HyWe: Unifying Search and Operation for Real-World Hybrid Web Tasks

Anonymous authors
Paper under double-blind review

Abstract

Real-world hybrid web tasks need both information gathering and page interaction, yet search-based and operation-based agents are developed and evaluated separately. To bridge this gap, we introduce HyWe, a framework that unifies both capabilities within one model and trajectory. Its two modes use dedicated prompts and tool sets to sustain sub-goal execution; explicit switching selects the needed capability, and shared history lets each mode build on the other's findings. To provide demonstrations of these behaviors, we develop HyWeCollector, a fully automatic pipeline that uses task prototypes, four switching patterns, and multi-agent synthesis to produce verified hybrid trajectories. We train HyWe-35B on these demonstrations to unify efficient information gathering and browser interaction in a single 35B web agent. To evaluate hybrid task completion, we introduce HyWeBench: 100 human-written tasks on websites distinct from training, with human-written sub-goal checklists and an outcome-based evaluation pipeline. Hybrid training enables HyWe-35B to achieve 47.0% success on this benchmark, improving on the untrained base model by 19.0 percentage points and outperforming existing search-based and operation-based agents. Even before training, the same base model solves 28.0% of tasks with HyWe, versus 11.0% with all tools under one prompt. Beyond hybrid tasks, HyWe-35B achieves 59.0% and 88.3% success on Online-Mind2Web and WebVoyager, outperforming existing open-source baselines and cutting action cost per successful task by 48.3% and 51.9% compared with Fara-7B.

1 Introduction

LLM agents have made substantial progress on web tasks, with approaches broadly falling into two groups. (i) Search agents use search and page-reading APIs to gather information efficiently across websites (OpenAI, 2025; Tongyi DeepResearch Team, 2025). (ii) Operation agents observe HTML or screenshots and interact with browsers through clicking and typing (He et al., 2024; Qin et al., 2025). In practice, however, many real-world web tasks are hybrid, and both types of agents struggle to solve them. Figure 1 illustrates a task that requires both finding the Bible verse beginning "*Blessed are the poor in spirit*" and playing its American Sign Language video on Deaf Bible. Search agents quickly find the verse but cannot play the video because their tools lack page interaction; operation agents face long browser navigation to locate it. Existing benchmarks also separate these task types: BrowseComp (Wei et al., 2025) and DeepSearchQA (Gupta et al., 2026a) focus on cross-website information gathering, while Online-Mind2Web (Xue et al., 2025) and WebVoyager (He et al., 2024) typically focus on operations within one website, leaving hybrid task completion insufficiently evaluated.

To quantify the limitations of existing search and operation agents, we sample real-world tasks from InSTA (Trabucco et al., 2025) and probe capability and efficiency. For capability, the search agent recovers only about 40% of facts obtained through browser interaction (Section 3.2). For efficiency, the search agent solves 29% of the sampled tasks within four steps, compared with 6% for the operation agent (Appendix B). **Search agents gather information faster; operation agents use browser actions to achieve required page states or obtain information search cannot provide.**

To unify these complementary capabilities, we introduce HyWe, a framework for general-purpose web agents that organizes execution around completing sub-goals. An agent should gather enough evidence or complete the required interaction for each sub-goal, while using earlier findings to guide

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

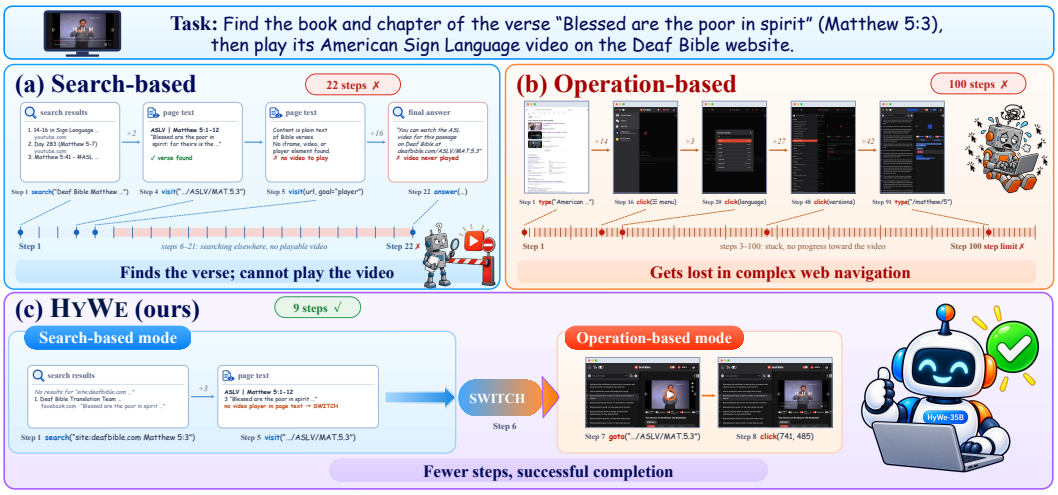

Figure 1: One real-world hybrid task from HyWeBench, three agents; key steps are taken from their actual trajectories. (a) A search agent finds the verse but cannot play the video. (b) An operation agent gets lost in complex web navigation and reaches the 100-step limit. (c) HyWe-35B finds the verse with search tools, switches to operation tools, and plays the video in 9 steps.

