MyPCBench: A Benchmark for Personally Intelligent Computer-Use Agents

Anonymous Authors[1]

Abstract

Current benchmarks for computer-use agents evaluate models in impersonal environments. This leaves a gap between evaluation and deployment as 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 benchmark six frontier and open-weight models under each provider's native CUA agent. Claude Opus 4.6 reaches 55.4%, the only model above 50%. Current agent failure modes focus on tasks that span many applications and long trajectories, where personalization stresses an assistant the most. We release the environment, the task set with rubrics, the agent harness, and the rubric-grading judge at [TBD].

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, together forming the record of a user's working and personal life. Current benchmarks for computer-use agents ignore this. Tasks run against empty desktops, generic application states, and minimally seeded databases. In most tasks, the agent is told exactly which application to open and the exact workflow to complete, with no realistic user data behind the application. 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 (Peter Steinberger, 2026) and Claude CoWork (Anthropic, 2026a)), evaluation has to keep up: it should test whether these systems are actually personal, and whether personalization improves or regresses with each new model release.

Existing agent benchmarks span the web (Zhou et al., 2024; Koh et al., 2024; Deng et al., 2023; He et al., 2024), full desktops (Xie et al., 2024; Bonatti et al., 2025; Yang et al., 2025), enterprise platforms (Drouin et al., 2024), and mobile devices (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. Web benchmarks in particular do not evaluate any site that requires logging in or variable personal information, which rules out a large fraction of what real users ask their assistants to do.

Simulated benchmarks have traded personalization for reproducibility because no benchmark has previously seeded a coherent user identity at the scale of a user's full personal computer. MyPCBench closes this gap. A single persona specification and a deterministic multi-application generator together produce an environment that is personal, consistent across applications, and reproducible.

Our canonical persona is Michael Scott, the regional manager of a paper company in Scranton, Pennsylvania. Michael's desktop is seeded with 1,429 bank transactions, 1,162 emails, 656 calendar events, 2,586 chat and workplace messages, 115 rideshare requests, 300 food-delivery orders, 126 retail orders, 28 restaurant reservations, and a Firefox profile with 35 bookmarks and 10,776 page-history visits, distributed across 17 pre-logged-in web applications and the

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review at the Second Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD) at ICML 2026. Do not distribute.

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

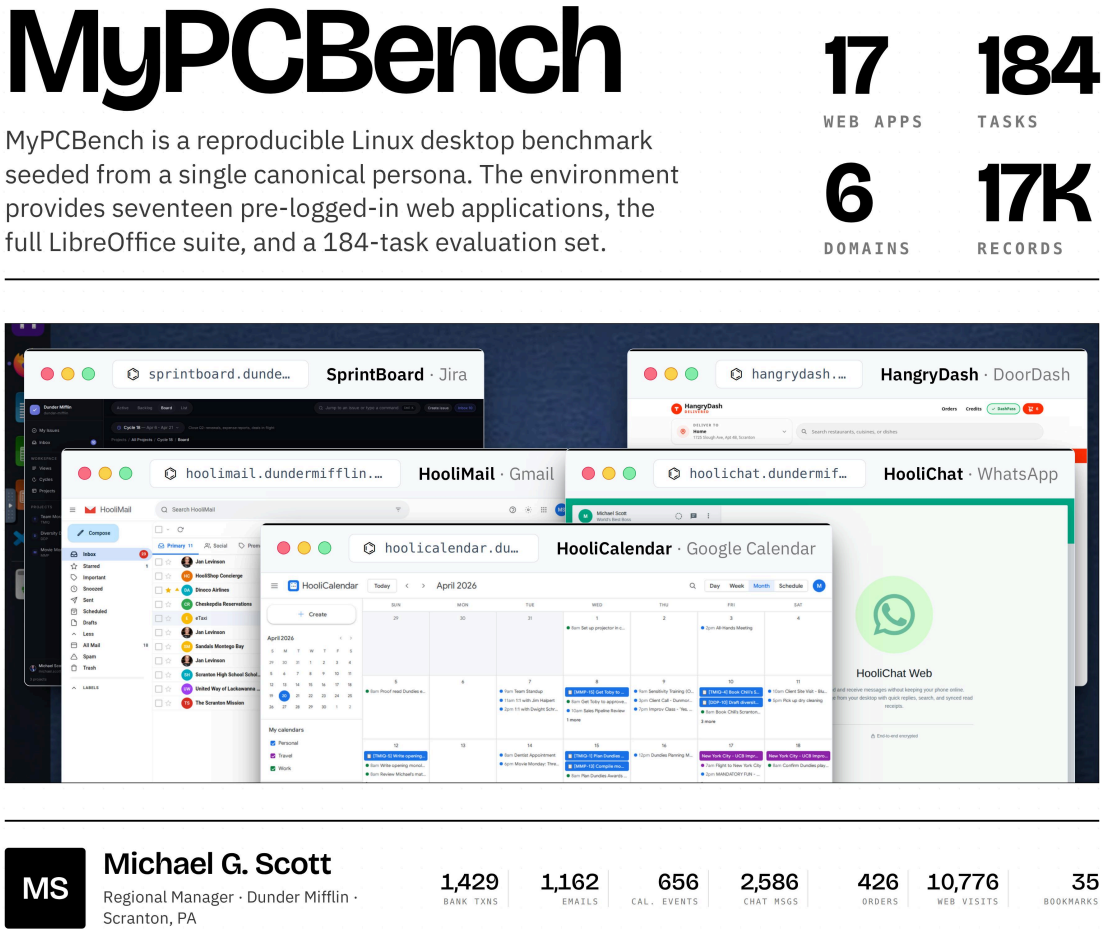

Figure 1. **Overview of MyPCBench.** A reproducible Linux-desktop benchmark for personally intelligent computer-use agents, seeded end-to-end from a single canonical persona (Michael Scott). The image hosts 17 pre-logged-in web apps mirroring real consumer products plus the full LibreOffice suite; the persona's records (1,429 bank txns, 1,162 emails, 656 calendar events, 2,586 chats, 426 orders, 10,776 web visits, 35 bookmarks; bottom strip) are cross-linked so that one trip leaves correlated records across every app that would plausibly book it.

surrounding desktop stack. The 17 apps and 184 tasks were chosen by manually inspecting the OpenClaw Discord, the largest personalized-LLM-agent community we are aware of, so that MyPCBench reflects the types of requests users actually issue to a personal assistant. Our contributions are:

1. A reproducible, cross-app-consistent desktop environment for evaluating personalized agents: 17 custom web apps and a full Linux desktop (Firefox, LibreOffice, file manager), 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, plus an agent harness that drives the standard CUA ReAct (Yao et al., 2023) loop against the environment and a rubric-grading LLM-as-a-judge.
3. Benchmarking of six closed- and open-weight models (Claude Opus / Sonnet 4.6, GPT-5.4 / mini, Qwen 3.5 35B-A3B / 9B) under each provider's native CUA agent, with a failure taxonomy and two scaling analyses (task length and personal-data load).

We release the environment image, the 184-task evaluation set with its rubrics, the agent harness, and the LLM-as-a-judge configuration at `[TBD]`. Our headline finding is that even the strongest current frontier agent (Claude Opus 4.6) perfects only 55.4% of MyPCBench tasks and only 36% of tasks that span 7 or more applications. GPT-5.4, Qwen 3.5 35B-A3B, and Qwen 3.5 9B all reach 0% perfect on the same 7+-app slice; the personalization regime stays wide open for future work.

2. Related Work

Web and desktop agent benchmarks. The first web benchmarks began with purely synthetic environments

(MiniWoB++ (Liu et al., 2018), WebShop (Yao et al., 2022)) and progressed to synthetic realistic websites (WebArena (Zhou et al., 2024), VisualWebArena (Koh et al., 2024)) and then to live Internet evaluations (Mind2Web (Deng et al., 2023), 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 manually handcrafted reward verifiers; Windows Agent Arena (Bonatti et al., 2025) and MacOSWorld (Yang et al., 2025) cover the other major operating systems. Static grounding benchmarks such as ScreenSpot (Li et al., 2025) and OmniAct (Kapoor et al., 2024) evaluate an LLM's ability to ground actions on desktop, browser, and mobile interfaces. LLM agents with personas have been evaluated in limited work settings: 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. Generative Agents (Park et al., 2023) studies a society of agents in a text-based environment and how they interact under separate personas.

Addressing the personalization gap. What every benchmark above shares is an impersonal environment. Desktop evaluations seed only what the task literally needs, so the agent never has to read across a real user's data or preferences. Web evaluations live on sites with no logged-in profile, so any task that requires personal information or pages behind a login is excluded by construction. You cannot, for example, evaluate calling an Uber, paying a friend back on Zelle, or fulfilling a user's usual DoorDash order. MyPCBench is a Linux-desktop benchmark with a fixed VM image and a deterministic snapshot reset (like OSWorld), but the desktop is seeded end-to-end with one user's data across every application, not just the data each task touches. The closest existing benchmark in spirit is TheAgentCompany, which simulates a software company populated by multiple coworker personas; MyPCBench differs by pinning a single user identity and by spanning the consumer-application surface that personal computers actually run (banking, travel, food delivery, calendar, messaging, etc.) rather than internal company tools. The result is that the same agent loop OSWorld-style evaluations already use can finally be pointed at tasks that require knowing who the user is.

