The Hard Part Comes After Search: Benchmarking Web Agents on Synthesizing, Organizing, and Displaying Knowledge

Alexander Gill **Md Farhan Ishmam** **Xuyen Nguyen** **Neha Bhat**
Parker Henry DeYoung **Fateme Hashemi Chaleshtori** **Nathan Stringham**
Kenneth Marino **Ana Marasović**
University of Utah
Correspondence: {alex.gill, farhan.ishmam, ana.marasovic}@utah.edu

Abstract

Existing computer-use agent benchmarks do not fully evaluate agents acting as assistants. A useful assistant retrieves information across complex, multi-step workflows, synthesizes it into artifacts (documents, presentations, spreadsheets), and navigates program interfaces to produce a coherent final product. Such workflows demand reasoning and synthesis, decomposition of complex tasks, as well as visual and spatial understanding. To study agents on workflows like these, we introduce KNOWS, a benchmark of open-ended, complex, browser-based tasks that jointly evaluate these capabilities, with each task culminating in a produced artifact. To write tasks, we develop a task design rubric and a protocol for ensuring that tasks meet the requirements. Each task is paired with an *evaluator*, a program that combines deterministic checks with LLM judgments to balance the richness, reliability, and automation tradeoff inherent to agent evaluation. We evaluate and analyze frontier computer-use agents and browser-based harnesses. They achieve moderate scores on partial-success metrics, but the best performer fully succeeds in fewer than 3% of our complex, long-horizon tasks. Failures on visual steps render the resulting artifacts unusable, even when agents complete more than 50% of other evaluation steps. Our results expose limitations of current agents acting as end-to-end assistants, and call for progress on tool use, visual understanding, and long-horizon reasoning.

1 Introduction

Amid a rapid deployment of agents as assistants in real-world workflows, a growing body of agent benchmarks has emerged (Yehudai et al., 2026; Kapoor et al., 2026). A convenient setup for evaluation is to produce tasks where success can be measured with a single, short, verifiable answer (Yoran et al., 2024; Wei et al., 2025). However, this is a poor match for the open-ended nature of real-world tasks. Deep research benchmarks (Du et al., 2026; Han et al., 2026) move closer to this setting by evaluating long-form textual reports, but in the real workspace, a plain textual report is rarely the final deliverable. Instead, findings are arranged into artifacts (slides, spreadsheets, and documents), whose effectiveness hinges on visual, spatial, and structural choices such as the placement and color of content, the selection of accompanying images, and the organization of tabular data. Properly executing these choices, on top of search and synthesis, makes the full task longer-horizon, and agents that excel at producing textual outputs may fail as assistants when these last-mile steps are also required.

To evaluate vision-and-language agents on these longer-horizon, open-ended tasks, we introduce KNOWS (**K**nowledge **N**avigation and **O**rganized **W**eb **S**ynthesis), the first benchmark spanning live web search, productivity tool use, and visual/spatial understanding (see §2 for a comparison with existing benchmarks). Figure 1 shows the overall setup. The agent receives a natural language instruction, uses a live web browser, a vision-and-language model (VLM), and a web-based office suite (specifically, Google Workspace) to produce a structured, coherent artifact (doc, sheet, slides) according to the user's specifications.

The more realistic, open-ended nature of the KNOWS tasks opens up a fundamental problem in agents: evaluation. In evaluating agents, we typically have a fundamental tradeoff between richness, reliability, and automation. LLM-as-a-judge evaluations (Zheng et al., 2023) scale, but are not fully reliable (Lù et al., 2025; Khalifa et al., 2026), at least not without sacrificing richness by constraining the space of possible outputs. One approach is to leave outputs unconstrained and tackle reliability directly with fine-grained evaluation protocols (Arora et al., 2025; Caldwell et al., 2025; Chen et al., 2026; Sharma et al., 2026; Han et al., 2026), but even then, reliably judging open-ended agent

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 45360–45400
October 24-29, 2026 ©2026 Association for Computational Linguistics

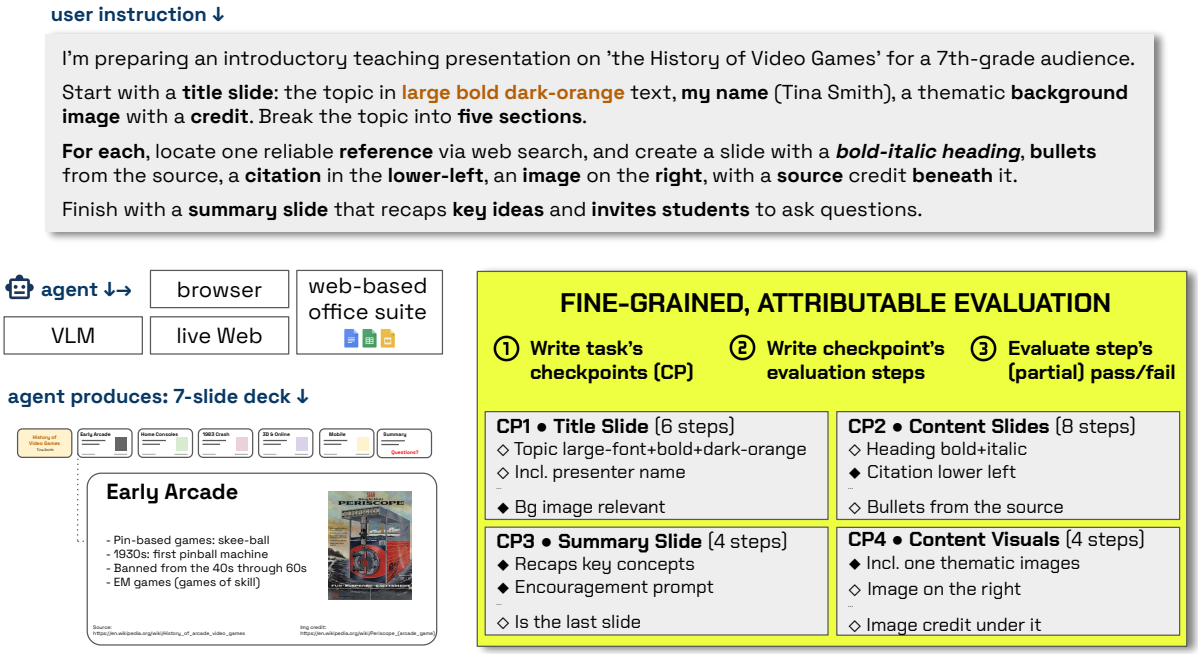


Figure 1: An overview of the KNOWS benchmark.

outputs remains difficult (Wang et al., 2026; Shen et al., 2026).

We, therefore, use a hybrid approach to balance this tradeoff with *evaluators*—programs that judge agent outputs using a combination of deterministic checks and LLMs. Consider the example in Fig. 1. The slide deck must satisfy criteria spanning layout (citation in the lower left), formatting (bold italic), source faithfulness (bullet points supported by the cited URL), and content relevance. We decompose each task into checkpoints, each checkpoint into evaluation steps, and check each step independently. Some steps, such as whether the image matches the theme, require a carefully prompted VLM or LLM judge. Others, like slide ordering or title color, can be checked deterministically with an API. The hybrid approach combines the richness of LLM judgment with the robustness we expect from deterministic checks.

Figure 2 overviews how we construct KNOWS. We first define a rubric specifying the requirements for a valid task in this benchmark: realism, information retrieval, artifact synthesis, visual/spatial understanding, and complexity. We then write an initial set of tasks. For quality control, each task is reviewed by another annotator against the rubric, and the task is revised in response. We write additional tasks inspired by the seed set, for a total of 110 tasks. Each task is paired with an evaluator, which is reviewed by at least two of the authors. One author then additionally validates the scripts on real artifacts by running an agentic browser on the task instruction, scoring the resulting artifact with the current scripts, inspecting the per-step verdicts for false positives and false negatives, and the scripts are revised accordingly.

We evaluate state-of-the-art (SOTA) computer-use agents across open-source and proprietary VLMs and harnesses, and find that even the strongest one achieves only <3% success rate on KNOWS tasks. Since complete success may underestimate agent capabilities, we report three partial-success metrics. Partial scores range from ≈35–70% for the best-performing baseline, but we show that the resulting artifacts are often unusable. Finally, our analysis shows that proprietary harnesses outperform open-source harnesses, and that agents fail more often on steps that require source fidelity and visual/spatial understanding. These failures call for advances in visual and spatial understanding of VLMs, long-horizon reasoning, open-source, and tool use. We publicly release KNOWS and the evaluation code to the community at `https://github.com/alexgill321/KNOWS-benchmark`.

2 Related Work

Web agent benchmarks such as WebShop (Yao et al., 2022), WebArena (Zhou et al., 2024), Mind2Web (Deng et al., 2023), and others (Koh

et al., 2024; Lù et al., 2024; Song et al., 2025) focus mostly on relatively simple and easily verifiable tasks. These generally test an agent's ability to perform basic tasks on websites, such as navigating to a page and finding a specific piece of information. They can typically be completed in a relatively small number of actions and do not involve creating complex artifacts or synthesizing large amounts of information, as in KNOWS.

As agents became more capable, agent benchmarks also involve more open-ended tasks. Many of these are categorized as **deep research benchmarks**. Datasets such as DeepResearchBench (Du et al., 2026), Mind2Web 2 (Gou et al., 2026), and DEER (Han et al., 2026) require agents to retrieve information from multiple websites and combine them, much as in KNOWS. However, unlike in KNOWS, these do not have the additional step of taking the results of this research and then sorting, analyzing, and synthesizing them into structured information. While they are required to reason from the information to produce a final answer (which can often be quite long), this does not necessitate manipulating web tools nor the kinds of spatial, visual, or structured reasoning required to create the outputs required in KNOWS.

Several datasets have tackled the task of **evaluating agents on productivity/office** suites. For instance, early OS benchmarks such as OSWorld (Xie et al., 2024) have required agents to do simple tasks in LibreOffice. OSWorld-MCP (Jia et al., 2026) includes similar spreadsheet operations and formatting tasks, text editing and formatting, slide manipulation, and media insertion. Prior work has built benchmarks specific to Microsoft Office (Wang et al., 2024), spreadsheets (Ma et al., 2024) or documents (Li et al., 2026), testing agents' ability to manipulate these programs, but none are complex or require notable amounts of information retrieval. OdysseyBench (Wang et al., 2025) is likely the most complex of these. However, this again mostly tests the ability to do multi-workflow office tasks rather than retrieving and synthesizing information. Our tasks, while requiring manipulation of these kinds of programs, also require information retrieval from the web, reasoning, and generating complex and coherent artifacts in these programs.

Most similar to ours is PresentBench (Chen et al., 2026), which, like some of our tasks, synthesizes slideshows. Their benchmark focuses on the specific problem of creating large, impressive presentations. However, their information retrieval is more limited (it retrieves from a curated set of academically rigorous sources), while we focus on broader information that also requires navigation of the live web. Our benchmark also focuses on more structural constraints (such as where images should be placed), whereas theirs focuses more on evaluating open-ended presentation generation.

One of our main contributions to the area of computer use agents is not only the dataset and tasks themselves, but our unique way of **agentic evaluation**, balancing the difficulty of having realistic and open-ended tasks with the practicalities of evaluation. Benchmarks with simple, easy to verify outputs (e.g., check if a particular item is in your cart; Yao et al., 2022) use simple automatic verification. For more ambiguous or difficult to programmatically define benchmarks, many papers use LLM-as-a-judge (Zheng et al., 2023) or VLM-as-a-judge (Chen et al., 2024), most prominently for web agents in Online-Mind2Web (Xue et al., 2025). As tasks have gotten more difficult, evaluations have started to add intermediate checkpoints (Pan et al., 2024) or use detailed rubrics (Chen et al., 2026; Sharma et al., 2026; Han et al., 2026) to evaluate tasks. KNOWS takes a hybrid approach, combining programmatic verifiers and LLM-as-a-judge, and breaks tasks into multiple discrete checkpoints and separate evaluation steps. This approach allows both more robust verification of complex tasks that can be divided into easily verifiable steps and the completion of more open-ended tasks.

3 KNOWS

The aim of KNOWS is to create tasks that test agents on realistic and complex artifact synthesis tasks. To this end, we follow a careful, time-intensive, iterative design process, overviewed in Figure 2. We start by laying out a rubric for task creation (§3.1). Next, we describe our task collection process, from designing a diverse set of high-quality seed tasks to using them to write a larger task set (§3.2). Then we describe how we build evaluators (§3.3) to score agents on these complex tasks. Finally, we present and analyze our dataset (§3.4).

3.1 Task Desiderata

We translate our core design requirements into a rubric that guides task creation and keeps new tasks

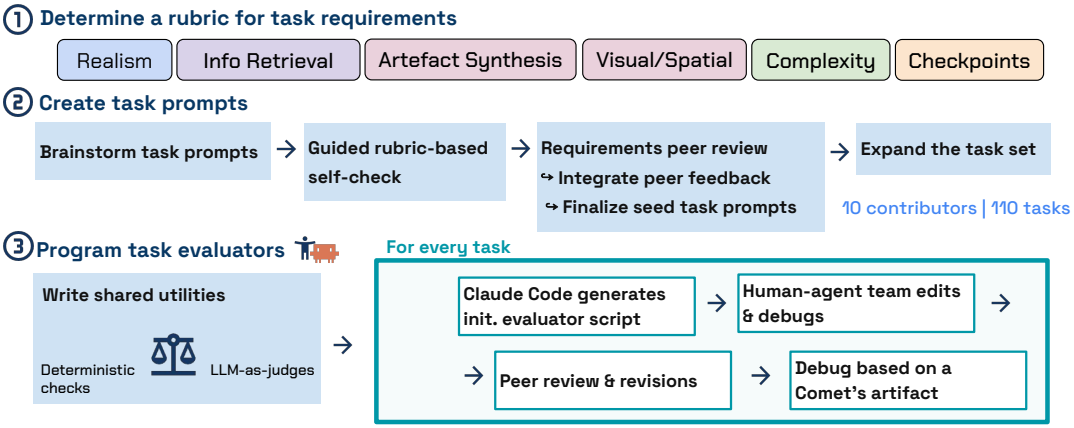


Figure 2: An overview of the construction of the KNOWS benchmark.

aligned with our motivation.

The first requirement, **realism**, follows recent web agent benchmarks. Tasks must be something a real user might want to do and would save time if automated. The next requirement, significant **information retrieval**, follows deep research benchmarks. Tasks must require browsing the live web with at least five (but often more!) pieces of information drawn from at least two websites. The next requirement, **artifact synthesis**, mandates that tasks produce a cohesive, well-organized artifact using a Google Workspace tool (slides, docs, or sheets). The agent solving our tasks must meaningfully synthesize retrieved information rather than just copy it directly into artifacts unedited. Our tasks also require some kind of **visual or spatial** component, such as identifying or editing specific elements, observing spatial relationships (left, right, up), or color-based modification (e.g., remove all red links). We again follow recent trends towards more **complexity** by requiring at least four steps of information retrieval, multiple steps to create the artifacts, and at least ten steps to incorporate and synthesize information. Finally, we require discrete **checkpoints** which break each task into measurable milestones in the creation of the artifact.

Google Workspace. We build our artifact synthesis tasks on Google Workspace (Drive, Docs, Sheets, and Slides). We choose this platform because it is used by a huge portion of internet users, supports realistic task scenarios in almost any domain, and provides a platform with API support for programmatic evaluations.

Live Web. Despite the known challenges with live web benchmarks (Akkil et al., 2026; Sun and Chen, 2025), we nevertheless focus our information retrieval on live web. It is more realistic as there are many possible sources of information, whereas a hosted intranet requires artificially limiting the set of available sources and therefore the breadth and realism of the tasks. Deep research benchmarks, which also look to evaluate agent performance in finding information on the web (Du et al., 2026; Gou et al., 2026; Han et al., 2026), all use live web as well in order to get the variety and richness of sources and distractors to test realistic information retrieval. Caching or freezing a subset of pages would remove the information retrieval challenge of KNOWS by artificially limiting the set of potential sources. Internet Archive, an alternative to caching, has issues with rate limiting of web-search agents and broken content for niche sites, so we avoid its use as well.[1]

Using the live web does create challenges for evaluation, particularly around websites changing with time. To account for this, we aim to curate tasks that involve information extraction that is independent of a single website and time-agnostic, meaning that the information required to complete a task can be found in multiple websites and will not change based on the date it is accessed. Since we nevertheless cannot guarantee that our evaluation scripts will remain functional indefinitely, we adopt a maintenance and deprecation policy for detecting and remedying outdated tasks, detailed in Appendix A.6.