the remaining work. The same model works in a search mode or an operation mode. Mode-specific prompts and isolated tool sets guide evidence gathering, page interaction, and recovery from failed attempts, helping the model focus on the current sub-goal. An explicit SWITCH action makes changing capabilities a deliberate decision: another mode may be needed to finish the current sub-goal or pursue the next one more efficiently. A unified action–observation format and shared history carry earlier findings forward so subsequent work builds on existing progress. Together, these designs help the agent complete sub-goals and choose when to change modes. Compared to exposing all tools under one prompt, our framework improves information and operation sub-goal completion from 36.9% and 15.8% to 74.8% and 43.7%, respectively, using the same base model (Section 4.4).

To learn how to complete sub-goals and switch between modes, we develop HyWeCollector, a fully automatic pipeline for collecting high-quality hybrid trajectories on real websites. (i) Task prototypes describe reusable sub-goals; (ii) four switching patterns capture capability limits, efficiency, and dependencies between sub-goals; and (iii) multi-agent trajectory synthesis coordinates task generation, execution, phase control, and review to produce verified examples of hybrid trajectories. We use these examples to train HyWe-35B. As a result, it achieves higher success rates and greater efficiency on operation tasks: 59.0% and 88.3% success on Online-Mind2Web and WebVoyager, using 48.3% and 51.9% fewer actions per successful task than Fara-7B (Awadallah et al., 2025). It also maintains strong deep search capabilities, with 64.0% accuracy on xbench-DeepSearch-2510 (Section 4.5).

We introduce HyWeBench to evaluate real-world hybrid task completion: 100 high-quality, human-written tasks on websites distinct from those used for training, human-written *info*/*op* checklists, and a corresponding evaluation pipeline. Each task includes at least one *op* sub-goal; the judge verifies outcomes from execution evidence regardless of the mode used. HyWe-35B achieves 47.0% success, up from 28.0% before training, and outperforms search and operation agents, including XYZ-Aquila-mini (22.0%) and Fara-7B (13.0%). Its success rate is comparable to Kimi-K3 (46.0%) and Qwen3.8-Max (44.0%), with $1/80$ and about $1/69$ of their total parameters, respectively (Section 4.3).

Our contributions are as follows.

(i) HyWe. A framework unifying search and operation capabilities through focused sub-goal execution, explicit mode switching, and shared evidence.

(ii) HyWeCollector. A fully automatic pipeline using task prototypes, switching patterns, and multi-agent synthesis to collect high-quality examples of hybrid trajectories.

(iii) HyWeBench. Human-written hybrid tasks and *info*/*op* checklists, with an evaluation pipeline.

(iv) HyWe-35B. Higher success and efficiency on operation tasks, and substantially better hybrid task performance than existing search and operation agents.

2 Related Work

Search-based agents and benchmarks. Search-o1 (Li et al., 2025a) and WebThinker (Li et al., 2025b) integrate web search into multi-step reasoning. Search-R1 (Jin et al., 2025) and DeepResearcher (Zheng et al., 2025) train search use through reinforcement learning, while WebExplorer (Liu et al., 2025) and Tongyi DeepResearch (Tongyi DeepResearch Team, 2025) scale training with synthetic information-seeking data. These methods focus on finding and reasoning over web information. HyWe unifies search-based tools with operation-based actions and observations in one trajectory, allowing the model to also obtain information produced by page interaction. For example, it can use retrieved requirements to configure a product and read the resulting price. HyWeCollector provides demonstrations of when to choose each mode and how to pass information between them.

Benchmarks used to evaluate search-based agents cover several aspects of information gathering. GAIA (Mialon et al., 2023) evaluates tool-assisted reasoning, BrowseComp (Wei et al., 2025) targets hard-to-find facts, and SealQA (Pham et al., 2025) tests reasoning with noisy or conflicting search results. DeepSearchQA (Gupta et al., 2026a) further measures whether agents retrieve complete answer sets. These benchmarks mainly grade final answers against reference answers, which does not establish whether an agent performed the page actions needed for a hybrid task. HyWeBench provides human-written tasks on real websites where retrieved information guides page actions, or interaction results guide further search. Its human-written checklists label information-gathering goals as *info* and operation goals as *op*, with at least one *op* sub-goal in every task. Our evaluator checks both the facts obtained and the actions completed.

