MyPCBench: A Benchmark for Personally Intelligent Computer-Use Agents

Anonymous authors
Paper under double-blind review

Abstract

Current benchmarks for computer-use agents evaluate models in impersonal environments. This leaves a gap between evaluation and deployment where personal assistants are expected to work across a user's whole digital life, including their context, historical data, and logged-in accounts. This gap is widest on web tasks, where live web evaluations cannot exercise sites that require logging in or personal information, the kind of site a real personal assistant has to drive. We introduce MyPCBench, which tests computer-use agents as personal assistants on a Linux desktop populated with 17 simulated real-world web applications and a full desktop stack, all seeded for one canonical persona, Michael Scott from *The Office*[1]. We define 184 tasks in this environment, each inspired by a real request drawn from the OpenClaw community, and evaluate ten closed and open-weight models, each given the same computer and shell tools. The highest perfect rate is 58.2% (Claude Opus 4.6), with GPT-5.6-sol following at 55.4%. We grade trajectories with an LLM-as-a-judge that agrees with human grading on 94.5% of sampled rubric items. Model failures cluster on tasks that span many applications and on long trajectories, where personalization stresses an assistant the most. We release the environment, task set, agent harness, and judge anonymously at `https://anonymous.4open.science/r/MyPCBench-21FF/`.

1 Introduction

A person's computer is not a blank slate. Bank transactions, calendar events, email threads, travel bookings, and work chats accumulate across many applications into a working and personal record. Current benchmarks for computer-use agents ignore this, with tasks running against empty desktops, generic application states, and minimally seeded databases. An agent that can place a delivery order but cannot find which restaurant the user actually orders from every Friday has not demonstrated useful capabilities as a personal assistant. As LLM-based assistants for personal computers move from research demos to consumer products (e.g., OpenClaw (Steinberger, 2026) and Claude Cowork (Anthropic, 2026a)), evaluation must test whether these systems are actually personal and whether personalization improves or regresses with each model release.

Existing agent benchmarks span the web (Zhou et al., 2024; Koh et al., 2024; Deng et al., 2023; He et al., 2024), desktops (Xie et al., 2024; Bonatti et al., 2025; Yang et al., 2025), enterprise platforms (Drouin et al., 2024), and mobile devices (Jang et al., 2026b; Rawles et al., 2025). Most are *simulated* so that grading is deterministic and reproducible. The cost of that reproducibility is impersonality. Each application carries only the data the current task literally needs, and there is no user history behind it. Live-web evaluations avoid sites that require logging in or personal information, and the benchmarks that do provide logged-in states, such as the self-hosted WebArena sites and AppWorld's API-level accounts, seed generic or minimal personal history rather than a deep, cross-application identity. This rules out a large fraction of what real users ask their assistants to do.

No prior benchmark seeds a coherent user identity at the scale of a full personal computer. AppWorld comes closest, but it seeds accounts at the API layer rather than in a lived-in desktop (§2). iOSWorld realistically seeds a personalized iPhone. MyPCBench closes this gap. A single persona specification and a deterministic multi-application generator produce a desktop environment that

[1] *The Office* (US), an American mockumentary sitcom developed by Greg Daniels and aired on NBC, 2005–2013. `https://www.imdb.com/title/tt0386676/`.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

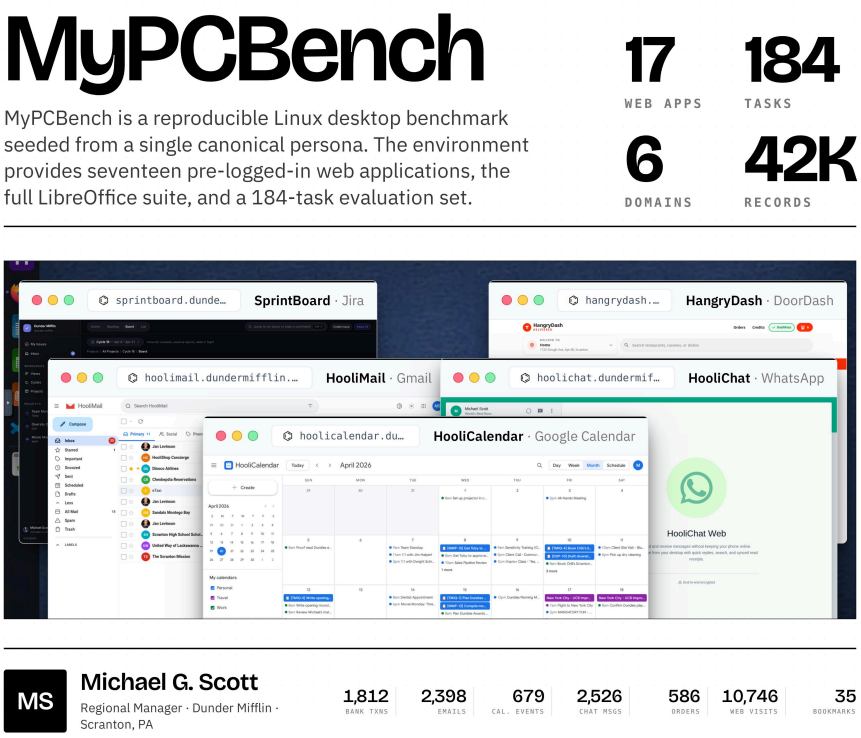


Figure 1: **Overview of MyPCBench.** A reproducible Linux-desktop benchmark for personally intelligent computer-use agents, seeded end-to-end from a single canonical persona (Michael Scott). The image hosts 17 pre-logged-in web apps mirroring real consumer products plus the full LibreOffice suite. The persona's records (bottom strip) are cross-linked so that one trip leaves correlated records across every app that would plausibly record it.

is personal, consistent across applications, and reproducible. Agents must operate over persistent identity and cross-app history, and we grade their trajectories on a desktop that is reset to the same snapshot before every task.

Our canonical persona is Michael Scott, the regional manager of a paper company in Scranton, Pennsylvania. Michael's desktop is seeded with 1,812 bank transactions, 2,398 emails, 679 calendar events with weekly recurrence, 2,526 chat and workplace messages, 126 rideshare requests, 402 food-delivery orders, 155 retail orders, 29 grocery orders, and 32 restaurant reservations. A Firefox profile adds 35 bookmarks and 10,746 page-history visits, and these records are spread across the 17 pre-logged-in web applications and the surrounding desktop stack (Figure 1).

Through MyPCBench, we make three contributions.

1. A reproducible, cross-app-consistent desktop environment for evaluating personalized agents, built from 17 custom web apps and a full Linux desktop, deterministically populated from one persona seed and packaged as a Docker container.
2. 184 tasks inspired by real OpenClaw personal-assistant requests, each with a natural-language rubric and verified solvable end-to-end by the authors, plus an agent harness that drives the standard computer-use agent (CUA) ReAct (Yao et al., 2023) loop against the environment and a rubric-grading LLM-as-a-judge validated against blinded human grading.
3. An evaluation of ten closed and open-weight models (Claude, GPT, Muse Spark, and Qwen families), each run under its provider's computer-use agent with the same `computer` and `bash` tools. We report confidence intervals and paired tests, a tool-access ablation, a persona-prior audit, a failure taxonomy, and scaling analyses across trajectory length and apps per task.

Our headline finding is that even the agent with the highest perfect rate, Claude Opus 4.6, fully solves only 58.2% of MyPCBench tasks and only 36% of tasks that span 7 or more applications, and its

lead over GPT-5.6-sol (55.4%) is not statistically significant. On that 7+-app slice, with every model using the same `computer` and `bash` tools, Muse Spark 1.1 is next at 27%, every remaining model perfects at most 18%, and GPT-5.4, GPT-5.4 mini, and both Qwen models perfect none.

2 Related Work

Web and desktop agent benchmarks. Web benchmarks progressed from synthetic environments (MiniWoB++ (Liu et al., 2018), WebShop (Yao et al., 2022)) to self-hosted realistic websites (WebArena (Zhou et al., 2024), VisualWebArena (Koh et al., 2024)) and static datasets of real-website tasks (Mind2Web (Deng et al., 2023)), and then to live Internet evaluations (WebVoyager (He et al., 2024), Online-Mind2Web (Xue et al., 2025)). Desktop benchmarks such as OSWorld (Xie et al., 2024) extend evaluation to Linux desktops with handcrafted reward verifiers, and OpenClaw-inspired benchmarks, including Claw-Eval (Ye et al., 2026), ClawBench (Zhang et al., 2026), and WildClaw-Bench (Ding et al., 2026), evaluate long-horizon, multi-turn, OpenClaw-style use cases. Most prior work that assigns agents a persona stays inside enterprise contexts, where TheAgentCompany (Xu et al., 2025) places agents in the role of an employee at a simulated software company, and WorkArena (Drouin et al., 2024) drives ServiceNow workflows. AppWorld (Trivedi et al., 2024) seeds a coherent user identity in the API layer, populating 9 simulated apps with one supervisor and a contact network for code agents. iOSWorld (Jang et al., 2026b) seeds a personalized iPhone.

Personalization benchmarks. LaMP (Salemi et al., 2024) scores LLMs over retrieved user profiles and LongMemEval (Wu et al., 2025) tests recall from long conversational histories. On the agent side, PersonalWAB (Cai et al., 2025) attaches user profiles and behavior logs to web agents at the web-function layer, and Persona2Web (Kim et al., 2026) grounds web tasks in long-span user histories. All of these hand the model its personal context as an explicit profile, memory store, or interaction history, whereas MyPCBench seeds the identity into a full desktop where the personal data lives inside the environment itself.

Addressing the personalization gap. Most of the benchmarks above run in the impersonal regime described in §1, excluding tasks that require personal data or pages behind a login (calling an Uber, paying back a friend on Zelle, reordering the usual DoorDash). MyPCBench keeps the OSWorld recipe of a fixed VM image with deterministic snapshot reset, but seeds the desktop end-to-end with Michael Scott's data across the consumer applications a personal computer actually runs, making it a benchmark for personal-assistant computer use rather than a stock-state desktop test.

3 MyPCBench

3.1 Environment

We release MyPCBench as a reproducible, open-source Linux desktop environment through a Docker image that runs a real QEMU/KVM Ubuntu 24.04 VM with GNOME Shell. The VM hosts 17 pre-logged-in websites (each modeled on a real-world analogue), LibreOffice (Writer, Calc, Impress), and a Firefox profile pre-loaded with a realistic browsing history and bookmark set. The home directory is populated with files relating to Michael's personal and work life. Figure 2 shows screenshots of all applications. MyPCBench is built around three properties for evaluating personalized agents.

(1) Cross-app consistency. Any trip, dinner, or client deal leaves correlated records in every application that would plausibly record it. Michael's New York City trip generates Gringotts (Chase) charges for the hotel and the Dinoco (Delta) flight, a HooliCalendar block, a confirmation email, an eTaxi ride to the airport, browsing history for the hotel, and a HooliChat (WhatsApp) message telling Holly about the trip. The seeding pipeline writes those records together, and runtime cross-app effects keep them in sync, so a HangryDash (DoorDash) order posts a charge to Gringotts and a confirmation in HooliMail (Gmail).

(2) Persona coherence. The user is a specific person, not a generic account, whose friends, coworkers, routines, and preferences are entangled across apps and data, and our environment preserves that entanglement. Because the persona is Michael Scott, we were able to use coding agents to draw on

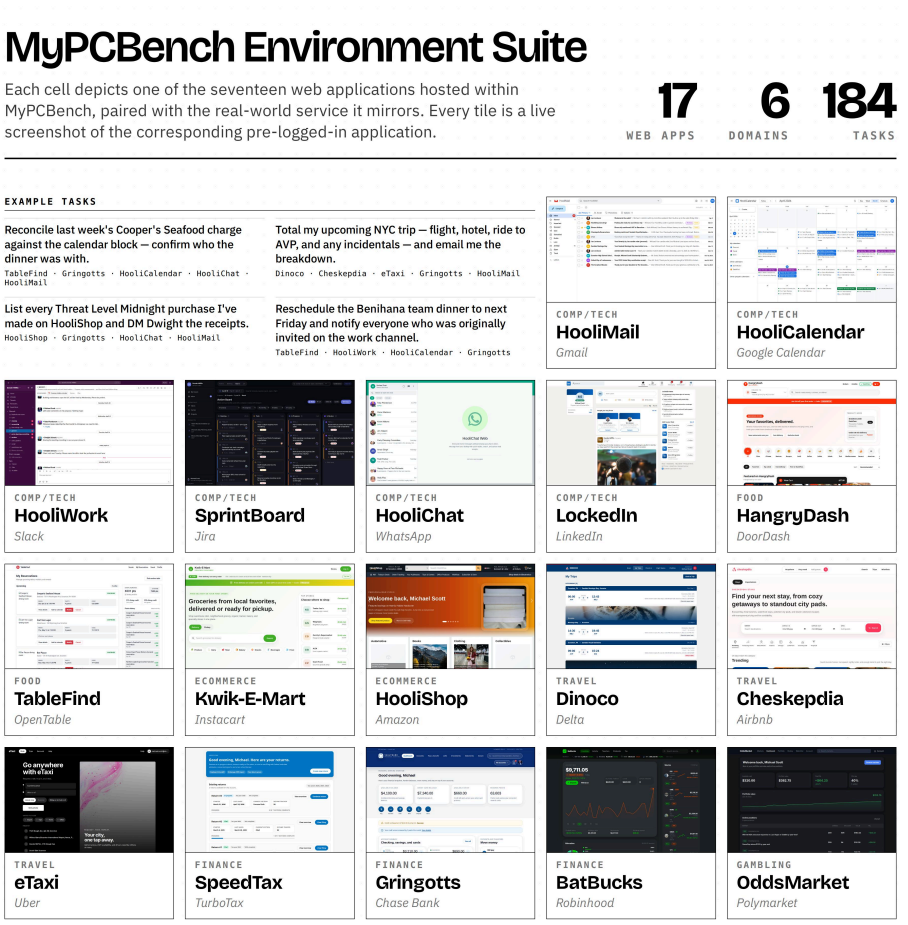


Figure 2: **MyPCBench environment suite.** The 17 logged-in web apps span six SimilarWeb top-level categories (Computers/Tech, Finance, Travel, Food, Ecommerce, Gambling). The four example tasks (top-left) each require combining several of these apps with Michael's seeded history.

The Office canon to populate the environment with large-scale, coherent, realistic data. Details from the show helped shape the seeded records, but the graded answers are almost always specific seeded values, such as amounts, dates, and bookings, that the show does not contain (§5.2).

(3) Real-world fidelity. Each web application is a local clone with the security and reproducibility constraints of a fixed VM, but its UI, navigation, and supported flows match the real-world analogue. The messaging apps are also reactive, as contacts reply to the agent with delayed, LLM-generated messages conditioned on their persona, and these replies were enabled in every evaluated run.