3. MyPCBench

3.1. Environment

We release MyPCBench as a reproducible, open-source Linux desktop 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 modelled on a real-world analogue), the full LibreOffice suite (Writer, Calc, Impress), and a Firefox profile pre-loaded with a realistic browsing history and bookmark set. Two of the web apps, HooliWork (Slack) and HooliChat (WhatsApp), are also exposed as native desktop apps, mirroring how a real user would have them open. 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 Philadelphia trip generates a Cheskepdia (Airbnb) booking, two Gringotts (Chase) charges, a HooliCalendar (Google Calendar) block, two Dinoco (Delta) boarding passes, browsing history for "Radisson Blu Warwick", three Travel-folder emails, and HooliChat (WhatsApp) messages referencing the trip. The seed pipeline writes those records together so they line up at boot, and runtime cross-app effects keep them in sync. For example, a HangryDash (DoorDash) order posts a charge to Gringotts and drops a confirmation in HooliMail (Gmail).

(2) Persona coherence. The user is a specific person, not a generic account. Friends, co-workers, routines, and preferences are interleaved characteristics of a user's data, and our environment reflects that. Because the persona is Michael Scott, frontier coding agents can draw on *The Office*[1] canon to populate it with coherent, realistic data.

(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. Every feature touched by a benchmark task was exercised end-to-end by a human author against the live VM during the QA pass (§4.1).

3.2. Environment Creation And Infrastructure

Synthetic website generation. We built 17 clones of real consumer products using Claude Code (Anthropic, 2026b), each a full Next.js build rather than a static mock, following prior work on coding-agent web cloning (Zhou, 2026). Gringotts supports transfers, bill pay, Zelle, and statement downloads; Dinoco Airlines generates boarding passes with QR codes; eTaxi uses OSRM for realistic routing over 700+ Scranton-area locations; TableFind exposes a reservation inventory of 3,360 slots with hold-and-release semantics. Across the canonical Michael Scott seed, the 17 applications expose 185 distinct database tables and roughly 18,000 rows of state, of which around 17,000 are the user-facing records itemised in Figure 1 (transactions, emails, events, messages, orders, browsing history).

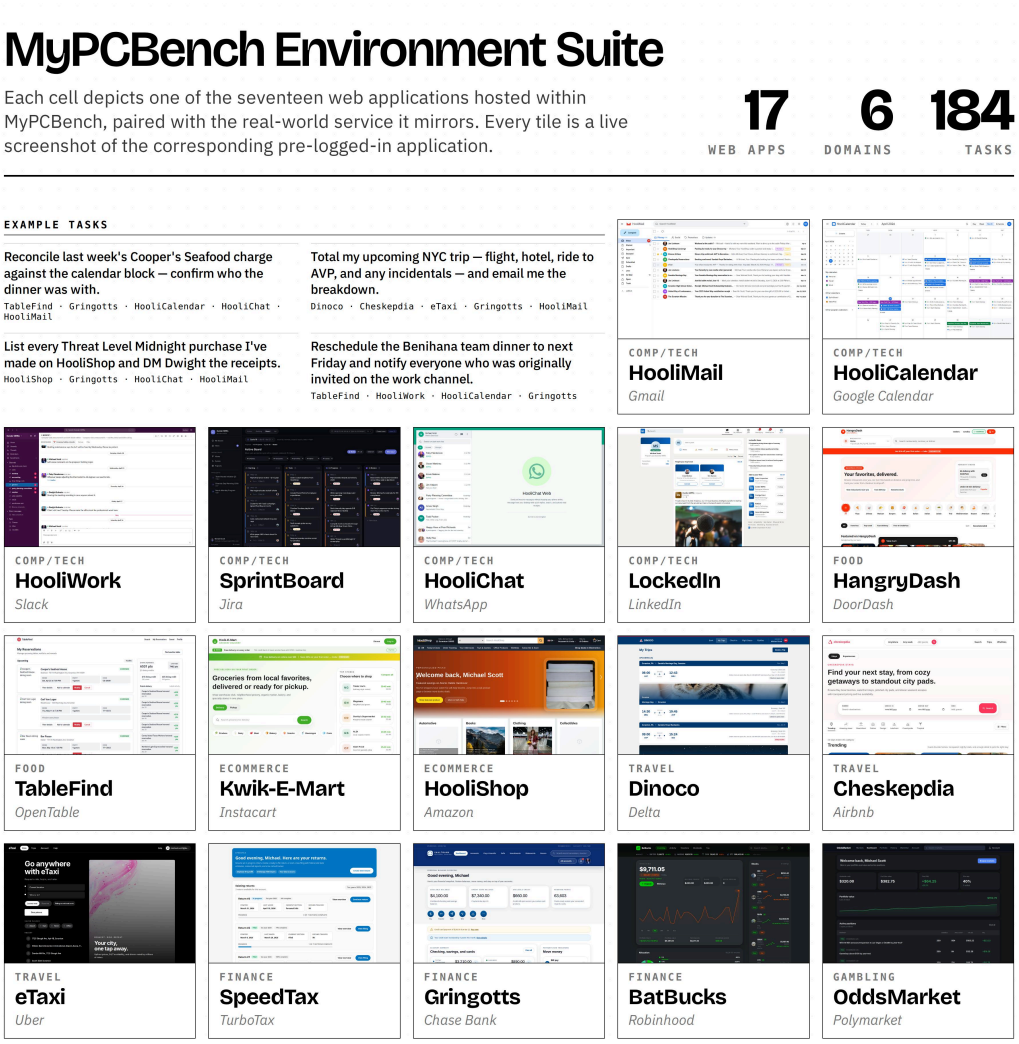

Figure 2. **MyPCBench environment suite.** The 17 pre-logged-in web apps span six SimilarWeb top-level domains (Computers/Tech, Finance, Travel, Food, Ecommerce, Gambling). The four example tasks (top-left) each require threading 3–5 of these apps together against Michael's seeded history.

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: SQLite databases for the 17 web apps with cross-consistent references, a Firefox profile with bookmarks/history/cookies/form-fields, and a filesystem of meeting notes, expense reports, trip itineraries, boarding-pass PDFs, and resume drafts.

Infrastructure. The default resource budget for a single virtual machine is 8 CPU cores and 16 GB of RAM. Boot-to-ready is about 90 seconds, and a base snapshot is captured after first boot and used to reset between tasks, avoiding state leakage. Adding personas or websites uses the same template (Appendix I).

4. Tasks and Evaluation Setup

4.1. Task Suite

MyPCBench includes 184 tasks, each one inspired by a real use case or request from the OpenClaw community. To ensure the task set reflects the true distribution of requests issued to personal-computer assistants, the authors manually sieved through 2,749 anonymized and paraphrased use-cases from the OpenClaw Discord, the largest community of users running personalized LLM agents on their own desktops to our knowledge. 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 remain-

Table 1. The six behavioural task types in MYPCBENCH, with counts and a representative instruction for each. The split separates analysis tasks (lookup, aggregation, inference, reconciliation; 72 tasks) from action tasks (bounded vs. orchestrated; 112 tasks). 68% of all tasks are multi-application. Full definitions in Appendix C.

Type	# Tasks (%)	Representative instruction
Bounded action	64 (35%)	*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%)	*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%)	*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%)	*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%)	*What's my current FlyMiles loyalty tier on Dinoco Airlines, and how many miles do I have in the bank?*
Pattern inference	11 (6%)	*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.*

ing requests were rewritten by Claude Code so the named entities (people, restaurants, dates, accounts) match Michael Scott's seeded data; in the same pass the coding agent generated a per-task rubric in the Odysseys (Jang et al., 2026) format. Both the rewrite and the rubric were then audited by the authors (§4.1). The final task set is stored as JSON, with each task carrying both its natural-language instruction and its rubric.

Quality assurance. Because coding agents generate the initial task drafts and the application clones, we manually verify both. Each task was reviewed by at least two authors through a custom web interface (Appendix H, 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 survived this round.

Domain coverage. We manually map each application to a top-level SimilarWeb[2] category by inspecting its real-world analogue, mirroring the categorization scheme used by Odysseys (Jang et al., 2026). The 17 apps span six SimilarWeb top-level categories and fourteen distinct subcategories. Computers, Electronics & Technology covers HooliMail, HooliCalendar, HooliWork, SprintBoard, HooliChat, and LockedIn. Finance covers Gringotts, BatBucks, and SpeedTax. Travel & Tourism covers Dinoco Airlines, Cheskepdia, and eTaxi. Food & Drink covers HangryDash and TableFind, Ecommerce & Shopping covers HooliShop and Kwik-E-Mart, and Gambling covers OddsMarket. The full per-app subcategory mapping, including Banking, Investing, Air Travel, and Restaurants & Delivery, is in Appendix A.

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

Apps per task. Tasks span from one to nineteen co-touched applications, and 68% are multi-application; 40% span at least two SimilarWeb top-level categories. The multi-application regime is what tests personalization, since the agent has to reconcile data across the persona's environment rather than drive a single tool in isolation. Figure 5 (Appendix B) gives the apps-per-task distribution, the per-domain task coverage, and the behavioural task-type split.

Task types. Independently of the SimilarWeb domain, every task is also assigned a behavioural *type* that captures what the agent must *do* with the persona's data, independent of which apps are involved (Table 1). We arrived at the type taxonomy by reading every task instruction and clustering by the primary capability under test. The resulting six categories cleanly split analysis tasks (lookup, aggregation, inference, reconciliation) from action tasks (bounded vs. orchestrated).