Operation-based agents and benchmarks. SeeAct (Zheng et al., 2024) uses HTML and screenshots to ground browser actions, while UI-TARS (Qin et al., 2025) learns screenshot-based interaction directly. UI-TARS-2 (Wang et al., 2025a) further adds terminal and file-system tools. OpenCUA (Wang et al., 2025b) scales training with human demonstrations, while InSTA (Trabucco et al., 2025) and NNetNav (Murty et al., 2024) generate browser trajectories through website interaction. Our method addresses the cost of gathering information through repeated browser navigation. HyWe lets the model use search-based tools to gather information across websites and operation-based tools for page interaction, with a shared history. A single search can then replace several site visits when the required facts are searchable. HyWeCollector trains these mode choices through four switching patterns grounded in capability probes.

Operation-based benchmarks differ in their environments and evaluation. Mind2Web (Deng et al., 2023) evaluates action prediction against recorded demonstrations, while WebArena (Zhou et al., 2024) checks task completion on self-hosted websites. Online-Mind2Web (Xue et al., 2025) and WebVoyager (He et al., 2024) provide tasks on live websites and use models to judge browser trajectories. Hybrid evaluation also needs evidence from search-based tools, which may never appear in browser screenshots. HyWeBench tests tasks that require both information gathering and page interaction on real websites. Our evaluator uses text tool observations and screenshots to verify each sub-goal in the human-written checklists. This supports agents with different tool sets and distinguishes completed work from unsupported claims in the final answer.

3 Method

Our method connects sub-goal execution, mode selection, and evidence transfer within one trajectory (Figure 2). HyWe includes mode-specific execution, mode switching, and shared context (Section 3.1); HyWeCollector collects demonstrations for this framework (Section 3.2); and supervised fine-tuning produces HyWe-35B (Section 3.3).

3.1 HyWe framework

To sustain progress on hybrid tasks, HyWe (Figure 2) guides execution within each mode, lets the model switch between search-based and operation-based modes as needed, and carries evidence

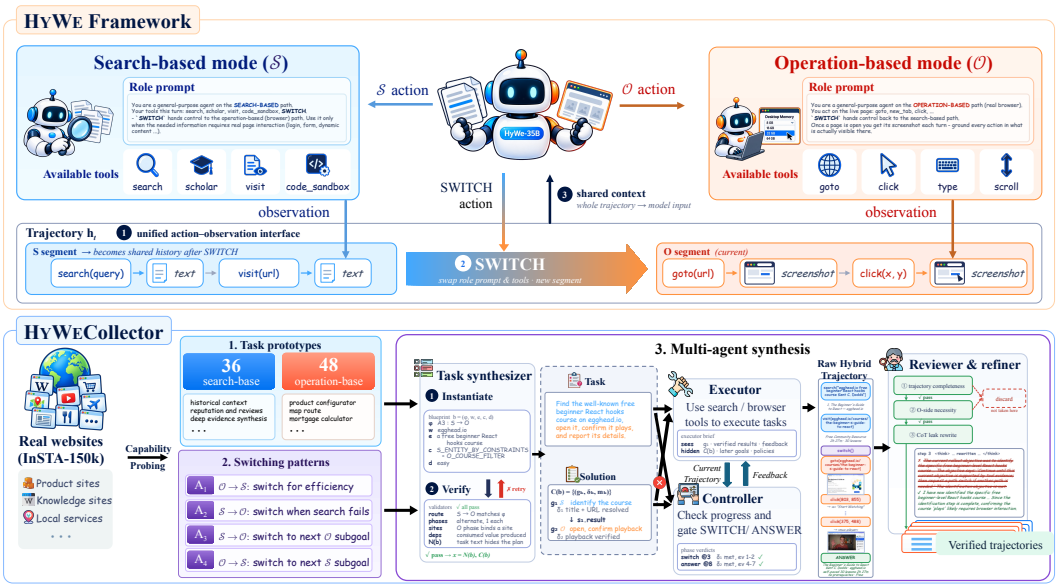

Figure 2: Overview of the HyWe framework. Mode-specific prompts and tools guide sub-goal execution; the SWITCH action lets the model move between search-based and operation-based modes as needed, and shared history carries evidence forward (Section 3.1). The HyWeCollector data pipeline supplies task prototypes, switching patterns, and website/entity bindings to a multi-agent framework for task synthesis, execution, control, and review (Section 3.2). Colors mark segments: consecutive steps in one mode, separated by mode switches. Prompts and tool calls are abbreviated.

across modes. Building on ReAct (Yao et al., 2022), one model reasons over observed progress to continue, switch, or answer (Algorithm 1).