3.2 Task Creation

Each task in KNOWS is composed of the following. First is the **user instruction** or **task prompt**,

[1] https://archive.org/

which is a detailed description in English of what the agent has to do. We aim for the task prompt to be clear and descriptive enough so that a detail-oriented human would be able to complete the task without further instructions. Next, each task contains a set of **checkpoints**. Checkpoints are discrete, measurable milestones for evaluating these tasks, each containing multiple separate evaluation steps. Tasks can be scored on overall completion or partial completion, based on the number of checkpoints and evaluation steps achieved. Finally, and most importantly, each task has an **evaluator**, a script that automatically checks the artifact at each checkpoint using a combination of hard-coded scripts, utility functions such as Google Workspace APIs, and LLMs and VLMs to grade the checkpoint. See Fig. 1 for a task example.

Using the rubric of task requirements, we write a Google Colab notebook (see Figures 7, 8, & 9 in Appendix A.1) to guide annotators through task creation. The notebook consists of the following steps. It first offers brainstorming suggestions for task prompts by displaying the Google Workspace template categories and subcategories, but it leaves an option to diverge from these. We allow annotators any tools at their disposal, including using LLMs, to assist in the brainstorming process. The notebook then urges annotators to check their task prompt against each of the required desiderata, and revise the task if any are not met. Finally, annotators write descriptions of the checkpoints and evaluation steps within these checkpoints.

While the workflow of self-checking each task requirement is part of the notebook, to further ensure that all tasks meet our requirements, another person reviews each task independently. They read, going back through the rubric, and verify that each is satisfied. In cases where they were not met, or there was an ambiguity or disagreement, the second annotator either edited or returned to the original annotator to either clarify or edit the task.

Once this initial set of tasks was collected, for additional coverage, we took these seed tasks and created additional tasks from them. For instance, we can take a task about preparing a presentation on the History of Video Games for a 7th-grade audience in Fig. 1 to a presentation on CRISPR-Cas9 Gene Editing for a 10th-grade audience, keeping many of the described steps the same, but drastically changing the websites and information in the task to create very different final outputs. In total, we collected 110 tasks.

Because of the complexity and length of the task creation pipeline (requiring in-person explanation and demonstration), these steps were done by authors whose work was then validated by other authors to ensure high-quality. We estimate that the creation of each task from start to finish takes around 9–18 hours of expert work to complete on average.

3.3 Evaluators

The final and most important step in task creation is writing evaluators. For each evaluation step in a task there must be a verifier for the successful completion of that step implemented by an evaluator. Each task has a dedicated evaluator script, which programmatically checks completion of each evaluation step, and returns the completion scores for each checkpoint (defined below). These evaluators are implemented by examining Google Workspace tool states, analyzing agent trajectories, and extracting data from the live web.

We first note that many evaluation steps share common requirements in implementation, so we create a suite of shared evaluation utilities available to any evaluator.[2] We organize these utilities around seven categories of evaluation step. **Structural checks** verify the existence, order, and hierarchy of elements via the Google Workspace API. **Information retrieval checks** compare extracted data values against gold-standard references using exact matching, fuzzy matching, or numerical comparison with tolerances. **Formatting checks** verify visual styling properties such as font weight, size, color, and alignment by querying the API directly. **Spatial checks** validate the physical layout of elements through their position, size, alignment, and whether they overlap or extend beyond the page. **Content checks** assess source fidelity and semantic accuracy of generated text. **Web visit checks** confirm that the agent actually visited relevant websites by inspecting its trace of visited URLs. **Visual checks** verify image content and provenance—whether an inserted image depicts the expected subject, originates from the correct source, or is visually distinct from other images.

Some checks that cannot be statically verified require careful use of LLM-as-a-judge (Zheng et al., 2023) or VLM comparison with formatting and rubrics such that they can be robustly evaluated. Additionally, for some deterministic checks, we use

[2] Similar to evaluation programs in Cho et al. (2023).

an LLM/VLM-as-judge as a backup check if the deterministic path fails. We describe the evaluation step categories in greater detail in Appendix A.4.

Next, we create an **evaluator** for each task. Each evaluator iterates through the task's checkpoints and verifies the success of every evaluation step within them, drawing on any of the seven evaluation check categories described above. As writing evaluators can be quite difficult and time-consuming, we use the following process. First a human creates a "gold standard" artifact for the task, acting as an ideal agent. We take the full task prompt and the codebase for the human-written evaluation utilities and feed these to Claude Code, along with an example of an evaluation script; the full prompt is given in the Appendix Figure 11. This generates an initial evaluator script that, unfortunately, always contains mistakes or misses steps. The evaluator's human creator edits and debugs the evaluator script against the gold artifact. One person reviews the evaluator and requests fixes to the evaluation script until it is fully correct on the gold artifact and approved. However, even then, bugs can persist. We therefore inspect how the evaluators score artifacts made by Comet (Yang et al., 2025), document the remaining issues, and finalize the evaluators with these final issues resolved. We provide more information on evaluator contributors in Appendix A.5.

Evaluator Validation. To measure the reliability of the finalized evaluators, an expert annotator, who did not implement the evaluators, judged whether artifacts sampled across baseline agents satisfy individual evaluation steps, for a total of 100 judgments. We find that evaluators agree with expert judgment with a Cohen's κ of 0.64 and 82% pairwise accuracy, in line with related open-ended agent benchmarks (Xue et al., 2025; Du et al., 2026; Han et al., 2026). We also validate expert annotator agreement on an independent set of 100 evaluator decisions consisting entirely of LLM/VLM-based steps. We find agreement with a Cohen's κ of 0.691 and 89% pairwise accuracy on these judgments.

3.4 Benchmark Overview

The KNOWS dataset contains 110 tasks across three artifact types: 25 documents, 40 slides, and 45 spreadsheets. Task examples are given in the Appendix Table 12. For a randomly chosen task, we present our decomposition of the task into checkpoints and of each checkpoint into evaluation steps in the Appendix Figure 10. We provide KNOWS's overall statistics in Table 1. One distinguishing feature is the length of these tasks. We see in Table 1 that the queries themselves for tasks are quite long (200 words on average) and the estimated time to complete the tasks (Appendix Table 11) is over 2 hours for humans. Tasks also span a broad range of domains. We use a taxonomy from Google Workspace use-case categories (§3.2), with our tasks covering 11 domains from the original taxonomy and an additional 9 use cases extending it. We list the full taxonomy and per-domain task counts in Appendix A.2.

We also report the distribution of evaluation step types in Table 2, which we calculated by prompting Claude Opus 4.6 to categorize each step's natural language description based on our definitions.[3]

Statistic	Value
Total Tasks	110
Prompt length	
Words (mean / median)	204.4 / 165.0
Sentences (mean / median)	13.7 / 12.0
Checkpoints per task	
Mean / Median	5.9 / 5.0
Min – Max	4 – 9
Evaluation steps per task	
Mean / Median	24.6 / 23.0
Min – Max	9 – 70

Table 1: Core statistics of the KNOWS dataset.

Step Type	# Steps	% Steps	% Inst.
Structural	835	30.8	90.9
Info. Retrieval	748	27.6	100.0
Formatting	327	12.1	78.2
Content	229	8.4	54.5
Spatial	199	7.3	59.1
Web Visit	189	7.0	68.2
Visual	184	6.8	63.6
Total	2,711	100.0	

Table 2: Distribution of types of checks KNOWS steps evaluate. % Steps is the share of all 2,711 evaluation steps; % Inst. is the share of 110 tasks containing at least one step of that type.

[3]The full prompt is given in the Appendix Figure 12.

Method	Docs				Sheets				Slides				Overall			
	SR	ASC	ACF	SF	SR	ASC	ACF	SF	SR	ASC	ACF	SF	SR	ASC	ACF	SF
Textual (Accessibility Tree)																
GPT 5.5	0	18.0	38.5	36.6	0	26.8	46.8	44.4	0	15.1	38.6	31.7	0	20.6	41.9	38.0
Claude Opus 4.7	0	7.7	19.9	21.9	0	3.0	12.3	10.0	0	9.3	20.3	12.5	0	6.4	17.0	13.6
DeepSeek V4 Pro	0	17.0	39.3	41.1	0	8.7	23.7	18.6	0	14.2	34.3	26.1	0	12.6	31.1	26.4
Multimodal (Accessibility Tree + Screenshot)																
GPT 5.5	0	21.7	44.9	46.2	0	25.0	50.1	48.2	0	18.1	51.7	45.3	0	21.8	49.5	46.7
Claude Opus 4.7	0	21.3	47.3	49.0	0	11.4	28.8	24.8	0	19.7	54.8	48.9	0	16.7	42.5	39.1
AI Browsers																
ChatGPT Atlas	0	22.3	43.6	40.6	0	25.9	51.5	48.6	0	16.3	39.7	31.6	0	21.6	45.4	40.6
Perplexity Comet	**12.0**	**41.0**	**68.7**	**66.3**	0	**35.8**	**73.0**	**68.3**	0	**31.3**	**67.5**	**59.7**	**2.7**	**35.4**	**70.0**	**64.7**

Table 3: Results on the KNOWS dataset. SR, ASC, ACF, and SF are metrics described in §4.1 across, with values reported as percentages (%). Baseline agents that are *not* AI Browsers use the BrowserGym-AgentLab harness.

4 Experimental Results

In this section, we introduce evaluation metrics (§4.1), baseline agents (§4.2), and finally, show how they perform on KNOWS (§4.3).

4.1 Evaluation Metrics

Let a task instance i consist of C_i checkpoints, where checkpoint j contains S_{ij} evaluation steps, of which k_{ij} are correct. We define four metrics:

Success Rate (SR) is the standard metric for *complete success*, where a task instance scores 1 only if every evaluation step across all checkpoints is correct. We define success rate for a task i as

$$\text{SR}_i = \mathbb{I}\left[\sum_j k_{ij} = \sum_j S_{ij}\right]. \quad (1)$$

Given the difficulty of our tasks, SR alone is too coarse. We introduce three *partial success* metrics, ordered from *strictest* to *loosest*.

Average Successful Checkpoints (ASC) assigns 1 to a checkpoint if *all* of its evaluation steps are correct, and averages across the checkpoints:

$$\text{ASC}_i = \frac{1}{C_i}\sum_{j=1}^{C_i} \mathbb{I}[k_{ij} = S_{ij}]. \quad (2)$$

Since each checkpoint represents a measurable milestone in artifact creation, ASC shows how many milestones an agent can fully complete.

Average Checkpoint Fraction (ACF) evaluates steps within each checkpoint, then averages the fraction across checkpoints, formulated as

$$\text{ACF}_i = \frac{1}{C_i}\sum_{j=1}^{C_i} \frac{k_{ij}}{S_{ij}}. \quad (3)$$

ACF is similar in spirit to ASC but less strict. Rather than requiring full checkpoint completion, it awards partial credit based on how much of each checkpoint was completed.

Step Fraction (SF) is simply the fraction of correct evaluation steps for each task:

$$\text{SF}_i = \sum_{j=1}^{C_i} k_{ij} / \sum_{j=1}^{C_i} S_{ij}. \quad (4)$$

SF is similar in strictness to ACF, but by aggregating evaluation steps rather than checkpoints, it naturally weights complex checkpoints (with more evaluation steps) more heavily.

4.2 Baselines

We benchmark SOTA proprietary models: Claude Opus 4.7 (Anthropic, 2026), GPT-5.5 (OpenAI, 2026), and an open-source model: DeepSeek V4 Pro (DeepSeek-AI, 2026). We benchmark models with the open-source BrowserGym-AgentLab harness (de Chezelles et al., 2025) across two modes: text-only using the accessibility tree (AXT) and multimodal using both screenshots (SS) and AXT.[4] We also evaluate closed-source harnesses (often called AI browsers): ChatGPT Atlas (OpenAI,

[4] At the time of writing, visual inputs were not available for the official DeepSeek V4 Pro API.

Figure 3: Output produced by the best-performing agent, Comet, on a `Slides` task. It received moderate partial scores, ACF (0.54) / SF (0.59), but the output remains practically unusable (indicated by a low ASC of 0.14).

2025) using a GPT agent,[5] and Perplexity's Comet (Yang et al., 2025), using a Claude Opus 4.7 agent. More details are in Appendix B. We do not report a controlled human baseline, as a single independent annotator per-task would cost roughly $5000 in labor alone (Appendix A.3). Every task was, however, completed by an author during construction, and resulting artifacts pass all evaluator criteria (§3.2), so agent failures we observe are not artifacts of unsolvable tasks.

4.3 Results and Analysis

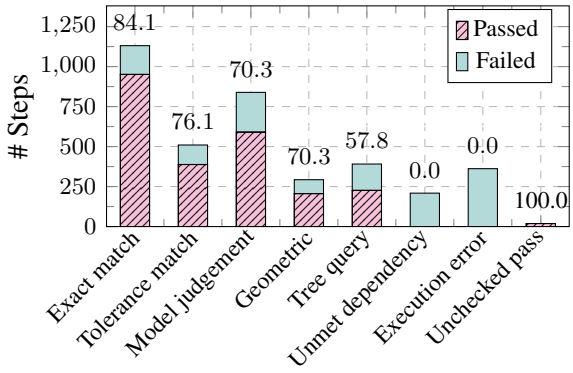
Table 3 shows the *full* and *partial* success across three splits: `Docs`, `Sheets`, and `Slides`, and the `Overall` performance.

Long-horizon tasks remain challenging. SR is near zero across most methods and splits, showing both the difficulty of our benchmark and the inability of SOTA agents on challenging, long-horizon tasks. The exception is `Docs`, where the best baseline, Comet, achieves a non-zero, albeit low, SR (12.0%). This result is unsurprising, as doc tasks resemble the conversational, text-only outputs best suited for the underlying LLMs.

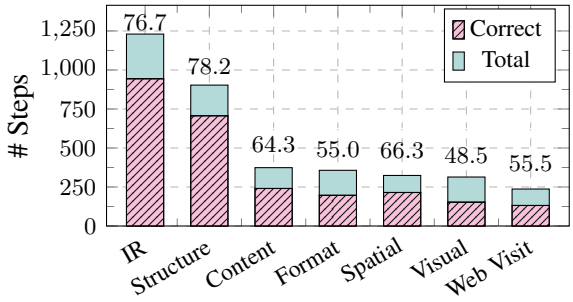
Partial success is also similarly low across most methods. Among the general agents, GPT-5.5 performs relatively well compared to other models. While Comet achieves the best performance, qualitative results show many of these outputs are often unusable, such as the one in Figure 3.

Visual modality is important. KNOWS contains a large fraction of tasks that require visual and spatial reasoning, which are often unsolvable from the AXT alone. Adding visual inputs as screenshots (SS) with the AXT can significantly improve performance, as much as (+10.3 ASC, +25.5 ACF, +25.5 SF) for Opus 4.7 over its AXT-only variant.

[5]The specific agent is auto-selected by the harness based on the task.



(a) W.r.t. the type of mechanism that checked step-completion correctness (see Appendix B.4).



(b) W.r.t. the types of checks steps evaluate; see §3.4.

Figure 4: Breakdown of Comet's correctness on 3,834 evaluation steps. Bar height is the number of evaluation steps, split into passed (hatched) and failed steps. The label above each bar gives the pass rate (%).

Harness matters as much as the model. Perhaps the starkest finding is the gap between two systems sharing the same backbone, e.g., Opus 4.7 with the Comet harness outperforms the Browsergym harness by (+2.7 SR, +18.7 ASC, +27.5 ACF, and +25.6 SF) overall. Proprietary harnesses show clear advantages in live web browsing, e.g., on complex webpages and with pop-up advertisements.

Step-wise performance analysis. Because of our hybrid evaluation framework, failures are directly traceable in KNOWS. Each failed evaluation step terminates under a specific condition in its evaluator script, which we automatically map to an error category. Fig. 4 breaks down the performance of the best agent, Comet: Fig. 4a by the type of mechanism that determines step-completion correctness, and Fig. 4b by the checks these steps evaluate (§3.4). These former types include EX-

ACT MATCH comaprisons (such as string equality), TOLERANCE MATCH (fuzzy string matching), MODEL JUDGMENT (VLM-determined content relevance), GEOMETRIC (bounding-box overlap), TREE QUERY (slide structure), UNMET DEPENDENCY (a missing element hinders a location check), EXECUTION ERROR (external API failure), UNCHECKED PASS (benefit-of- the-doubt branches). See Table 9 (Appendix) for detailed descriptions and examples.[6]

Comet is strongest on deterministic EXACT MATCH (84.1%) and TOLERANCE MATCH (76.1%) steps, middling on MODEL JUDGEMENT and GEOMETRIC steps (70.3% each), and weakest on TREE QUERY steps (57.8%), the latter driven almost entirely by one task in which Comet produced a deck without the requested slide structure. A further 651 steps (TRACE MATCH, UNMET DEPENDENCY, and EXECUTION ERROR) fail without the evaluator reaching a verdict on the artifact itself.