3.2 Environment Creation and Infrastructure

Synthetic website generation. We built 17 clones of real consumer products with Claude Code (Anthropic, 2026b), each a full Next.js build rather than a static mock app, following prior work on coding-agent web cloning and scaled synthesis of browser environments (Zhou, 2026; Murty et al., 2025). The clones implement real workflows, so Gringotts supports transfers, bill pay, Zelle, and statement downloads, Dinoco Airlines generates boarding passes with QR codes, eTaxi routes between seeded locations with OSRM, and TableFind holds a 4,128-slot reservation inventory with hold-and-release semantics. Across the Michael Scott seed, the 17 apps expose 226 database tables and roughly 42,000 rows of user-facing state (headline record types in Figure 1, per-app details in Appendix A). Product images are real photos (Wikimedia/Wikipedia) rather than synthetic placeholders, with image–title alignment checked by a vision–language model.

Persona generation. The persona is specified as a JSON document covering identity, financial profile, social network, travel history, work context, routines, preferences, and recent and upcoming life events. A deterministic Python pipeline populates every part of the desktop from this spec. It writes cross-consistent SQLite databases for the 17 web apps, a Firefox profile with history and bookmarks, and a filesystem of meeting notes, expense reports, itineraries, boarding passes, and resume drafts.

Infrastructure. A single virtual machine uses 4 vCPUs and 8 GB of RAM. Boot-to-ready on the VM takes about 90 seconds, and a base snapshot captured after first boot resets the desktop between tasks, avoiding state leakage. Each image build runs the full generation pipeline end-to-end before the boot snapshot is taken, so the entire environment is reproducible from the spec alone. New personas or websites can be easily added with the same template (Appendix L).

4 Tasks and Evaluation Setup

4.1 Task Suite

MyPCBench includes 184 tasks, each inspired by a real request from the OpenClaw community. The authors manually sifted through 2,749 anonymized and paraphrased use cases from the OpenClaw Discord. We dropped requests that (i) were near-duplicates of an already-kept request, (ii) were infeasible inside any deterministic VM (e.g., "call my mom"), or (iii) required an app outside the 17 we host. The remaining requests were rewritten so the named entities match Michael Scott's seeded data, and a coding agent generated per-task rubrics in the Odysseys (Jang et al., 2026a) format. Both the rewrite and the rubric were then audited by the authors, and the final task set is stored as JSON with each task carrying its instruction and rubric.

Quality assurance. Since coding agents drafted the tasks and the application clones, each task was reviewed by at least two authors through a custom web interface (Appendix K, Figure 8). Reviewers ran each task end-to-end on the live VM and confirmed that (a) every named entity exists in the seeded environment, (b) the expected answer is obtainable from the environment alone, (c) each rubric criterion is individually checkable from a step-level screenshot, and (d) the task is not a near-duplicate of another in the suite. All 184 tasks passed this review. Authors with expert knowledge of the environment also completed all 184 tasks end-to-end on the live VM, with a second author independently verifying every required action and rubric outcome, so every task is executable and scoreable as specified.

Domain coverage. We map each application to a top-level SimilarWeb[2] category by inspecting its real-world analogue, mirroring the categorization scheme of Odysseys (Jang et al., 2026a). The 17 apps span six top-level categories and fourteen subcategories (Table 2).

4.2 Agent Evaluations

We model MyPCBench as a partially observable Markov decision process (Kaelbling et al., 1998). At each turn the agent receives an observation (a screenshot of the guest desktop) and emits one or more actions, which the harness executes on the guest through the OSWorld-compatible (Xie et al., 2024) HTTP control API (Appendix N). Our harness, a thin extension of the OSWorld runner, boots the MyPCBench Docker image, restores a fresh QEMU snapshot before every task so each run begins from an identical desktop state, and drives the standard agent loop until the agent emits `DONE` or `FAIL` or exhausts the turn budget.

Observation space. At each turn the harness captures a 1280×800 screenshot of the full Linux desktop. Claude's agent is defaultly configured to resize it to 1280×720, the Qwen agent passes it through the Qwen-VL token budget, and OpenAI's agent receives it as captured. Following the OSWorld agent convention, the context keeps the full textual action history but only the 20 most recent screenshots.

Action space. The action space is the unmodified OSWorld `pyautogui` action set (click, type, key, scroll, drag, wait, screenshot, done, fail). Each provider's computer-use API defines its own action vocabulary, which we map onto the OSWorld action set (e.g., Claude's `computer.click` becomes

[2] `https://www.similarweb.com/category/`

`click`). Anthropic's Computer Use bundle (Anthropic, 2024) already includes a native `bash` tool, so for parity we give each other agent the shell tool its provider supports (the OpenAI Responses-API `shell`, a Qwen function-call `bash`, and a `bash` tool for the Muse Spark agent). Every model therefore runs with the same `computer` and `bash` tools, and Claude also keeps its native file-edit tool (under 0.5% of its actions). Appendix D gives the controlled computer-only vs. computer+bash comparison and Appendix N the full action table, per-provider mapping, system prompts, and per-model action distribution, including when `bash` helps versus when it triggers the console-script shortcut of §5.5.

What each model evaluation measures. Our evaluations measure one model driven by its provider's computer-use interface inside this shared harness, not an agent tuned for this benchmark. The only changes we make, applied the same way to every task within a provider, are the action mapping above, the shell tool, and each provider's recommended system prompt. Every agent also receives the same persona, verification, and task-completion guidance, provided in Appendix N.

4.3 GRADING

We grade each trajectory against its rubric with the full-trajectory-per-rubric LLM-as-a-judge (Zheng et al., 2023) of Odysseys (Jang et al., 2026a). This judge model and protocol reached $\kappa = 0.79$ agreement with human graders on rubric items and $\kappa = 0.85$ on task success over 120 Claude Opus 4.6 trajectories in Odysseys. Every task in MYPCBENCH ships 3 to 13 natural-language rubric criteria (mean 6.5, 1,191 in total), all audited during the QA pass. Because the rubrics were drafted against the seeded data, they name the exact records a correct answer must use, so the judge holds privileged information the agent must find for itself, as in Gym-Anything (Aggarwal et al., 2026), whose checklist judge agreed with humans on 93.3% of trajectories against 43.3% for model-written programmatic verifiers. The judge runs once per rubric item over the full trajectory, receiving the task instruction, the rubric item, the complete action history (every GUI action, every shell command and its output, and the agent's text responses), and every screenshot (one per turn, at most 100) in chronological order, so it grades from the agent's full observable behavior rather than screenshots alone. Our budget is counted in turns, where each turn is one LLM call and a run is capped at 100 turns. A step is one executed action. A single turn can emit several `pyautogui` actions, so per-task step counts (the Avg steps in Table 1) can exceed the turn cap. The judge returns "success" or "failure" per item, $s_{i,r} \in \{0, 1\}$, and we let $s_i = \frac{1}{N_i} \sum_r s_{i,r}$ denote the per-task score, the fraction of the N_i rubric items passed. The judge model is `gemini-3.1-flash-lite-preview` (Google, 2026).

We report three evaluation metrics per model. The rubric score $\overline{s} = \frac{1}{T} \sum_i s_i$ averages the per-task fraction of rubrics passed and credits partial completion. The stricter perfect rate $\frac{1}{T} \sum_i \mathbf{1}[s_i = 1]$ requires every rubric in a task to pass and is our defining metric. Trajectory Efficiency, also from Odysseys, measures how much rubric score the agent extracts per step, Traj. Eff. $= \frac{1}{T} \sum_{i=1}^{T} s_i / n_i$, where n_i is the number of steps on task i, reported scaled by 100. Alongside each rubric and perfect score we report a 95% interval from a nonparametric bootstrap over the 184 tasks, which reflects the sampled tasks rather than run-to-run variance (§5.2). The full judge prompt is in Appendix M.

5 EXPERIMENTS AND ANALYSIS

5.1 MAIN RESULTS

We evaluate ten models on the full 184-task suite, each driven by its provider's computer-use (CUA) agent with the shared `computer` and `bash` tools. Throughout, a model name stands for the whole agent a provider ships around that model, so differences between scores compare those agents rather than bare models. The eight closed-weight models are Claude Opus 4.6 and Claude Sonnet 4.6 (Anthropic, 2026c;d), the two GPT-5.6 models, GPT-5.6-sol and GPT-5.6-luna (OpenAI, 2026d), GPT-5.5 (OpenAI, 2026c), GPT-5.4 (OpenAI, 2026a), GPT-5.4 mini (OpenAI, 2026b), and Muse Spark 1.1 (Meta, 2026), and the two open-weight models are Qwen 3.5 (Qwen Team, 2026) 35B-A3B and 9B. Every row is a single run per task with no reruns under the 100-turn budget and the same judge. Table 1 reports the three evaluation metrics of §4.3 and the average steps consumed.

Closed-weight frontier agents lead by a wide margin. Claude Opus 4.6 reaches **58.2%** perfect at 85.1% rubric score, the highest perfect rate in the suite, with GPT-5.6-sol (55.4%, 85.3% rubric) and

Table 1: Main results on the 184-task suite under each provider's CUA agent with `computer+bash` enabled (100-turn budget, same judge and persona context). Perfect (every rubric passed) is our headline metric, Rubric score gives partial credit, and Traj. Eff. is rubric score per agent step (§4.3). Brackets are 95% bootstrap intervals over tasks.

Model	Perfect ↑		Rubric score ↑		Avg steps	Traj. Eff. ↑
API (closed-weight)						
Claude Opus 4.6	**58.2**	[51.1, 65.2]	85.1	[81.7, 88.3]	46.5	**3.78**
GPT-5.6-sol	55.4	[48.4, 62.5]	**85.3**	[82.0, 88.4]	57.2	2.26
GPT-5.6-luna	55.4	[48.4, 62.5]	83.2	[79.6, 86.6]	63.5	2.00
Claude Sonnet 4.6	50.5	[43.5, 57.6]	78.5	[74.1, 82.7]	45.8	3.68
GPT-5.5	45.1	[38.0, 52.2]	83.6	[80.6, 86.5]	43.5	2.80
Muse Spark 1.1	44.0	[37.0, 51.1]	80.7	[77.1, 84.2]	45.2	2.73
GPT-5.4†	28.3	[21.7, 34.8]	51.5	[45.8, 57.4]	35.4	2.09
GPT-5.4 mini	23.9	[17.9, 30.4]	53.5	[47.7, 59.3]	49.5	1.64
Open weights						
Qwen 3.5 35B-A3B	7.6	[3.8, 11.4]	42.4	[37.4, 47.4]	66.0	1.41
Qwen 3.5 9B	6.5	[3.3, 10.3]	24.1	[19.4, 28.9]	69.2	1.09

GPT-5.6-luna (55.4%) close behind. Paired tests do not separate Opus from either GPT-5.6 model. The exact McNemar test gives $p = 0.61$ for both comparisons, and the 95% interval on the Opus–Sol rubric difference is $[-4.5, 3.9]$ (Appendix F).

Trajectory Efficiency adds a view on how efficient the agents are, especially given that they have a step budget. The Claude models extract the most rubric score per step (Opus **3.78**, Sonnet 3.68), while the GPT-5.6 models spend 11–17 more steps per task for similar rubric scores.

5.2 How Robust Is the Comparison?

Tool access. We ran the GPT-5.4, GPT-5.4 mini, and Qwen baselines on all 184 tasks with and without the shell (Appendix D). The shell moves GPT-5.4 from 23.9% to 28.3% perfect and GPT-5.4 mini from 19.0% to 23.9%, moves the Qwen models by at most 3.3 points, and changes no ordering of model performance rankings. Separately, on 72 tasks sampled proportionally across task types, Opus without `bash` perfects 50.0% of tasks against 30.6% for GPT-5.4 with it, so shell access does not explain the gap between models. We keep the shell as it mirrors deployment, where assistants such as OpenClaw hold logged-in sessions and programmatic access to the same accounts. Because the judge sees every shell command and its output, it grades shell actions as well as what appears in the screenshots. In the shell logs, we inspected the trajectories, agents used the shell to read application state and made no direct writes or attempts to change the local databases in a reward-hacking manner(Appendix N).

Construction bias. The app clones, task rewrites, and rubric drafts were produced with Claude Code, which could favor Claude models. Four observations limit this concern. First, the tasks derive from real OpenClaw requests and at least two authors validated every task and rubric on the live VM. Second, the judge is a non-Anthropic model. Third, GPT-5.6-sol ties Opus on rubric score and leads it on three of six task types (§5.3). Fourth, the ordering of model performance is not specific to MyPCBench, since an independent report evaluating agents on both OSWorld 2.0, whose tasks were written by human annotators, and MyPCBench places Claude Opus first and that report's own Qwen-CUA agent second on both (Lu et al., 2026). These observations cannot rule out every remaining effect, but none of them indicates a systematic Claude-family advantage.

Persona priors. Because Michael Scott is a public character, a model could exploit memorized knowledge of *The Office*. Under explicit rules checked against the rubrics, only 9 of 184 tasks have any graded step that knowledge of the show could plausibly help with, and none can be completed from the show alone (Appendix G). On a deliberately broad 73-task split covering every task that involves habits, preferences, or relationships, every model perfects fewer tasks than on the other 111, by 5.6 to 20.7 points, although these are also the inference-heavy tasks, so this split bounds a canon effect rather than isolating it.

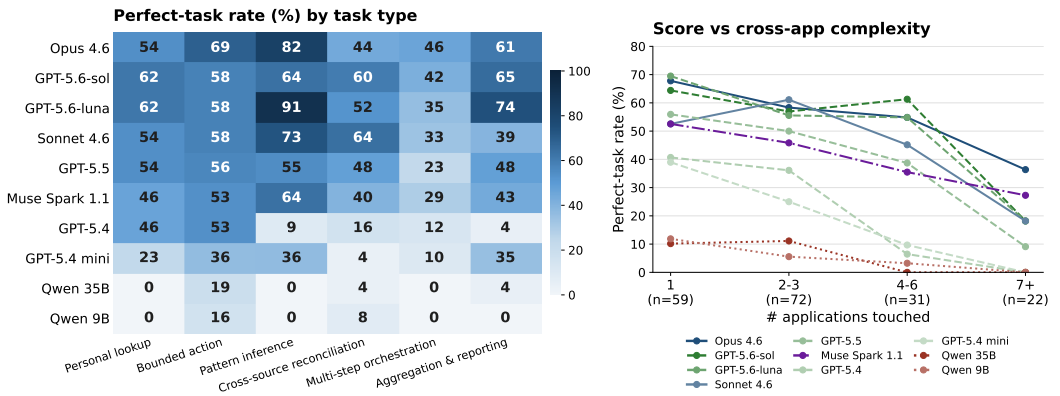


Figure 3: **Left:** perfect rate (%) by task type. **Right:** perfect rate by the number of distinct apps a task touches. Rubric scores for both are in Appendix E.