Formulation. A sub-goal specifies a required fact or page outcome. A segment groups consecutive steps in one mode; each executed SWITCH ends the current segment and starts the next. A sub-goal can span segments, and a segment can advance several sub-goals. For task x, model π_θ produces trajectory $\tau = (o_0, a_1, o_1, \ldots, a_T)$, choosing action a_t in mode $m_t \in \{\mathcal{S}, \mathcal{O}\}$ and receiving observation o_t:

$$\mathcal{A}(m_t) = \mathcal{A}_{m_t} \cup \{\text{SWITCH}, \text{ANSWER}(y)\}, \tag{1}$$

$$a_t \sim \pi_\theta\big(\cdot \mid \mathcal{R}(h_t, m_t)\big), \qquad h_t = (x, (m_i, a_i, o_i)_{i<t}). \tag{2}$$

Here $\mathcal{A}_{m_t}$ contains the active mode's tools (Appendix I, Table 10), h_t is the raw history, and $\mathcal{R}$ renders the model's context.

Mode-specific execution. Each mode exposes its own prompt and tools to focus execution on the current sub-goal. Search-based guidance asks the model to verify facts from source pages and revisit unchecked constraints; operation-based guidance asks it to verify page changes and recover from failed actions. Both prompts require assessing remaining work and checking evidence before answering (Appendix K.1). A common step format (m_t, a_t, o_t) lets the model reason over text search results and browser screenshots within one trajectory.

Explicit mode switching. The model can switch when another mode supplies a missing capability or advances a sub-goal more efficiently, including when earlier results enable a dependent sub-goal. It states the reason and calls the argument-free SWITCH action, which changes m_t to $\bar{m}_t$, activates the other prompt and tool set, and starts a new segment without an environment action. At each step, the model decides whether to continue in the current mode, switch modes, or answer.

Shared context across modes. Later sub-goals may depend on entities, constraints, or results obtained earlier. To preserve this evidence, steps preceding the current segment are stored in a shared history block $H_{<s(t)}$, where $s(t)$ starts the current segment. The model's context is

$$\mathcal{R}(h_t, m_t) = \Big[S_{m_t} \; ; \; U\big(x, H_{<s(t)}\big) \; ; \; (a_i, o_i)_{i=s(t)}^{t-1}\Big]. \tag{3}$$

Algorithm 1 HyWe inference loop

Require: task x; initial mode m_1 (start URL if $m_1 = \mathcal{O}$); step budget $T_{\max}$

$h_1 \leftarrow (x)$; $s(1) \leftarrow 1$
for $t = 1, \ldots, T_{\max}$ **do**
  $c_t \leftarrow \mathcal{R}(h_t, m_t)$ ▷ render context: Eq. (3)
  $(r_t, a_t) \sim \pi_\theta(\cdot \mid c_t), \quad a_t \in \mathcal{A}(m_t)$ ▷ assess progress; choose one action
  if $a_t = \text{ANSWER}(y)$ **then return** y
  if $a_t = \text{SWITCH}$ **then**
    $o_t \leftarrow$ hand-off notice; $m_{t+1} \leftarrow \bar{m}_t$; $s(t{+}1) \leftarrow t{+}1$
  else
    $o_t \leftarrow \text{Env}_{m_t}(a_t)$; $m_{t+1} \leftarrow m_t$; $s(t{+}1) \leftarrow s(t)$
  end if
  $h_{t+1} \leftarrow h_t \oplus (m_t, a_t, o_t)$ ▷ append the observed step
end for
return forced answer ▷ only ANSWER is admitted at the budget limit

Here S_{m_t} is the mode-specific prompt and U holds the task and history. On a switch, completed steps enter the history block with their modes, actions, reasoning, and observations. The next segment thus uses earlier evidence to assess unfinished work and continue from existing progress. Context compression and screenshot handling are detailed in Appendix I.3.

3.2 HyWeCollector trajectory synthesis pipeline

To our knowledge, no open-source work has unified search-based and operation-based trajectories for web tasks. We develop HyWeCollector (Figure 2) to collect demonstrations for HyWe through three fully automatic stages: abstracting reusable sub-goals, designing switching patterns, and coordinating agents to synthesize tasks and verified trajectories on real websites.