4.2. Agent Harness

The harness lets us point standard CUA agents at the MYPCBENCH environment with as little adaptation as possible. We model MYPCBENCH as a partially observable Markov decision process (Kaelbling et al., 1998) $\mathcal{E} = (\mathcal{S}, \mathcal{A}, \Omega, T)$. At each step t the agent receives an observation $o_t \in \Omega$ from the guest VM and emits an action $a_t \in \mathcal{A}$, which the harness executes through the OSWorld (Xie et al., 2024) VNC bridge. Our harness is a thin extension of the OSWorld runner. It boots the MYPCBENCH Docker image, restores a fresh QEMU snapshot before every task so each run begins from an identical desktop state, attaches to the VNC display, and drives the standard agent loop until the agent emits `DONE` or `FAIL` or exhausts the step budget.

Observation space. At each step the agent receives a 1280×800 screenshot of the full Linux desktop, augmented with the running tool-call history. Screenshots are passed unmodified to vision-language model APIs.

Action space. The action space is the unmodified OSWorld `pyautogui` surface (click, type, key, scroll, drag, wait, screenshot, done, fail). Frontier computer-use APIs (Claude Computer Use, OpenAI CUA) each define their own action vocabularies, and we map them onto this surface through a unified translation layer (for example, Claude's `computer.click` becomes `click`, and CUA's drag path becomes `drag`). Anthropic's Computer Use bundle additionally ships with a native `bash` tool and a `str_replace_based_edit_tool`, and we expose those through the same VNC channel when running Claude. The OpenAI CUA and Qwen CUA APIs do not expose equivalent tools; we therefore do not provide them to those agents, since adding tools they were not trained against would be both unfair and out of distribution. The full per-action table, the per-provider mapping, and the system prompts are in Appendix K.

4.3. Grading

We grade each trajectory against each rubric using an LLM-as-a-Judge, following the scheme proposed by Odysseys (Jang et al., 2026). Every task ships with its own set of rubrics: a list of natural-language criteria $\{r_1, \ldots, r_N\}$ authored alongside the task and audited during the QA pass. Across the suite, rubrics range from 3 to 13 items, with a mean of 6.5 per task and 1,191 rubric items in total. The judge runs once per rubric item over the *full* trajectory. It is shown the task instruction for context, that one rubric item, the agent's complete action history (the running tool-call log), and every screenshot the trajectory produced. The judge prompt caps the screenshot list at the most recent 200 as a defensive limit on context length; in practice no run in this paper hit that cap because we use a 100-step budget and capture one screenshot per step. The full action history is always provided in text form regardless of the cap. The judge returns "Status: success" or "Status: failure" for that item. A rubric item is considered satisfied iff the judge returns "success", and we denote this $s_r \in \{0, 1\}$. We write s_i for the per-task average $(\sum_r s_{i,r})/N_i$ on task i. We use `gemini-3.1-flash-lite-preview` as the judge model throughout; we use a single judge for all runs (Appendix M discusses the resulting correlation in judge errors across rubrics on the same trajectory).

We report three metrics per model. The *rubric score* $\overline{s} = \frac{1}{T}\sum_i s_i$ is the per-task average of rubric pass rates (each rubric weighted equally) averaged across tasks; it credits partial completion. The stricter *perfect rate* $\frac{1}{T}\sum_i \mathbb{1}[s_i = 1]$ requires every rubric in a task to pass. *Trajectory Efficiency*, also from Odysseys, measures how much rubric score the agent extracts per step,

$$\text{Traj. Eff.} \;=\; \frac{1}{T}\sum_{i=1}^{T}\frac{s_i}{n_i},$$

where n_i is the number of agent steps on task i. Trajectory Efficiency penalises agents that arrive at a correct outcome only after substantial wasted effort, and we report it scaled by 100 (percent of the rubric satisfied per agent step) for readability. The full judge prompt is in Appendix J.

5. Experiments and Analysis

5.1. Main Results

We evaluate six models on the full 184-task suite using each provider's native computer-use (CUA) agent (§4.2). The four frontier closed-weight models are Claude Opus 4.6 and Claude Sonnet 4.6 (Anthropic, 2026c), GPT-5.4 and GPT-5.4 mini (OpenAI, 2026). The two open-weight models are Qwen 3.5 (Qwen Team, 2026) 35B-A3B and 9B, picked for contrasting scale within the same family. Every run uses a 100-step budget and the shared persona-and-environment context reproduced in Appendix K, and all runs are graded by the same Gemini judge (§4.3). Table 2 reports the three metrics from §4.3 alongside the average and median step counts each agent actually consumed.

Table 2. Main results on the 184-task suite under each provider's native CUA agent (`gemini-3.1-flash-lite-preview` judge, 100-step budget, shared persona context). *Perfect* is the fraction of tasks for which every rubric in the task passed. *Rubric* is the per-task average of rubric pass rates. *T.Eff.* is rubric score per agent step (Odysseys) in percent. Claude Opus 4.6 leads on every metric, with a perfect rate more than 2× any non-Claude model and 13× Qwen 9B.

Model	Perfect ↑	Rubric ↑	Steps	T.Eff. ↑
API (closed-weight)				
Claude Opus 4.6	**55.4**	**81.8**	46.5	**3.61**
Claude Sonnet 4.6	39.1	65.4	45.8	3.03
GPT-5.4	23.9	50.5	41.2	2.26
GPT-5.4 mini	18.5	47.5	**30.8**	2.31
Open weights				
Qwen 3.5 35B-A3B	10.3	39.0	53.2	1.38
Qwen 3.5 9B	4.3	20.2	42.8	0.78

Closed-weight frontier agents lead by a wide margin. Claude Opus 4.6 reaches **55.4%** perfect at 81.8% rubric score, the only model above 50% perfect and more than twice the perfect rate of any non-Claude model. Within the API tier the Opus, Sonnet, GPT-5.4, GPT-5.4 mini ordering is preserved across both metrics, with Sonnet sitting roughly halfway between Opus and the GPT-5.4 family on

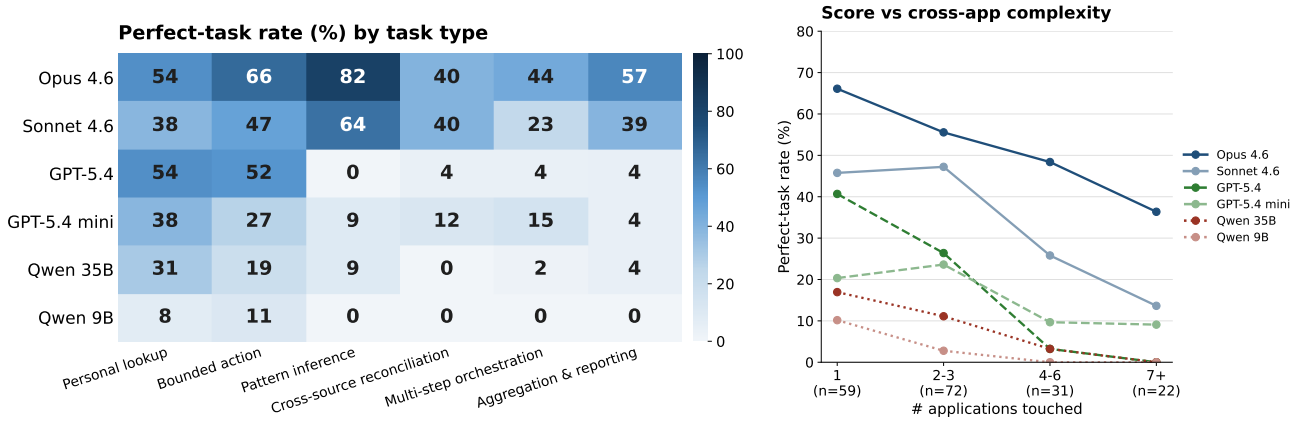

Figure 3. **Left:** per-task-type perfect-task rate (%), models × task types. Pattern inference is the largest single-category gap (Opus 82%, GPT-5.4 0%); on the three analysis categories (aggregation, multi-step orchestration, cross-source reconciliation) the GPT-5.4 family and Qwen all stay below 16% perfect. **Right:** perfect-task rate versus the number of distinct applications a task touches. At 7+ apps, GPT-5.4, Qwen 35B, and Qwen 9B all reach 0% perfect, and only the Claude tier and GPT-5.4 mini perfect any 7+-app task at all.

both. The open-weight Qwen 3.5 models trail every closed-weight model. Within the Qwen family the larger 35B-A3B more than doubles 9B on perfect rate (10.3% vs. 4.3%) and on rubric score (39.0% vs. 20.2%); both still fall well below GPT-5.4 mini.

Trajectory Efficiency adds a step-budget view. Opus extracts **3.61** rubric points per step, almost 5× Qwen 3.5 9B (0.78). GPT-5.4 mini consumes the fewest steps per task (30.8) but converts them less productively than the Claude tier, so its Traj. Eff. (2.31) still trails Sonnet (3.03) by 0.7 points. Average step count therefore does not by itself predict efficiency: low step counts can reflect either tight execution (Sonnet, 45.8 steps, Eff. 3.03) or premature stopping (Qwen 9B, 42.8 steps, Eff. 0.78), and the two are distinguished by the failure-mode breakdown in §5.4.

5.2. Performance by Task Type

The aggregate gap in Table 2 hides large differences across the six task types. Figure 3-left shows the per-type perfect-task rate for every model.