Judge reliability and run variance. Two authors blind to the judge's labels each graded half of 36 representatively sampled GPT-5.6-sol trajectories (220 rubric items). Human–judge agreement is 94.5% ($\kappa = 0.73$) at the rubric level and 83.3% ($\kappa = 0.64$) on perfect-task labels, against $\kappa = 0.79$ and 0.85 for the same judge on Odysseys, and re-judging the Claude Opus 4.6 leaderboard trajectories three more times gives rubric scores within 0.9 points of each other (Appendix F). Three independent 184-task runs of Muse Spark 1.1 (Table 1 reports the first) score 80.7, 80.8, and 80.0 rubric (SD 0.4) but 44.0, 48.9, and 42.4 perfect, so the run-to-run spread in perfect rate is comparable to the gaps between adjacent rows in our results table, and those gaps should be read through the paired tests (Appendix F).

5.3 Performance by Task Type

Tasks fall into six behavioral types, defined with examples in Table 3. Two categories separate the models least. On *personal lookup* (Figure 3-left) the five strongest models all clear 53% perfect (both GPT-5.6 models lead at 62%), and on *bounded action* the six strongest span 53–69%, with Opus ahead. The remaining four categories require reasoning over persona history or coordinating writes across apps, and the spread between models widens there. *Multi-step orchestration* is the hardest type overall (lowest cross-model rubric mean, 53.5). Opus leads at 46% perfect, followed by GPT-5.6-sol at 42%, while Muse Spark falls to 29%, GPT-5.5 to 23%, and both Qwen models perfect none of the 48 tasks. The frontier models also differ in where they are strong, as GPT-5.6-sol leads Opus on *personal lookup*, *cross-source reconciliation*, and *aggregation*, and Opus leads on *bounded action* and orchestration. On *pattern inference*, which asks the agent to infer an unstated rule from many records ("what do I usually tip?"), GPT-5.6-luna reaches 91% and Opus 82%, while GPT-5.4 and the Qwen models reach at most 9% on the same 11 tasks.

5.4 Performance Scaling by Steps and Apps

App count and trajectory length both expose horizon limits. From single-app to 7+-app task types the perfect rate falls from 68% to 36% for Opus and from 64% to 18% for GPT-5.6-sol, and the frontier models are nearly interchangeable up to six apps before separating at 7+, where Opus perfects 36% against 27% for Muse Spark and at most 18% for every other model (Figure 3-right). Rubric scores fall more gently (Table 7), so agents on long cross-app tasks usually complete most of the work and miss a piece. On the step axis (Figure 6) Opus and the GPT-5.6 models are still climbing at the budget, while the GPT-5.4 models flatten after step 70 and the Qwen curves after step 40.

5.5 Personalization-Specific Failures

We classify every failed-rubric judge explanation for the ten leaderboard rows against keyword rules for five failure modes (Appendix I), so an item can match several modes or none. Skipped required app (659 hits) and premature `DONE` (566) are the most common modes, followed by surface-error abandonment (232), partial artifact (84), and hallucinated persona data (73). The families concentrate

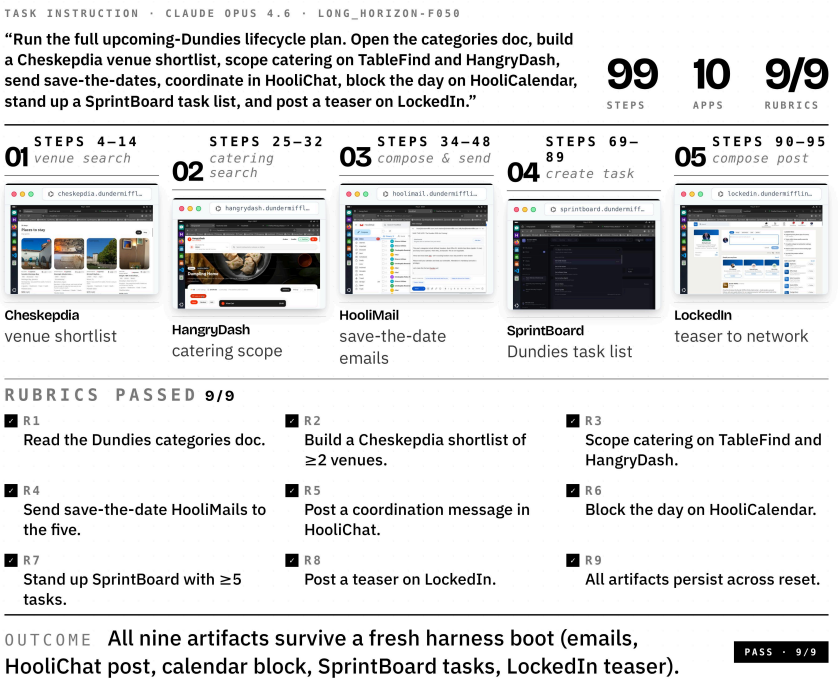


Figure 4: **One full successful Opus trajectory, the Dundies-lifecycle plan on `long_horizon-f050`** (99 steps, 10 apps, 9/9 rubrics). Cells are real screenshots from the steps where the agent drives each app, and the bottom strip lists the nine rubric criteria the judge marked passed.

in different failure modes. The GPT family accounts for 392 of the 566 premature-`DONE` failures, while Qwen accounts for 211 of the 659 skipped-app failures, and Qwen 35B also leads persona-data hallucination. Claude instead takes console-script shortcuts, trying to complete tasks through `bash` rather than the UI and leaving the graded UI side effect undone. Zero-score trajectories split into two regimes, in which Sonnet and GPT-5.4 abandon early (mean 14–20 steps), while the Qwen models keep working past the point where the rubric is recoverable (mean 85–88 steps). For contrast, Figure 4 shows a clean 10-app Opus run.

6 Conclusion

Our benchmark provides improvement points for every model evaluated in the lens of personally intelligent computer-use agents. The gaps on MyPCBench resolve into three family-shaped failure patterns (§5.5). Claude unsuccessfully attempts to shortcut past the UI through `bash`, the older GPT models emit a premature `DONE` before producing the rubric-graded side effect, and within Qwen the 35B hallucinates persona values while the 9B often emits malformed tool calls. These are specific modes that future agent designs can target. The best agents we evaluate perfect barely more than half the suite, the cross-app and long-horizon slopes are steep for every model, and the hardest categories (multi-step orchestration, aggregation & reporting) remain the weakest. Additionally, we find that the ability for models to leverage their coding abilities in personal assistant capabilities while balancing with computer-use actions will be very important going forward, with gaps showing currently. We hope that MyPCBench pushes computer-use agents and general personal assistant research towards a personalized lens, where we believe there is currently a large gap between deployment and evaluation. We release the MyPCBench environment, tasks with rubrics, harness, and rubric-grading judge (`https://anonymous.4open.science/r/MyPCBench-21FF/`) as a baseline for work on personal computer-use agents.

AI USE STATEMENT

We used generative AI tools in building this benchmark. Claude Code built the 17 web-application clones, helped populate the persona data, and drafted the task rewrites and per-task rubrics from human-selected OpenClaw requests, and every task and rubric was then validated by at least two authors on the live VM (§4.1). A vision–language model checked the alignment of product images with their titles, and an LLM assisted in labeling tasks for the persona-prior audit under explicit rules, with every positive label checked by hand against the rubrics (Appendix G). We also used AI assistants to help write analysis scripts and to edit the text of this paper. The authors reviewed all AI-assisted work, checked every reported number against the underlying task-level scores, and take responsibility for the content of this paper, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

MYPCBENCH contains no real personal data. Every record is synthetic and tied to a public fictional character, and every web application runs locally inside the VM, so the credentials and form values the agent handles belong only to local services. The OpenClaw requests that inspired the tasks were anonymized and paraphrased before use. The skills MYPCBENCH measures are the same skills an agent would need to act on a real user's accounts, so results on this benchmark should not be read as clearance to deploy agents on production accounts (Appendix Q).

REPRODUCIBILITY STATEMENT

The environment image, generation pipeline, task set with rubrics, harness, and judge configuration are released through the anonymized links in Appendix R. §4.2 and Appendix N give the action space, per-provider mappings, and the system prompts, Appendix M reproduces the judge prompts, and Appendices L and J describe the deterministic seeding pipeline and persona schema. Task-level scores for every leaderboard row ship with the release so that all intervals and paired tests can be recomputed, and the agent trajectories and per-rubric judge outputs will be released with the camera-ready version.

REFERENCES

Pranjal Aggarwal, Graham Neubig, and Sean Welleck. Gym-Anything: Turn any software into an agent environment. *arXiv preprint arXiv:2604.06126*, 2026.

Anthropic. Introducing computer use, a new Claude 3.5 Sonnet, and Claude 3.5 Haiku. *Anthropic Blog*, 2024.

Anthropic. Claude Cowork. `https://claude.com/product/cowork`, 2026a.

Anthropic. Claude Code. `https://docs.anthropic.com/en/docs/claude-code/overview`, 2026b.

Anthropic. Claude Opus 4.6. `https://www.anthropic.com/news/claude-opus-4-6`, 2026c.

Anthropic. Claude Sonnet 4.6. `https://www.anthropic.com/news/claude-sonnet-4-6`, 2026d.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Yadong Lu, Justin Wagle, Kazuhito Koishida, Arthur Bucker, et al. Windows Agent Arena: Evaluating multi-modal OS agents at scale. *ICML*, 2025.

Hongru Cai, Yongqi Li, Wenjie Wang, Fengbin Zhu, Xiaoyu Shen, Wenjie Li, and Tat-Seng Chua. Large language models empowered personalized web agents. *WWW*, 2025.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2Web: Towards a generalist agent for the web. *NeurIPS*, 2023.

Shuangrui Ding, Xuanlang Dai, Long Xing, Shengyuan Ding, Ziyu Liu, Jingyi Yang, Penghui Yang, Zhixiong Zhang, Xilin Wei, Xinyu Fang, et al. WildClawBench: A benchmark for real-world, long-horizon agent evaluation. *arXiv preprint arXiv:2605.10912*, 2026.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, et al. WorkArena: How capable are web agents at solving common knowledge work tasks? *ICML*, 2024.

Google. Gemini 3.1 Flash-Lite. https://blog.google/innovation-and-ai/models-and-research/gemini-models/gemini-3-1-flash-lite/, 2026.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. *ACL*, 2024.

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

Lawrence Keunho Jang, Mareks Woodside, Geronimo Carom, Andrew Keunwoo Jang, Jing Yu Koh, and Ruslan Salakhutdinov. iosworld: A benchmark for personally intelligent phone agents, 2026b. URL https://arxiv.org/abs/2606.09764.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

Serin Kim, Sangam Lee, and Dongha Lee. Persona2Web: Benchmarking personalized web agents for contextual reasoning with user history. *ICML*, 2026.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. *ACL*, 2024.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. Reinforcement learning on web interfaces using workflow-guided exploration. *ICLR*, 2018.

Dunjie Lu, Shuai Bai, Tianyi Bai, Sicheng Fan, Chang Gao, Jian Guan, Feng Hu, Mianqiu Huang, Xingyang Huang, Yizhen Jiang, Yuheng Jing, Dehui Kong, Ning Li, Dayiheng Liu, Shixuan Liu, Zheng Liu, Que Shen, Bowen Wang, Junli Wang, Chencan Wu, Rui Xie, Tianbao Xie, Zhihui Xie, Haiyang Xu, An Yang, Tao Yu, Wenzhen Yuan, Xi Zhang, Zhenru Zhang, Mingkang Zhu, Zhaoqing Zhu, Yizhong Cao, Kai Dang, Binyuan Hui, Kaixin Li, Junyang Lin, Haiquan Wang, Zekun Wang, Yiheng Xu, Fan Yan, Mengqi Yuan, Danyang Zhang, Jiajun Zhang, Zhipeng Zhang, and Fan Zhou. Qwen-CUA: Native computer use for (almost) everything. *arXiv preprint arXiv:2608.02352*, 2026.

Meta. Introducing Muse Spark 1.1. https://ai.meta.com/blog/introducing-muse-spark-meta-model-api/, 2026.

Shikhar Murty, Hao Zhu, Dzmitry Bahdanau, and Christopher D. Manning. NNetNav: Unsupervised learning of browser agents through environment interaction in the wild. *arXiv preprint arXiv:2410.02907*, 2025.

OpenAI. GPT-5.4. https://developers.openai.com/api/docs/models/gpt-5.4, 2026a.

OpenAI. GPT-5.4 mini. https://developers.openai.com/api/docs/models/gpt-5.4-mini, 2026b.

OpenAI. GPT-5.5. https://developers.openai.com/api/docs/models/gpt-5.5, 2026c.

OpenAI. GPT-5.6 Sol and GPT-5.6 Luna. https://developers.openai.com/api/docs/models/gpt-5.6 and https://developers.openai.com/api/docs/models/gpt-5.6-luna, 2026d.

Qwen Team. Qwen3.5-35B-A3B. https://huggingface.co/Qwen/Qwen3.5-35B-A3B, 2026.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. AndroidWorld: A dynamic benchmarking environment for autonomous agents. *ICLR*, 2025.

Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. LaMP: When large language models meet personalization. *ACL*, 2024.

Peter Steinberger. OpenClaw. https://openclaw.ai/, 2026.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. *ACL*, 2024.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. LongMemEval: Benchmarking chat assistants on long-term interactive memory. *ICLR*, 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *NeurIPS*, 2024.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. TheAgentCompany: Benchmarking LLM agents on consequential real world tasks. *NeurIPS*, 2025.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. *COLM*, 2025.

Pei Yang, Hai Ci, and Mike Zheng Shou. MacOSWorld: A multilingual interactive benchmark for GUI agents. *NeurIPS*, 2025.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. *NeurIPS*, 2022.

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

Bowen Ye, Rang Li, Qibin Yang, Yuanxin Liu, Linli Yao, Hanglong Lv, Zhihui Xie, Chenxin An, Lei Li, Lingpeng Kong, Qi Liu, Zhifang Sui, and Tong Yang. Claw-Eval: Toward trustworthy evaluation of autonomous agents, 2026.

Yuxuan Zhang, Yubo Wang, Yipeng Zhu, Penghui Du, Junwen Miao, Xuan Lu, Wendong Xu, Yunzhuo Hao, Songcheng Cai, Xiaochen Wang, Huaisong Zhang, Xian Wu, Yi Lu, Minyi Lei, Kai Zou, Huifeng Yin, Ping Nie, Liang Chen, Dongfu Jiang, Wenhu Chen, and Kelsey R. Allen. ClawBench: Can AI agents complete everyday online tasks? *arXiv preprint arXiv:2604.08523*, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. *NeurIPS*, 2023.