Qualitative error analysis. While the best-performing agents can achieve decent partial success scores, several of the resulting outputs often remain practically unusable. As seen in Figure 3, the agent achieves 0.54 and 0.59 in ACF and SF, respectively, while the resulting artifact is a visual mess. This failure pattern is consistent with the weaker GEOMETRIC and TREE QUERY scores in Fig. 4a, which check the layout and structure of the artifact. We observe similar errors in structuring documents (Figure 6), applying formulae in spreadsheets, structuring tables, and creating proper visualizations (Figure 5).

Cross-platform performance. In real use cases, assistant agents are not only restricted to creating artifacts in one platform environment. We perform a small cross-platform check to test whether our results are due to agent or harness optimization on Google Workspace. Specifically, we have Comet, our top performing model–harness combination, complete a subset of 5 of our tasks using Microsoft 365 web apps. We then grade these attempts manually using the same checkpoints and evaluation steps. We find that Comet performs better on 2 tasks and worse on another 2 tasks. Our results indicate that performance is not driven by optimization on the Google Workspace environment. Full results are in Appendix B.5.

[6] TRACE MATCH has been excluded from our analysis because Comet does not return browsing traces, and the corresponding evaluation steps were performed manually.

5 Conclusions

From the results on KNOWS, we can say that current web agents are quite strong at information retrieval, but still struggle with synthesizing, organizing, and displaying that information into artifacts that make retrieved information useful. KNOWS tests this gap directly, evaluating web agents on complex tasks that require all of these capabilities.

Our experiments reveal a clear capability gap. The best performing model, Comet, passes most deterministic value checks, but is weaker on checks of the fidelity, layout, and structure of the artifact (Figure 4a). No model successfully completes more than 3% of the total KNOWS task set (Table 3). Even when partial success metrics suggest moderate progress, the resulting artifacts are often practically unusable (Figure 3). This gap suggests that the bottleneck for web agents acting as useful assistants is not knowledge retrieval, but the reasoning capabilities, tool knowledge, and visual understanding required to produce coherent final products.

Our results also highlight that the agent harness matters as much, if not more, than the underlying model. Claude Opus 4.7, using two different harnesses, shows a large performance gap, pointing to infrastructure, action spaces, and tool interfaces as key areas driving and potentially hindering the performance of current web agents. We release KNOWS and all evaluation code to support future work on closing these gaps.

Limitations

Domain coverage. While KNOWS selectively collects 110 realistic tasks, it may not cover the full breadth of domains encountered in real-world web use. One design decision in KNOWS is its focus on depth over scale. While some benchmarks, such as WebArena (Zhou et al., 2024) and VisualWebArena (Koh et al., 2024), prioritized broader task coverage, KNOWS instead emphasized task complexity and length. Other recent benchmarks, such as Mind2Web 2 (Gou et al., 2026) and WorkArena (Drouin et al., 2024), share a similar philosophy. Each of our 110 tasks required approximately 9–18 person-hours to construct, including evaluator creation and debugging, reflecting the care taken to ensure each task is meaningful, well-validated, and representative of real-world complexity. This rigor also has practical implications at evaluation time, as a single baseline run takes

at least 10 hours to complete (depending on the benchmarking hardware and parallelization).

Live web evaluation. The challenges of live web benchmarks have been well known in the web-use domain (Akkil et al., 2026; Sun and Chen, 2025). Most significantly, this makes the benchmark depend on the current state of the web, which can change. Web changes can take the form of simple UI changes, the updating and outdating of information, and, occasionally, the loss of websites entirely. This dependence means our benchmark's validity is partially contingent on the current state of the web at evaluation time, a limitation present across all live web benchmarks.

We nevertheless believe that the live web aspect was crucial to properly evaluate information retrieval. In real information retrieval tasks, you must find a particular piece of information on an Internet-scale corpus of potential knowledge and extract the necessary information, where there may be many sources for that piece of information. Limiting to a small subset of locally hosted websites would not realistically test information retrieval. Virtually all DeepResearch-type benchmarks also make this choice (Du et al., 2026; Gou et al., 2026; Han et al., 2026). To mitigate the challenges associated with live web, we design our benchmark by carefully curating tasks for which the required information is available on multiple websites and that generally do not change over time, e.g., we avoid tasks where the agent needs to find the front-page story. Our evaluation scripts pull from the live web, and we acknowledge they may break. To address this, we commit to periodically testing all scripts after release and to promptly respond to reports of outdated tasks via the project's GitHub page (Appendix A.6).

Artifact scope. Our benchmark is limited to three artifact types: Google Sheets, Slides, and Docs, leaving the broad landscape of real-world web tools untested. The resulting coverage is nevertheless broad: three artifact types spanning tabular data, visual presentations, and long-form documents support a wide range of task structures, and this diversity is comparable to, or even greater than, that of prior datasets (see §2). Our artifact choice also gives models explicitly trained on Google Workspace a structural advantage over those trained on other platforms. To partially mitigate the training confound, we deliberately exclude Google's own models from our evaluation.

We additionally perform a small cross-platform check with our top performing model to determine the extent that training with Google Workspace affects our findings (§4.3). A full cross-platform check is not within the scope of this benchmark due to many of the deterministic evaluator checks relying on Google API utilities to function. That said, the design of our shared evaluation utilities could be ported to extend to other platforms in future work building on KNOWS. While our evaluators rely on Google Workspace APIs, web agents completing the tasks only interact with workspace artifacts through the rendered browser UI via generic actions (Table 5), so any training on Google Workspace APIs does not affect our results.

Ethical Considerations

One set of ethical considerations for KNOWS fall in the category of general risks for web agents. This benchmark looks to benchmark and push the envelope on the capabilities of agents to perform meaningful, long-term useful work. Improved agents raise a number of safety and privacy concerns, such as unsupervised agents leaking a user's personal information, spamming or performing harmful actions on websites, purchasing items without authorization, and more. This benchmark specifically gives access to a Google account, which might be connected to a user's professional or personal information.[7] There are real concerns about the speed of progress in the field and how much access users are giving agents before the work on security and trustworthiness has fully caught up. We evaluate tasks that do not ask the agent to send emails and messages, post on blogs, use credit cards, buy products, or generally interact with others on the web, so we may avoid many of these issues. However, continued development of these agents will inevitably bring these issues up in the future as agents become more capable.

A more specific concern relating to the kinds of tasks in this dataset is that it allows work to be automated, which might introduce issues. If, for instance, educators use agents to automate some parts of education (examples of which can be found in this dataset), this could lead to the particular biases and viewpoints inherent in these models to be reflected more broadly. It may also cause issues related to mistakes, hallucinations, lack of nuance

[7] We recommend that people evaluating on our dataset make a new account to avoid this issue.

or other pedagogical losses due to agents lacking the ability to tailor their content. Shifting tasks to agents blurs the accountability and transparency in these processes. This concern is one instance of a larger problem yet to be fully addressed: how LLM technology has shaped the education of millions around the world before society has had the time to investigate the effects of the technology or shape new norms.

Personally Identifiable Information is another ethical consideration for this dataset. As several of the tasks involve collecting information from the web, sometimes this includes PII of individually identifiable people. All of the PII in this dataset comes directly from the live web. We do not introduce or add any new PII that was not already directly available on the web before KNOWS was created. In addition, to ensure compliance with rules and norms around PII, we contacted the owners of all PII and received explicit permission to use the information for the purpose of creating the dataset. The information involved is not of a sensitive nature, but it is the kind of information that people might put on their personal website (as indeed they do in one of our tasks), such as emails and affiliations.

Acknowledgments

We thank Google for providing funding for the construction of this benchmark. We thank Anthropic and OpenAI for their donation of API credits for benchmarking models. We thank Ivan Andhika, Matthew Lee, Ryhor Pryslopski, Spike Cheng and Qui Ngo for their contributions to the codebase, Shrusti Ghela and Ana Alvarez Lopez for providing valuable annotations, and everyone who consented to our use of their information in this benchmark. We also thank everyone involved in the review process for providing valuable feedback on KNOWS.

References

Deepak Akkil, Mowafak Allaham, Amal Raj, Tamer Abuelsaad, and Ravi Kokku. 2026. Emergence webvoyager: Toward consistent and transparent evaluation of (web) agents in the wild. *Preprint*, arXiv:2603.29020.

Anthropic. 2026. System Card: Claude Opus 4.7.

Rahul K. Arora, Jason Wei, Rebecca Soskin Hicks, Preston Bowman, Joaquin Quiñonero-Candela, Foivos Tsimpourlas, Michael Sharman, Meghan Shah, Andrea Vallone, Alex Beutel, Johannes Heidecke, and Karan Singhal. 2025. Healthbench: Evaluating large language models towards improved human health. *Preprint*, arXiv:2505.08775.

Shane Caldwell, Max Harley, Michael Kouremetis, Vincent Abruzzo, and Will Pearce. 2025. Pentestjudge: Judging agent behavior against operational requirements. *Preprint*, arXiv:2508.02921.

Dongping Chen, Ruoxi Chen, Shilin Zhang, Yaochen Wang, Yinuo Liu, Huichi Zhou, Qihui Zhang, Yao Wan, Pan Zhou, and Lichao Sun. 2024. MLLM-as-a-judge: Assessing multimodal LLM-as-a-judge with vision-language benchmark. In *Forty-first International Conference on Machine Learning*.

Xin-Sheng Chen, Jiayu Zhu, Pei-lin Li, Hanzheng Wang, Shuojin Yang, and Meng-Hao Guo. 2026. Presentbench: A fine-grained rubric-based benchmark for slide generation. *arXiv preprint arXiv:2603.07244*.

Jaemin Cho, Abhay Zala, and Mohit Bansal. 2023. Visual programming for text-to-image generation and evaluation. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Thibault Le Sellier de Chezelles, Maxime Gasse, Alexandre Lacoste, Massimo Caccia, Alexandre Drouin, Léo Boisvert, Megh Thakkar, Tom Marty, Rim Assouel, Sahar Omidi Shayegan, Lawrence Keunho Jang, Xing Han Lù, Ori Yoran, Dehan Kong, Frank F. Xu, Siva Reddy, Graham Neubig, Quentin Cappart, Russ Salakhutdinov, and Nicolas Chapados. 2025. The browsergym ecosystem for web agent research. *Transactions on Machine Learning Research*. Expert Certification.

DeepSeek-AI. 2026. DeepSeek-V4: Towards highly efficient million-token context intelligence. Technical report, DeepSeek. Preview Release. Technical report available at https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro/blob/main/DeepSeek_V4.pdf.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. 2024. WorkArena: How capable are web agents at solving common knowledge work tasks? In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 11642–11662. PMLR.

Mingxuan Du, Benfeng Xu, Chiwei Zhu, Licheng Zhang, Xiaorui Wang, and Zhendong Mao. 2026. Deepresearch bench: A comprehensive benchmark

for deep research agents. In *The Fourteenth International Conference on Learning Representations*.

Fabrizio Gilardi, Meysam Alizadeh, and Maël Kubli. 2023. Chatgpt outperforms crowd workers for text-annotation tasks. *Proceedings of the National Academy of Sciences*, 120(30).

Boyu Gou, Zanming Huang, Yuting Ning, Yu Gu, Michael Lin, Weijian Qi, Andrei Kopanev, Botao Yu, Bernal Jimenez Gutierrez, Yiheng Shu, Chan Hee Song, Jiaman Wu, Shijie Chen, Hanane Nour Moussa, TIANSHU ZHANG, Jian Xie, Yifei Li, Tianci Xue, Zeyi Liao, and 7 others. 2026. Mind2web 2: Evaluating agentic search with agent-as-a-judge. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Janghoon Han, Heegyu Kim, Changho Lee, Dahm Lee, Min Hyung Park, Hosung Song, Stanley Jungkyu Choi, Moontae Lee, and Honglak Lee. 2026. Deer: A benchmark for evaluating deep research agents on expert report generation. *Preprint*, arXiv:2512.17776.

Hongrui Jia, Jitong Liao, Xi Zhang, Haiyang Xu, Tianbao Xie, Chaoya Jiang, Ming Yan, Si Liu, Wei Ye, and Fei Huang. 2026. OSWorld-MCP: Benchmarking MCP tool invocation in computer-use agents. In *The Fourteenth International Conference on Learning Representations*.

Sayash Kapoor, Benedikt Stroebl, Peter Kirgis, Nitya Nadgir, Zachary S Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, Franck Ndzomga, Dheeraj Oruganty, Sophie Luskin, Kangheng Liu, Botao Yu, Amit Arora, Dongyoon Hahm, Harsh Trivedi, Huan Sun, and 12 others. 2026. Holistic agent leaderboard: The missing infrastructure for AI agent evaluation. In *The Fourteenth International Conference on Learning Representations*.

Muhammad Khalifa, Lajanugen Logeswaran, Jaekyeom Kim, Sungryull Sohn, Yunxiang Zhang, Moontae Lee, Hao Peng, Lu Wang, and Honglak Lee. 2026. Gaming the judge: Unfaithful chain-of-thought can undermine agent evaluation. *Preprint*, arXiv:2601.14691.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905, Bangkok, Thailand. Association for Computational Linguistics.

Xiangyi Li, Kyoung Whan Choe, Yimin Liu, Xiaokun Chen, Chujun Tao, Bingran You, Wenbo Chen, Zonglin Di, Jiankai Sun, Shenghan Zheng, and 1 others. 2026. Clawsbench: Evaluating capability and safety of llm productivity agents in simulated workspaces. *arXiv preprint arXiv:2604.05172*.

David G Lowe. 1999. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee.

Xing Han Lù, Zdeněk Kasner, and Siva Reddy. 2024. Weblinx: real-world website navigation with multi-turn dialogue. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stanczak, Peter Shaw, Christopher Pal, and Siva Reddy. 2025. Agentrewardbench: Evaluating automatic evaluations of web agent trajectories. In *Second Conference on Language Modeling*.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. 2024. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

OpenAI. 2025. Introducing ChatGPT Atlas. `https://openai.com/index/introducing-chatgpt-atlas/`. Accessed: 2026-05-25.

OpenAI. 2026. GPT-5.5 System Card.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, and 1 others. 2024. Webcanvas: Benchmarking web agents in online environments. *arXiv preprint arXiv:2406.12373*.

Manasi Sharma, Chen Bo Calvin Zhang, Chaithanya Bandi, Clinton Wang, Ankit Aich, Huy Nghiem, Tahseen Rabbani, Ye Htet, Brian Jang, Sumana Basu, Aishwarya Balwani, Denis Peskoff, Marcos Ayestaran, Sean M. Hendryx, Brad Kenstler, and Bing Liu. 2026. Researchrubrics: A benchmark of prompts and rubrics for evaluating deep research agents. In *The Fourteenth International Conference on Learning Representations*.

William F. Shen, Xinchi Qiu, Chenxi Whitehouse, Lisa Alazraki, Shashwat Goel, Francesco Barbieri, Timon Willi, Akhil Mathur, and Ilias Leontiadis. 2026. Rethinking rubric generation for improving llm judge and reward modeling for open-ended tasks. *Preprint*, arXiv:2602.05125.

Yixiao Song, Katherine Thai, Chau Minh Pham, Yapei Chang, Mazin Nadaf, and Mohit Iyyer. 2025. BEARCUBS: A benchmark for computer-using web agents. In *Second Conference on Language Modeling*.

Zihao Sun and Ling Chen. 2025. Webarxiv: Evaluating multimodal agents on time-invariant arxiv tasks. *Preprint*, arXiv:2507.00938.

Leyao Wang, Yanan He, Peng Chen, Asaf Yehudai, Yixin Liu, Rex Ying, Michal Shmueli-Scheuer, and Arman Cohan. 2026. Time to reflect: Can we trust llm judges for evidence-based research agents? *Preprint*, arXiv:2605.19196.

Weixuan Wang, Dongge Han, Daniel Madrigal Diaz, Jin Xu, Victor Rühle, and Saravan Rajmohan. 2025. Odysseybench: Evaluating llm agents on long-horizon complex office application workflows. *arXiv preprint arXiv:2508.09124*.