Two categories localise the gap. *Bounded action* and *personal lookup* are the only types where every model in the API tier clears 38% perfect; GPT-5.4 matches Sonnet on personal lookup (both 54%) and overtakes it on bounded action (52% vs. 47%). The remaining four categories all require either reasoning over persona history or coordinating writes across multiple apps, and the gap to Opus widens accordingly. The largest single-category gap is *pattern inference*, where Opus reaches 82% perfect and GPT-5.4 reaches 0% perfect on the same 11 tasks. These tasks ask the agent to infer an unstated rule from many records ("what do I usually tip?"), and the rubric only credits answers that match the rule the seeded history supports. On the three remaining analysis categories (aggregation, multi-step orchestration, cross-source reconciliation), the GPT-5.4 family and both Qwen models stay below 16% perfect, and Qwen 9B records zero perfect tasks across the three combined.

5.3. Performance Scaling by Steps and Apps

The two scaling axes are (a) the number of distinct applications a task touches and (b) the number of agent steps the trajectory consumes. The Figure 3-right curve covers the apps axis (per-bin numbers in Table 6, Appendix D); a step-budget scaling law (cumulative perfect-task rate at each step budget) is in Figure 6 (Appendix D).

Apps and steps both stress horizon. Every model degrades as apps-touched grows, and the drop is sharper on perfect rate than on rubric score. From single-app to 7+-app bins, the perfect rate falls from 66%→36% for Opus, 50%→14% for Sonnet, 41%→0% for GPT-5.4, and 23%→9% for GPT-5.4 mini; the Qwen 35B model and Qwen 9B both end at 0% in the 7+-app bin (Figure 3-right; per-bin numbers and rubric-score equivalents in Table 6). The step axis (Figure 6) sorts the cumulative perfect-rate curves into three regimes: Opus is still adding 7pp between steps 60 and 100, the GPT-5.4 family flattens by step 60 ($<$ 2pp in the last 40 steps), and Qwen saturates by step 25. The two axes interact with the task-type findings: 68% of MyPCBench tasks span 2+ apps and 35% touch 4+, so the steep cross-app slope for the GPT and Qwen families directly produces the low perfect rates on the analysis categories of §5.2.

5.4. Personalization-Specific Failures

We read every failed-rubric judge explanation and tagged it with up to five categories: skipped required app, premature DONE, surface error as terminal, partial artifact, and hallucinated persona data (definitions and per-model counts in Appendix F). Skipped required apps (578 hits) and prema-

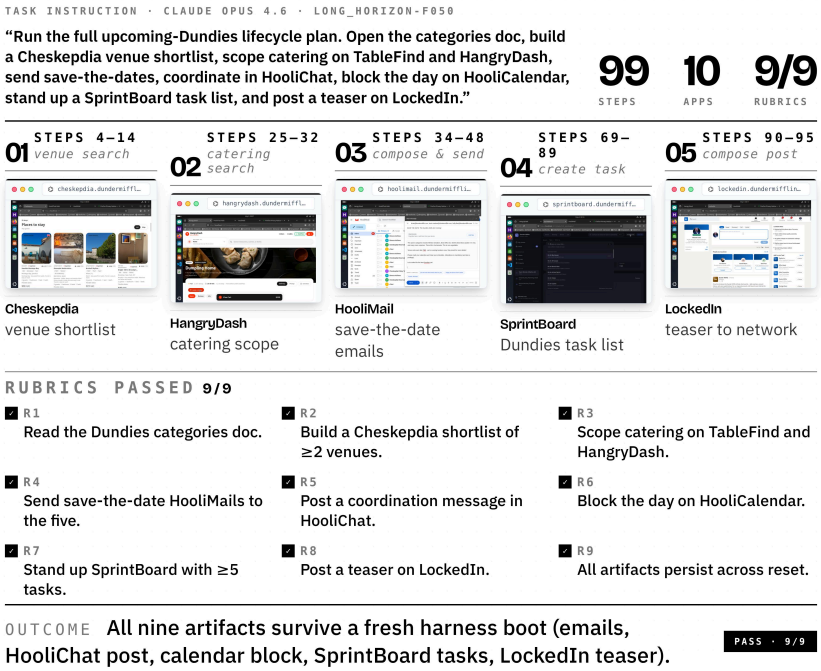

Figure 4. **One full successful Opus trajectory: Dundies-lifecycle plan on `long_horizon-f050`** (99 steps, 10 apps, 9/9 rubrics). Cells are real screenshots from the steps where the agent is actively driving each app; the bottom strip enumerates the nine rubric criteria the judge marked passed.

ture DONE (532) account for most of the loss; surface-error abandonment (123), partial artifact (90), and hallucinated persona data (36) follow. The three model families concentrate in different modes: GPT in premature DONE (307 of 532 hits), Qwen in persona-data hallucination (24 of 36), and Claude in console-script shortcuts that hit app REST endpoints with its native `bash` tool instead of driving the visible UI. Mean step count on zero-score trajectories splits the families the same way: GPT-5.4 (17.7), GPT-5.4 mini (19.7), and Sonnet (29.7) abandon early, while Opus (52.1), Qwen 35B (52.6), and Qwen 9B (39.2) keep going until they hit the step budget. Figure 4 traces a clean Opus run end-to-end for contrast; per-family pass/fail vignettes are in Appendix L.

6. Conclusion

MyPCBench seeds the empty desktop of computer-use evaluation with a coherent persona: 17 cross-consistent web apps, 17,000 personal records, and 184 tasks inspired by OpenClaw use cases. Claude Opus 4.6 leads at 55.4% perfect. We release the environment, tasks, harness, and judge for MyPCBench at [TBD]: We hope MyPCBench serves as a shift towards evaluating personalized computer-use agents and provides a baseline for identifying and improving the performance of frontier agents as truly personal assistants.

Impact Statement

A personally intelligent agent is, by construction, an agent that can act on a user's full digital life. We see two broad classes of impact. 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 catalogue gives developers a concrete checklist (premature DONE, surface-error abandonment, skipped apps, hallucinated persona values, console-script shortcuts) to drive against. 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 immediate dual-use surface 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 credentials and form values cannot be exfiltrated to the live web during evaluation, and (iii) recommending offline benchmarking against the released image as the intended use, and only that.

References

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.

Bonatti, R., Zhao, D., Bonacci, F., Dupont, D., Abdali, S., Li, Y., Lu, Y., Wagle, J., Koishida, K., Bucker, A., et al. Windows agent arena: Evaluating multi-modal os agents at scale. *ICML*, 2025.

Deng, X., Gu, Y., Zheng, B., Chen, S., Stevens, S., Wang, B., Sun, H., and Su, Y. Mind2web: Towards a generalist agent for the web. *NeurIPS*, 2023.

Drouin, A., Gasse, M., Caccia, M., Laradji, I. H., Del Verme, M., Marty, T., Boisvert, L., Thakkar, M., Cappart, Q., Vazquez, D., et al. Workarena: How capable are web agents at solving common knowledge work tasks? *ICML*, 2024.

He, H., Yao, W., Ma, K., Yu, W., Dai, Y., Zhang, H., Lan, Z., and Yu, D. Webvoyager: Building an end-to-end web agent with large multimodal models. *ACL*, 2024.

Jang, L. K., Koh, J. Y., Fried, D., and Salakhutdinov, R. Odysseys: Benchmarking web agents on realistic long horizon tasks. *arXiv preprint arXiv:2604.24964*, 2026.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

Kapoor, R., Butala, Y. P., Russak, M., Koh, J. Y., Kamble, K., Alshikh, W., and Salakhutdinov, R. Omniact: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web. *ECCV*, 2024.

Koh, J. Y., Lo, R., Jang, L., Duvvur, V., Lim, M., Huang, P.-Y., Neubig, G., Zhou, S., Salakhutdinov, R., and Fried, D. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *ACL*, 2024.

Li, K., Meng, Z., Lin, H., Luo, Z., Tian, Y., Ma, J., Huang, Z., and Chua, T.-S. Screenspot-pro: Gui grounding for professional high-resolution computer use. *MM*, 2025.

Liu, E. Z., Guu, K., Pasupat, P., Shi, T., and Liang, P. Reinforcement learning on web interfaces using workflow-guided exploration. *ICLR*, 2018.

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

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., and Bernstein, M. S. Generative agents: Interactive simulacra of human behavior. *UIST*, 2023.

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

Qwen Team. Qwen3.5-35b-a3b. https://huggingface.co/Qwen/Qwen3.5-35B-A3B, 2026.

Rawles, C., Clinckemaillie, S., Chang, Y., Waltz, J., Lau, G., Fair, M., Li, A., Bishop, W., Li, W., Campbell-Ajala, F., et al. Androidworld: A dynamic benchmarking environment for autonomous agents. *ICLR*, 2025.

Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T. J., Cheng, Z., Shin, D., Lei, F., et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *NeurIPS*, 2024.

Xu, F. F., Song, Y., Li, B., Tang, Y., Jain, K., Bao, M., Wang, Z. Z., Zhou, X., Guo, Z., Cao, M., Yang, M., Lu, H. Y., Martin, A., Su, Z., Maben, L., Mehta, R., Chi, W., Jang, L., Xie, Y., Zhou, S., and Neubig, G. Theagentcompany: Benchmarking llm agents on consequential real world tasks. *NeurIPS*, 2025.

Xue, T., Qi, W., Shi, T., Song, C. H., Gou, B., Song, D., Sun, H., and Su, Y. An illusion of progress? assessing the current state of web agents. *COLM*, 2025.