Shuyan Zhou. WebArena-Infinity: Generating browser environments with verifiable tasks at scale. https://github.com/web-arena-x/webarena-infinity, 2026.

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

A Application Details

Table 2 summarizes the 17 web applications hosted within the MyPCBench environment image, the real-world service each one mirrors, and the SimilarWeb top-level category and subcategory inherited from that analogue.

Table 2: The 17 web applications hosted within the MyPCBench environment image, with the SimilarWeb top-level category and subcategory each one inherits from its real-world analogue. Screenshots of every app are in Figure 2.

App	Analogue	SimilarWeb category	Subcategory	Description
HooliMail	Gmail	Computers, Electronics & Tech	Email	Gmail-style web client over a local Maildir, with 2,398 seeded messages across Inbox, Sent, and four labeled folders.
HooliCalendar	Google Calendar	Computers, Electronics & Tech	Productivity	Google-Calendar-style scheduling app exposing 679 personal and work events with recurrence and attendee lists.
HooliWork	Slack	Computers, Electronics & Tech	Programming & Developer Software	Slack-style team messenger with the persona's branch channels, DMs, and read state.
SprintBoard	Jira	Computers, Electronics & Tech	Programming & Developer Software	Jira-style issue tracker holding the persona's running sprints, tickets, and assignees.
HooliChat	WhatsApp	Computers, Electronics & Tech	Telecommunications	WhatsApp-style messenger with one-to-one and group threads spanning friends, family, and coworkers.
LockedIn	LinkedIn	Computers, Electronics & Tech	Social Media Networks	LinkedIn-style professional network exposing the persona's profile, feed, and connections, with 18 jobs across 9 companies and 13 fully populated profiles, plus connection- and company-authored feed posts on top of the user's own.
HangryDash	DoorDash	Food & Drink	Restaurants & Delivery	DoorDash-style food delivery surface covering 28 restaurants with 165 menu items, plus order history and active carts.
TableFind	OpenTable	Food & Drink	Restaurants & Delivery	OpenTable-style restaurant reservation app over a 4,128-slot pre-computed inventory across 31 dates and 19 restaurants, with past bookings and search.
Kwik-E-Mart	Instacart	Ecommerce & Shopping	Marketplace	Instacart-style grocery delivery spanning 7 Scranton-area stores (Wegmans, Price Chopper, ALDI, Gerrity's, Weis, Trader Joe's, Giant) with 109 curated SKUs total, multi-store carts, and an order log.
HooliShop	Amazon	Ecommerce & Shopping	Marketplace	Amazon-style retail front with a 90-SKU catalog of real-photo product imagery, personalized recommendations, orders, and a cart.
Dinoco	Delta	Travel & Tourism	Air Travel	Delta-style airline app exposing flights, loyalty, baggage, and check-in, with 14 booked flights across 5 AVP hubs and full real-Delta seat parity, plus a dynamic alternative-flight search pool.
Cheskepdia	Airbnb	Travel & Tourism	Accommodation & Hotels	Airbnb-style stays / experiences booking surface with trips, wishlists, and search.
eTaxi	Uber	Travel & Tourism	Ground Transportation	Uber-style ride-hail app with ride history, saved places, and active requests.
SpeedTax	TurboTax	Finance	Accounting & Auditing	TurboTax-style tax preparation surface with prior-year returns, W-2s, 1099s, and a current-year draft.
Gringotts	Chase Bank	Finance	Banking, Credit & Lending	Chase-style bank dashboard covering checking, savings, credit card, and a 1,812-row transaction log.
BatBucks	Robinhood	Finance	Investing	Robinhood-style brokerage with a holdings view, watchlist, and position-level history.
OddsMarket	Polymarket	Gambling	Other	Polymarket-style prediction-market exchange with the persona's open positions and watchlist.

B TASK-DISTRIBUTION PLOTS

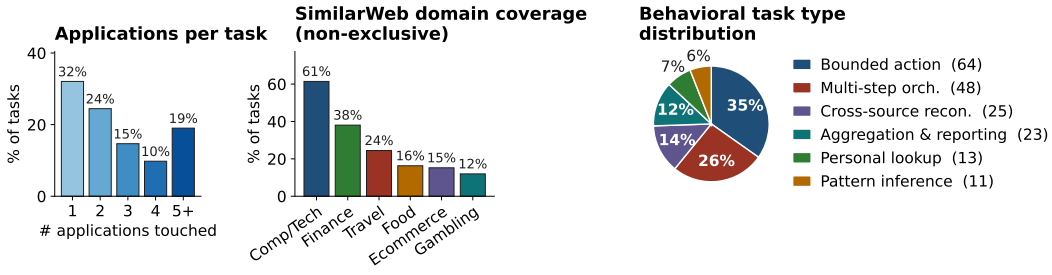


Figure 5: **Left:** distribution of tasks by the number of distinct applications they touch. **Middle:** fraction of tasks that touch at least one application in each SimilarWeb top-level category (non-exclusive, so a single multi-app task can contribute to several bars). **Right:** behavioral task-type split (exclusive 1-of-6 categorization per task).

Temporal and social coverage. The seeded history spans roughly one year and includes, among other records, 1,812 bank transactions and 10,746 page-history visits. Of the 184 tasks, 29 (16%) explicitly require looking back over several months, for example computing miles flown over the past 12 months or ranking Zelle recipients over the two most recent complete calendar months. Pattern-inference tasks also depend on accumulated history, such as months of delivery orders. Half of the tasks (92) involve a named contact or a person-centric interaction, such as checking with a coworker before a transfer, selecting and inviting relevant connections, posting to a group channel, or notifying attendees after rescheduling an event. The shared social graph spans HooliChat, HooliMail, HooliWork, and LockedIn, and the chat apps return delayed, LLM-generated replies conditioned on each contact's persona.

C TASK-TYPE DEFINITIONS

Table 3: The six behavioral task types in MYPCBENCH, with definition, count, and a representative instruction for each. Counts cover all 184 tasks, and the per-task type mapping ships with the release as the `category_v2` field of `tasks/final/all_tasks.json`.

Type	n (%)	Definition	Representative instruction
Bounded action	64 (35%)	Execute one concrete write action, or a tightly-scoped sequence within a single application, grounded in the persona's current state.	*Zelle Pam a hundred bucks. She covered me last weekend. Check HooliChat first to make sure Pam Beesly is on my contacts, then put a memo on the transfer.*
Multi-step orchestration	48 (26%)	Chain reads and writes across multiple applications, typically producing artifacts (LibreOffice docs, decks) plus coordinated side-effects (calendar blocks, sent messages, board updates).	*The Threat Level Midnight Fan Club has been dormant. Peek at the group chat, scroll my LockedIn contacts for Dunder Mifflin folks to recruit, draft them an invitation email, and book a watch party on my calendar for next month.*
Cross-source reconciliation	25 (14%)	Reconcile a claim, assumption, or hypothetical against the persona's data by pulling from several apps and quantifying the gap. Covers both contradiction-finding and counterfactual feasibility.	*I've got the Jamaica trip AND the Barbados trip booked about four weeks apart. Given my credit-card balance, can I actually afford both, or am I about to max out?*
Aggregation & reporting	23 (12%)	Compute a total, distribution, ranking, chart, or rollup from many persona records, typically delivered into a LibreOffice document.	*How much am I sending via Zelle each month, and who's getting the money? Check the most recent two complete calendar months and rank the recipients in a LibreOffice Calc spreadsheet.*
Personal lookup	13 (7%)	Surface a specific named value, file, or record from the persona's environment. Single-fact retrieval, no rollup, no inference.	*What's my current FlyMiles loyalty tier on Dinoco Airlines, and how many miles do I have in the bank?*
Pattern inference	11 (6%)	Infer an unstated habit, preference, or stylistic pattern from historical persona data, never explicitly stored anywhere in the environment.	*What do I usually tip on food delivery, in dollars and as a percent? I want to set a smart default so I'm not thinking about it every order.*

D COMPUTER-ONLY VS. COMPUTER+BASH TOOL ACCESS

The main results (Table 1) evaluate every model with both the `computer` and `bash` tools. Anthropic's native computer-use agent already includes `bash`, and for every other agent we add a `bash` or `shell` tool to an otherwise computer-only setup. The ablation below compares the OpenAI and Qwen agents with and without this added tool.

Table 4: Effect of adding the documented `bash`/`shell` tool to the computer-only OpenAI and Qwen agents, on the full 184-task suite (identical persona context, 100-turn budget with one LLM call per turn, same Gemini judge). Every cell is a single run per task with no best-of-N selection. Δ is computer+bash minus computer-only.

	Perfect		Rubric score		Δ	
Model	**computer**	**+bash**	**computer**	**+bash**	**Perfect**	**Rubric**
GPT-5.4	23.9	28.3	50.5	51.5	+4.4	+1.0
GPT-5.4 mini	19.0	23.9	47.8	53.5	+4.9	+5.7
Qwen 3.5 35B-A3B	10.9	7.6	42.7	42.4	−3.3	−0.3
Qwen 3.5 9B	6.0	6.5	24.3	24.1	+0.5	−0.2

Adding the shell never changes the perfect-rate ordering of these four models (Table 4). The two GPT models gain 4.4 and 4.9 perfect points, and GPT-5.4 mini also gains 5.7 rubric points. The two Qwen models change by at most 3.3 perfect points and 0.3 rubric points, which is within the run-to-run and judge variation of Appendix F. Qwen 9B trajectories often contain malformed tool calls that mix the `bash` and `computer` schemas, which is consistent with the shell adding nothing for this model. Beyond the GPT perfect-rate gains, we therefore make no strong claim that `bash` helps or hurts these models, only that equalizing the tools does not overturn the perfect-rate ordering.

Claude without the shell. To test whether Claude's lead comes from its shell, we removed `bash` from Opus 4.6 on 72 tasks sampled proportionally across the task types and compared it with GPT-5.4 on the same tasks (Table 5). Opus without the shell still perfects 19.4 points more tasks than GPT-5.4 with it, and the shell adds 8.3 perfect points to Opus, about 1.5× its effect on GPT-5.4 on the same tasks. We have not run this ablation for the GPT-5.6 models.

Table 5: Opus 4.6 and GPT-5.4 with and without the shell on the same 72 tasks, sampled proportionally across task types.

	Perfect		Rubric score	
Model	**computer**	**+bash**	**computer**	**+bash**
Claude Opus 4.6	50.0	58.3	82.6	85.5
GPT-5.4	25.0	30.6	49.2	53.0

E Per-Task-Type, Cross-App, Rubric-Count, and Step-Budget Breakdowns

Tables 6 and 7 give the rubric scores that accompany the perfect rates in Figure 3, and Table 8 breaks perfect rate down by rubric count, all computed from the task-level scores of the ten rows of Table 1. Figure 6 shows how many tasks each model perfects within a given number of steps.

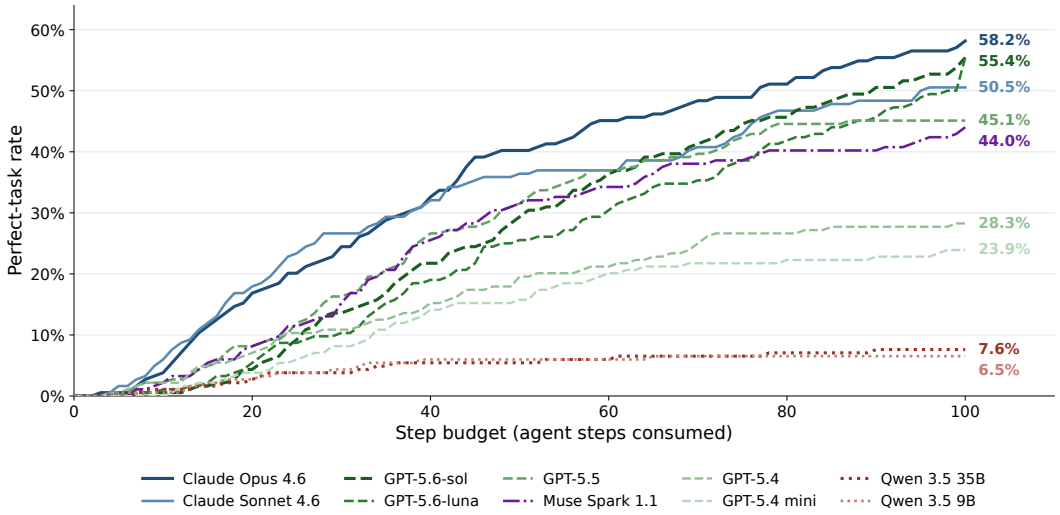


Figure 6: **Perfect rate by steps consumed.** For each model, the curve at X steps is the fraction of the 184 tasks that the model perfects with $\leq X$ agent steps consumed. Curve shape, not just the peak scores, separates the families, since the Opus and GPT-5.6 curves are still rising at the budget while the GPT-5.4 and Qwen curves have flattened.

Table 6: Rubric score (%) by task type for all ten leaderboard rows. Rows ordered by descending cross-model average. *Mean* is the unweighted average across the ten models. Best per row in **bold**.

Task type	n	Opus	5.6-sol	5.6-luna	Sonnet	5.5	Muse	5.4	5.4 mini	Q-35B	Q-9B	Mean
Bounded action	64	**90.3**	88.5	87.7	84.6	87.7	86.6	69.8	62.7	58.9	39.6	75.6
Pattern inference	11	94.7	94.3	**97.0**	86.4	87.6	84.5	58.8	62.7	35.9	8.3	71.0
Cross-source reconc.	25	80.1	**90.6**	84.4	86.4	82.7	80.8	55.3	63.8	37.6	30.3	69.2
Personal lookup	13	83.8	**90.0**	88.5	81.4	80.8	79.2	69.5	40.0	33.2	31.2	67.8
Aggregation & reporting	23	86.3	**94.2**	87.4	64.5	86.8	83.0	31.0	74.4	24.7	15.9	64.8
Multi-step orchestration	48	**78.3**	70.8	70.0	70.5	76.9	71.4	28.5	27.4	35.3	5.8	53.5

Table 7: Rubric score (%) versus the number of distinct applications a task touches, all ten leaderboard rows. 68% of MYPCBENCH tasks are multi-app. Best per row in **bold**.