Zilong Wang, Yuedong Cui, Li Zhong, Zimin Zhang, Da Yin, Bill Yuchen Lin, and Jingbo Shang. 2024. Officebench: Benchmarking language agents across multiple applications for office automation. *arXiv preprint arXiv:2407.19056*.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. 2025. Browsecomp: A simple yet challenging benchmark for browsing agents. *Preprint*, arXiv:2504.12516.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

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

Jeremy Yang, Noah Yonack, Kate Zyskowski, Denis Yarats, Johnny Ho, and Jerry Ma. 2025. The adoption and usage of ai agents: Early evidence from perplexity. *Preprint*, arXiv:2512.07828.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Asaf Yehudai, Lilach Eden, Alan Li, Guy Uziel, Yilun Zhao, Roy Bar-Haim, Arman Cohan, and Michal Shmueli-Scheuer. 2026. Survey on evaluation of llm-based agents. *Findings of the Association for Computational Linguistics: ACL 2026*.

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. 2024. AssistantBench: Can web agents solve realistic and time-consuming tasks? In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 8938–8968, Miami, Florida, USA. Association for Computational Linguistics.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*.

A KNOWS Collection

A.1 Task Creation Colab

Following our rubric of task requirements, we wrote a Google Colab notebook to guide annotators through task creation. For legibility, we report the notebook as text in Figures 7, 8, and 9.

A.2 Task Domain Taxonomy

We report the full taxonomy and per-domain task coverage counts in Table 4. This table shows the Google Workspace use-case category that tasks fall under, and whether or not that use case comes from the original taxonomy or an additional use case. In total, there are 20 unique domains, each consisting of at least 5 tasks.

A.3 Human Completion Times

To establish a human baseline for task difficulty, we measured the time required for human annotators to complete each task by analyzing edit activity logs from the Google Drive Activity API. For each task template, we examined the edit history of the gold document used for debugging when creating the original evaluators. Edit events are recorded at approximately three-minute intervals by the API. We segmented editing activity into sessions by treating gaps longer than one hour as session boundaries, then summed the duration of all sessions attributed to the primary annotator. We show the completion time to create a gold artifact for one instance of each seed task (Table 11).

As discussed in Section 4.2, we do not report a controlled human baseline in our experiments. From Table 11 we estimate each task would take an average of 3 hours per task. At $15/hr pay, 110 tasks would cost $4950, not accounting for any platform fees, and this is with only a single annotator for each task.

A.4 Evaluator Details

Each task instance includes a programmatic evaluator that decomposes the task requirements into fine-grained evaluation steps. Many evaluation steps share common implementation requirements, so we developed a suite of shared evaluation utilities accessible to all evaluators. Below we describe the implementation techniques used for each category of evaluation step defined in Section 3.3.

A.4.1 Structural Checks

Many tasks require organizing a Google Workspace document in a specific manner: creating particular columns in a spreadsheet, ordering sections in a document, or including required elements such as charts or bullet lists. Structural checks verify the existence, order, and hierarchy of these elements—not their styling or position, but whether the right pieces are present and correctly arranged. For example, the investment tracker task checks that columns exist for stock name, ticker symbol, and price, while the recipe task verifies that an "Ingredients" column appears first. To implement these checks, we leverage the Google Workspace API, which provides structural information with ordered sequential data for all elements (text, images, etc.) in a Workspace object. Using this, we locate elements in an object's structure and compare their position relative to other elements.

A.4.2 Information Retrieval Checks

Since every task requires live web information retrieval, a substantial portion of evaluation steps verify that the agent retrieved correct data. These steps compare values in the completed document against gold-standard reference data, checking that numbers, names, dates, and other facts match expected values within appropriate tolerances. For example, the investment tracker task verifies that stock prices match current market data, and the food composition task checks that nutritional values match USDA database entries. Implementation relies on deterministic comparisons: exact string matching, fuzzy text matching, and numerical comparison with configurable error tolerances.

A.4.3 Formatting Checks

These steps verify visual styling properties of document elements: font weight, size, and color, cell background colors, text alignment, merged cells, and other presentation attributes. For example, the event poster task checks that the event name is bold, at least 20pt, and black, while the food composition task verifies that column headers are bolded and group headers are centered and italicized. These checks query styling properties directly through the Google Workspace API and are fully deterministic.

A.4.4 Content Checks

While information retrieval checks verify exact data values, content checks assess whether information is faithfully tied to the correct source, whether summaries accurately represent source material, or whether claims are properly substantiated. For example, the event poster task checks that a speaker biography is "clearly tied to" the speaker's field of expertise and that cited sources correspond to specific claims. Because these judgments require semantic understanding, content checks are the category most reliant on LLM-as-a-Judge (Zheng et al., 2023). We format each requirement as a binary yes/no question and use an LLM to assess whether the content meets the criterion, following the robust performance of LLMs on binary classification (Gilardi et al., 2023).

A.4.5 Spatial Checks

While structural checks verify element ordering within the document structure, spatial checks validate an element's physical position in the rendered document as seen by a human, such as where something appears on the page, how large it is, or its spatial relationship to other elements. For example, "a logo must be in the top right of the slide" or "no text or images are overlapping each other." For Google Slides and Sheets, we extract element positions directly from the Google Workspace API, which provides each element's translation offset and size in English Metric Units (EMUs). We convert these to bounding boxes and use geometric operations such as overlap ratios, containment checks, and center-point comparisons to evaluate spatial relationships. For example, to verify that a text box appears at the top of a slide, we check whether its vertical offset falls within the top 20% of the slide height. To verify that elements do not overlap, we compute pairwise bounding box intersections. For Google Docs, where the API does not expose positional coordinates, we render PDF exports as images at 300 DPI and use scale-invariant template matching (Lowe, 1999) to locate specific elements within the rendered page.

A.4.6 Web Visit Checks

As discussed in Section 3.1, each task is designed to require live web information retrieval. However, LLMs may memorize correct information during pre-training, potentially enabling agents to complete tasks without actually browsing the web. To verify that agents perform genuine web retrieval, we require a list of all URLs visited during task execution to be passed to our evaluators. We do not require that specific URLs be visited, since our tasks leave the choice of website open to the agent. Instead, we pull HTML data from each visited URL and determine whether the information in the completed Workspace document could have been extracted from the set of visited pages. This verification uses either exact match or LLM-as-a-Judge, determined on a case-by-case basis depending on the complexity of the information. For some tasks, the agent is required to cite source links in the completed artifact. In those cases, we verify that the sources appear in their browsing history.

A.4.7 Visual Checks

These steps verify that a specific image is present in the document or that an image matches expected content. For example, the Wikipedia photos task checks that "the wiki image of Tom Hanks is present on the slide," and the paper sorting task verifies that generated figures match gold-standard chart images. Implementation varies by task: some steps use deterministic image comparison (pixel-level matching or perceptual hashing), while others require a VLM to identify whether an image depicts the expected content.

A.5 Evaluator Contributor Details

Students from the authors' university helped develop the evaluator scripts. We recruited them through a mix of word-of-mouth and advertising the project in class. Task prompts themselves were written by the authors, contributors worked exclusively on implementing, debugging and evaluating the per-task evaluator scripts against gold artifacts. Contributors participated through one or more of the following arrangements: research-assistant role paid at $20/hr, course credit for research, or paid hourly work at $15/hr.

A.6 Benchmark Maintenance and Deprecation Policy

Live web tasks may be affected by changes or updates to websites over time. We intentionally limit the susceptibility of KNOWS to website changes by aiming to curate tasks that are time-agnostic and independent of any single website. Nevertheless, we cannot guarantee that every task and evaluation script will remain functional indefinitely, so we adopt the following maintenance and deprecation policy, modeled after the policy of Mind2Web 2 (Gou et al., 2026).

We have curated a list of 23 URLs and 20 tasks with website-extracted gold data that our evaluators rely on, and we commit to manually checking these at periodic intervals after release. We prioritize reliability over ease of running these checks, and therefore commit to manual verification. In parallel, we will validate automatic freshness checks against the manual ones, and will switch to automated checking once it proves consistently accurate.

If substantial website changes or unavailability significantly alter a task's difficulty or solvability, we will update the affected task or replace it with a new one of similar complexity and scope. Any changes to tasks or evaluators will be logged as a new version of the benchmark on GitHub, and we will archive each prior version so that results reported against it remain interpretable and reproducible. We include instructions on the project's GitHub page `https://github.com/alexgill321/KNOWS-benchmark` for community reporting of any outdated tasks or evaluator issues.

B Experimental Details

Our tasks require the creation of new Google Workspace artifacts, and the evaluation scripts directly use the Workspace API for evaluation. To run this benchmark, we first authenticate with a Google account and, for each task instance, the framework creates blank output artifacts to be filled in by agents. Some tasks also bundle supplementary input data—such as reference images, recipe PDFs, or CSV files—in a local `data/` directory accessible to the agent. Once baselines are run, we then use these created artifacts for our evaluator scripts (see §3.3) to score agent performance.

B.1 Live Web Harness

For evaluating our LLM baselines, we use the BrowserGym ecosystem (de Chezelles et al., 2025), which provides an easy common interface for web browsing tasks. This not only provides the infrastructure, but also a standard interface for observation spaces including accessibility trees and action

Category	Use Case	Modalities	Tasks	Source
Letters	Business Letter	Docs	5	Taxonomy
Personal	Recipes	Docs, Sheets	10	Taxonomy
Education	Lesson Plan	Docs, Slides	10	Taxonomy
Education	Class Notes / Reference List	Docs	5	Taxonomy
Education	Research Report	Docs	5	Taxonomy
Education	Book Report	Slides	5	Taxonomy
Education	Research Library	Sheets	5	Extension
Personal	Investment Tracker	Sheets	5	Taxonomy
Personal	Travel Planner	Sheets	5	Taxonomy
Personal	Wedding Planner	Sheets	5	Taxonomy
Personal	Fitness Analytics	Sheets	5	Extension
Personal	Outdoor Activity Planner	Sheets	5	Extension
Personal	Housing Search	Sheets	5	Extension
Personal	Media Recommendation	Sheets	5	Extension
Personal	Lookbook	Slides	5	Taxonomy
Personal	Party Invite / Event Poster	Slides	5	Taxonomy
Personal	Product Comparison	Slides	5	Extension
Personal	Purchase Decision	Slides	5	Extension
Work	Photo Directory	Slides	5	Extension
(Cross)	Slide Editing	Slides	5	Extension
20 unique domains			110	11 T / 9 E

Table 4: Domain coverage of the KNOWS benchmark. Each row is a distinct use-case domain identified by (category, use case). Domains are drawn from a Google Workspace use-case taxonomy defined in the task creation colab notebook (Appendix A.1). 11 fall within the pre-defined taxonomy ("Taxonomy") and 9 are extensions we introduced to capture live web-heavy use cases the original design under-specified ("Extension"). Every domain is represented by at least 5 individual tasks via our template$\times$instance structure.

spaces, which allows for both low-level actions (e.g., `click`) as well as more general, high-level interactions (e.g., `goto`). We report the action space in Table 5. Some baselines, such as Comet (Yang et al., 2025) are integrated into browsers and are thus pre-packaged with their own harness, and we use these instead.

B.2 Google Authentication.

To avoid bot detection, Google authentication state is captured fresh before each task and stored per worker `PID`, giving each parallel Ray worker its own isolated Playwright storage state. Shared snapshots and persistent Chromium profiles are deliberately avoided to maintain worker isolation. Should a session expire mid-task, inline re-authentication instructions serve as a fallback.

B.3 Evaluation Cost

Here we report the monetary cost of running our baselines on the KNOWS tasks. Token- and step-level efficiency of baselines cannot be reported uniformly across all agents because Comet and Atlas are consumer products that don't expose token consumption or action traces, and are priced as flat-rate subscriptions. For Comet and Atlas, wall-clock time is not directly comparable as those agents will occasionally stop to request user assistance. While we do not provide any assistance in our benchmarking, we do need to prompt them to continue the task with no assistance in these instances. The one metric that is comparable across every system is monetary cost, which we report in Table 6.

B.4 Quantitative Error Analysis

Every evaluator in KNOWS has verdicts produced by explicit checks. We categorize all scoring call sites across evaluator instances so that each step records which type of mechanism decided a pass/-fail verdict, along with the verdict itself. We fix mechanism categorizations under the conditional branches an evaluation step terminates at, so assignment is always deterministic.

Categories. Table 9 defines the scoring mechanism taxonomy. Eight categorizations can occur on failure or pass verdicts, while UNCHECKED PASS occurs only on passing steps.

Assignment rules. Many evaluation steps have multiple fallback checks (e.g., exact pixel match $\rightarrow$ perceptual hash $\rightarrow$ VLM comparison). In these cases, the step records the last mechanism that ran, since it makes the final call on the step's success verdict. When an LLM extracts the value under test, but a deterministic or tolerance-based comparison renders the verdict, the step records the com-

Action	Description
`click`	Single left-click on an element
`dblclick`	Double-click on an element
`drag_and_drop`	Drag an element to a target location
`fill`	Clear and fill a text input
`go_back`	Navigate to the previous page
`go_forward`	Navigate to the next page
`goto`	Navigate to a given URL
`hover`	Move cursor over an element
`keyboard_insert_text`	Insert text at the cursor position
`keyboard_press`	Press a key or key combination
`keyboard_type`	Type text character by character
`mouse_click`	Click at absolute screen coordinates
`mouse_dblclick`	Double-click at screen coordinates
`mouse_drag_and_drop`	Drag between two screen coordinates
`mouse_move`	Move mouse to screen coordinates
`mouse_upload_file`	Upload a file via mouse interaction
`new_tab`	Open a new browser tab
`noop`	Perform no operation
`report_infeasible`	Flag the task as infeasible
`scroll`	Scroll the page or an element
`select_option`	Select an option from a dropdown
`send_msg_to_user`	Send a message to the user
`tab_close`	Close the current tab
`tab_focus`	Switch focus to a specific tab
`upload_file`	Upload a file to an input element

Table 5: Action space of KNOWS.

parator's category. When a single step scores N sub-items whose verdicts may come from different mechanisms, the step records the majority category among the failing items if any item failed, and the majority category among all items otherwise.

Step-wise analysis details. We grade the artifacts produced by the best performing baseline, Comet, on all 110 tasks. No browser history is taken, as Comet does not provide it. From the 3,834 evaluation steps, Comet passes 2,380 steps (62.1%) and fails 1,454. Figure 4a summarize the distribution. Table 7 breaks pass rates and failure shares down by artifact type, and Table 8 attributes the failures. Below, seed tasks are referred to by artifact type and number (e.g., Docs-11), matching the identifiers of our released tasks. Three observations stand out: (a) the same mechanism behaves very differently across artifact types: TREE QUERY steps are near-perfect in `Sheets` (97%, column and row existence and ordering) but the weakest mechanism in `Slides` (41%, required slides present and in order) and `Docs` (50%, section layout of recipe pages and reference lists), (b) GEOMETRIC checks barely exist outside `Slides`, where they are a genuine weakness (67%), (c) `Docs` is the only artifact type in which EXACT MATCH failures dominate.

Failure attribution. Only 798 of the 1,454 failed steps (55%) are a check that ran on the artifact and rejected it (Table 8). All 80 TRACE MATCH steps, plus MODEL JUDGEMENT and EXECUTION ERROR steps whose source-fidelity checks need the visited URLs. We excluded the TRACE MATCH steps from our analysis because they require the browsing trace, which Comet does not provide. We instead manually evaluate the browsing traces. The remaining 427 (29%) are EXECUTION ERROR or UNMET DEPENDENCY steps, and most of them are evaluator-side.

Some of the API failures include: (a) a failed apartment-listing page fetch in Sheets-38 cascades through every downstream check of that listing (121 UNMET DEPENDENCY and 23 EXECUTION ERROR steps), (b) a vehicle-statistics lookup returns nothing in Slides-29 (75 steps), and (c) a film-database lookup fails in Sheets-55 (17 steps worth 171 of that task's 360 points). Together, these three lookups account for roughly 236 failed steps and 390 points, so Comet's ideal score would rise by a few points. Because most of these are evaluator-side, the EXECUTION ERROR and UNMET DEPENDENCY bars in Figure 4a slightly overstate the actual failures of Comet.

System	Configs	Total
Usage-based (API)		
Claude Opus 4.7	AXT, AXT+SS	$2,200
GPT 5.5	AXT, AXT+SS	$1,700
DeepSeek V4 Pro	AXT	$80
Flat-rate (subscription)		
Comet	—	$200
Atlas	—	$100

Table 6: Cost of one full evaluation pass over all 110 tasks. Totals are not comparable across rows: the API systems differ in the number of configurations run, and subscription tiers are billed monthly and are independent of usage volume.

B.5 Cross-Platform Full Results

Here we report full results for 5 task artifacts created in Microsoft Office by Comet, as discussed in §4.3. Table 10 shows the results comparing the scores for Comet artifacts created on the same tasks using both Google Workspace and Microsoft Office as the platforms.