Yang, P., Ci, H., and Shou, M. Z. Macosworld: A multilingual interactive benchmark for gui agents. *NeurIPS*, 2025.

Yao, S., Chen, H., Yang, J., and Narasimhan, K. Webshop: Towards scalable real-world web interaction with grounded language agents. *NeurIPS*, 2022.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models. *ICLR*, 2023.

Zhou, S. Webarena-infinity: Generating browser environments with verifiable tasks at scale. *shuyanzhou.com*, 2026.

Zhou, S., Xu, F. F., Zhu, H., Zhou, X., Lo, R., Sridhar, A., Cheng, X., Ou, T., Bisk, Y., Fried, D., et al. Webarena: A realistic web environment for building autonomous agents. *NeurIPS*, 2024.

A. Application Details

Table 3 summarises 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. Live screenshots of every application are in Figure 2 (§3).

App	Analogue	SimilarWeb category	Subcategory	Description
HooliMail	Gmail	Computers, Electronics & Tech	Email	Gmail-style web client over a local Maildir; 667 seeded messages across Inbox, Sent, and four labelled folders.
HooliCalendar	Google Calendar	Computers, Electronics & Tech	Productivity	Google-Calendar-style scheduling app exposing 637 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 co-workers.
LockedIn	LinkedIn	Computers, Electronics & Tech	Social Media Networks	LinkedIn-style professional network exposing the persona's profile, feed, and connections.
HangryDash	DoorDash	Food & Drink	Restaurants & Delivery	DoorDash-style food delivery surface with order history and active carts.
TableFind	OpenTable	Food & Drink	Restaurants & Delivery	OpenTable-style restaurant reservation app with past bookings and search.
Kwik-E-Mart	Instacart	Ecommerce & Shopping	Marketplace	Instacart-style grocery delivery with multi-store carts and an order log.
HooliShop	Amazon	Ecommerce & Shopping	Marketplace	Amazon-style retail front with personalised recommendations, orders, and a cart.
Dinoco	Delta	Travel & Tourism	Air Travel	Delta-style airline app exposing flights, loyalty, baggage, and check-in.
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 1,370-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.

Table 3. 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.

B. Task-Distribution Plots

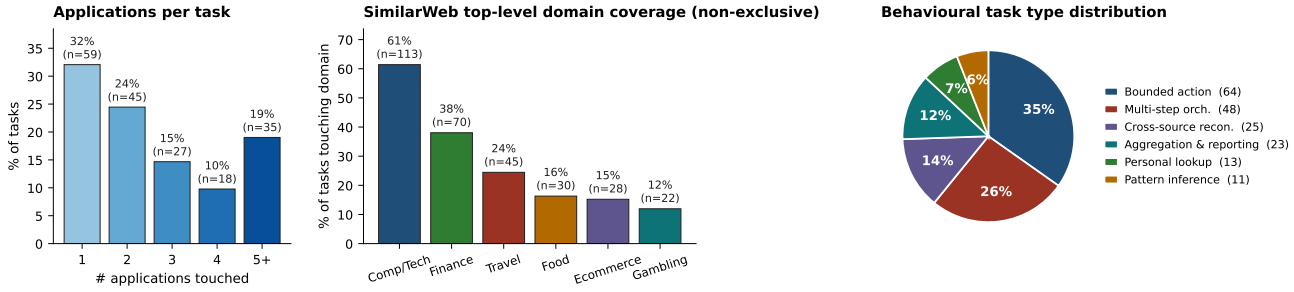

Figure 5. **Left:** distribution of tasks by the number of distinct applications they touch. **Middle:** fraction of tasks that touch at least one application in each SimilarWeb top-level category (non-exclusive; a single multi-app task can contribute to several bars). **Right:** behavioural task-type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

C. Task-Type Definitions

Table 4. The six behavioural task types in MYPCBENCH: definition, count, and a representative instruction. Counts cover all 184 tasks. (Main paper Table 1 reproduces only the counts and example instructions.)

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 artefacts (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. Per-Task-Type, Cross-App, and Step-Budget Scaling

Tables 5 and 6 are the raw numbers behind Figure 3. Figure 6 reads the step axis as a scaling law: at each step budget X on the horizontal axis, the curve plots the fraction of the 184 tasks the model graded perfect with $\leq X$ agent steps consumed.

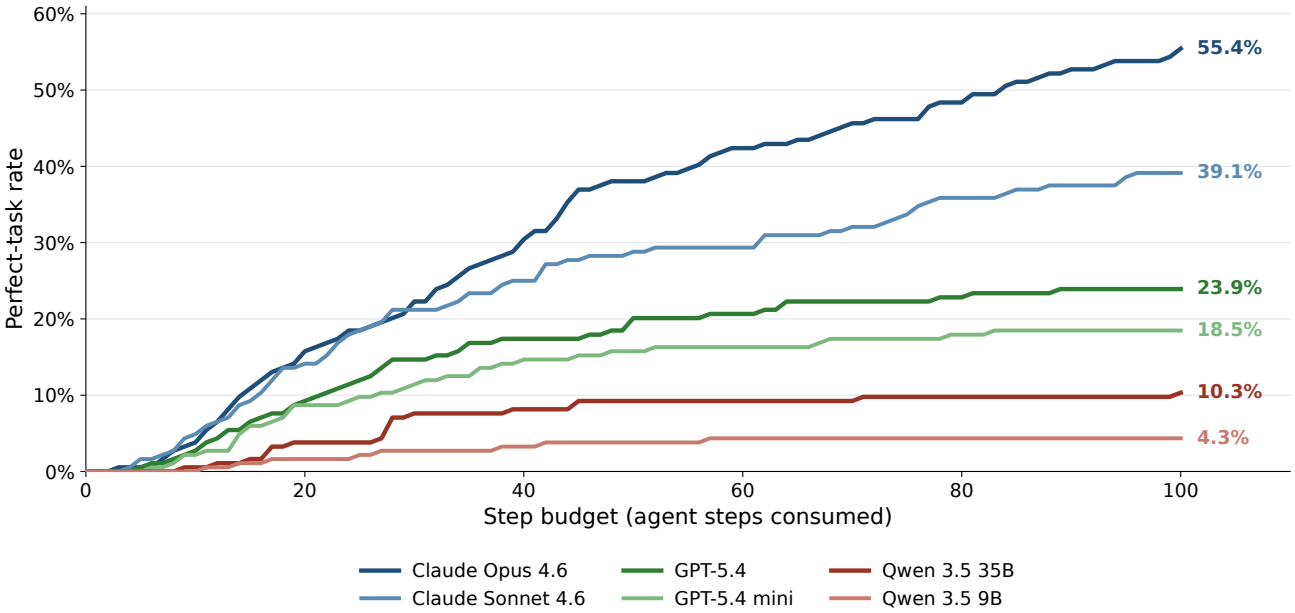

Figure 6. **Step-budget scaling law.** For each model, the curve at step budget X is the fraction of the 184 tasks the model graded perfect with $\leq X$ agent steps consumed. Curve shape, not just height, separates the families: Claude Opus is still climbing at the 100-step cap; GPT-5.4 saturates near 60 steps; Qwen flatlines below 25 steps.

Table 5. Rubric score (%) by task type. Rows ordered by descending cross-model average. *Mean* is the unweighted average across the six models. Best per row in **bold**.

Task type	n	Opus	Sonnet	GPT-5.4	GPT-5.4 mini	Qwen 35B	Qwen 9B	Mean
Personal lookup	13	**83.8**	66.0	71.9	69.0	58.1	32.3	63.5
Bounded action	64	**85.3**	70.4	73.2	60.6	48.2	31.0	61.5
Pattern inference	11	**94.7**	77.3	41.3	36.3	28.0	18.5	49.4
Cross-source reconciliation	25	**76.1**	61.8	41.1	41.4	27.1	13.1	43.4
Multi-step orchestration	48	**76.6**	59.5	31.5	39.2	37.5	11.4	42.6
Aggregation & reporting	23	**81.9**	61.9	29.4	28.6	23.6	10.3	39.3

Table 6. Rubric score (%) versus the number of distinct applications a task touches. 68% of MyPCBench tasks are multi-app.

Apps touched	n	Opus	Sonnet	GPT-5.4	GPT-5.4 mini	Qwen 35B	Qwen 9B
1	59	87.4	69.9	59.7	51.6	44.7	31.8
2–3	72	82.4	66.8	57.3	54.2	37.7	16.0
4–6	31	79.8	61.8	31.7	35.9	34.1	15.9
7+	22	67.9	54.1	29.9	30.9	34.5	9.1
Δ (1 $\rightarrow$ 7+)		−19.5	−15.8	−29.8	−20.7	−10.2	−22.7

E. Family-Signature Plots

Figure 7 breaks failed-rubric hits out by failure mode and groups them by family (left), and shows a per-model error budget (right) with the share of zero-score tasks that terminated under 20 steps (premature termination) versus those that hit the 99-step budget without success.