Apps touched	n	Opus	5.6-sol	5.6-luna	Sonnet	5.5	Muse	5.4	5.4 mini	Q-35B	Q-9B
1	59	89.8	91.4	**92.0**	77.8	85.2	86.3	59.2	68.4	44.8	32.0
2–3	72	86.2	88.0	84.0	87.1	**88.7**	80.8	64.1	56.8	48.8	25.3
4–6	31	83.4	**88.7**	82.9	76.5	80.7	78.3	36.8	46.2	33.0	20.4
7+	22	**71.1**	55.3	57.3	55.2	66.5	68.7	10.4	12.9	28.3	4.2
Δ (1 $\rightarrow$ 7+)		-18.7	-36.1	-34.7	-22.6	-18.7	-17.6	-48.8	-55.5	-16.5	-27.8

Table 8: Perfect rate (%) by number of rubric items per task, with the point-biserial correlation r between rubric count and the per-task perfect label. Perfect rate is lower in the 9+ bin than in the 3–4 bin for every model and the correlation is negative throughout, as expected because a perfect score requires every rubric to pass and longer rubrics accompany tasks that touch more apps.

Rubrics	n	Opus	5.6-sol	5.6-luna	Sonnet	5.5	Muse	5.4	5.4 mini	Q-35B	Q-9B
3–4	39	74.4	71.8	69.2	71.8	66.7	53.8	66.7	23.1	17.9	12.8
5–6	58	56.9	62.1	63.8	48.3	56.9	55.2	24.1	36.2	6.9	8.6
7–8	61	59.0	54.1	52.5	52.5	34.4	39.3	16.4	19.7	4.9	3.3
9+	26	34.6	19.2	23.1	19.2	11.5	15.4	7.7	7.7	0.0	0.0
Point-biserial r		-0.22	-0.29	-0.30	-0.28	-0.39	-0.24	-0.41	-0.18	-0.19	-0.16

F Statistical Tests and Judge Validation

Intervals and paired tests. The row intervals in Table 1 come from a nonparametric percentile bootstrap over the 184 tasks with 50,000 resamples and a fixed seed. They describe how much the numbers depend on which tasks were sampled, not the run-to-run randomness of the agents. To compare models, we do not check whether row intervals overlap. We instead resample shared task IDs jointly to bootstrap the paired rubric difference, and we test perfect labels with the exact two-sided McNemar test on the discordant tasks (Table 9). The leading group cannot be separated. Opus and GPT-5.6-sol have 34 tasks perfected only by Opus and 29 perfected only by Sol ($p = 0.61$), and Sol and Luna split 23/23. The comparison between Sonnet and GPT-5.5 has opposite signs on the two metrics, which is why Table 1 reports both. The large drops, from GPT-5.5 to GPT-5.4 and from GPT-5.4 mini to Qwen 35B, are significant at $p < 10^{-4}$.

Table 9: Paired comparisons on the 184 shared tasks. Δ is the first model minus the second. *Discordant* counts tasks perfected only by the first / only by the second model. p is the exact two-sided McNemar test on perfect labels. The rubric interval is a 95% paired bootstrap over tasks.

Comparison	Δ Perfect	Discordant	p	Δ Rubric	95% CI
Claude Opus 4.6 vs. GPT-5.6-sol	$+2.7$	34 / 29	0.61	-0.2	$[-4.5, 3.9]$
Claude Opus 4.6 vs. GPT-5.6-luna	$+2.7$	34 / 29	0.61	$+1.9$	$[-2.3, 6.0]$
Claude Opus 4.6 vs. Claude Sonnet 4.6	$+7.6$	41 / 27	0.11	$+6.6$	$[1.8, 11.4]$
GPT-5.6-sol vs. GPT-5.6-luna	$+0.0$	23 / 23	1.00	$+2.1$	$[-0.9, 5.2]$
GPT-5.6-sol vs. GPT-5.5	$+10.3$	40 / 21	0.02	$+1.7$	$[-1.7, 5.1]$
Claude Sonnet 4.6 vs. GPT-5.5	$+5.4$	42 / 32	0.30	-5.1	$[-9.9, -0.4]$
GPT-5.5 vs. GPT-5.4	$+16.8$	45 / 14	6.5×10^{-5}	$+32.1$	$[26.6, 37.6]$
GPT-5.5 vs. GPT-5.4 mini	$+21.2$	49 / 10	2.7×10^{-7}	$+30.1$	$[24.4, 35.9]$
GPT-5.4 vs. GPT-5.4 mini	$+4.3$	30 / 22	0.33	-2.0	$[-9.3, 5.6]$
GPT-5.4 mini vs. Qwen 3.5 35B-A3B	$+16.3$	36 / 6	2.8×10^{-6}	$+11.1$	$[4.3, 17.9]$
Qwen 3.5 35B-A3B vs. Qwen 3.5 9B	$+1.1$	11 / 9	0.82	$+18.3$	$[12.2, 24.5]$
Claude Opus 4.6 vs. Qwen 3.5 9B	$+51.6$	96 / 1	1.2×10^{-27}	$+61.0$	$[55.3, 66.7]$

Judge design. We inherit the judge from Odysseys (Jang et al., 2026a), and concurrent work on checklist judges for computer-use agents independently supports several of its choices (Aggarwal et al., 2026). First, the judge decides each rubric item in its own call instead of issuing one verdict per task. A single holistic call can overlook one unmet requirement in a long trajectory, while a per-item call localizes each failure and gives partial credit, and Gym-Anything reports 93.3% task-level human agreement for a per-item checklist judge against 81.7% for a direct holistic judge. Second, we grade with a model rather than with checks on the final database state. Many rubric items grade how the agent reached its answer, such as reading Michael's meetings from the HooliCalendar UI for a standup summary or checking HooliChat for overnight messages, and these leave no trace in the final state. Many others grade content that no query can score, such as whether a summary covers the three requested topics in paste-ready bullets. In Gym-Anything, programmatic verifiers written by a model agreed with humans on only 43.3% of trajectories. Third, the judge holds privileged information. The rubrics were drafted against the seeded data and name the specific records a correct answer must use, such as the boarding-pass file for flight DN1562, so the judge knows what a correct answer contains while the agent must discover it. Fourth, the judge sees the full trajectory, including every shell command and its output, so work done through the shell is graded as well as work visible in screenshots. Finally, the judge is a Gemini model, which keeps grading independent of the Claude Code tooling used to build the environment, and our rubric score and perfect rate correspond to Gym-Anything's average score and pass rate.

Human–judge agreement. To validate the judge on MyPCBench, we sampled four GPT-5.6-sol trajectories from each of the nine source categories (e.g., `retrieval`, `long_horizon`). These source categories are encoded in task IDs, which exist alongside the six behavioral task types. The resulting 36 tasks and 220 rubric items were drawn with a fixed seed and without looking at the judge's verdicts. Two authors each graded a non-overlapping half (18 tasks) from the full action history and screenshots with the judge's labels hidden, so inter-annotator agreement is not available. Pooled agreement is 94.5% ($\kappa = 0.73$) at the rubric level and 83.3% ($\kappa = 0.64$) on the derived

perfect-task labels (Table 10). Because the sample comes from one model and consists mostly of passes (195 of 220 items), κ is the more informative number. The judge catches 19 of the 25 items the humans failed, and rubric-level disagreements are symmetric, with 6 items the judge passed and the humans failed and 6 the judge failed and the humans passed. The same full-trajectory-per-rubric judge was validated for Odysseys (Jang et al., 2026a), where `gemini-3.1-flash-lite-preview` reached $\kappa = 0.79$ (F1 0.95) against human graders on rubric items and $\kappa = 0.85$ (F1 0.93) on task success over 120 Claude Opus 4.6 trajectories and 1,225 rubric items. Together the two studies cover both a Claude and a GPT agent, and our agreement ($\kappa = 0.73$ and 0.64) falls in the same range on a harder, desktop-scale set of trajectories. Our rubric-level agreement of 94.5% is also in line with the 90.9% per-item agreement that Gym-Anything reports for its checklist judge on 60 trajectories.

Table 10: Pooled human versus judge labels on 220 rubric items and the 36 corresponding perfect-task labels.

	Rubric items		**Perfect-task labels**	
	Judge pass	**Judge fail**	**Judge pass**	**Judge fail**
Human pass	189	6	20	4
Human fail	6	19	2	10

Judge repeatability and run variance. To measure judge sampling noise alone, we re-judged the 184 Claude Opus 4.6 trajectories behind Table 1 three more times with the same judge and rubrics. The three passes give rubric scores of 84.9, 85.8, and 85.2 (sample SD 0.5) and perfect rates of 57.1, 58.7, and 56.5 (sample SD 1.1), against 85.1 and 58.2 in the table. Across the three passes, 93.5% of the 1,191 rubric items receive the same label and 135 of the 184 task scores are identical. To measure agent variance, we ran Muse Spark 1.1 three times on all 184 tasks under the same configuration. The runs score 80.7, 80.8, and 80.0 rubric (sample SD 0.4) and 44.0, 48.9, and 42.4 perfect (sample SD 3.4). Rubric score is therefore stable across agent runs, while the perfect rate moves by several points, because a task's perfect label flips whenever any one of its rubric items changes. The Muse row of Table 1 reports the first of these runs, so that every leaderboard row is one run per task.

G Persona-Prior Audit

Because the persona is a public fictional character, we audited whether knowledge of *The Office* could substitute for reading the environment. We labeled a task *canon-could-help* only when knowledge of the show could plausibly supply or prime an identifiable graded rubric requirement, such as choosing coworkers from the show as candidates or judging Michael's likely expertise. Character names, places, and Office-themed entities did not qualify on their own, and we neither used the task category as a signal nor counted information already supplied in the task instruction. An LLM applied these rules to every task with a written rationale, and we then checked every positive label by hand against the finalized rubrics. Under these strict rules, only 9 of the 184 tasks (4.9%) have any graded component that canon could prime. In all nine, the remaining rubric items still require seeded records or a live action, such as locating and editing a seeded notes file, extracting candidates from seeded LockedIn connections, computing a verdict from live order counts, or creating invitations, posts, and bookings in the applications.

The strict set is too small to compare models, so we also use a deliberately broad 73-task *persona-sensitive* split that includes every task involving habits, preferences, or relationships, where canon could conceivably help even if it is not sufficient. The remaining 111 tasks form the *record* split in Table 11. If memorized character knowledge drove performance, models should score higher on the persona-sensitive split. Every model perfects fewer tasks on it, by 5.6 to 20.7 points, and the Opus lead over GPT-5.4 is as large on the record split (30.6 points) as overall. Because the persona-sensitive tasks are also the inference-heavy ones, this split bounds rather than isolates any canon effect, which is why we pair it with the strict audit above.

Table 11: Perfect rate and rubric score (%) on the 111 record-determined tasks versus the broad 73-task persona-sensitive split, for all ten leaderboard rows.

	Perfect			Rubric score	
Model	**Record (111)**	**Persona (73)**	**Δ**	**Record**	**Persona**
Claude Opus 4.6	62.2	52.1	−10.1	88.1	80.4
GPT-5.6-sol	57.7	52.1	−5.6	86.7	83.2
GPT-5.6-luna	58.6	50.7	−7.9	84.0	82.1
Claude Sonnet 4.6	55.9	42.5	−13.4	81.7	73.7
GPT-5.5	52.3	34.2	−18.0	85.6	80.6
Muse Spark 1.1	52.3	31.5	−20.7	82.0	78.8
GPT-5.4	31.5	23.3	−8.2	53.4	48.7
GPT-5.4 mini	26.1	20.5	−5.6	53.1	54.1
Qwen 3.5 35B-A3B	10.8	2.7	−8.1	43.3	41.0
Qwen 3.5 9B	9.9	1.4	−8.5	25.9	21.3

H Family-Signature Plots

Figure 7 complements §5.5 by breaking failed-rubric hits down by failure mode and model family (left) and showing each model's error budget (right).

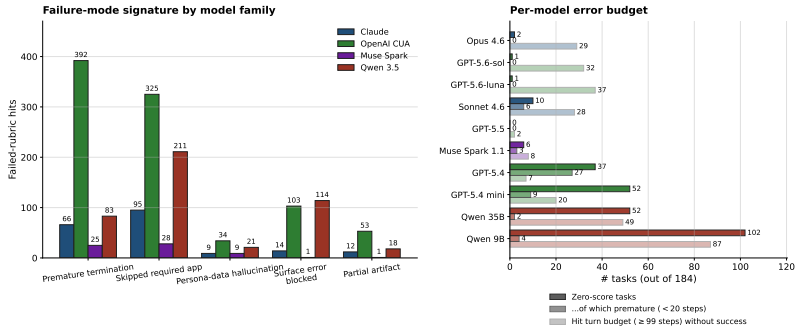


Figure 7: **Left:** failed-rubric hits by failure mode, grouped by family for the ten leaderboard rows, where Claude covers Opus 4.6 and Sonnet 4.6, OpenAI CUA covers GPT-5.6-sol, GPT-5.6-luna, GPT-5.5, GPT-5.4, and GPT-5.4 mini, Muse Spark covers Muse Spark 1.1, and Qwen covers 3.5 35B-A3B and 9B. **Right:** per-model error budget. The darkest bar counts zero-score tasks, the middle bar the subset that terminated under 20 steps, and the lightest bar trajectories that hit the 100-turn budget without success.

I Detailed Failure Modes

Table 12: Failure-mode counts on rubric items the judge marked failed, aggregated across the ten leaderboard rows of Table 1. Each failed item is matched against keyword rules over the judge's written reasoning, so an item can hit more than one mode. Per-family breakouts are in Figure 7.

Failure mode	Hits	Description
Skipped required app	659	Multi-app task closed after only some apps are visited, so the unvisited app's rubrics fail.
Premature `DONE`	566	Agent emits `DONE` before the trajectory satisfies the remaining rubric items.
Surface-error abandonment	232	Agent hits a captcha, console error, slow page, or modal and quits instead of recovering.
Partial artifact	84	Artifact started but not saved/exported (e.g., spreadsheet opened but never saved).
Hallucinated persona data	73	Agent fabricates a value instead of reading the seeded source.

Why the perfect rate falls faster than the rubric score. Failed rubric items tend to occur together. Skipped-app failures often appear alongside a premature `DONE` in the same trajectory, because the agent considers itself finished after the apps it did open. Surface errors likewise lead to partial-artifact failures, such as a spreadsheet that is opened but never saved. Because the perfect rate requires *every* rubric item to pass, a single one of these co-occurring failures costs the task its perfect score.

Per-family breakdown. The per-model counts behind Table 12 and Figure 7 show three family-level trends.