Step type	Passed / total steps				Share of failed steps (%)			
	Docs	Sheets	Slides	All	Docs	Sheets	Slides	All
EXACT MATCH	148/219 (68%)	556/629 (88%)	247/283 (87%)	951/1,131 (84%)	26.1	13.0	5.8	12.4
TOLERANCE MATCH	137/157 (87%)	155/238 (65%)	96/115 (83%)	388/510 (76%)	7.4	14.8	3.1	8.4
MODEL JUDGEMENT	117/165 (71%)	189/282 (67%)	284/392 (72%)	590/839 (70%)	17.6	16.6	17.4	17.1
GEOMETRIC	20/20 (100%)	28/36 (78%)	158/237 (67%)	206/293 (70%)	0.0	1.4	12.7	6.0
TREE QUERY	41/82 (50%)	100/103 (97%)	85/206 (41%)	226/391 (58%)	15.1	0.5	19.5	11.3
TRACE MATCH	0/15	0/5	0/60	0/80	5.5	0.9	9.6	5.5
UNMET DEPENDENCY	0/9	0/137	0/63	0/209	3.3	24.5	10.1	14.4
EXECUTION ERROR	0/68	0/158	0/136	0/362	25.0	28.2	21.9	24.9
UNCHECKED PASS	6/6	12/12	1/1	19/19	0.0	0.0	0.0	0.0
All	469/741 (63%)	1,040/1,600 (65%)	871/1,493 (58%)	2,380/3,834 (62%)	Failed: 272 / 560 / 622 / 1,454			

Table 7: Comet's evaluation steps by scoring mechanism and artifact type. Left: passed / total steps (pass rate). Right: share of each artifact type's failed steps decided by each mechanism. Last row gives number of failed steps.

Failure class	Steps	%	Docs	Sheets	Slides
Check ran and rejected the artifact	798	54.9	180	260	358
Requires the browsing trace	229	15.7	25	119	85
Execution error / dependency skip	427	29.4	67	181	179
All failed steps	1,454	100.0	272	560	622

Table 8: Attribution of Comet's 1,454 failed evaluation steps. Only the first class is a verdict on the artifact. The second is unscorable without the browsing trace, and the third is dominated by evaluator-side external lookups.

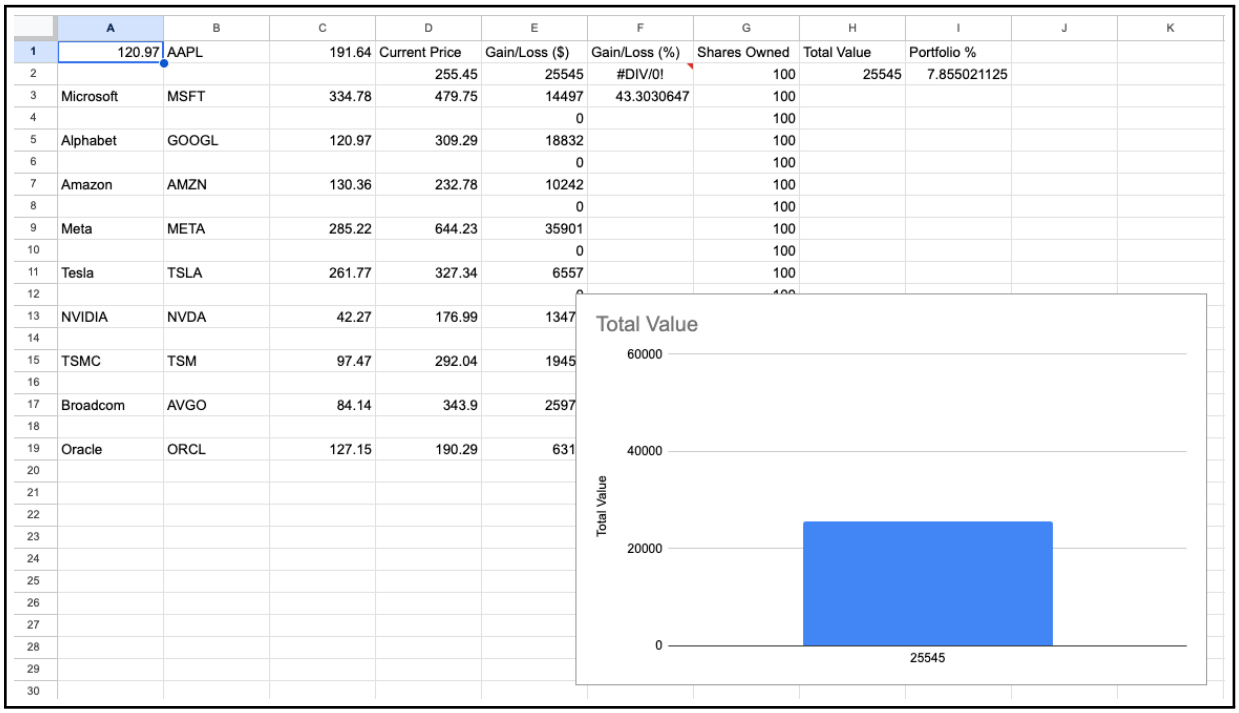


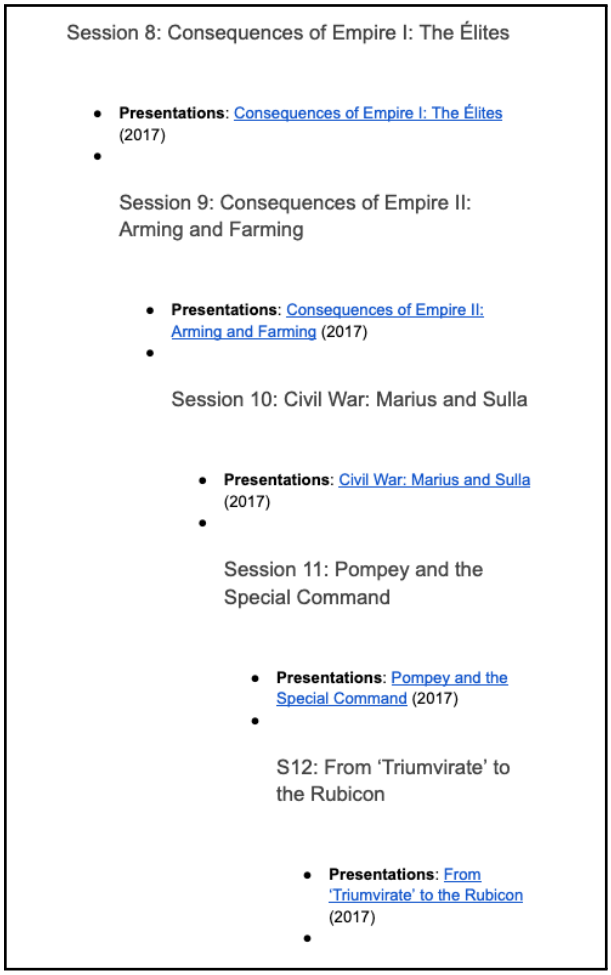
Figure 5: Output produced by the best performing agent, Comet, on a `Sheets` task. The agent makes several errors, including formula errors, inconsistent table structure, misplaced formulas, and an inability to generate visualizations correctly.

Category	Deciding mechanism
EXACT MATCH	Exact or normalized comparison: string equality, regular expressions, exact numeric equality, pixel-exact image match, API-reported formatting flags.
TOLERANCE MATCH	Tolerance-based, non-geometric comparison: fuzzy string matching, perceptual image hashing, numeric comparison within a relative or absolute tolerance, color-distance thresholds.
MODEL JUDGMENT	An LLM or VLM verdict decided the step (content relevance, semantic equivalence, image–description agreement).
GEOMETRIC	A geometric test over rendered coordinates decided the step: bounding-box containment or overlap, page-region position, area coverage, OCR-localized text position.
TREE QUERY	An artifact structural test decided the outcome of a step: element counts, ordering, position within the document tree, slide or column structure.
TRACE MATCH	A string or pattern match over the agent's browsing history decided the step. These measure process that the agent properly complies to website visits a task requires.
UNMET DEPENDENCY	The step was deemed a failure because a prerequisite step failed (e.g., a location check cannot be performed because the corresponding text is missing from the artifact). Failures in this category are cascades, not independent errors.
EXECUTION ERROR	The check could not run due to missing or unreachable input data, external API failure, or an exception in the evaluator. These reflect harness or environment faults rather than agent behavior.
UNCHECKED PASS	(Success only.) The step passed although no check actually ran, e.g., benefit-of-the-doubt branches when reference data was unavailable.

Table 9: Failure-category taxonomy. Each evaluation step records the category of the mechanism that decided its outcome.

Task	**MS 365**	**Google Workspace**	Δ
docs_1_formal_letter	8/10	8/10	0
docs_31_education_lesson_plan	99/145	132/145	-33
slides_29_buy_car_pres	57/68	62/68	-5
slides_42_personal_none_product_comparison	43/57	26/57	$+17$
slides_51_event_announcement_poster	175/185	100/185	$+75$
Combined	**382/465 (82%)**	**328/465 (71%)**	$+54$

Table 10: Cross-platform comparison of agent performance on five KNOWS tasks implemented in both Microsoft 365 and Google Workspace. Scores are evaluation steps passed out of total steps. Δ is the difference in steps passed (MS $-$ GWS).



Session 8: Consequences of Empire I: The Élites

- **Presentations**: Consequences of Empire I: The Élites (2017)
-

Session 9: Consequences of Empire II: Arming and Farming

- **Presentations**: Consequences of Empire II: Arming and Farming (2017)
-

Session 10: Civil War: Marius and Sulla

- **Presentations**: Civil War: Marius and Sulla (2017)
-

Session 11: Pompey and the Special Command

- **Presentations**: Pompey and the Special Command (2017)
-

S12: From 'Triumvirate' to the Rubicon

- **Presentations**: From 'Triumvirate' to the Rubicon (2017)
-

Figure 6: Output produced by the best performing agent, Comet, on a `Docs` task. The agent generates deeply nested bullet points, making the document difficult to navigate and unpresentable.

Task	**Time**
docs_5_influential_papers	24m
slides_20_Illustrated_Book_Report	44m
docs_31_education_lesson_plan	55m
sheets_6_investmenttracker	1.4h
docs_1_formal_letter	1.6h
slides_39_Personal_Lookbook_PaintColors	1.6h†
sheets_55_Movie_Recommendation	1.9h
sheets_2_personal_recipe_foodcomposition	1.9h
docs_11_personal_recipe_ocr	1.9h
sheets_25_skitourplan	2.1h
slides_17_removeimagesaddplaceholders	2.2h
sheets_38_apartment_finder	2.4h
sheets_10_paper_sorting	3.1h
sheets_7_running_analysis	3.1h
slides_29_buy_car_pres	3.2h
slides_51_event_announcement_poster	3.4h
slides_30_Work_Wikipedia_Photos	3.5h
slides_26_basic_educational_slide_deck	4.1h
sheets_45_wedding_color_pallette	4.4h
sheets_28_personal_travel_planner	7.3h
docs_37_reference_list	14.5h
slides_42_personal_none_product_comparison	18.8h
Median	2.3h
Mean	3.8h

Table 11: Human completion times for each task template (instance 1), measured from Google Drive Activity API edit logs. Times are sorted in ascending order.

Figure 7: The brainstorming phase of the task-creation notebook. The annotator first picks a Google Workspace tool and a optionally (top-level, low-level) category, then writes a brief overview and a first-draft task prompt. They are explicitly told they may diverge from the suggested categories.

Step 1 of the notebook: Brainstorm a task prompt

```
=== Choose Type / Categories ===

> What type of template is it?
  Options: Docs (d)  Sheets (sh)  Slides (sl)
< sh
Your template type is Sheets

> Now choose the top level template type.
  Note: you are not restricted to these (select None instead) and some
  categories may be hard to write templates for.
  Options: Personal (pe)  Work (w)  Project Management (pr)
           Education (e)  None (n)
< pe

> Now choose low level category
  Options: To-do List (to)  Annual Budget (annual b)  Monthly Budget (m)
           Google Finance Investment Tracker (g)  Annual Calendar (annual c)
           Schedule (s)  Travel Planner (tr)  Wedding Planner (w)
           Team Roster (te)  Pros and Cons (p)  None (n)
< s
You have chosen: Personal | Schedule

=== Brainstorm ===

Now that you have your categories, brainstorm a bit. If your idea doesn't
quite match the categories you chose above, that's fine -- go back and
rerun those to match the idea. Think of things you might want a digital
assistant to do in your own life, or put yourself in someone else's shoes.

> Please give a brief overview of the task you are asking the agent to do.
< Make a playlist sheet for an event for my favorite artist.

=== Draft Template ===

Example Task:
  Structure a formal letter written in my name using Google Docs. Use my
  website for my info: {link}. Add the logo of my {"university",
  "department", "company", "employer"} to the upper left if available on
  the Web. Take a photo of my signature from the drive {location} and add
  it to the end of the letter.

> Write the most straightforward prompt for what you want the template to
  do. For now, don't worry about the requirements.
< I need to make a playlist for my {EVENT}. I want all of the songs to be
  from {ARTIST/BAND/COMPOSER}. First, in one tab, list at least 20 songs
  ... [annotator's draft prompt]
```

Figure 8: The desiderata-check phase. The notebook walks the annotator through a True/False rubric grouped into five categories (utility and realism, document synthesis, information retrieval, visual / spatial understanding, complexity). When a required check fails, the annotator is asked to revise the prompt and re-do the checks.

Step 2 of the notebook: Check the prompt against desiderata

```
For each question below the annotator answers True / False.  If any
*required* check is False, they are sent back to revise the prompt.
```

```
=== Utility and Realism (all required) ===
[ ] Task represents something a real user might want to do
    (it has practical value for typical users).
[ ] Task would save users time if automated.

=== Document Synthesis (all required) ===
[ ] Task involves creating or editing a Google Workspace tool
    (Docs, Sheets, Slides, Drive, ...).
[ ] Task requires meaningful synthesis of gathered information
    (not just copying).
[ ] Final product is cohesive and well-organized.

=== Information Retrieval (>= 2 of 3 required) ===
[ ] Task requires retrieving 5+ distinct pieces of information.
[ ] Information must come from 2+ different websites / sources.
[ ] Task explicitly requires browsing the live web.

=== Visual / Spatial Understanding (>= 1 of 3 required) ===
[ ] Task requires identifying and editing specific document elements.
[ ] Task requires spatial-arrangement requirements
    (positioning elements left / right / etc.).
[ ] Task requires color-based instructions or modifications.

  If none are met, the notebook suggests randomized ideas, e.g.:
    "Add web images related to part of the document and place them at a
     given location."
    "(Sheets) Ask user to color-code some elements of the sheet."
    "Ask for certain words to be highlighted some color."
    "Ask to strike out certain words."
    "(Slides) Ask for slides matching some definition to have animation."

=== Complexity (all required) ===
[ ] Task requires 4+ steps for information retrieval
    (search + visit/extract per source).
[ ] Task requires 1+ step to create the document / template.
[ ] Task requires 10+ steps to incorporate information into the document.
[ ] Task cannot be completed by simply pasting information into a
    blank document.

>>  If any required check fails:
    "Maybe go back through previous steps and edit your prompt until it
     meets this requirement. Look at the example above (and in the
     template sheet)."
```

Figure 9: The checkpoint-writing phase. The notebook shows detailed instructions and a worked example, then has the annotator write their own checkpoints in Markdown and answer a final True/False rubric to verify that each checkpoint is measurable and ties to a concrete document state change.

Step 3 of the notebook: Write and check the checkpoints

```
=== Instructions for Checkpoints ===
  * Each checkpoint is written in English.
  * Each checkpoint describes a measurable milestone.
  * Checkpoints describe specific state changes in the document.
  * Checkpoints specify one or more required actions:
      - Finding / extracting specific information from websites.
      - Placing / adjusting information in the document in a specific way.
  * For each checkpoint, include:
      - A number
      - A general description
      - Explanation of how to evaluate
      - Eval template(s)

=== Good example (excerpt) ===
```

```
  # Checkpoints
  This task has 3 points in total.

  ## Checkpoint 1:
  Check that common information about the letter writer is placed at a
  reasonable location in the document letter.

  ### Outcome Evaluation:
  - Exact match to check that the document includes the writer's right
    name, title, and address.
  - Use OCR to locate the text and ensure it is on the left-hand side of
    the document, beneath the logo.

  ### Eval Template(s)
  - Text Exact Match, Text Location Match

=== Check the checkpoints ===
For each question below the annotator answers True / False.

[ ] Each checkpoint is written in English.
[ ] Each checkpoint describes a measurable milestone.
[ ] Checkpoints describe specific state changes in the document.
[ ] At least one checkpoint requires finding / extracting specific
    information from websites.
[ ] At least one checkpoint requires adding / adjusting information in
    the document in a specific way.
[ ] Task requires 4+ evaluation steps.
```

Figure 10: Decomposition of one task into checkpoints, and of each checkpoint into evaluation steps. The example task asks the agent to build a wedding color-palette spreadsheet; it carries six checkpoints and twenty-nine evaluation steps. Steps are a mixture of deterministic checks (URL validity, column structure, exact-text matches) and model-judged checks (LLM judges for topical relevance, VLM judges for image content and color-fidelity verification).