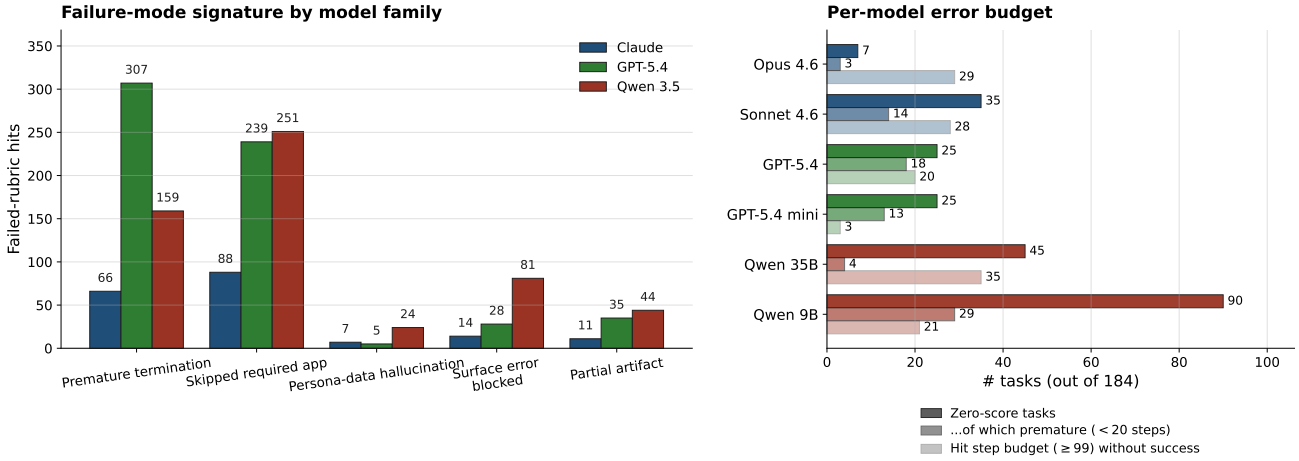

Figure 7. **Left:** failed-rubric hits by failure mode, grouped by family (Claude=Opus+Sonnet; GPT-5.4=GPT-5.4+mini; Qwen=35B-A3B+9B). Each family's bar set traces a distinct trend. **Right:** per-model error budget. Top dark bar: zero-score tasks; middle: subset that terminated under 20 steps; lightest: trajectories that hit the 99-step budget without success.

F. Detailed Failure Modes

This section expands the family-level failure trends summarised in §5.4, the compounding effect that makes perfect rate fall faster than rubric score, and the two failure modes that are catalogued only briefly in the main paper.

Why the perfect rate falls faster than the rubric score. A failed rubric is rarely an isolated event. Skipped-app failures co-occur with premature-`DONE` on the same trajectory (the agent quits because it considers itself done after the apps it did open), and surface errors trigger partial-artifact failures (an opened spreadsheet that is never saved). Because perfect-task rate requires *every* rubric to pass, even one such co-occurring failure zeroes the task. This compounding is why the perfect-rate gap (Opus 55.4%, GPT-5.4 23.9%, Qwen 9B 4.3%) is wider than the rubric-score gap (81.8 / 50.5 / 20.2).

Per-family breakdown. Three trends fall out of the per-model counts in Table 7 and Figure 7.

Table 7. Failure-mode counts on rubric items the judge marked failed, aggregated across all six models. Per-family breakouts are in Figure 7.

Failure mode	Hits	Description
Premature termination	532	Agent emits `DONE` before the trajectory satisfies the remaining rubric items.
Skipped required app	578	Multi-app task closed after only some apps are visited; unvisited app's rubrics fail.
Surface error as terminal	123	Agent hits a captcha, console error, slow page, or modal and quits instead of recovering.
Partial artifact	90	Artifact started but not saved/exported (e.g., spreadsheet opened but never saved).
Hallucinated persona data	36	Agent fabricates a value instead of reading the seeded source.

- **The GPT family stops too early.** 307 of the 532 premature-`DONE` hits are in the GPT family (GPT-5.4 mini 170, GPT-5.4 137), at least 3.6× either Claude model on its own. This lines up with the early-stopping step regime in §5.4 and with the family's flat step-budget curve in §5.3.
- **The Qwen family makes things up.** 24 of the 36 persona-data-hallucination hits are in Qwen (9B 14, 35B 10), versus 7 in Claude and 5 in GPT. Qwen 9B alone accounts for 39% of all hallucination hits and an additional 58 surface-error abandonments (47% of all surface-error hits).
- **The Claude family takes UI shortcuts.** The Anthropic Computer Use API exposes a native `bash` tool. Claude trajectories use it to query app REST endpoints (e.g., 55 `curl` calls to the SprintBoard backend on `hard_app-f026`) instead of driving the visible UI. Rubrics that require a user-visible side-effect (move a card, save a file from the menu) then fail even though the agent retrieved the correct value. We do not observe this in the GPT or Qwen runs because the harness exposes no equivalent tool to those agents (§4.2).

Console-script shortcuts (Claude-specific). A failure pattern unique to the two Claude models is the console-script shortcut: the agent opens a JavaScript console (or, given Anthropic's native `bash` tool, hits the app's REST endpoint directly with `curl`) and reads the persona's data without driving the visible UI. When the rubric only requires that the agent *know* the value, this satisfies it. When the rubric requires a user-visible side-effect (drag a card, open the project, save a file from the menu), the script reads the data and `DONE`s the task without producing the artifact. `hard_app-f026` is canonical: 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." Opus issues 55 `curl http://localhost:3013/api/projects ...` calls on that trajectory.

Skipped-app concrete examples. On 228 failed rubrics, the agent finishes a multi-app task without ever opening one of the named apps. `long_horizon-f066`: the agent archives a HooliChat conversation and never opens OddsMarket. `long_horizon-f074`: visits nine apps, never opens TableFind to make the required reservation (Figure 10, row 04). `aggregation-f010`: searches HooliChat extensively but never reads the Dundies categories file in `~/Documents`. These are exactly the failures predicted by the cross-app scaling collapse in §5.3.

G. Persona Specification and Event Chains

The canonical Michael Scott persona is stored as a single JSON document (`personas/michael_scott.json` in the released code) with fifteen top-level sections that 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":       [ ... ]
}
```

The four sections that drive cross-app consistency are `cross_app_events` (cross-app side-effects: a trip seeds rows in six apps), `planted_contradictions` (deliberate red herrings that test whether agents read all sources), `planted_dependencies` (records the rubric needs the agent to chain through), and `app_overrides` (per-app tuning, e.g. a Cheskepdia booking that a later HangryDash record references).

Annotated event chain. A single `cross_app_events` entry produces correlated rows across every application that would plausibly record the event. Below is the canonical Cooper's Seafood House dinner-plan event taken verbatim from the seed.