- **The GPT family stops too early.** Of the premature-`DONE` hits attributed to the GPT family in §5.5, GPT-5.4 mini accounts for 129, GPT-5.4 for 115, GPT-5.5 for 104, GPT-5.6-luna for 28, and GPT-5.6-sol for 16, compared with 30 for Opus and 36 for Sonnet. Each of the three older GPT models stops early roughly three times as often as either Claude model, while the GPT-5.6 models fall in the Claude range.
- **The Qwen family fails in different ways.** The two Qwen models account for 114 of the 232 surface-error abandonments (59 for Qwen 9B and 55 for Qwen 35B) and 211 of the 659 skipped-app hits, 128 of them from Qwen 9B alone. Qwen 35B also leads persona-data hallucination with

13 of the 73 hits, ahead of Muse Spark with 9 and each GPT-5.6 model with 8. Qwen 9B often emits malformed tool calls that mix the `computer` and `bash` schemas, so many of its failed items have no explanation that the keyword rules can classify.

- **The Claude family takes console-script shortcuts.** Every model had the same `bash` tool (Appendix D), and in the action logs from our initial evaluation the GPT models actually use it *more*, for 52% and 44% of their actions against 24% and 16% for the two Claude models (Table 14). What sets Claude apart is how it uses the shell rather than how often. Claude tends to read app state through `bash` instead of producing the UI side effect that the rubric grades, which is the console-script shortcut described below.

Console-script shortcuts (Claude only). We observed one failure pattern only in Claude trajectories. Instead of driving the visible UI, the agent opens a JavaScript console, or calls the app's REST endpoint directly with `curl`, and reads the persona's data from there. This is enough when the rubric only requires the agent to *know* a value. When the rubric requires a user-visible side effect, such as dragging a card, opening a project, or saving a file from the menu, the agent reads the data and declares the task `DONE` without producing that effect. A representative example is the trajectory on `hard_app-f026` from our initial evaluation, which uses only the console and API calls. The judge notes that the agent "investigates SprintBoard throughout the trajectory using API calls in the browser console… but it never actually opens the three SprintBoard projects."

Examples of skipped apps. The skipped-required-app mode covers 659 failed rubric items in which the agent finishes a multi-app task without ever opening one of the named apps. On `long_horizon-f066`, the agent archives a HooliChat conversation but never opens OddsMarket. On `long_horizon-f074`, in the computer-only GPT-5.4 run shown in the first row of Figure 11, the agent visits nine apps but never opens TableFind to make the required reservation. On `aggregation-f010`, the agent searches HooliChat extensively but never reads the Dundies categories file in `~/Documents`.

J Persona Specification and Event Chains

The Michael Scott persona is stored as a single JSON document (`personas/michael_scott.json`, included in the release) with fifteen top-level sections, which the generator reads in dependency order. Every seeded record in the 17-app environment can be traced back to one of these sections.

Top-level schema.

```
{
  "identity":           { name, age, city, address, employer, role,
                          salary, email, phone, gender, bio, ... },
  "contacts":           [ { name, relationship, frequency, email, phone,
                            shared_activities, apps_present_in,
                            message_personality, birthday, address }, ... ],
  "financial":          { checking_balance, savings_balance, credit_limit,
                          credit_used, monthly_income_net,
                          recurring_charges },
  "investments":        { cash_balance, holdings, order_history, dividends },
  "prediction_markets": { balance, total_invested, net_pnl,
                          active_positions, watchlist },
  "routines":           { commute, exercise, meals, improv_class },
  "trips":              [ { destination, dates, hotel, flights, ... }, ... ],
  "work":               { projects: [ ... ] },
  "tax_info":           { tax_year, w2, freelance_1099, deductions,
                          state_code },
  "planted_contradictions": [ ... ],
  "planted_dependencies":   [ ... ],
  "browsing_patterns":      { research_threads, routine_browsing,
                              humor_searches },
  "shopping":               { online_orders, wishlist },
  "app_overrides":          { hoolishop, lockedin, batbucks, speedtax,
                              hoolichat, etaxi, hangrydash, tablefind,
                              ... },
  "cross_app_events":       [ ... ]
}
```

Four of these sections drive cross-app consistency. The `cross_app_events` section defines events with side effects in several apps, so that a trip, for example, seeds rows in six apps. The `planted_contradictions` section adds deliberate red herrings that test whether agents read all sources, and `planted_dependencies` holds the records a rubric expects the agent to chain through. Finally, `app_overrides` holds per-app adjustments, such as a Cheskepdia booking that a later HangryDash record refers to.

Annotated event chain. Each `cross_app_events` entry produces correlated rows in every application that would plausibly record the event. The example below is the Cooper's Seafood House dinner-plan event, reproduced from the seed, with app names shown as the seed's internal lowercase identifiers.

```
{
  "type": "dinner_plan",
  "description": "Romantic dinner at Cooper's Seafood House for Holly",
  "date": "2026-03-28",
  "time": "7:30pm",
  "apps": ["tablefind", "hoolichat", "gringotts", "hoolimail",
           "hoolicalendar"],
  "generates": {
    "hoolichat_mention": {
      "contact": "Jim Halpert",
      "context": "Jim. JIM. I need your help. I'm taking Holly to Cooper's
       tonight. What do I wear? Should I bring flowers? Is it too much if
       I also bring a boombox? Please respond immediately."
    },
    "browser_history": [
      "coopers seafood house scranton reviews",
      "romantic restaurants scranton",
      "how to be charming at dinner wikihow",
      "what wine goes with steak date night"
    ],
    "calendar_event": true
  }
}
```

Each seeder then writes its share of this event. The web-app seeder writes the TableFind reservation and the Gringotts charge, the calendar seeder writes the HooliCalendar block, the browser seeder writes the Firefox history rows, and the chat seeder writes the message thread. Because every seeder reads the same event record, the whole chain stays internally consistent.

K TASK-REVIEW INTERFACE

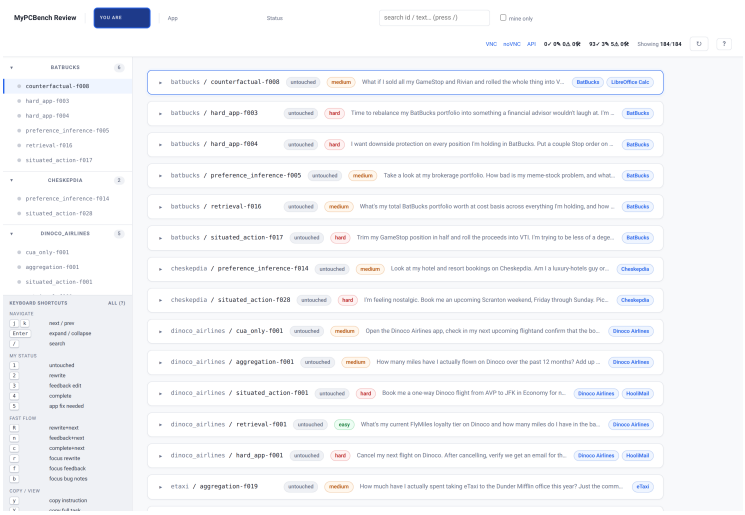


Figure 8: The MYPCBENCH task-review interface used during quality assurance. The left pane is grouped by primary application. Each row shows the task identifier, review state, difficulty, the verbatim instruction preview, and pills for the apps the task touches.

We built a single-page web reviewer (Figure 8) for the quality-assurance pass described in §4.1. It lists every task grouped by primary application, shows each task's instruction, difficulty, and apps inline, and tracks review state, which reviewers update with one-key keyboard shortcuts. Selecting a task opens a side pane with its rubric items and a link to the corresponding live application URL inside the VM, so a reviewer can run the task end-to-end without leaving the page.

L DATA GENERATION PIPELINE

The pipeline turns the persona JSON of Appendix J into a fully populated Linux desktop image. A single deterministic entry point runs one seeder for each kind of data, in dependency order. Because every seeder reads the same persona document, adding a new persona requires only a new JSON file.

Seeders. The pipeline first resolves the active persona and expands the JSON into derived fields such as paystub line items. The core seeders then write each kind of data from the spec. The web-app seeder writes the SQLite databases for the 17 Next.js apps and follows `cross_app_events`, so a single trip leaves correlated rows across Gringotts, Dinoco, HooliCalendar, HooliMail, HooliChat, and eTaxi. The remaining seeders build the on-disk Maildir at `/home/user/Maildir` and its HooliMail mirror, write HooliCalendar events together with the `.ics` files that LibreOffice reads, populate the Firefox profile and the per-app session cookies that keep every app logged in, and place the persona's documents (meeting notes, expense reports, itineraries, boarding passes, and resumes) under `/home/user`. A final step writes a Markdown summary of the persona for the task reviewers.

Determinism. Each seeder draws from a deterministic random number generator keyed on the persona name and the seeder's identifier. The reference time defaults to the date the image is built, so that the seeded data reads as current, but it can also be pinned to a fixed date. With the reference time pinned, identical inputs produce byte-identical outputs across runs and machines.

From persona to image. Once all seeders have run, a single `docker build` packages the populated home directory, the Firefox profile, and the 17 Next.js apps into the released environment image. A base snapshot is captured on the first boot of the QEMU guest, and every task starts from this snapshot, so the agent always sees the same initial state.

M GRADING AND RUBRIC PROMPTS

This appendix reproduces, verbatim from the released code, the prompts used to grade every task.

Judge system prompt.

```
You are an expert evaluator of desktop-agent trajectories.

You will receive:
- The user task (for context).
- ONE specific rubric item with a criterion and (optional) verification description.
- The agent's full action history (one line per step).
- Every screenshot from the trajectory, in chronological order.

Your goal is to decide whether this single rubric item is satisfied by the trajectory.

Evaluation rules:
- Judge ONLY the one rubric item you are given; ignore all other implicit requirements.
- Ground your judgment in what the screenshots and actions actually show. Do not invent
↪ state.
- Filtering / sorting / form requirements must be applied AND confirmed (visible) to count
↪ as satisfied.
- If the agent was blocked (captcha, access denied, crash, etc.) and therefore could not
↪ satisfy the rubric, report failure.
- If a later step UNDID the rubric (e.g. user-visible state was correct, then was
↪ overwritten with wrong data), report failure.

Respond in exactly this format:

Thoughts: <your reasoning, citing specific steps/screenshots>
Status: "success" or "failure"
```

Judge user prompt (one call per rubric). The user message is instantiated from the template below with the task instruction, the single rubric item being evaluated, and the agent's compacted action history, which lists every GUI action, every shell command with its returned output, and the agent's intermediate and final text responses. All screenshots from the trajectory (at most 100 under the turn budget) are attached in chronological order in the same message. When the rubric carries a `verification` note or a non-default `weight` (weights are shown to the judge but not used in any reported metric), two additional lines (`Verification: ...` and `Weight: ...`) are emitted between `Requirement` and `Full Action History`.

```
User Task (context only): {task_instruction}

Evaluate ONLY this rubric item:
Rubric ID: {rubric_id}
Requirement: {rubric_criterion}

Full Action History:
{action_history}

Screenshots attached below: {n_screenshots} (trajectory had {n_steps} total step(s)).

Decide whether the rubric ({rubric_id}) is satisfied. Use the required 'Thoughts:' / 'Status
↪ :' format.
```

Aggregation. Letting $s_r \in \{0, 1\}$ denote whether the judge returned "success" for rubric r of a task with N rubrics, the two reported per-task metrics are

$$\text{rubric score} = \frac{1}{N}\sum_{r=1}^{N} s_r, \qquad \text{perfect} = \mathbf{1}[\forall r : \; s_r = 1]\,.$$

N AGENT HARNESS

This appendix documents the released agent harnesses, including its interface, action space (Table 13), turn budget, snapshot reset, and the system prompts used for each model family.

Harness interface. The harness starts the Docker image (Ubuntu 24.04 with a GDM auto-login GNOME session and the 17 Next.js apps running as systemd services), waits until the desktop is ready (typically $\sim$90 s), and then enters the agent loop. At each turn it fetches a screenshot through the guest's OSWorld-compatible HTTP control API on port 5000 (`GET /screenshot`) and builds the agent message from the system prompt, the task instruction, the screenshot, and the accumulated history. It sends this message to the model's native API, parses the returned action, and executes it on the guest through the same control API (`POST /execute`) or its shell endpoint. The loop repeats until the agent emits `DONE` or `FAIL` or the turn budget runs out. Claude's screenshots are resized to 1280$\times$720 by configuration, Qwen's pass through the Qwen-VL token budget, and OpenAI's are sent as captured. Between tasks the harness restores a fresh base snapshot by rebuilding the copy-on-write overlay, which matches OSWorld's `revert_to_snapshot`, so every task starts from an identical state. VNC and noVNC are exposed only so that humans can watch a running agent. Two interchangeable backends are available. The default, `-backend qemu`, drives a QEMU/KVM guest directly from the host, while `-backend docker` runs the same guest inside the released Docker image for portability. Because the app databases are SQLite files inside the guest, an agent with the shell can read them directly as well as through the apps' HTTP endpoints. In the shell logs we inspected, agents read application state only through the apps' UIs or HTTP endpoints and we observed no direct database writes, which rubrics that require a user-visible side effect would not credit in any case.

Action space. Table 13 lists every action available to an evaluated agent. The top block is the unmodified OSWorld `pyautogui` action set, onto which we map each provider's action vocabulary, and the bottom block lists the one tool outside `pyautogui`.

Table 13: The MYPCBENCH action space.

Action	Parameters	Available to
`click`, `double_click`, `right_click`	x, y pixel coordinates	all CUA agents
`type`	text string	all CUA agents
`key`	key combination (e.g., `ctrl+c`)	all CUA agents
`scroll`	x, y, scroll amount	all CUA agents
`drag`	start (x, y), end (x, y)	all CUA agents
`wait`	duration (seconds)	all CUA agents
`screenshot`	—	all CUA agents
`done` / `fail`	—	all CUA agents
`bash`	shell command string	all computer+bash agents

Action distribution by model. To see how each model uses the shared tools, we recover the full action sequence that each of the six models from our initial evaluation emitted on all 184 tasks. The Claude and Qwen logs come from the leaderboard runs, while the GPT-5.5 and GPT-5.4 mini logs come from earlier full runs of those models, so their action totals differ from the step counts in Table 1. For Claude we read actions from the `tool_use` blocks in each `messages.json`. For GPT and Qwen we take one record per executed action from each `traj.jsonl` and match each task's action log to its scored trajectory by recorded step count, and the caption of Table 14 notes how many tasks differ. We map each executed `pyautogui` call back to the provider's `computer.*` action that produced it. Because the OSWorld-style Qwen agent emits `mouse_move` as a separate action, we merge each `moveTo` followed by a click into a single click so that click counts are comparable across agents. Any tool call outside `pyautogui` counts as a `bash` call, since the shell is the only non-`computer` tool in computer+bash mode. Figure 9 plots the resulting distribution, and Table 14 gives the per-action shares.