Example task with checkpoints and evaluation steps

```
=== Task Prompt ===

I want to make a Google Sheet to help choose color templates for my wedding. First, do a web
    search and find at least 3 articles on wedding colors. From those articles, extract at
    least 10 distinct shades of purple, yellow, or orange. Then, at the top left of the sheet,
    list all the colors vertically. In the column to the right of the color name, put the link
    to the article that mentioned the color. Then, in the next column to the right, fill in
    that cell with a background color matching the named color as closely as possible using
    the color's standard hex code (e.g. from colorhexa.com). Then, in the next column to the
    right, find a page or link to that specific color at a paint store and paste the link.
    Below this list, on the same sheet, add a second table with the same colors as the list
    above as column headers, in the same order. The rows are different wedding objects with at
    least 3 objects. Fill in each cell with a real photograph from the internet of that
    decoration in that color and display it in using =IMAGE() in the sheet. If you can't find
    an image for an object in a specific color, skip it, but fill in at least half of the
    cells. Finally, in a new tab, create at least 10 different rows of full-color palettes.
    For each row, choose three colors from the list above and fill each cell's background
    color with one of those three colors to show different color palettes.

=== Checkpoints ===

## Checkpoint 1: Color Extraction
The agent found and extracted wedding color information from at least three articles and color
    names are listed vertically in the top-left area of the main sheet.

### Evaluation Steps:
- All extracted color names appear in a vertical list in a single column.
- The color list starts in the top-left area of the sheet (rows 1-15 approximately).
- At least 10 unique color names/shades are present, extracted from articles.
- The agent searched for and found at least 3 articles about wedding colors (agent trace, LLM
    judge that the articles are about weddings).
- All extracted colors belong to the specified purple, yellow, or orange categories (LLM judge
    that each color falls into the category).

## Checkpoint 2: Article Source Links
Article links are placed in the column immediately to the right of each color name.

### Evaluation Steps:
- The article column is a URL/link column (every populated cell contains a URL, not free text).
- Each color name has a corresponding article link in the same row.
- The article links are functional and reachable.
- The links lead to relevant wedding-color content for that color.

## Checkpoint 3: Color Cell Formatting
Cells are filled with colors matching the color names in the third column.

### Evaluation Steps:
- The fill column is a colour-fill column (every populated cell has a background fill, not just
    typed text).
- The fill colors match the named colors or are extremely close to them (VLM judge).
- The hex value for the cell color matches the color shade hex found in
    https://www.colorhexa.com/.
- Each color name has a corresponding colored cell.

## Checkpoint 4: Paint Store References
Paint store links are provided in the fourth column next to each color.
```

```
### Evaluation Steps:
- The store column is a URL/link column (every populated cell contains a URL, not free text).
- Each color has an associated paint store link.
- The paint store links are functional and reachable.
- The links lead to relevant paint store / color content for that color.

## Checkpoint 5: Wedding Decoration Matrix
A wedding decoration matrix is created below the color list with images.

### Evaluation Steps:
- At least 3 types of wedding decorations are listed in the leftmost column (API for location,
    LLM judge for content).
- Column headers contain the same color names from the original list (exact text match).
- The matrix column headers appear in the same order as the original color list.
- At least half of the matrix cells contain images.
- Images show the specified decoration type in the corresponding color (VLM judge).

## Checkpoint 6: Color Palette Tab
A separate tab contains at least 10 color palette combinations.

### Evaluation Steps:
- A new sheet/tab was created for color palettes.
- At least 10 rows of color combinations exist.
- Each row contains exactly 3 colored cells representing a palette.
- Colors are filled as background colors (not just text).
- Color combinations use colors from the original extracted list.
```

Table 12: Example task prompts from KNOWS. One randomly chosen instance per task; URLs replaced with `[redacted]`.

Shortname	Task Instruction
`docs_1_formal_letter`	Structure a formal letter written in my name using Google Docs. Use my website for my info: [redacted]. Include my employers logo in the upper left corner if available online. Include the following information in the letter header: - My Name - My Email Address - My Job/Position Title Take the photo of my signature from the drive folder here: [redacted] and add it to the end of the letter.
`docs_11_personal_recipe_ocr`	Create a recipe from the Recipe Coral template in Google Docs. First, replace the name of the recipe with the correct recipe name by looking at the image saved in the drive at [redacted]. Then, copy the ingredient list into the template and paste the instructions into the corresponding field. Get tips from an online source, and make sure to cite source urls for tips below the tips entries. Use direct quotes for all tips. Replace the default image with a cropped image from the original drive link that shows the recipe text clearly. Fill out the rest of the template with relevant information from the recipe. Then search the internet for at least 3 similar recipes and put them on separate pages with the same format as the first recipe. Put the link for the source of each additional recipe on a line below the title fitting with the formatting. Make sure each page of recipes is visually distinct from the others. Finally, check the other online recipes and fill in the tips with some useful information that you found.
`docs_31_education_lesson_plan`	I am an Elementary School educator, and I have a free class to teach a fun topic. My class is on Deep Learning. Can you help me make a lesson plan document appropriate for my students? First, come up with 2-3 fun topics in Deep Learning that the class might enjoy but is still related and list those as the main bullets. Then, under each of these, find 4 unique websites that have information about the topic and put their URLs as sub-bullets. Then, for each of these websites, list 5 facts as sub-bullets to the url. Finally, list all of the facts below the bullets and color code them by which topic they are about with the highlighter. Use a different color for each topic. Write a summary of the lesson using the facts collected. Finally, go online to find images related to each of the topics. Put them all side-by-side at the bottom of the last page of the document.
`docs_37_reference_list`	Create a Google Doc reference list from this course webpage: [redacted]. For each lecture, add a Heading 3 with 'Module N: Lecture title', where N is the module number. Things like exams and holidays should not appear in the reference list. Under each title, create a separate bullet list for tutorials, textbooks, and videos. Each bullet list should start with "Tutorials", "Textbooks", or "Videos" in bold, normal font. If a given type has no hyperlinks, delete that category. Each bullet is a hyperlink to a resource referenced in the lecture slides, using the resource title as the anchor text instead of the raw URL. Include author(s) in parentheses after the hyperlink, formatted as "(First author)". Next to the hyperlink, add "Slide: <list of slide numbers separated by commas>" listing the slides where the link is referenced. No duplicates: if something is referenced multiple times, then add all slide numbers. If a link is referenced on more than one slide, bold that hyperlink (and only the hyperlink) in the corresponding bullet list, italicize it, color it 'dark cyan 1', and set its font size to 12pt. If the link is inactive (dead link) do not include it in the google doc.
`docs_5_influential_papers`	Create a Google Docs with five influential papers on parameter-efficient fine-tuning (PEFT). Include links to arxiv.org for each of the papers. The papers should have at least 100 citations, have been first published within the last three years, and must have an active page on semantic-scholar.org. Include the name of the paper, a link below the name, and the abstract of each of the papers below the link.
`sheets_10_paper_sorting`	Create a new google sheets document for all the papers in my drive folder [redacted]. Make a blank google sheets table and for each paper, in column A put the title of the paper, in column B put the paper authors, in column C put the abstract, in column D put a link to the arxiv abstract website, in column E put the links to the google drive pdfs. Additionally, search the web for at least three more papers on arxiv where the first author of each existing paper is also listed as the first author (or less if they do not have at least 3 first-authorships on arxiv) and put the same information. For new papers, download the PDFs and upload them to the pdfs folder in this drive folder: [redacted] and link them in column E. Extract or take a screenshot of Figure 1 for each paper, upload the image to the figures folder in this drive folder: [redacted] and display it in column F using =IMAGE() with the Drive URL. In column G put a checkbox and check for each paper that is a new paper. Highlight papers that have a related works sections which mention dark energy in yellow, and make sure to group these papers at the top rows of the sheet. Ensure all of the text in the table is visible and not cut off.

continued on next page

Table 12 – continued from previous page

Task	Prompt
sheets_2_personal_recipe_foodcomposition	From the recipe [redacted], pull the average amount of iron, carbohydrates, fat, sugar, potassium, vitamin A, protein, sodium, fiber, calcium, and vitamin C for each ingredient (based on its quantity) using the USDA food database. When searching the USDA, use the food code that best matches the ingredient; more general codes are usually preferable, unless the recipe specifies a matching brand. For instance, if a listed ingredient is just “Paneer”, you would use “Cheese, paneer” instead of “Palak Paneer”. You may use any of the five USDA food search tabs. Place the results into a Google Sheet as follows. Group nutrients into macros, minerals, and vitamins. Within each group, sort nutrients alphabetically. The first column should be titled "Ingredients" and the second “Link”, followed by sorted nutrient columns starting with macros, then minerals, then vitamins. Above each group, add a merged, centered, italicized header cell labeled "Macros", "Minerals", or "Vitamins" spanning all associated columns. Bold all column titles, excluding the group header cells. Color all cells in each group with a light background color, making sure each group has a distinct color. Place a horizontal line under column titles. List ingredients in the first column in the order they appear in the recipe. In the “Link” column, add the USDA data link you used to obtain the quantity of a nutrient in a specific food. Bold any nutrient values exceeding 10% DV according to FDA guidelines. For ingredients you can’t find in the USDA database, leave centered N/A. For ingredients with a range of values as the amount, use the minimum value. Ignore ingredients that are listed as optional or don’t have a specified amount.
sheets_28_personal_travel_planner	A family of 4 people is traveling to Washington D.C this May for 3 days, and they need a detailed day-by-day itinerary. Please prepare this travel plan and put everything into a Google Sheet so it’s easy to follow. They will stay at the same hotel Hyatt Regency Washington on Capitol Hill for all 3 days of the trip and use Public Transportation for travel during the trip. They plan to do a weekday trip. Here’s what they are looking for: - Activities & Sightseeing: Each day should have a mix of morning, afternoon, and evening plans at popular attractions in the city. Please include the opening hours, suggested duration, and any entrance fees. (For Activities & Sightseeing, there should be three rows per day: morning, afternoon, evening.) - Food Stops: Include two food stops per day (lunch and dinner). For each, add the type of cuisine, an average price estimate. - The food stops must be in the order: Lunch between Morning and Afternoon activity, Dinner between Afternoon and Evening activity. - Time of Day: Each row must specify the time slot — one of: Morning, Afternoon, Evening (for activities), or Lunch, Dinner (for food stops). - Start Time and Departure Time: Each row must include a start time (when the visit/meal begins) and a departure time (when the family leaves). These times must align with the Time of Day slot (e.g., Morning activities start in the morning hours, Lunch starts around midday). Start and departure times must not overlap with adjacent events and must leave enough time for transportation between destinations. The schedule across each day should flow sequentially without gaps or conflicts. - Transportation: For every entry (activity or food), note how they’d get between places (walking, metro, bus, taxi, etc.) and the typical travel time. Clearly indicate whether each day is a weekday or weekend, and adjust transit time estimates accordingly (e.g., account for rush hour delays on weekdays and reduced service or increased congestion on weekends). Ensure transit time correctness reflects these differences. - Costs: Provide an estimated cost for each main activity and meal, and color-code the cost column: under $100 = green, $100–$200 = yellow, above $200 = orange - The review link for both Activity and Food should be from Google Maps. - Alternatives: For every destination (food or activity), suggest one backup option (for weather, crowds, or unexpected closures). Put these in the last column and format them in blue. - Please avoid repeating the same destination or restaurant across the trip. - If two places are far apart, adjust the order so the day makes geographic sense, and the pacing feels realistic (not jam-packed, but no big idle gaps). - Include the following columns: Date, Time of Day, Destination (Food/Activity), Cuisine, Opening time, Start time, Departure Time, Duration, Review link, Transportation Mode, Travel Time, Cost (with color coding), Alternative Option (in blue). - Freeze the top row (header). - Individual transit times should be <= 30 minutes and keep total transit times under 90 min per day

continued on next page

Table 12 – continued from previous page

Task	Prompt
sheets_25_skitourplan	Create a ski tour plan for 3 possible runs 32 degrees or less using the wasatch backcountry ski guide. Organize the route information for each run in a google sheet including the run name, run link, starting location, gps coordinates of the run location, typical vertical of the run, slope aspect, and slope angle as columns. Then look up the danger color rating for this run according to the Utah Avalanche Center forecast for 12/28/2024. Add columns for forecast date and forecast link and a screenshot of the full danger rose. These 3 columns should be merged vertically since they apply to all runs. Then using this information, determine the danger color rating for each run. Color the cell of each run name with the color of its danger rating.
sheets_38_apartment_finder	I am looking to rent a 3 bed 2 bath unit in Chicago, Illinois. Your task is to check www.craigslist.org and create a google sheet with at least 6 listings. My budget is 3000-4500 USD/month and I would like to move in within the next month. The unit should have the features: air-conditioning and optionally has at least 1200 sq ft. For each unit include the necessary columns (including the address) and a link to the listing in the spreadsheet and use conditional formatting on the numeric features (green to red where green means better and red means worse). Add an additional column at the end on interesting positive features of the listing and another column on potential dealbreakers (if there are none, keep it empty). Create a separate "summary statistics" table in the top-right corner (starting at column L) while the main data starts at A1. Populate the summary statistics table with data about the listings table using sheets equations that will update the statistics automatically. Ensure all of the text in both tables is visible and not cut off. Both tables must be inserted as blank Google Sheets Tables and then populated with the data.
sheets_45_Personal_WeddingPlanner_weddingcolorpallette	I want to make a Google Sheet to help choose color templates for my fall wedding. First, do a web search and find at least three articles discussing fall or autumn wedding colors. From those articles, extract at least 15 different specific shades of burgundy, rust, or mustard. Then, at the top left of the sheet, list all the colors vertically. In the column to the right of the color name, put the link to the article that mentioned the color. Then, in the next column to the right, fill in that cell with a background color matching the named color as closely as possible using the color's standard hex code (e.g. from colorhexa.com). Then, in the next column to the right, find a page or link to that specific color at a paint store and paste the link. Then, in the next column, add a short one-sentence description of the mood or feeling that color evokes. Below this list, on the same sheet, add a second table with the same colors as the list above as column headers, in the same order. The rows are different wedding objects with at least 4 objects. Fill in each cell with a real photograph from the internet of that decoration in that color and display it in using =IMAGE() in the sheet. If you can't find an image for an object in a specific color, skip it, but fill in at least half of the cells. Finally, in a new tab, create at least 15 different rows of full-color palettes. For each row, choose three colors from the list above and fill each cell's background color with one of those three colors to show different color palettes. Add a label in the first column of each row giving the palette a creative name.
sheets_55_Movie_Recommendation	I want to watch a movie tonight. I prefer the following genres: Action, Drama, Thriller, Sci-Fi, and Comedy. Find the genre based on the IMDb listing of the movies. I want to watch award-winning movies, and I prefer movies that are highly rated and well-reviewed (IMDb $\geq$ 6.5). Look for movies that have won at least one Oscar award for Best Actor, Best Actress, Best Director, Best Original Screenplay, Best Adapted Screenplay, or Best Cinematography. Please make a Google Sheet with at least 5 movie recommendations that won at least one of the mentioned awards. Each movie's primary genre on IMDb — the first standard genre IMDb lists for the movie — must be one of the above listed genres. Use IMDb's standard genre list, not UI tags. Each row should contain information about one movie. Use these column header names, in this order: Movie Title Genre MPA/Age Rating IMDb Score Release Year Duration Oscar Awards Won MPA/Age Rating is the MPA/age rating, e.g., PG, PG-13, R. In the Oscar Awards Won column, list all of the qualifying Oscar categories above that the movie won, separated by commas. Do not include other awards. Sort the list based on Duration in descending order. Highlight the highest IMDb score in the list with green fill + bold text. Highlight the lowest IMDb score with red fill + bold text.