Stage 1: task prototypes from capability probing. To identify sub-goals suited to each mode, an operation-based exploration agent visits InSTA websites (Trabucco et al., 2025) and an extractor records facts such as prices and filtered counts. A search-based agent tries to recover them within budget B. For the stable, reliable fact pool F, a judge defines

$$\ell(f) = \mathbf{1}\big[\text{Probe}_{\mathcal{S}}(f; B) \text{ recovers } f\big], \quad F_{\mathcal{S}} = \{f \in F : \ell(f) = 1\}, \quad F_{\mathcal{O}} = F \setminus F_{\mathcal{S}}. \tag{4}$$

Of all collected facts, 40% enter $F_{\mathcal{S}}$ and 31% enter $F_{\mathcal{O}}$: 18% depend on interaction and 13% reflect probe limits. From $F_{\mathcal{S}}$ and $F_{\mathcal{O}}$, we abstract 36 search-based and 48 operation-based task prototypes (Appendix L), such as looking up requirements or configuring a product. Each prototype specifies a sub-goal and typed inputs and outputs for linking it to later sub-goals. Website and entity bindings ground these prototypes during task synthesis.

Stage 2: switching pattern design. Our pilot study (Appendix B) shows that search-based agents gather information faster, with low token costs even on failed attempts, while operation-based agents can solve tasks that search-based agents cannot. This motivates two patterns that change capabilities while pursuing the same goal: A_1, $\mathcal{O} \to \mathcal{S}$ for efficiency, and A_2, $\mathcal{S} \to \mathcal{O}$ after a bounded search attempt fails. Two further patterns connect dependent sub-goals: A_3, $\mathcal{S} \to \mathcal{O}$ with search results guiding page interactions, and A_4, $\mathcal{O} \to \mathcal{S}$ with interaction results guiding further search. Stage 3 instantiates these patterns using the prototypes and site bindings from Stage 1 (Appendix D).

Stage 3: multi-agent trajectory synthesis. Four agent roles turn these sub-goals and switching patterns into demonstrations of sustained execution and evidence transfer.

(i) *Task synthesizer.* It composes prototypes with matching input/output types, binds them to real sites and entities, and builds a task blueprint and private plan:

$$b = (\phi, w, e, c, d), \qquad C(b) = \big((g_k, \delta_k, m_k)\big)_{k=1}^{K}. \tag{5}$$

Here ϕ is the switching pattern, w the target sites, e the entity, c the prototype composition, and d the difficulty. Phase k specifies goal g_k, condition δ_k for ending the phase, and mode m_k. The synthesizer writes $x = N(b)$ without revealing the plan; validators check site bindings and dependencies (Appendix J.1).

(ii) *Executor.* It acts through HYWE using the current goal, verified earlier results, and feedback without plan details; the full plan remains hidden.

(iii) *Controller.* It checks δ_k against observed progress, approving a switch when the phase condition is met and an answer after all phases finish. For A_2, the condition is a failed bounded search attempt. Execution and control are expressed as

$$a_t \sim \pi_E\big(\cdot \mid x, g_k, \beta, \sigma(u), \tau_{<t}\big), \qquad v_k = \nu(\delta_k, \tau_k). \tag{6}$$

Here π_E is the executor, β holds verified earlier results, and $\sigma(u)$ removes plan details from controller feedback u. The executor observes history $\tau_{<t}$; the controller ν evaluates phase evidence τ_k and returns a binary verdict v_k. Their outputs are a raw trajectory τ and a feedback record.

(iv) *Reviewer & refiner.* This role verifies task completion independently of the planned route, using the feedback record to identify steps L containing guidance leakage. Refinement rewrites these steps without changing facts or decisions, followed by acceptance checks:

$$\widetilde{\tau} = \mathrm{Refine}(\tau, L), \qquad \mathrm{Keep}(\widetilde{\tau}) = \mathrm{Success}(x, \widetilde{\tau}) \wedge \mathrm{Valid}(\widetilde{\tau}, C(b)). \tag{7}$$

$\mathrm{Success}$ checks task completion; Valid checks evidence support, required interactions, switching conditions, and residual leakage.

In all, we probe facts from $1{,}110$ real websites and construct $35{,}784$ distinct tasks. We run the multi-agent framework on $25{,}662$ of these tasks and retain $3{,}735$ trajectories after review, refinement, and deduplication. Appendix F provides the full pipeline (Algorithm 2) and detailed executor–controller interaction (Algorithm 3).

3.3 TRAINING

Hybrid training samples. We convert the $3{,}735$ trajectories collected by HYWECOLLECTOR into $8{,}055$ multi-turn training samples. We split each trajectory τ at executed switches and construct one sample per segment using the context renderer $\mathcal{R}$ in Eq. (3). At each supervised step t, the model receives $\mathcal{R}(h_t, m_t)$, with earlier segments stored in the history block $H_{<s(t)}$; only assistant turns in the current segment are supervised. Thus, each trajectory yields

$$|\mathcal{D}(\tau)| = n_{\mathrm{sw}}(\tau) + 1, \tag{8}$$

where $\mathcal{D}(\tau)$ is the set of exported training samples and $n_{\mathrm{sw}}(\tau)$ the number of executed switches. This jointly supervises work within a mode and decisions to SWITCH or ANSWER, conditioned on evidence from earlier segments.