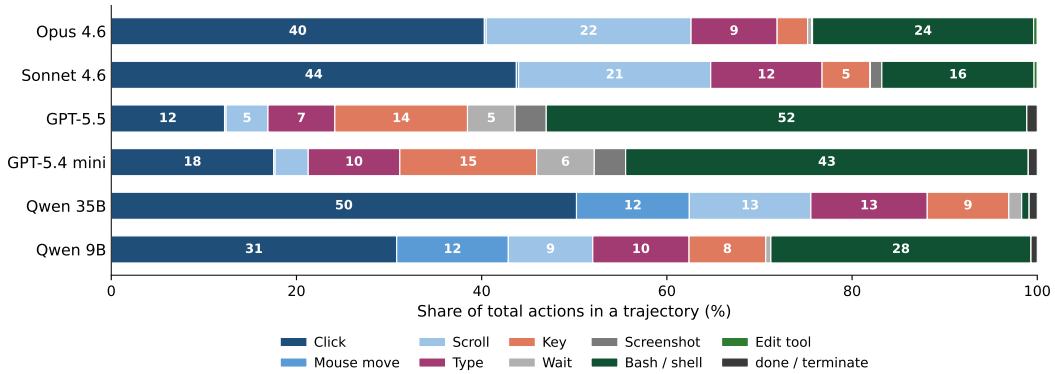


Figure 9: Per-model action distribution across all 184 trajectories, grouped into the categories in the legend. All six models from our initial evaluation have the same `computer` and `bash` tools, but the families use them very differently.

Table 14: Per-model action distribution across all 184 trajectories for the six models of our initial evaluation. Each cell is the share of that model's total actions, and the bottom rows give trajectory counts and absolute totals. For GPT and Qwen, the `bash` row counts every tool call outside `pyautogui`, so it measures reliance on the shell rather than the number of individual commands. Only Claude's action set includes `triple_click`. The OSWorld-style Qwen agent emits `mouse_move` as a separate action, whereas Claude's and OpenAI's APIs fold movement into the click, which raises Qwen's mouse-only share to about 12%. Each model was run once per task and covers all 184 tasks. For GPT-5.4 mini, Qwen 35B, and Qwen 9B, the logged action count differs slightly from the published step count on 3, 21, and 26 tasks, respectively.

Action	Opus 4.6	Sonnet 4.6	GPT-5.5	GPT-5.4 mini	Qwen 35B	Qwen 9B
`left_click`	37.6%	41.7%	11.8%	16.9%	**48.1%**	**29.5%**
`double_click`	0.5%	0.1%	0.2%	0.6%	1.1%	1.3%
`triple_click`	2.0%	1.9%	0.0%	0.0%	0.0%	0.0%
`right_click`	0.1%	0.0%	0.0%	0.0%	0.4%	0.1%
`drag`	0.0%	0.1%	0.2%	0.1%	0.7%	0.0%
`mouse_move`	0.2%	0.2%	0.2%	0.2%	12.2%	12.1%
`scroll`	22.1%	20.7%	4.5%	3.6%	13.1%	9.1%
`type`	9.3%	12.0%	7.2%	9.9%	12.6%	10.4%
`key`	3.3%	5.2%	14.3%	14.8%	8.8%	8.3%
`wait`	0.4%	0.0%	5.1%	6.2%	1.4%	0.6%
`screenshot`	0.1%	1.2%	3.4%	3.4%	0.0%	0.0%
`done/terminate`	0.0%	0.0%	1.1%	1.0%	0.9%	0.7%
`bash`	23.9%	16.4%	**51.9%**	**43.5%**	0.8%	28.1%
`str_replace_based_edit_tool`	0.4%	0.3%	0.0%	0.0%	0.0%	0.0%
Trajectories analyzed	184	184	184	184	184	184
Total actions	8,575	8,584	8,420	8,148	12,587	13,623

How the families use the same tools. Figure 9 shows that the families use the same `computer` and `bash` tools very differently on the same 184 tasks. Claude mixes UI actions and shell calls at a steady rate. On Opus, around 70% of actions go through the UI (click, scroll, type) and roughly 24% through `bash`, and Sonnet shifts toward more clicks and less `bash`. Part of this `bash` share is the console-script shortcut described in §5.5, in which the agent reads app state with `curl` without touching the visible UI. GPT-5.5 and GPT-5.4 mini rely on the shell the most. Shell calls make up 52% and 44% of their actions, about double Claude's share, and their UI footprint is correspondingly small (12–17% clicks, 4% scrolls, and 14–15% `key` presses). The two Qwen models behave differently from each other. Qwen 35B almost ignores the shell (0.8% `bash`) and relies heavily on clicks and `mouse_move` (48% and 12%), reflecting the OSWorld-style interface it was trained on. Its failures come from the hallucination mode it leads and the surface-error mode it shares with Qwen 9B (§5.5), not from tool access. Qwen 9B's 28% `bash` share, in turn, often reflects the malformed tool calls that mix the two schemas (Appendix I) rather than productive shell use.

When `bash` helps and when it hurts. Both Claude models use `bash` widely, with 155 of 184 Opus trajectories and 152 of 184 Sonnet trajectories invoking it at least once, but shell use does not predict success overall. Opus perfects 72.4% of tasks (21/29) when it never invokes `bash` and 55.5% (86/155) when it does. This gap is correlational rather than causal, because Claude turns to `bash` more often on the harder multi-app tasks, where the perfect rate is lower to begin with. It does, however, illustrate the failure mode described in §5.5. The shell is fast and reliable for *reading* app state through REST endpoints, but it is the wrong tool when the rubric requires a user-visible side effect such as moving a card or saving a file from the menu, as the console-and-API-only trajectory on `hard_app-f026` shows (Appendix I).

Implications for hybrid CUA designs. The action distributions (Figure 9) show that the same tool produces opposite behaviors in different families. A hybrid agent therefore needs a policy for choosing between `bash` and the UI based on *what the task must leave behind*, not only on what answer it must produce, because rubrics that grade user-visible side effects can penalize a correct `bash`-only path. Providing the tool is not enough on its own, and the harness or prompt should state when UI grounding is required instead of leaving that choice entirely to the agent. Since the shell helps some families and hurts others, a useful next step is per-family guidance, or harness-side constraints on side-effect-graded rubrics, rather than a single shared configuration.

Shared persona-and-environment context (all agents). Every evaluated agent, regardless of model family, receives the persona-and-environment block below, appended to its system prompt or, for OpenAI agents, to the first user message. This shared block keeps every agent grounded in the same persona, applications, and conventions, and it is reproduced verbatim from the released code.

```
## Persona

- Name: Michael Scott
- Email: `michael.scott@dundermifflin.com`
- Linux user: `user` (sudo password: `{CLIENT_PASSWORD}`)

## Environment

- Ubuntu 24.04 GNOME desktop. Browser: Firefox (pre-logged-in to every
  web app via the bookmarks toolbar).
- Pinned to dock: HooliChat, HooliWork, Firefox, LibreOffice Writer/Calc/Impress, VS Code.
- `/home/user/Documents/`, `/home/user/Downloads/`, `/home/user/Maildir/` hold persona files.
↪
- Python 3.12 and the LibreOffice CLI are available in the VM.

## Web apps

Each is served at `http://localhost:PORT`, pre-authenticated as the persona.

| Port | App             | Domain                                                 |
|------|-----------------|--------------------------------------------------------|
| 3001 | Gringotts       | personal banking: accounts, transactions, transfers |
| 3002 | BatBucks        | stock / crypto trading: portfolio, orders           |
| 3003 | OddsMarket      | prediction markets: bets, positions                 |
| 3004 | HooliChat       | direct + group messaging                            |
| 3005 | HooliWork       | workplace channels                                  |
| 3006 | eTaxi           | ride hailing: trips, drivers                        |
| 3007 | HangryDash      | food delivery: orders, restaurants                  |
| 3008 | TableFind       | restaurant reservations                             |
| 3009 | Kwik-E-Mart     | grocery orders, inventory                           |
| 3010 | HooliShop       | e-commerce: orders, carts, products                 |
| 3011 | Dinoco Airlines | flight bookings, itineraries                        |
| 3012 | Cheskepdia      | short-term rental bookings                          |
| 3013 | SprintBoard     | project tasks, sprints                              |
| 3014 | LockedIn        | professional networking, jobs, connections          |
| 3015 | SpeedTax        | tax returns, filings                                |
| 3016 | HooliMail       | email inbox                                         |
| 3017 | HooliCalendar   | events, invitations                                 |

## Output

- Place your final answer (numbers, text, file paths) as plain text in
  your last assistant turn before any stop signal.
```

Task-completion discipline (all agents). A second shared block, `COMPLETION_DISCIPLINE` in `agents/prompts.py`, tells the agent when *not* to terminate and is also appended to every system prompt.

```
Task completion discipline:
- Do NOT emit `DONE`, `terminate`, or any stop signal until you have actually completed the
↪ task. A task is only complete when you have produced the specific output the user asked
↪ for AND verified it looks correct.
- Use all available steps --- plan, act, observe, iterate. Don't bail out early just because
↪  the first approach didn't work.
- If something fails, try a different approach (different coordinates, different app,
↪ different bash command). Never give up on the first error.
- Always write your final answer (numbers, text, file contents) before terminating --- the
↪ grader reads your last response to check correctness.
- Only emit `FAIL` if the task is genuinely impossible (required data literally does not
↪ exist). Never use `FAIL` as a shortcut when the task is just hard.
```

Claude Computer Use system prompt. Claude Opus 4.6 and Claude Sonnet 4.6 use the released prompt `CLAUDE_CUA_SYSTEM_PROMPT` in `agents/prompts.py`. It follows Anthropic's recommended XML-tagged scaffold (`<role>`, `<instructions>`, `<safety>`, and `<environment>`) and inserts the shared `VERIFICATION_GUIDANCE`, `PARALLEL_TOOL_HINT`, `SHELL_NON_INTERACTIVE_NOTE`, and `KEYBOARD_SHORTCUT_HINT` blocks into `<instructions>`, with the persona block above placed inside `<environment>`. We reproduce the leading `<role>` and `<instructions>` text below.

```
<role>
You are an AI agent operating a Linux workstation. Your tools are `computer` (screenshot +
↪ mouse/keyboard), `bash` (shell commands in the VM), and `str_replace_based_edit_tool` (
↪ file view/create/str_replace/insert).
</role>

<instructions>
Stop signal: state your final answer in plain text, then emit ```DONE``` (or ```FAIL``` / `[
↪ INFEASIBLE]` if impossible).
...
</instructions>
```

OpenAI CUA operator prompt. For the OpenAI computer-use agents (every GPT row), the operator prompt is sent as the text portion of the first user message. The released constant, `OPENAI_CUA_OPERATOR_PROMPT` in `agents/prompts.py`, joins the lead-in below with `SAFETY_PREAMBLE`, `VERIFICATION_GUIDANCE`, `PARALLEL_TOOL_HINT`, `SHELL_NON_INTERACTIVE_NOTE`, and `KEYBOARD_SHORTCUT_HINT`, and the caller then appends the shared persona context.

```
You are an agent on a Linux desktop. Your tools are `computer` and `bash`.

Stop signal: state your final answer in plain text, then emit ```DONE``` (or ```FAIL``` / `[
↪ INFEASIBLE]` if the task is impossible).
```

Qwen tool-call system prompt. The open-weight agents (Qwen 3.5 35B-A3B and Qwen 3.5 9B) use the `qwen35vl_agent` vendored from OSWorld (Xie et al., 2024), which emits one structured XML `<tool_call>` per turn that the harness parses into the corresponding OSWorld `pyautogui` call. The system prompt below is reproduced verbatim from `agents/vendored_paper_results/qwen35vl_agent.py`. The shared persona context is appended to the same system message, together with a description of the `bash` tool in computer+bash mode.

```
You are a multi-purpose intelligent assistant. Based on my requests, you can use tools to
↪ help me complete various tasks.

# Tools

You have access to the following functions:
```

```
<tools>{tools_json}</tools>

If you choose to call a function ONLY reply in the following format with NO suffix:

<tool_call>
<function=example_function_name>
<parameter=example_parameter_1>
value_1
</parameter>
<parameter=example_parameter_2>
This is the value for the second parameter
that can span
multiple lines
</parameter>
</function>
</tool_call>

<IMPORTANT>
Reminder:
- Function calls MUST follow the specified format: an inner <function=...></function> block
↪ must be nested within <tool_call></tool_call> XML tags
- Required parameters MUST be specified
- You may provide optional reasoning for your function call in natural language BEFORE the
↪ function call, but NOT after
- If there is no function call available, answer the question like normal with your current
↪ knowledge and do not tell the user about function calls
- The current date is {today}.
- Collapsed screenshots appear as text: {collapse_text}
</IMPORTANT>

# Response format

Response format for every step:
1) Action: a short imperative describing what to do in the UI.
2) A single <tool_call>...</tool_call> block.

Rules:
- Output exactly in the order: Action, <tool_call>.
- Be brief: one sentence for Action.
- Do not output anything else outside those parts.
- If finishing successfully, use action=terminate with status=success in the tool call.
- If the task is infeasible or impossible, use action=terminate with status=failure in the
↪ tool call.
```