continued on next page

Table 12 – continued from previous page

Task	Prompt
sheets_6_investmenttracker	Make a stock tracker google sheet for the top 10 highest market cap stocks in tech from the end of Q2 2023. Insert a blank Google Sheets Table and then populate it with the data. Add the name and ticker of each of these to my watchlist and compare their past price to the current price for each stock to determine how much I would have gained or lost on each stock. Finally, put that I currently own 100 shares of each of these stocks and make a bar chart showing each of my stocks and their total value as a percentage of the overall portfolio.
sheets_7_running_analysis	Gather my running data from the provided Drive link: [redacted] and put relevant data into a table in a Google Sheet. Create 2 plots using this data, one that shows my average running speed in (min/mile) over time, and one that shows the cumulative distance I've ran in miles over time. The running speed plot should have all runs as circular points and the distance plot should be a tracking the increasing distance over time. Include these plots in the same sheet tab as the data table. Find online and add in the average male age 25 running speed for a 5k as a baseline dotted line in the average running speed plot. Compare this to the average running speed for Eliud Kipchoge in his top 3 races in the Marathon. Make sure everything in the google sheet is fully unobscured and put the sources for the average running times you retrieve below the relevant plot.
slides_17_removeimagesaddplaceholders	Take the presentation at [redacted] and help me replace all the images. First, make a copy of the original presentation and put in into [redacted]. Then identify all of the images in the document and save them all as image files and place them into [redacted] for possible later use. Delete all the images, replacing each one with a description of the image in a text box at the same spot in big red text. Finally, for each of the images, find a similar image online and place it in the right spot. For each of these, make sure to put the URL for the image address below each new image to credit it.
slides_20_Illustrated_Book_Report	I need to make a book report slideshow about A Cat At the End of the World, can you help me get started? First, on the title slide put my name Anisija Mihaljević on the bottom, the title of the book above my name, and a photo of the cover of the book above that. Next, make a slide for at least 5 of the major characters with their name as the title and bulletpoints describing their characteristics below. At the bottom of each slide, include a section where you link to the source that you got the information for that character from. Make sure each character slide has a different background color. Finally, make a slide for the author of the book with their name and a photo of them. At the end, create a references slide and include links to at least 3 different sites that you used to create the presentations. Ensure that all information you enter in the slides is from these references.
slides_26_basic_educational_slide_deck	I'm preparing an introductory teaching presentation on 'Mercury' for a 7th-grade audience, and I'd like help designing the slide content from scratch. Start with a title slide that highlights the Mercury in large, bold, dark-orange lettering, includes my name (Mrs. Emily Carter), and uses a thematically relevant background image. Add a small-font source credit beneath the image. Next, break the Mercury into four student-friendly sections. For each section, locate one reliable online reference through web search, and create a slide that includes: - A bold-italic section heading - A short list of bullet-point explanations based on the source you found - A clear citation for the source in small type in the lower-left corner - An illustrative online image, placed on the right side, with its source credit beneath it in small text. Finish with a slide that summarizes the key ideas and encourages the student to ask questions.
slides_29_buy_car_pres	Create a slides presentation comparing different cars that can be categorized as a sports car. First, find an online article talking about at least 5 different 2010 models. Make a title slide called "Comparing Different Cool Cars to Buy" with font size atleast 30pt. Then for each of at least 5 cars of that category, make a slide for each. Start with the make and model as the slide title. Then, for each car, go to Kelly Blue Book and find the sticker price, fuel efficiency and horsepower. Then, go to Kelly Blue Book for user reviews and get the average rating for each. Include the link to the review website in the slide. For each car, add a picture of the car that takes up at least 50% of the slide. The picture must be from the article or Kelly Blue Book. Finally, on the last slide, list which car had the best stats in each category (sticker price, fuel efficiency, horsepower, user rating).

continued on next page

Table 12 – continued from previous page

Task	Prompt
slides_30_Work_Wikipedia_Photos	I represent several clients in Science and want to help replace their bad Wikipedia pages. I need to make a presentation on this. Could you please look at my client list at [redacted] and make a presentation for me? The title should be "Why we need new Wikipedia headshots" and have a picture on the top right taking the majority of the slide of Tom Hanks from their wikipedia page. Then, for each of my clients, make a slide with their name at the top, their current wikipedia picture on the left. Then, find images on the internet of each client and put it on the right side so it is symmetric with their wikipedia picture. Finally, on the last slide, copy all of the wikipedia urls of the client pages and then the webpage of a photographer in San Francisco.
slides_39_Personal_Lookbook_PaintColors	Create a Google slides presentation to help us choose colors for our home office. First, on the title slide, find an image of a home office by searching "home office" and choosing one of those images. Make home office the title text of the title slide and make the image as big as possible. Then, for the next 5-10 slides, choose a color. Make the color the title of each of those slides. Then for each slide, search for images of that color and home office and choose images with that color. Put two of these images on the slide, one on the bottom left and one on the bottom right. Ensure that each image's source URL is present in the image's ALT text. Finally, on the final slide, choose whichever color from the previous slides you like best. Then on that final slide write in big font "COLOR is the best choice" and make the background color of only that slide that same color.
slides_51_event_announcement_poster	Create a one-page Google Slides formatted as a visually appealing event announcement poster. This is the Breaking the Silence: Understanding the Youth Mental Health Crisis event, organized by the American Psychological Association. It will take place on 10/03/2026 at the Ronald Reagan Building and International Trade Center, Washington, D.C. The main theme is the growing mental health crisis among adolescents and young adults, and the featured speaker is Dr. Lisa Damour. Retrieve reliable background information about the growing mental health crisis among adolescents and young adults that is closely related to Dr. Lisa Damour's work from a few stable, reputable sources (e.g., universities, professional associations, encyclopedias, major news outlets, and the speaker's website). Summarize the topic in 1-2 paragraphs that highlight why the event is worth attending and cite the websites you used. Also, introduce the speaker, including their education and current affiliation, and mention 3 of their main professional achievements. Poster Structure: Header (top, centered): Breaking the Silence: Understanding the Youth Mental Health Crisis, large, bold, in color Deep Teal. Subheader (below header, left-aligned): 10/03/2026, Ronald Reagan Building and International Trade Center, Washington, D.C., and hosted by the American Psychological Association, styled in a smaller italic font. Logo (top, right): Retrieve the logo of the American Psychological Association and insert it to the top right of the poster. Make sure the logo size isn't too big or too small. Central Section (main body, centered): Synthesized background on the growing mental health crisis among adolescents and young adults (engaging, persuasive, 1-2 short paragraphs). Right Sidebar (shaded box, color Warm Amber): Speaker bio (Dr. Lisa Damour), including her credentials, achievements, and relevance to youth mental health. Footer (bottom, right-aligned): Contact info of the American Psychological Association, Dr. Lisa Damour's webpage. Styling Requirements: Ensure visual contrast between header, subheader, and body text. Use at least two colors: Deep Teal for header elements and Warm Amber for sidebars/highlights. Maintain consistent spacing and alignment so the document reads like a poster, not just plain text. Ensure that citations are placed in speaker notes and say exactly what they are citing in the poster.

continued on next page

Table 12 – continued from previous page

Task	Prompt
slides_42_personal_none_product_comparison	My colleague, James, has just been promoted to Senior Account Manager and will be starting his new role at Deloitte next quarter. I want to buy him a tablet as a congratulatory gift and need help deciding between three options: iPad Pro (M4), Samsung Galaxy Tab S10 Ultra, and Microsoft Surface Pro 11. Objective Create a comprehensive, visually appealing slide deck that compares these three devices specifically for a corporate professional's use. The presentation should be polished and persuasive rather than dry or overly technical, helping both my colleague and me make an informed decision. Target Audience Primary: Newly promoted corporate professional entering a client-facing role Secondary: Colleague making the purchase decision Design Requirements Theme: Use Deloitte's official colors (Deloitte Green and black) throughout the presentation Tone: Professional, confident, and easy to understand Focus: Business professional needs and workplace lifestyle Slide Structure and Content Slide 1: Title Slide Title (bold): "A Gift for James!" Subtitle: "A Comparison of the iPad Pro (M4), Samsung Galaxy Tab S10 Ultra, and Microsoft Surface Pro 11" Image: Photo representing Deloitte (office or branding scene) The title and subtitle are to the left, and the image is to the right of the slide Slide 2: The Challenge & The Goal Explain the decision-making challenge Define what we're trying to achieve Slide 3: How We'll Judge (The Criteria) List evaluation criteria relevant to business professionals: Client presentation capability Battery life for long workdays and travel Portability and weight for commuting Performance for productivity and multitasking Security and enterprise compatibility Other relevant factors Slides 4-6: Deep Dive on Each Device One slide per device (iPad Pro M4, Samsung Galaxy Tab S10 Ultra, Microsoft Surface Pro 11) Content per slide: Key features and specifications Pros and cons with respect to professional/corporate use Two product images showing different angles All the information should come from the official website of each option. Cite the resources when needed. Slide 7: Side-by-Side Comparison Format: Three-column comparison table, with devices as column headers Content: Feature-by-feature comparison using the established criteria Color coding system: Green: Best performer in each category Yellow: Middle performer Red: Lowest performer in that category Slide 8: Who Is Each Device For? (The Recommendation) Summarize information on each option and match each device to different professional types/needs Provide clear recommendations based on different priorities Additional Guidelines Keep technical jargon to a minimum Use business-relatable examples and scenarios Include practical considerations for corporate life (travel, client meetings, remote work) Ensure all information is current and accurate Make the presentation visually engaging with appropriate use of images, colors, and formatting

Figure 11: The prompt we use with Claude Code to draft an evaluator script for a given task. The generated scripts were a useful starting point, but finalizing them and making them reliable took substantial effort of multiple authors.

Prompt for drafting evaluator scripts with Claude Code

```
# Task Creation Guidelines

> **IMPORTANT: Before writing or modifying any evaluator, read
    [`eval_utils/EVAL_UTILS_DIRECTORY.md`](../eval_utils/EVAL_UTILS_DIRECTORY.md) in full.**
    It is a comprehensive directory of every shared utility function with descriptions and
    real usage examples. Always reuse existing utilities from `eval_utils/` rather than
    writing new helper functions. This is essential for maintaining consistency across
    evaluators and avoiding duplicate code.

## Directory Structure

tasks/
+-- <template_name>/
    +-- utils.py              # Shared utilities (optional)
    +-- DEV.md                # Development state tracking (required when developing)
    +-- test/                 # Test suite (optional)
    +-- instance_1/
        +-- task.md           # Task description
        +-- checkpoints.md    # Evaluation rubric
        +-- evaluator.py      # Evaluation logic
        +-- id.txt            # Unique identifier
        +-- gold_instances.csv
        +-- data/             # Task-specific assets (optional)

### Required Files (Instance Level)
- **`task.md`**: Human-readable task description
- **`checkpoints.md`**: Evaluation criteria with point values
- **`evaluator.py`**: Implements `grade_checkpoints()` returning a `Result` object
- **`id.txt`**: Unique task identifier
- **`gold_instances.csv`**: Maps instance names to Google document IDs

### Optional Files
- **`utils.py`** (template level): Shared utilities - check `eval_utils/` first before
    creating new ones
- **`data/`**: Reference datasets, gold images, expected outputs
- **`test/`**: Unit and integration tests

---

## DEV.md - Development State Tracking

**CRITICAL**: When developing or modifying an evaluator, maintain a `DEV.md` file at the
    template level. This file tracks development state, issues, and decisions.

### Requirements
- **Maximum 300 lines** - keep it concise and relevant
- **Read DEV.md** at the start of each session working on the evaluator
- **Update DEV.md** after significant changes or when issues are discussed
- **Delete irrelevant sections** as issues are resolved

### DEV.md Structure
# <Task Name> - Development Notes

## Current Status
Brief summary of evaluator state (working/in-progress/blocked)

## Known Issues
- Issue 1: description and status
- Issue 2: description and status

## Recent Changes
```

```
- YYYY-MM-DD: What changed and why

## Implementation Decisions
Key decisions made and their rationale

## TODO
- [ ] Remaining work items

## Notes from Discussions
Relevant context from user conversations

### When to Update DEV.md
- After implementing a checkpoint
- When encountering a bug or edge case
- After discussing issues with the user
- When making non-obvious implementation decisions
- Before ending a development session

---

## Checkpoint Mapping

**Each criterion in `checkpoints.md` maps to exactly ONE `add_step()` call.**

Exception: Repeated validations (e.g., 5 slides) use loops with clear naming like `f"Slide
    {i+1} - Title"`.

### Correct Example
# checkpoints.md
## Checkpoint 1 (3pt): Title slide elements
- Student name present
- Book title present
- Cover image present

def grade_checkpoint_1():
    checkpoint = Checkpoint(total=3, result=0, name="Title Slide")
    checkpoint.add_step("Student Name", check_name(), 1, "...")
    checkpoint.add_step("Book Title", check_title(), 2, "...")
    checkpoint.add_step("Book Cover", check_cover(), 3, "...")
    return checkpoint

### Incorrect - Don't Split Criteria
# DON'T: One criterion becoming multiple steps
checkpoint.add_step("Bullet Count", has_bullets, 1, "...")
checkpoint.add_step("Bullet Content", valid_content, 2, "...")  # Combine these!

---

## Planning Evaluation Steps

For each step, document:

1. **Summary**: 1-2 sentence implementation description
2. **Utilities**: List from `eval_utils/`, `utils.py`, or other evaluators
3. **Hierarchy**: Fallback chain (exact -> fuzzy -> LLM-based)
4. **New Utilities**: If needed, specify name, purpose, and location

Example:
### Step 1.2: Book title (fuzzy match)
**Summary**: Validate title appears on slide with minor variation tolerance.
**Utilities**: `text_fuzzy_match_contained()`, `get_slide_text_content()`
**Hierarchy**: exact match -> fuzzy (85%) -> LLM validation
**New Utilities**: None

---

## Code Patterns
```

```
### Imports
import os, time
from typing import List

from rapidfuzz import fuzz
from src.browsergym.eval.eval_utils.scoring import Checkpoint, Result
from src.browsergym.eval.eval_utils.text_utils import (
    keyword_exact_match,
    keywords_exact_match,
    keywords_match_robust,
)

### LLM Queries
from src.browsergym.eval.eval_utils.models import load_model

model = load_model("gemma-google-ai")
messages = [
    {"role": "system", "content": [{"type": "text", "text": "Instructions"}]},
    {"role": "user", "content": [
        {"type": "image", "image": "/path/to/image.png"},  # For vision
        {"type": "text", "text": "Query"}
    ]}
]
response = model(messages)

### Evaluator Structure
def grade_checkpoints(workspace_doc_id, cached_models=None, browsing_history=None):
    total_start = time.time()
    checkpoints = [grade_checkpoint_1(), grade_checkpoint_2()]
    return Result(checkpoints, total_execution_time=time.time() - total_start)

def grade_checkpoint_1():
    start = time.time()
    checkpoint = Checkpoint(total=5, result=0, name="Checkpoint Name")

    step_start = time.time()
    success = perform_validation()
    checkpoint.add_step("Step Name", success, 1, "Details", time.time() - step_start)

    checkpoint.execution_time = time.time() - start
    return checkpoint

---

## Best Practices

1. **One criterion = one step** (except loops)
2. **Reuse before create** - check `eval_utils/` first
3. **Track timing** for performance monitoring
4. **Specific exceptions** - no bare `except`
5. **Detailed failure messages** for debugging
6. **Clean up temp files** after evaluation
7. **Support cached models** to avoid reloading

## Common Pitfalls

- [X] Multiple steps for one criterion
- [X] Bare `except` statements
- [X] Missing failure details
- [X] Hardcoded paths
- [X] Duplicating existing utilities
- [X] Forgetting to update DEV.md

---

## Rules

**These rules are mandatory and must always be followed.**
```

```
1. **No utility methods in `evaluator.py`**: Never define helper functions or utility methods
     directly in evaluator files. All utilities must go in:
   - `eval_utils/` - for functions reusable across multiple tasks
   - `utils.py` (template level) - for task-specific shared utilities

   Evaluators should only contain `grade_checkpoints()` and `grade_checkpoint_N()` functions.

2. **Run eval-utils-reviewer agent after major evaluator changes**: After making significant
     updates to an evaluator (adding checkpoints, modifying evaluation logic, refactoring), run
     the `eval-utils-reviewer` agent to verify compliance with utility placement rules and
     identify any helper functions that should be moved.

---

## Running Evaluators

### Prerequisites
source <path-to-venv>/bin/activate

### Via VSCode (Recommended)
Check `.vscode/launch.json` for debug profiles with required args.

### Command Line
python src/browsergym/eval/tasks/<template>/instance_1/evaluator.py --workspace_doc_id="<id>"

---

## Checklist

- [ ] `task.md` describes the task clearly
- [ ] `checkpoints.md` lists criteria with points
- [ ] Each criterion maps to one step in `evaluator.py`
- [ ] `id.txt` contains unique identifier
- [ ] `DEV.md` tracks development state (when developing)
- [ ] Utilities tested and documented
- [ ] Error handling is robust
- [ ] Temp files cleaned up
```

Figure 12: The prompt we used to produce categorization in Table 2.