Search-based training data. To strengthen search-based capabilities, we supplement the hybrid samples with the $7{,}000$ public search-based trajectories of XYZ Agentic Team (2026), adapting their tool names, observations, and prompts to HYWE. The hybrid trajectories already contain operation-based interactions, so we use no additional operation-based training corpus.

Both sources use the inference context format. We fine-tune the language-model parameters of Qwen3.6-35B-A3B (Qwen Team, 2026) with next-token cross-entropy, keeping the vision encoder and projector frozen. Sample processing and training settings are in Appendix G.

4 EXPERIMENTS

4.1 SETUP

Benchmarks. We evaluate operation-based success and efficiency on Online-Mind2Web (Xue et al., 2025) and WebVoyager (He et al., 2024), hybrid task completion on HYWEBENCH, and deep search on HLE (Phan et al., 2025), xbench-DeepSearch-2510 (Chen et al., 2025), and BrowseComp-EN (Wei et al., 2025) (Appendix C).

Baselines. We compare with open-source search-based and operation-based agents and published frontier results. On HYWEBENCH, frontier models also run within HYWE. The Qwen3.6-35B-A3B base model (Qwen Team, 2026) provides training, framework, and mode controls. Baseline details are in Appendix H.

4.2 SUCCESS AND EFFICIENCY ON OPERATION-BASED BENCHMARKS

Table 1 compares success, efficiency, and switching. Baseline success rates are published results; efficiency is measured on our re-runs (Appendix C.4).

Table 1: Success and efficiency on Online-Mind2Web and WebVoyager. SR: task success rate (%); Steps/SR: total actions (including failures) per success in our re-runs, using separate success labels from the SR column; Switch: tasks with a mode switch (%). Bold/underline: best/second-best SR and Steps/SR among open-source and HyWe rows. Small values: changes from base (rates in percentage points); green/red/gray: improvement/decline/switch-rate changes. ↑/↓: higher/lower is better; –: unreported.

Model / system	Size	Online-Mind2Web			WebVoyager		
		SR↑	Steps/SR↓	Switch	SR↑	Steps/SR↓	Switch
Frontier LLMs							
GPT-5.5	–	74.7	–	–	85.1	–	–
GPT-5 (SoM)	–	57.7	–	–	90.6	–	–
o3 (SoM)	–	55.4	–	–	79.3	–	–
Claude-Opus-4.6	–	–	–	–	88.0	–	–
Seed2.1-Pro	–	59.2	–	–	70.6	–	–
GLM-5V-Turbo	–	–	–	–	88.5	–	–
Kimi-K2.6	–	–	–	–	76.8	–	–
Qwen3.8-Max	2.4T-A95B	68.9	–	–	77.8	–	–
Gemini 2.5 Computer Use	–	57.3	–	–	–	–	–
OpenAI Operator	–	58.3	–	–	87.0	–	–
Yutori Navigator (n1)	–	64.7	–	–	–	–	–
Opensource operation-based agents							
Fara-7B	7B	34.1	61.5	0	73.5	34.9	0
UI-TARS-1.5-7B	7B	31.3	417.1	0	66.4	194.1	0
GUI-Owl-1.5-8B-Thinking	8B	48.6	207.5	0	78.1	87.3	0
GUI-Owl-1.5-32B-Thinking	32B	–	–	–	<u>82.1</u>	59.5	0
MolmoWeb-8B	8B	35.3	149.8	0	78.2	102.2	0
HyWe Framework							
Qwen3.6-35B-A3B (base)	35B-A3B	<u>51.7</u>	**30.9**	37.3	76.8	<u>20.8</u>	20.8
HyWe-35B	35B-A3B	**59.0** +7.3	<u>31.8</u> +0.9	93.7 +56.4	**88.3** +11.5	**16.8** −4.0	87.9 +67.1

Success and efficiency. HyWe-35B leads the evaluated open-source agents with 59.0% and 88.3% success on Online-Mind2Web and WebVoyager. Even before training, the base model in HyWe uses fewer actions per success than every listed operation-based agent: 30.9 and 20.8 Steps/SR, versus at least 61.5 and 34.9. Training raises success by 7.3 and 11.5 points, lowers WebVoyager Steps/SR to 16.8, and keeps Online-Mind2Web efficiency comparable (31.8). Thus, the success gains retain the framework's advantage in execution cost, including actions spent on failed tasks.

