study: Radius Raid, a hostile real-time game, with GPT-6.1 Sol in real time. (a) VISTA at 6:42, shortly before its eighth failure. (b) An abridged version of AutoPILOT's controller, which tracks the ship and enemies by color. (c) Best progress, model calls, steps and failures over game time. VISTA loses its progress at every failure, while AutoPILOT wins at 1:02.

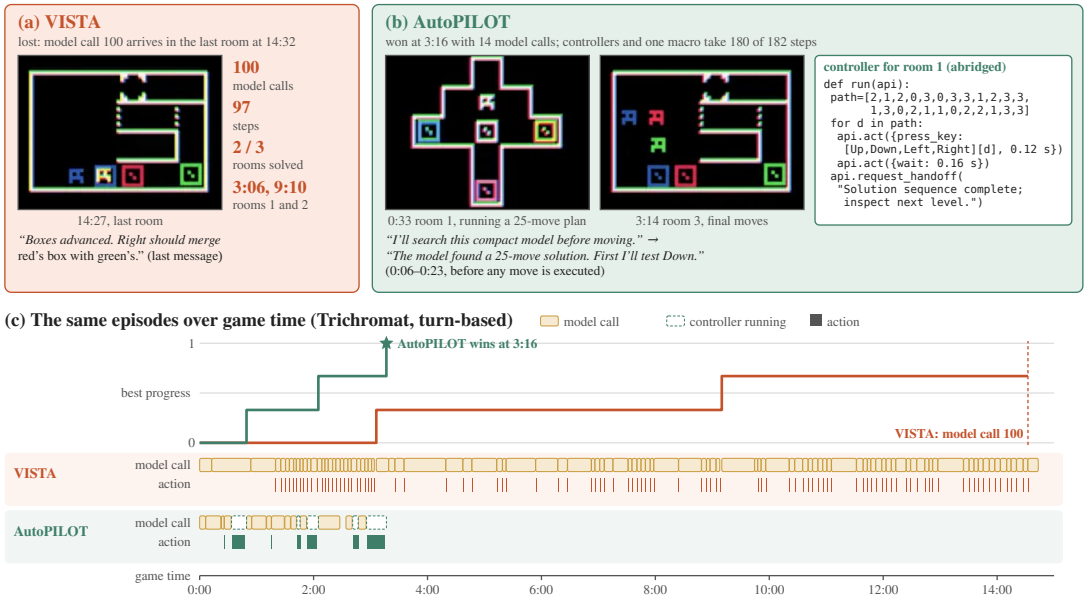

Figure 10 Case study: Trichromat, a turn-based game, with GPT-6.1 Sol in real time. (a) VISTA moves once per model call and reaches the last room with its last calls. (b) AutoPILOT searches a Python model of each room and executes the solution with controllers, shown abridged for room 1. (c) Best progress, model calls, controller runs and steps over game time. Each room adds a third to progress.

H Limitations

AutoPILOT relies on model-written perception and control code, which is easier in 2D browser games than in cluttered 3D scenes. In hostile real-time environments most of its wins come after a restart, so winning an unseen real-time game at the first attempt remains open. Models also rarely let a controller keep playing after a handoff, and doing so by default did not add wins, as Sec. F.3 reports.