```
{
  "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
  }
}
```

The seeders fan out: `seed_webapps.py` writes the TableFind reservation and the Gringotts charge; `seed_calendar.py` writes the HooliCalendar block; `seed_browser.py` writes the Firefox history rows; the HooliChat seeder writes the message thread. Because every seeder reads from the same event record, the entire chain stays internally consistent: TableFind, Gringotts, HooliCalendar, Firefox history, and HooliChat all reference the same date, time, restaurant identifier, and contact.

H. Task-Review Interface

We built a single-page web reviewer (Figure 8) for the quality-assurance pass described in Section 4.1. The interface lists every task in the suite grouped by primary application, surfaces the instruction, difficulty, and the apps the task touches inline, and exposes per-task review state (unsorted / unsolved / fix-needed / complete / app-fix-needed) with one-key keyboard shortcuts. Selecting a task expands a side-pane with the rubric items and a deep-link to the corresponding live application URL inside the VM, so a reviewer can run the task end-to-end without leaving the page.

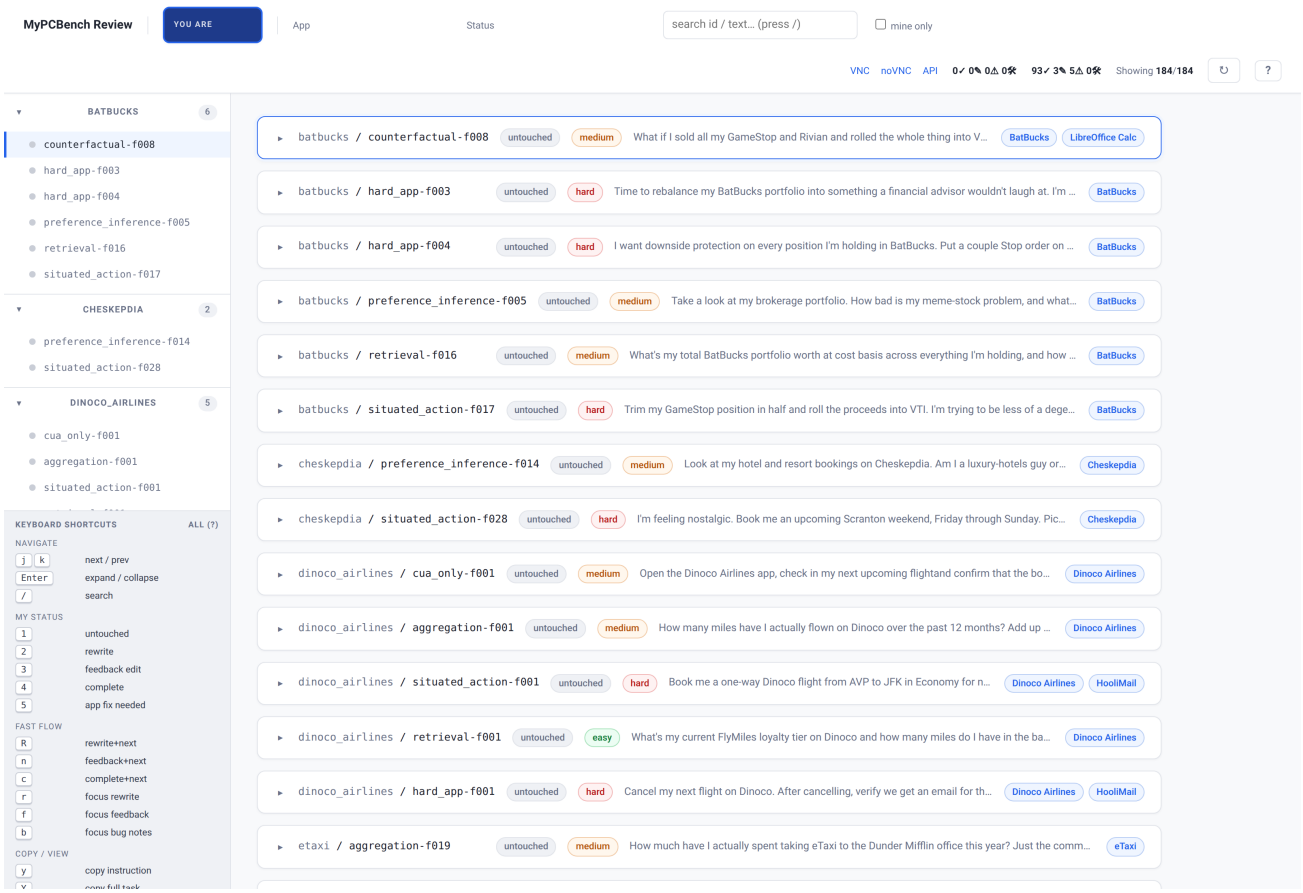

Figure 8. The MYPCBENCH task-review interface used during quality assurance. The left pane is grouped by primary application; each row shows the task identifier, review state, difficulty, the verbatim instruction preview, and pills for the apps the task touches. Reviewer-identifying fields have been removed for double-blind review.

I. Data Generation Pipeline

The pipeline turns the persona JSON in Appendix G into a fully populated Linux desktop image. It is a single Python entry point (`generator/generate.py`) that calls one seeder per data surface in dependency order. Every seeder consumes the same persona document, so adding a new persona is a one-file change.

Seeder modules.

- `persona_registry.py`: resolves the active persona (default Michael Scott) and exposes identity helpers used by every downstream seeder.
- `persona_data.py` and `enrich_personas.py`: expand the JSON into derived fields (e.g. paystub line items inferred from `salary` and `tax_info`).
- `seed_webapps.py`: writes the SQLite databases for the 17 Next.js apps. Honours `cross_app_events` so a single trip leaves correlated rows across Cheskepdia, Dinoco, eTaxi, Gringotts, HooliMail, and HooliCalendar.
- `seed_email.py`: builds the on-disk Maildir (`/home/user/Maildir`) and the HooliMail SQLite mirror.
- `seed_calendar.py`: writes HooliCalendar events plus the `.ics` files that the LibreOffice toolchain reads.
- `seed_browser.py` and `inject_firefox_cookies.py`: populate the Firefox profile, history, bookmarks, form-fields, and the per-app session cookies that keep every web app pre-logged-in.
- `seed_filesystem.py`: writes the persona's documents (meeting notes, expense reports, trip itineraries, boarding-pass PDFs, resume drafts) into `/home/user/Documents` and `/home/user/Downloads`.
- `document_personas.py`: emits a per-persona Markdown summary used by reviewers and by the LLM-as-a-judge as background context.

Determinism. Every seeder is seeded from a deterministic RNG keyed on the persona name and the seeder identifier, so identical inputs produce byte-identical outputs across runs and machines. The default reference time is the persona's `tax_info.tax_year` end-of-day, overridable through the `MYPCBENCH_REFERENCE_TIME` environment variable for time-travel debugging.

From persona to image. After all seeders run, a single `docker build` bakes the populated home directory, the Firefox profile, and the 17 Next.js apps into the released environment image. The first boot of the QEMU guest captures a base snapshot; every subsequent task starts from this snapshot, so the agent always sees the same initial state. Adding a new persona requires (i) writing a JSON document conforming to the schema in Appendix G and (ii) rerunning `generator/generate.py` with the new persona slug; no seeder code has to change.

J. Grading and Rubric Prompts

This appendix reproduces, verbatim from the released code, the prompts used to grade every task. Each task's rubric is a list of natural-language criteria authored alongside the task and audited during the QA pass (§4.3). The judge runs once per rubric item over the agent's full trajectory; we use `gemini-3.1-flash-lite-preview` as the judge model.

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; up to the most recent 200 screenshots from the trajectory are attached in chronological order in the same message.

```
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, the two reported metrics are

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

K. Agent Harness

This appendix documents the release of the agent harness: its interface, action space (Table 8), step budgets, snapshot reset, and the actual system prompts used for each evaluated model family.

Harness interface. The harness spins up the Docker image (Ubuntu 24.04, XFCE over VNC on port 5901, supervisord managing the 17 Next.js services), waits for desktop-ready (typically ∼90 s), and enters the step loop: capture screenshot over VNC; construct the agent message (system prompt + task instruction + screenshot + accumulated history); dispatch to the model's native API; parse the returned action; execute on the guest via VNC or shell; repeat until the agent emits `done`/`DONE` or the step budget is reached. A fresh base snapshot is restored between tasks so each task sees an identical initial state.

Action space. Table 8 enumerates every action exposed to an evaluated agent. The top block is the unmodified OSWorld `pyautogui` surface and is mapped onto every provider's CUA vocabulary. The bottom block lists the two tools that ship natively with Anthropic's Computer Use API; the harness exposes them when running Claude but not for the OpenAI or Qwen CUA agents.

Table 8. 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	Claude (native)
`str_replace_based_edit_tool`	view / create / replace / insert	Claude (native)

Shared persona-and-environment context (all agents). Every evaluated agent, regardless of model family, receives the following persona-and-environment block appended to its system prompt. This is the shared frame that keeps the agents grounded on the correct persona, applications, and conventions; it is reproduced verbatim from the released code (`agents/prompts.py`, `MYPCBENCH_CONTEXT`).

```
## 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 single shared block on *when not* to terminate, 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. Prepended to the Claude Computer Use request before the shared persona context. Used by Claude Opus 4.6 and Claude Sonnet 4.6.

```
You are an agent on a Linux workstation. Your tools are `computer`, `bash`, and `str_replace_based_edit_tool`.

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

OpenAI CUA operator prompt. Injected as the text portion of the first user message for OpenAI computer-use agents (GPT-5.4, GPT-5.4 mini); the shared persona context is appended.

```
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).
```

Vision-only screenshot-to-pyautogui prompt. Used for the open-weight agents (Qwen 3.5 35B-A3B and Qwen 3.5 9B); ported from OSWorld (Xie et al., 2024) with MyPCBench-specific additions in the shared persona context. The agent emits `pyautogui` code, which the harness executes on the guest.

```
You are an agent which follows my instruction and perform desktop computer tasks as instructed.
You have good knowledge of computers and good internet connection and assume your code will run on a computer for
↪ controlling the mouse and keyboard.
For each step, you will get an observation of an image, which is the screenshot of the computer screen and you
↪ will predict the action of the computer based on the image.
```

```
You are required to use ‘pyautogui‘ to perform the action grounded to the observation, but DONOT use the ‘
↪ pyautogui.locateCenterOnScreen‘ function to locate the element you want to operate with since we have no image
↪ of the element you want to operate with. DONOT USE ‘pyautogui.screenshot()‘ to make screenshot.
Return one line or multiple lines of python code to perform the action each time, be time efficient. When
↪ predicting multiple lines of code, make some small sleep like ‘time.sleep(0.5);‘ interval so that the machine
↪ could take; Each time you need to predict a complete code, no variables or function can be shared from history
You need to to specify the coordinates of by yourself based on your observation of current observation, but you
↪ should be careful to ensure that the coordinates are correct.
You ONLY need to return the code inside a code block, like this:
```python
# your code here
```
Specially, it is also allowed to return the following special code:
When you think you have to wait for some time, return ```WAIT```;
When you think the task can not be done, return ```FAIL```, don’t easily say ```FAIL```, try your best to do the
↪ task;
When you think the task is done, return ```DONE```.

My computer’s password is ’{CLIENT_PASSWORD}’, feel free to use it when you need sudo rights.
First give the current screenshot and previous things we did a short reflection, then RETURN ME THE CODE OR
↪ SPECIAL CODE I ASKED FOR. NEVER EVER RETURN ME ANYTHING ELSE.
```

L. Example Trajectories

Figures 9 and 10 pair one passing and one failing trajectory from each of the three evaluated model families (Claude, GPT, Qwen) at higher fidelity than the main-paper Figure 4. Each vignette row reproduces the verbatim task instruction issued to the agent, three screenshots from the actual trajectory (one early, one mid, one near the end), and a short observed-behaviour note explaining how the trajectory arrives at the judge verdict shown in the pill. The pairing surfaces intra-family contrast: the same model family can both complete a task cleanly and exhibit one of the failure modes catalogued in Table 7. The six selected runs cover `aggregation-f001`, `hard_app-f011`, `retrieval-f009`, `long_horizon-f074`, and `situated_action-f036`, drawn from three of the failure modes and the family-level trends discussed in §5.4. The Qwen pair places the larger MoE model (Qwen 3.5 35B-A3B) and the smaller dense model (Qwen 3.5 9B) on the same aggregation task to surface scale-driven differences within the family.