How switching helps. Search avoids long browser navigation for retrievable facts (A_1); switching back completes interaction-dependent goals (A_2). All trajectories start in $\mathcal{O}$. On Online-Mind2Web/WebVoyager, training raises switch rates from 37.3%/20.8% to 93.7%/87.9% and mean switches from 0.44/0.23 to 2.01/1.18. Still, 79%/95% of HyWe-35B trajectories switch at most twice: 48%/71% follow $\mathcal{O} \rightarrow \mathcal{S}$ and 25%/12% follow $\mathcal{O} \rightarrow \mathcal{S} \rightarrow \mathcal{O}$. These patterns indicate focused use of search and page interaction with limited back-and-forth.

4.3 Hybrid capability

HyWeBench. Our benchmark contains 100 high-quality, human-written hybrid tasks on websites distinct from training, paired with human-written completion checklists. Items specify information to obtain (*info*) or interactions and page states to achieve (*op*); every task includes at least one *op* item. Task success requires completing all items.

Evaluation protocol. An LLM judge verifies each item's outcome from text observations and screenshots, regardless of the tools or mode used. All systems share the same tasks, budget, and checklists, keeping evaluation fair across agent types. Checklist statistics and scoring details are in Appendix C.1.

Comparison with existing single-mode agents. HyWe-35B achieves 47.0% success, versus 22.0% for XYZ-Aquila-mini and 13.0% for Fara-7B (Table 2), more than doubling the best single-mode result. It also leads the single-mode agents on both information (82.2%) and operation (68.8%)

Table 2: Performance on HYWEBENCH. $\mathcal{S}$/$\mathcal{O}$ denote search-based/ operation-based modes. Success: tasks with all checklist items achieved (%); Info. and Op.: achieved information and operation checklist items (%), respectively. Sw.: mean switches per task; %Sw.: tasks with at least one switch (%). Bold/underlined values mark the best/second-best scores; ↑: higher is better; –: unavailable or inapplicable.

System	Size	Mode	Success↑	Info.↑	Op.↑	Sw.	%Sw.
SEARCH-BASED AGENTS							
XYZ-Aquila-mini (XYZ Agentic Team, 2026)	35B-A3B	$\mathcal{S}$	22.0	71.2	26.7	–	–
Agents-A1 (Bai et al., 2026)	35B-A3B	$\mathcal{S}$	13.0	70.9	23.1	–	–
OPERATION-BASED AGENTS							
GUI-Owl-1.5 (Xu et al., 2026)	32B	$\mathcal{O}$	7.0	34.3	20.2	–	–
Fara-7B (Awadallah et al., 2025)	7B	$\mathcal{O}$	13.0	37.5	35.2	–	–
MolmoWeb (Gupta et al., 2026b)	8B	$\mathcal{O}$	2.0	11.7	9.7	–	–
UI-TARS-1.5 (Qin et al., 2025)	7B	$\mathcal{O}$	6.0	24.6	21.1	–	–
HYWE Framework							
Kimi-K3 (zero-shot)	2.8T	$\mathcal{S}+\mathcal{O}$	<u>46.0</u>	**83.2**	<u>64.4</u>	0.57	57
Qwen3.8-Max (zero-shot)	2.4T	$\mathcal{S}+\mathcal{O}$	44.0	77.3	62.3	0.59	54
Qwen3.7-Plus (zero-shot)	–	$\mathcal{S}+\mathcal{O}$	33.0	77.0	51.8	0.44	42
Qwen3.6-35B-A3B (base)	35B-A3B	$\mathcal{S}+\mathcal{O}$	28.0	74.8	43.7	0.47	40
HYWE-35B (SFT)	35B-A3B	$\mathcal{S}+\mathcal{O}$	**47.0**	<u>82.2</u>	**68.8**	2.10	99

Table 3: Framework and mode ablations on HYWEBENCH (all 100 tasks). Mode: available tools ($\mathcal{S}$ search-based, $\mathcal{O}$ operation-based). Succ.: tasks with all sub-goals achieved (%); Info./Op.: completion rates of information/operation sub-goals (%). Bold: best base-model scores within each panel; HYWE-35B is a training reference.

System	Mode	Succ.↑	Info.↑	Op.↑
(a) Framework design				
Base model, naive merging	$\mathcal{S}+\mathcal{O}$	11.0	36.9	15.8
Base model, HYWE	$\mathcal{S}+\mathcal{O}$	**28.0**	**74.8**	**43.7**
(b) Modes within HYWE				
Base model	$\mathcal{S}$	7.0	73.8	15.8
Base model	$\mathcal{O}$	26.0	57.6	**47.8**
Base model	$\mathcal{S}+\mathcal{O}$	**28.0**	**74.8**	43.7
HYWE-35B (SFT reference)	$\mathcal{S}+\mathcal{O}$	47.0	82.2	68.8

sub-goals, turning progress on both sides into more complete hybrid solutions. Training improves task and operation sub-goal completion by 19.0 and 25.1 points, respectively. Its success rate is comparable to Kimi-K3 (46.0%) and Qwen3.8-Max (44.0%), with $1/80$ and about $1/69$ of their total parameters, respectively.