Categorization of Evaluation Step Check

```
# Eval Step Taxonomy Agent Prompt

You are categorizing every evaluation step across a benchmark dataset of Google Workspace
    tasks. Your goal is to produce one JSON file per task instance that classifies each eval
    step into a taxonomy category and flags whether the step primarily uses an LLM/VLM for
    evaluation.

## Task Overview

The benchmark lives under `src/browsergym/knows/eval/tasks/`. There are 22 tasks, each with 5
    instances (`instance_1` through `instance_5`). For each instance you will:

1. Read **three files** to understand what the eval step checks and how it is implemented:
   - `checkpoints.md`  describes what each step evaluates in plain language
   - `evaluator.py`  implements the evaluation logic (shows which utility functions are called)
   - Task-level `utils.py` (one directory up from the instance, e.g.
     `src/browsergym/knows/eval/tasks/<task_name>/utils.py`)  shared helpers called by the
     evaluator. This file may not exist for every task; skip if absent.

2. For each eval step listed in `checkpoints.md`, assign exactly one **category** from the
    taxonomy below.

3. For each eval step, determine whether an **LLM or VLM is the primary evaluation mechanism**
    (see rules below).

4. If a step does not fit cleanly into one category, **flag it** for human review.

5. Write a JSON output file to `output/taxonomy/<task_name>__<instance>.json`.

---

## Taxonomy Categories

### 1. `visual_check`
The step requires a Vision-Language Model (VLM) as its primary check, or compares an image
    against another image (pixel-level, perceptual hash, or template matching).

**Examples from the benchmark:**
- "The wiki image of Tom Hanks is present on the slide." (slides_30, checkpoint 1)
- "An image from the client's Wikipedia page is found on the left side of the slide."
    (slides_30, checkpoint 2)
- "Logo is the University of Utah's logo" (slides_51, checkpoint 2)
- "Figure 1 images match gold data" (sheets_10)

**Implementation signals in evaluator.py:**
- Calls to `binary_judge_image()`, `image_exact_match()`, `match_image_tiered()`,
    `perceptual_hash_match()`, `verify_image_in_region()`
- Direct VLM model calls asking "Does this image show X?"

### 2. `structural_check`
The step verifies the existence, order, or hierarchy of elements  not their styling or
    position, but whether the right pieces are present and in the right arrangement.

**Examples from the benchmark:**
- "There is a column/row with the name of each stock." (sheets_6, checkpoint 1)
- "'Ingredients' column exists and is first." (sheets_2, checkpoint 1)
- "Macros columns are in alphabetical order." (sheets_2, checkpoint 1)
- "A bar chart was created in the spreadsheet." (sheets_6, checkpoint 4)
- "There are no empty bullet lists." (docs_37, checkpoint 2)
- "No duplicate reference links." (docs_37, checkpoint 3)
- "Subheader exists below the main header" (slides_51, checkpoint 1)
```

```
**Implementation signals in evaluator.py:**
- Column existence checks, row ordering logic
- Checks for presence/absence of elements (charts, headers, bullet lists)
- `match_columns()` calls (checking if columns exist)

### 3. `formatting_check`
The step verifies visual styling properties: bold, italic, font size, cell colors, bullet
    points, highlight colors, text alignment, merged cells, text overflow.

**Examples from the benchmark:**
- "The title of each lecture should be of size 'Heading 3'" (docs_37, checkpoint 1)
- "Group headers are centered and italicized." (sheets_2, checkpoint 2)
- "Column titles are bolded (excluding group headers)." (sheets_2, checkpoint 2)
- "Hyperlinks of references that appear in multiple slides are colored 'dark green 2'"
    (docs_37, checkpoint 4)
- "Event name text has bold font" (slides_51, checkpoint 1)
- "Event name text has font size >= 20" (slides_51, checkpoint 1)
- "Event name text color is black" (slides_51, checkpoint 1)
- "Macro columns share same background color." (sheets_2, checkpoint 3)
- "Values exceeding 10% DV are bolded per FDA guidelines." (sheets_2, checkpoint 7)
- "A shaded right-hand sidebar in grey exists" (slides_51, checkpoint 4)
- "Horizontal line under column titles exists." (sheets_2, checkpoint 2)

**Implementation signals in evaluator.py:**
- `is_cell_bold()`, `is_cell_italic()`, `is_cell_centered()`
- `get_cell_background_color()`, `colors_are_similar()`, `colors_are_distinct()`
- `get_text_style_from_shape()`, font size comparisons
- `get_paragraph_alignment()`

### 4. `spatial_check`
The step validates the physical position or layout of an element on the page/slide/document
    where something appears, how large it is, or its spatial relationship to other elements.

**Examples from the benchmark:**
- "The Tom Hanks image is positioned in the top right area of the slide." (slides_30,
    checkpoint 1)
- "The Tom Hanks image takes up more than 50% of the slide." (slides_30, checkpoint 1)
- "The Wikipedia image and the internet image are approximately symmetric in vertical
    position." (slides_30, checkpoint 2)
- "Logo is inserted in the top right of the slide" (slides_51, checkpoint 2)
- "Logo does not overlap or crowd header/subheader text" (slides_51, checkpoint 2)
- "Footer is positioned at the bottom right of the poster" (slides_51, checkpoint 5)
- "Event name text box appears at the top of the page" (slides_51, checkpoint 1)
- "No text or images are overlapping each other" (slides_51, checkpoint 7)
- "No text or images are off of the slide" (slides_51, checkpoint 7)
- "Background summary is placed in the main, central body of the poster" (slides_51, checkpoint
    3)

**Implementation signals in evaluator.py:**
- `is_text_in_title_position()`, `get_element_bbox()`, `get_slide_dimensions()`
- `is_upper_left()`, `is_upper_right()`, `is_mostly_inside()`
- `get_image_area_percentage_from_api()`
- Bounding box arithmetic, overlap detection

### 5. `content_check`
The step verifies that information came from the correct source, or that content is a direct
    quote, paraphrase, or faithful representation of a specific source. This is about **source
    fidelity**, not numerical accuracy.

**Examples from the benchmark:**
- "Summary clearly tied to Pavel Panchekha's work or field of expertise" (slides_51, checkpoint
    3)
- "Summary is engaging and highlights why the event is worth attending" (slides_51, checkpoint
    3)
- "At least 2 reputable, stable sources are used" (slides_51, checkpoint 3)
- "Speaker's current affiliation is explicitly named and correct according to a reputable
    source" (slides_51, checkpoint 4)
- "3 distinct, verifiable professional achievements are listed" (slides_51, checkpoint 4)
```

- "Each link is categorized correctly i.e. a link that is a blog should be categorized as a blog" (docs_37, checkpoint 3)
- "The name of each link is relevant to the webpage it opens." (docs_37, checkpoint 3)
- "Sources cited correspond to specific claims in the topic summary" (slides_51, checkpoint 6)

Implementation signals in evaluator.py:
- LLM calls that judge relevance, accuracy of claims, or source quality
- `evaluate_with_llm()` for content verification
- Web fetching to verify content against source pages
- `fetch_page_text_content()` used to validate claims against source material

6. `information_retrieval`
The step verifies that data values are correct by comparing against gold/ground-truth data. This is about **data accuracy** the numbers, names, or facts match expected values.

Examples from the benchmark:
- "Each of the 10 stocks matches a top 10 highest market cap stock in tech from the end of Q2 2023" (sheets_6, checkpoint 2)
- "The past price of each stock is correct." (sheets_6, checkpoint 2)
- "The current price of each stock is correct." (sheets_6, checkpoint 2)
- "Carbohydrates values match gold labels within tolerance (7 pts)." (sheets_2, checkpoint 6)
- "Raw Cashews link valid and matches ingredient (via HTML parsing)." (sheets_2, checkpoint 5)
- "The slide number list for each reference matches the gold list of slides" (docs_37, checkpoint 3)
- "The bar chart correctly shows the total value of each stock as a percentage of the overall portfolio." (sheets_6, checkpoint 4)
- "All reference links under each category of each lecture are present." (docs_37, checkpoint 3)
- "Each lecture has an associated title in 'Month/Day: Lecture title' format." (docs_37, checkpoint 1)
- "Event name reads 'Foundations of Web Browser Internals'" (slides_51, checkpoint 1)
- "Subheader contains date (07/27/2026)" (slides_51, checkpoint 1)

Implementation signals in evaluator.py:
- Comparisons against gold CSV/JSON data files
- `numerical_match_with_error()`, `text_exact_match_contained()`, `text_fuzzy_match_contained()`
- `keywords_match_robust()`, `keyword_exact_match()`
- Gold label lookups and tolerance checks
- `validate_chart_values_match()`, `validate_chart_categories_match()`

7. `web_visit_check`
The step checks whether specific URLs appear in the agent's browsing history. **Important:** Even if the implementation fetches page content and uses an LLM to verify that the visited page contains relevant information, the step is still a `web_visit_check` as long as its purpose is verifying browsing history. The LLM in this case is just a smarter URL-matching mechanism the step's intent is "did the agent visit the right page?", not "is the content correct?".

Examples from the benchmark:
- "The browsing history contains a visit to Tom Hanks' Wikipedia page." (slides_30, checkpoint 4)
- "The website trace contains a URL that has information about the top 10 highest market cap stocks in tech from the end of Q2 2023." (sheets_6, checkpoint 3)
- "Recipe URL (rainbowplantlife.com) visited (1 pt)." (sheets_2, checkpoint 8)
- "USDA database URL visited for Raw Cashews (1 pt)." (sheets_2, checkpoint 8)

Implementation signals in evaluator.py:
- Browsing history list iteration
- URL pattern matching against history entries
- `keywords_match_robust()` applied to URL strings
- LLM calls that fetch and analyze page content to verify the visited URL is relevant (this is still `web_visit_check`, not `content_check`)

LLM/VLM Primary Flag Rules

Set `llm_vlm_primary` to `true` ONLY when an LLM or VLM is the **primary** evaluation mechanism for that step meaning the core pass/fail decision depends on the model's judgment, not on deterministic logic.

`true` LLM/VLM is primary
- `binary_judge_image()` VLM makes the yes/no decision
- `evaluate_with_llm()` LLM judges content quality/relevance
- Direct `model(messages)` calls where the model's response determines success
- `verify_image_in_region()` when VLM is used for image identification (not just position)
- `match_image_tiered()` uses VLM as the final tier, but the step's purpose is image verification so VLM is the intended primary method

`false` LLM/VLM is fallback or not used
- `keywords_match_robust()` keyword matching first, LLM only as fallback
- `match_columns()` keyword matching first, LLM fallback for fuzzy column name matching
- `text_exact_match_contained()`, `text_fuzzy_match_contained()` deterministic string operations
- `keyword_exact_match()` pure string matching
- `numerical_match_with_error()` arithmetic comparison
- `image_exact_match()`, `perceptual_hash_match()` deterministic image comparison (no LLM)
- Color, font, position checks all deterministic API data inspection
- Browsing history URL checks string matching

How to determine
1. Find the eval step in `evaluator.py` locate the code that adds the `EvaluationStep` or calls `checkpoint.add_step()`.
2. Trace what function(s) determine the `success` value for that step.
3. If the function is deterministic (string match, numeric comparison, pixel comparison, API property check), set `false`.
4. If the function calls an LLM/VLM and that call's output directly determines pass/fail, set `true`.
5. If there is a tiered approach (try deterministic first, fall back to LLM), set `false` the primary mechanism is deterministic.
6. When in doubt, check the task-level `utils.py` to trace helper function implementations.

Handling Ambiguous Steps

Some eval steps may not fit cleanly into a single category. Common ambiguities:

- **Structural vs. Information Retrieval**: "Column X exists" is structural. "Column X contains the right values" is information retrieval. If a step checks both existence AND correctness, categorize by its primary purpose.
- **Formatting vs. Spatial**: "Text is centered" is formatting (alignment property). "Text box is at the top of the page" is spatial (position on canvas). "Text is left-aligned" is formatting.
- **Content vs. Information Retrieval**: "Summary is tied to the speaker's work" is content (source fidelity). "The date is 07/27/2026" is information retrieval (exact value match).
- **Visual vs. Spatial**: "Image is present on the slide" is visual (identifying the image). "Image is in the top right" is spatial (position). If a step checks both ("Image X is in the top right"), categorize as spatial if position is the primary thing being validated, or visual if image identity is the primary thing.

When a step is genuinely ambiguous, add it to the `flagged_steps` array with:
- The checkpoint name
- The step name
- The full step description from checkpoints.md
- Your reason for flagging (e.g., "Could be structural or information_retrieval checks both column existence and value correctness in the same step")

Output Format

For each instance, write a JSON file to `output/taxonomy/<task_name>__<instance>.json`:

```json
```

```
{
  "task_name": "sheets_6_investmenttracker",
  "instance": "instance_1",
  "eval_steps": [
    {
      "checkpoint": "Checkpoint 1",
      "step_name": "Stock Name Column",
      "category": "structural_check",
      "llm_vlm_primary": false
    },
    {
      "checkpoint": "Checkpoint 2",
      "step_name": "Stocks Match Gold Data",
      "category": "information_retrieval",
      "llm_vlm_primary": true
    }
  ],
  "flagged_steps": [
    {
      "checkpoint": "Checkpoint 3",
      "step_name": "Some Ambiguous Step",
      "description": "Full step description from checkpoints.md",
      "reason": "Could be content_check or information_retrieval  verifies data but also checks
     source"
    }
  ]
}
```

Field details:
- `checkpoint`: The checkpoint header (e.g., "Checkpoint 1", "Checkpoint 2")
- `step_name`: A short name for the step. Use the step description from checkpoints.md,
     truncated to a concise label if it's long.
- `category`: Exactly one of: `visual_check`, `structural_check`, `formatting_check`,
     `spatial_check`, `content_check`, `information_retrieval`, `web_visit_check`
- `llm_vlm_primary`: `true` or `false` per the rules above
- `flagged_steps`: Array of steps you couldn't cleanly categorize. Can be empty.

Process

1. **Work through each task one at a time.** For each task, process all 5 instances.

2. **Treat each instance independently.** Do not assume that instances of the same task have
     identical checkpoints some tasks have slight variations across instances (different
     numbers of items, different point values, different step counts).

3. **For each instance:**
   a. Read `checkpoints.md` to get the list of eval steps and their descriptions.
   b. Read `evaluator.py` to understand how each step is implemented.
   c. Read the task-level `utils.py` (if it exists) to trace any helper functions.
   d. For each eval step, assign a category and LLM/VLM flag.
   e. Write the JSON output file.
   f. **Verify step count**: The number of entries in your `eval_steps` array MUST exactly
     match the number of bullet points under `### Outcome Evaluation` sections in
     `checkpoints.md`. Count only bullets under Outcome Evaluation headers do NOT count
     bullets under `### Eval Template(s)` or other sections. If your count doesn't match,
     re-read the checkpoints.md and find the steps you missed or double-counted.

   Warning: Different instances of the same task can have different numbers of eval steps
     (e.g., one instance may have 3 contact fields while another has 4). Do not copy step
     counts from one instance to another always count from the actual checkpoints.md.

4. **After processing all instances of a task**, briefly report:
   - How many total steps were categorized
   - How many steps were flagged as ambiguous
   - Any patterns or issues you noticed
```
```

5. **After processing ALL tasks**, provide a summary:
   - Total steps categorized across all 110 instances
   - Total flagged steps
   - List all flagged steps grouped by task, so I can review them and tell you how to resolve them

---

Task Directory Structure

All tasks are under: `src/browsergym/knows/eval/tasks/`

```
<task_name>/
  utils.py              (shared helpers  may not exist)
  instance_1/
    checkpoints.md
    evaluator.py
    task.md
    id.txt
    data/               (gold data files)
  instance_2/
    ...
  instance_3/
    ...
  instance_4/
    ...
  instance_5/
    ...
```

The 22 tasks are:
- docs_1_formal_letter
- docs_5_influential_papers
- docs_11_personal_recipe_ocr
- docs_31_education_lesson_plan
- docs_37_reference_list
- sheets_2_personal_recipe_foodcomposition
- sheets_6_investmenttracker
- sheets_7_running_analysis
- sheets_10_paper_sorting
- sheets_25_skitourplan
- sheets_28_personal_travel_planner
- sheets_38_apartment_finder
- sheets_45_Personal_WeddingPlanner_weddingcolorpallette
- sheets_55_Movie_Recommendation
- slides_17_removeimagesaddplaceholders
- slides_20_Illustrated_Book_Report
- slides_26_basic_educational_slide_deck
- slides_29_buy_car_pres
- slides_30_Work_Wikipedia_Photos
- slides_39_Personal_Lookbook_PaintColors
- slides_42_personal_none_product_comparison
- slides_51_event_announcement_poster

Output directory: `output/taxonomy/`