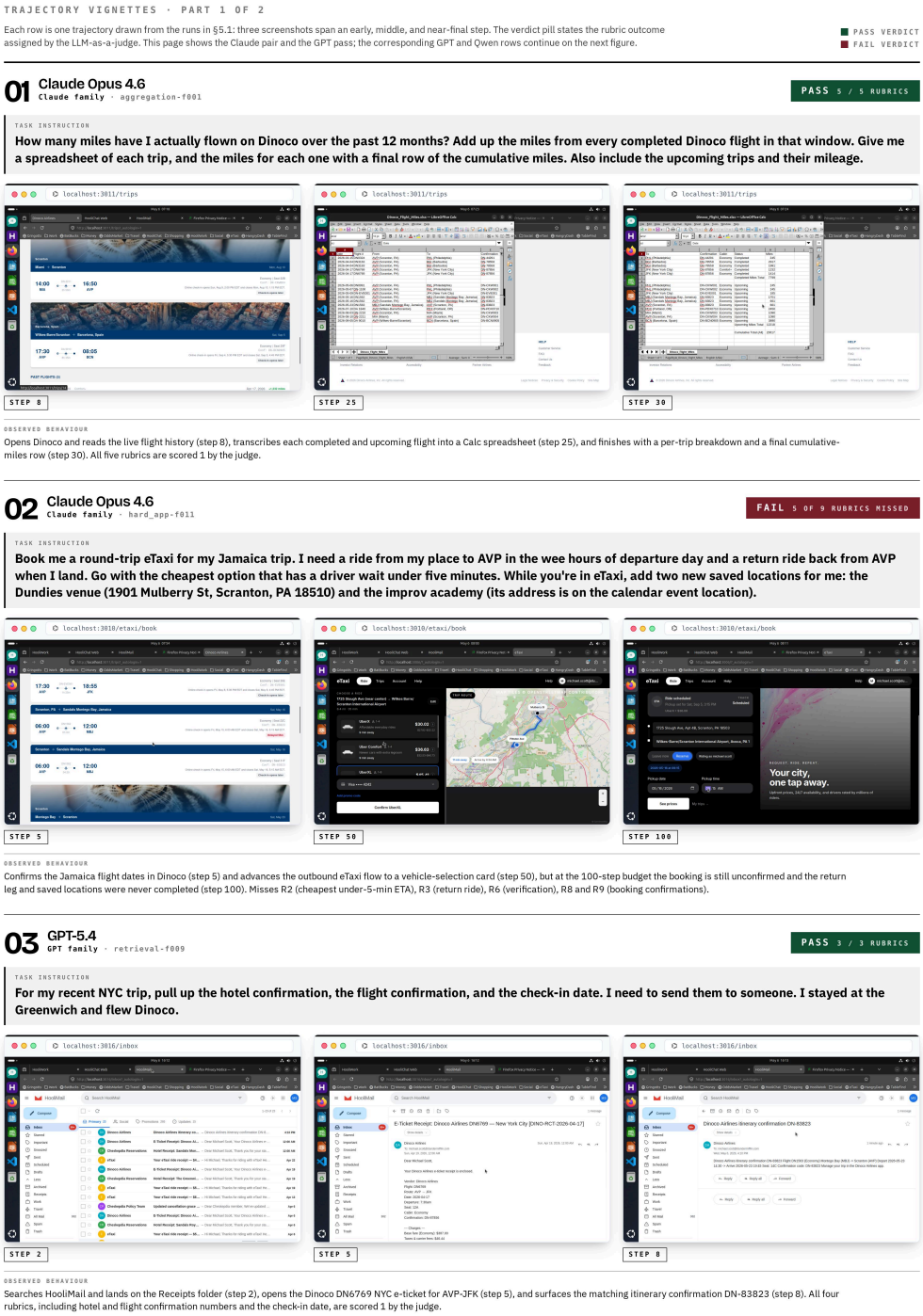

Figure 9. Trajectory vignettes, part 1 of 2. Rows: Claude Opus 4.6 PASS on `aggregation-f001`, Claude Opus 4.6 FAIL on `hard_app-f011`, GPT-5.4 PASS on `retrieval-f009`. Each row shows the verbatim task instruction, three screenshots (early, mid, near-final step), and an observed-behaviour note linked to the rubric outcome assigned by the LLM-as-a-judge.

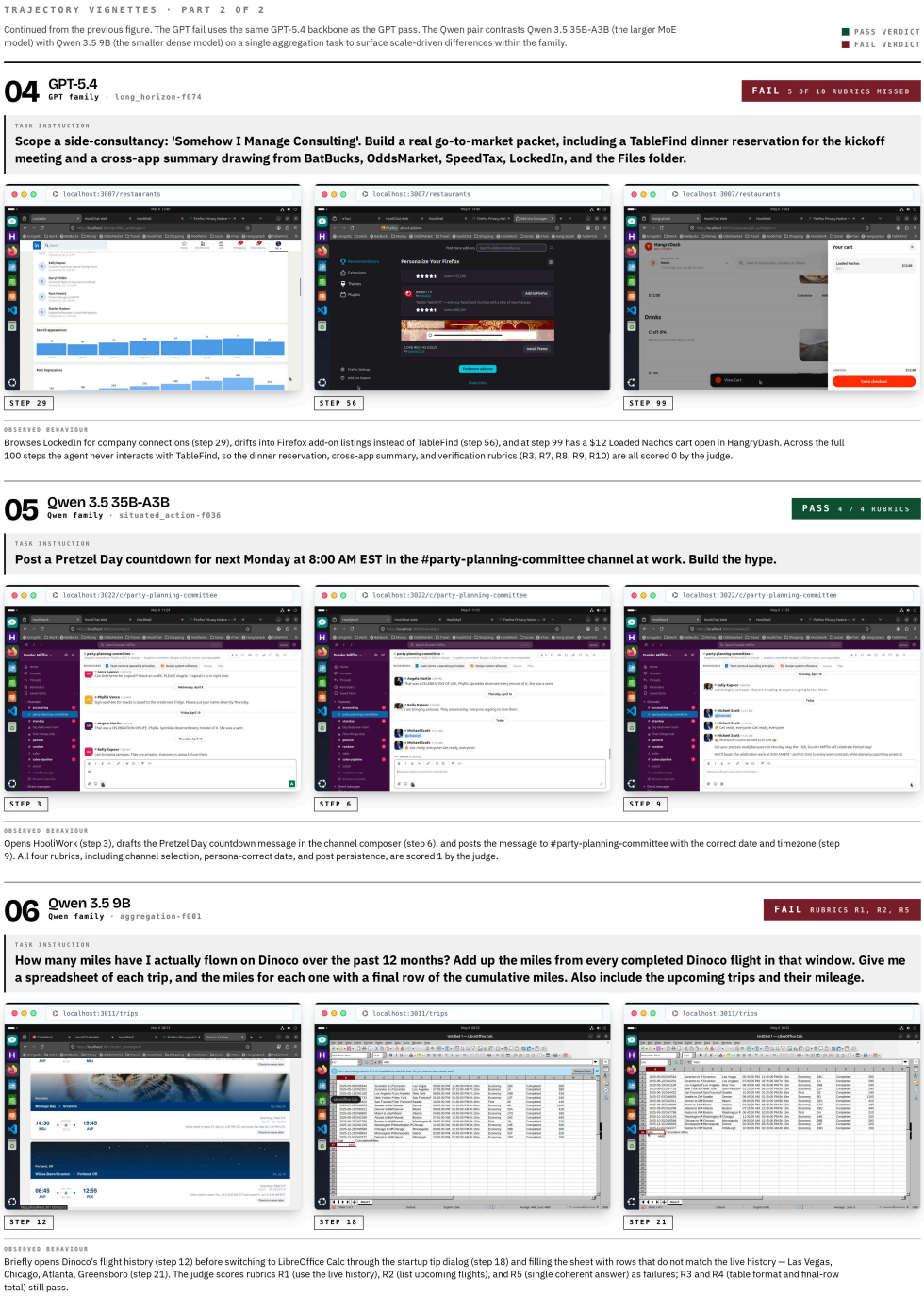

Figure 10. Trajectory vignettes, part 2 of 2 (continued from Figure 9). Rows: GPT-5.4 FAIL on `long_horizon-f074`, Qwen 3.5 35B-A3B PASS on `situated_action-f036`, Qwen 3.5 9B FAIL on `aggregation-f001`. Same layout convention as Figure 9.

M. Limitations

MYPCBENCH commits to one canonical persona (Michael Scott) and one Linux/GNOME/Firefox software stack; quantitative patterns specific to other demographics, locales, or stacks may not transfer. Grading uses a single Gemini judge, and the judge's errors are correlated across rubrics on the same trajectory; absolute failure-mode counts (Table 7) should be read as a structural breakdown across three models, not as a precise prevalence estimate. The seeded persona is intentionally low-sensitivity (a public fictional character), so behaviours that emerge only on agents reasoning over genuinely sensitive personal data are not exercised by this benchmark; we view that as an explicit out-of-scope choice.

N. Release Artifacts

The following artifacts are located at the URL given in §1: (i) the environment image (Docker + QEMU snapshot), (ii) the set of task evaluations 184, (iii) the rubrics per-task, (iv) the agent harness that connects standard CUA agents to the environment, and (v) the configuration of the rubric-grading judge. We do *not* release the agent trajectories or per-rubric judge outputs produced by the runs in this paper; any future work using the same harness and judge can reproduce them on the released image.