Muse Spark system prompt. Muse Spark 1.1 follows the provider's reference computer-use agent and its cookbook `build_system_prompt`. We adapted the capability block to the MYPCBENCH GNOME desktop and added a guest `bash` tool alongside the `computer` tool. The provider's prompt itself tells the model to prefer `bash` for code, data, and file inspection and to make one tool call per turn. As for every other agent, the shared persona context and task-completion discipline are appended. The prompt below is reproduced verbatim from `SYSTEM_PROMPT` in `agents/meta_bash.py`.

```
Current date: {date}.

You are an AI assistant specialized in computer use.
You perceive the screen via screenshots and control the desktop using a fixed set of APIs.

<SYSTEM_CAPABILITY>
* You are using an Ubuntu virtual machine with a {width}x{height} display and a GNOME
↪ desktop.
* Applications (Firefox, LibreOffice, VS Code, Files, Terminal, chat apps) are launched from
↪  the dock or desktop icons.
* You also have a `bash` tool that runs shell commands inside this same VM. Prefer `bash`
↪ over the GUI for code/data/file inspection tasks.
* DO NOT ask users for clarification. Take action using available tools.
* Note: the machine's timezone may differ from the user's expectation. Check the visible
↪ clock if precise local time matters.
* The password for sudo is "{client_password}".
</SYSTEM_CAPABILITY>

# Available Tools
The tool schema is provided separately by the API. Use `computer.computer` for computer
↪ actions, `bash` for shell commands inside the VM, and `computer.stop` when the task is
↪ complete or you cannot proceed safely. Make exactly ONE tool call per turn.

# Screen And Coordinates
```

```
Coordinates are normalized integers in [0, 1000], with (0, 0) at the top-left of the screen
↪ and (1000, 1000) at the bottom-right, regardless of the actual pixel resolution. Always
↪ read coordinates off the screenshot rather than guessing.

# Task Execution Strategy
- Focus on ONE action at a time.
- Only use the tools available to you.
- After each action, a screenshot is automatically returned showing the result.
- When you believe the task is done, VERIFY the result. Take a screenshot and confirm the
↪ change covers the ENTIRE target, not just part of it.
- If you notice something is incomplete or wrong during verification, fix it before
↪ proceeding. Do NOT stop with a known issue.

<IMPORTANT>
# 1. Understand The Task
* Before acting, re-read the task instructions carefully. Pay attention to exact wording.
↪ Application-specific terms may differ from everyday language.
* Follow the task literally. If a specific website or application is named, use it. Do not
↪ substitute a faster tool or script.
* Complete ALL requirements before stopping. If the task says "do X AND Y", both must be
↪ done.

# 2. Verify. Never Guess, Believe, Or Assume
* Every action that matters must be verified through a concrete check, not by interpreting
↪ ambiguous screenshots. If you find yourself thinking "probably worked", "appears to", "
↪ hard to tell", "seems like", or "better to trust", you have NOT verified.
* After making a change, confirm it took effect by reading the result back.
* After completing a task, verify the visible or functional result.

# 3. Interaction Principles
* The user's data lives across the VM's applications (mail, chat, banking, shopping,
↪ documents). Find and correlate it; do not invent facts.
* Do not guess or invent URLs. Use the app's visible navigation, menus, links, or search to
↪ find pages.
* For precise text selection, use keyboard navigation such as arrow keys with Shift rather
↪ than guessing pixel coordinates between characters.

# 4. Efficiency
* Once all requirements are met, stop. Do not add extra changes beyond what was asked.
* Stop when you finish.
</IMPORTANT>
```

O EXAMPLE TRAJECTORIES

Figures 10 and 11 together show one passing and one failing trajectory for each of three evaluated model families (Claude, GPT, and Qwen), complementing the full Opus run in Figure 4. Each row gives the verbatim task instruction issued to the agent, three screenshots from the actual trajectory (early, middle, and near the end), and a short note on the observed behavior that explains how the trajectory reaches the judge verdict shown in the pill. The six runs cover five tasks (`aggregation-f001`, `hard_app-f011`, `retrieval-f009`, `long_horizon-f074`, and `situated_action-f036`) and illustrate three of the failure modes and the family-level trends discussed in §5.5. The Qwen pair contrasts the larger MoE model (Qwen 3.5 35B-A3B) on a bounded action with the smaller dense model (Qwen 3.5 9B) on an aggregation task to show how the two sizes differ.

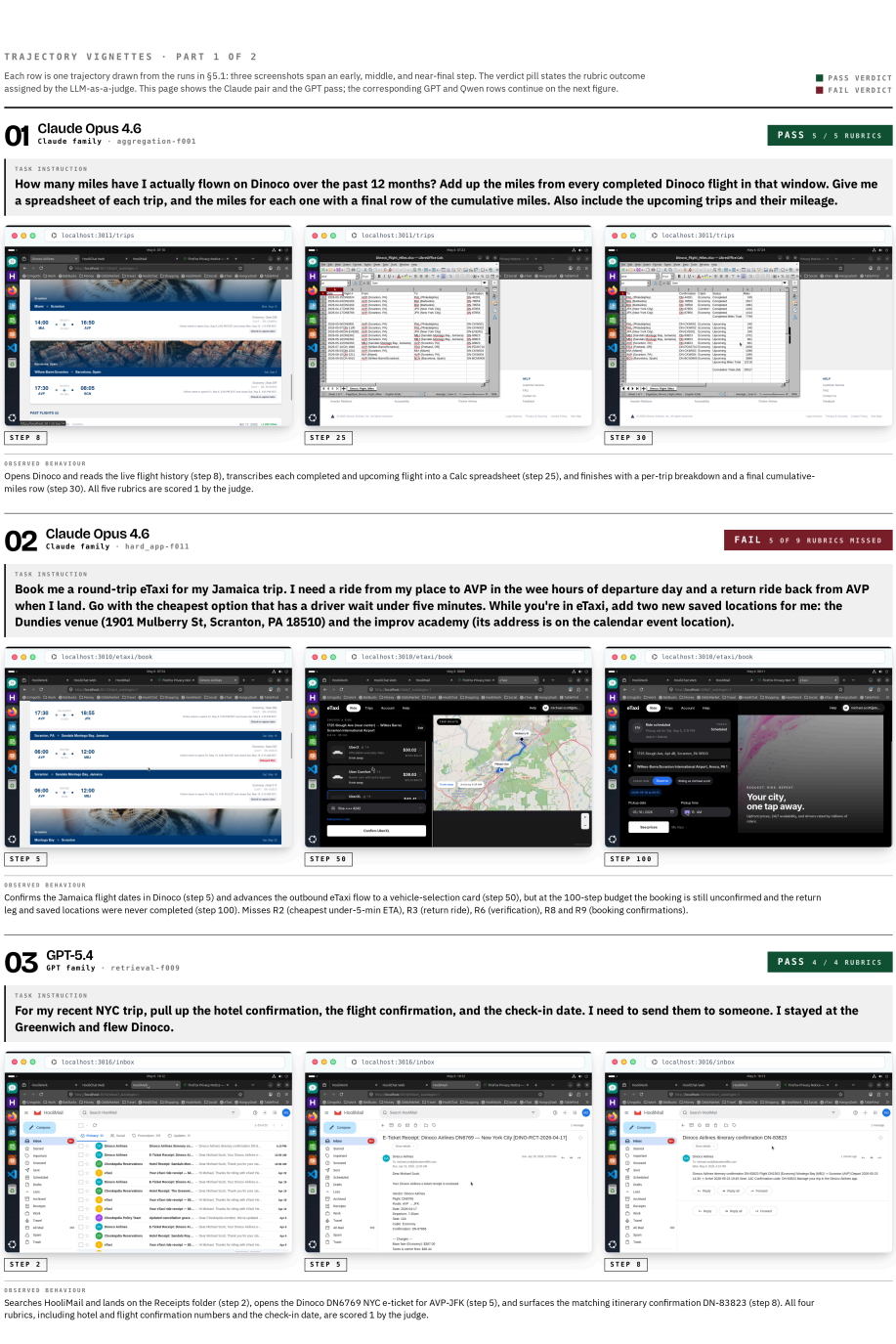


Figure 10: Trajectory vignettes, part 1 of 2. The rows show Claude Opus 4.6 PASS on `aggregation-f001`, Claude Opus 4.6 FAIL on `hard_app-f011`, GPT-5.4 PASS on `retrieval-f009`. Each row shows the verbatim task instruction, three screenshots (early, mid, near-final step), and an observed-behavior note linked to the rubric outcome assigned by the LLM-as-a-judge. The GPT row is taken from the computer-only configuration runs of the appendix ablation, chosen as representative of the family behaviors discussed in the main results.

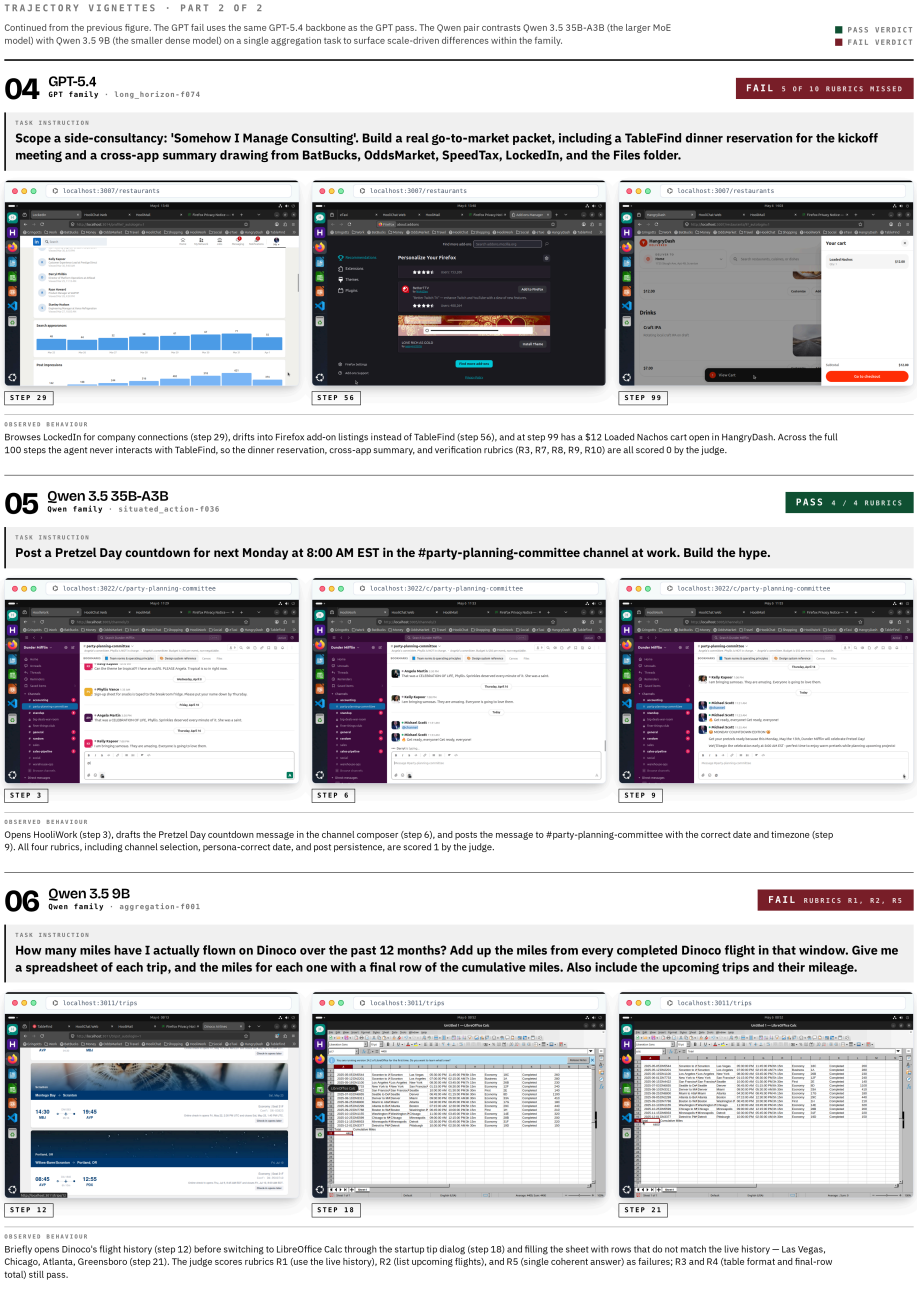


Figure 11: Trajectory vignettes, part 2 of 2 (continued from Figure 10). The rows show GPT-5.4 FAIL on `long_horizon-f074`, Qwen 3.5 35B-A3B PASS on `situated_action-f036`, Qwen 3.5 9B FAIL on `aggregation-f001`. Same layout convention as Figure 10. The GPT and Qwen rows are taken from the computer-only configuration runs of the appendix ablation, chosen as representative of the family behaviors discussed in the main results.

P LIMITATIONS

MYPCBENCH commits to one persona (Michael Scott) and one Linux/GNOME/Firefox software stack, choosing depth over persona diversity. It measures whether agents can use one coherent personal computer deeply, not whether their performance generalizes across demographics, locales, or device stacks. The persona is also intentionally low-sensitivity, since it is a public fictional character, so behaviors that emerge only when agents reason over genuinely sensitive personal data are not exercised by this benchmark. We treat that as an explicit choice about scope.

Grading relies on a single Gemini judge that sees the full action history, shell outputs, and agent responses alongside the screenshots. The judge agrees with blinded human grading on 94.5% of sampled rubric items (Appendix F), but its errors are correlated within a task. The failure-mode counts in Table 12 come from keyword rules over the judge's reasoning, so they should be read as a structural breakdown across the ten leaderboard rows rather than as a precise prevalence estimate. Each leaderboard row is a single run per task, and three independent Muse Spark runs (Appendix F) show that the perfect rate moves by several points between runs, so adjacent rows should be compared through the paired tests of Appendix F rather than through their point estimates. We report no timed human performance baseline, because the authors' end-to-end completion of every task (§4.1) shows only that each task can be done. Finally, the best agents' rubric scores are already close to saturation on every task type except multi-step orchestration (90–97, Table 6). The remaining room lies in the perfect rate (58% at best), multi-step orchestration (46%), and the 7+-app slice (36%), and we expect the perfect rate and the cross-app slope, together with new personas built from the released template, to keep separating models as they improve.

Q BROADER IMPACT

A personally intelligent agent is, by construction, an agent that can act on a user's full digital life. On the upside, a benchmark that explicitly tests cross-app, cross-history personalization should make it harder to ship assistants that look competent on stock-state demos but fail the moment they meet real personal data, and the failure-mode catalog identifies concrete behaviors for developers to measure and reduce (premature `DONE`, surface-error abandonment, skipped apps, hallucinated persona values, console-script shortcuts). On the downside, the same skills that drive a clean Dundies-lifecycle plan against Michael Scott's seeded desktop are the skills required to drive an agent against a real user's logged-in accounts. Numbers on this benchmark should not be read as a clearance to deploy CUA agents on production accounts. We mitigate the most direct misuse risks by (i) seeding only synthetic data tied to a public fictional persona, so the released image contains no real PII or real correspondence, (ii) hosting every web application locally inside the QEMU guest, so the credentials and form values the agent handles refer only to local services, and (iii) stating that benchmarking against the released image is the only intended use. The evaluation VM does retain outbound network access, and an optional host-provided `OPENAI_API_KEY` can be injected to generate in-character replies from contacts in the messaging apps, a feature enabled for every run reported in this paper.

R RELEASE ARTIFACTS

The anonymized code repository (`https://anonymous.4open.science/r/MyPCBench-21FF/`) and dataset (`https://huggingface.co/datasets/anonymousmypcbench/mypcbench-baseline`) host (i) the environment image (Docker + QEMU snapshot), (ii) the 184 task definitions, (iii) the per-task rubrics, (iv) the agent harness that connects standard CUA agents to the environment, (v) the configuration of the rubric-grading judge, (vi) the persona specification (`personas/michael_scott.json`), and (vii) the task-level scores behind every leaderboard row. With the camera-ready version we will also release the agent trajectories and the per-rubric judge outputs behind every leaderboard row, so that the leaderboard, the human–judge study, and the failure taxonomy can be audited independently.