Why search-based agents receive operation credit. The *op* labels specify required outcomes, not mandatory tools. Search-based agents receive credit when HTTP requests, parameterized URLs, or retrieved content provide evidence that the required outcome was achieved.

4.4 FRAMEWORK AND MODE ABLATIONS

Table 3 tests whether mode organization helps hybrid task completion and what each mode contributes, using the same base model.

Framework ablation. We compare HYWE with *naive merging*, which exposes search-based and operation-based tools under one prompt (Table 3, panel a). HYWE raises success from 11.0% to 28.0%. Information and operation completion rise from 36.9%/15.8% to 74.8%/43.7%. These results show that our framework based on mode switching is more effective than simply merging the search and operation action spaces.

Mode ablation within HYWE. The modes have complementary strengths (panel b): search-based mode completes more information sub-goals, and operation-based mode more operation sub-goals. Before training, using both modes raises success only slightly over operation-based mode alone

Table 4: Deep search performance (accuracy, %; higher is better). HLE: Humanity's Last Exam; DS-2510: xbench-DeepSearch-2510; BC-en: BrowseComp-EN. Superscript t marks text-only HLE results; others use the full set; –: unreported.

Model / system	Size	HLE↑	DS-2510↑	BC-en↑
FRONTIER LLMS				
GPT-5.6 Sol	–	58.0	–	90.4
Claude Fable 5	–	64.5	–	88.0
Qwen3.8-Max	2.4T	56.2	–	–
Kimi-K3	2.8T	56.0	–	91.2
Kimi-K2.6	1T	54.0	–	83.2
GLM-5.2	744B	54.7^{t}	–	–
DeepSeek-V4-Pro (Preview)	1.6T	48.2^{t}	–	83.4
OPENSOURCE SEARCH-BASED AGENTS				
Tongyi DeepResearch	30B	32.9^{t}	55.0	43.4
OpenSeeker-v1	30B	–	–	29.5
REDSearcher	30B	34.3	–	57.4
MiroThinker-1.7-mini	30B	36.4^{t}	57.2	67.9
FORT-Searcher	30B	–	57.2	72.2
QUEST-35B	35B	37.2^{t}	–	64.6
Agents-A1	35B	47.6^{t}	–	75.5
XYZ-Aquila-mini	35B	51.1^{t}	–	78.8
HYWE Framework				
Qwen3.6-35B-A3B (base)	35B-A3B	30.5	49.0	59.2
HYWE-35B	35B-A3B	47.8	64.0	74.6

(28.0% vs. 26.0%). Training with high-quality trajectories from HYWECOLLECTOR raises success to 47.0%, substantially improving the model's ability to complete tasks requiring both information gathering and page interaction.

Case study. In the reward-chart task (Appendix A.1), HYWE-35B finds the correct builder through search, then switches to browser interaction and completes all seven sub-goals. The base model under the same framework stops after gathering information; this contrast illustrates how training helps turn retrieved evidence into completed page actions.

4.5 RETENTION OF DEEP SEARCH CAPABILITIES

Alongside higher success on operation-based and hybrid tasks, HYWE-35B substantially improves over the base model on search tasks: from 30.5% to 47.8% on HLE, 49.0% to 64.0% on xbench-DeepSearch-2510, and 59.2% to 74.6% on BrowseComp-EN (Table 4). The agent rarely calls SWITCH, so these evaluations primarily measure deep research in HYWE's search mode (Appendix C.5).

Its 64.0% on xbench-DeepSearch-2510 is the highest among the reported open-source baselines in our comparison. On BrowseComp-EN, HYWE-35B (74.6%) is close to Agents-A1 (75.5%) and above FORT-Searcher (72.2%), showing strong search capability alongside its hybrid gains.

5 CONCLUSION

We presented HYWE, HYWECOLLECTOR, and HYWEBENCH to unify search-based and operation-based capabilities, collect hybrid demonstrations, and evaluate real-world hybrid task completion. The central design is to sustain progress on sub-goals within each mode and carry evidence forward when capabilities change. Framework ablations show that this organization improves completion over exposing all tools together; hybrid training further strengthens their joint use. The resulting HYWE-35B outperforms the evaluated search-based and operation-based agents on hybrid tasks, achieves higher success and efficiency on operation-based tasks, and preserves deep search capabilities. Our ablations assess the framework as a whole, and the capability probe depends on its search tools and budget. Isolating individual design contributions and improving robustness to changing websites remain open directions.

AI USE STATEMENT

AI tools assisted with manuscript preparation, language polishing, literature search, LaTeX formatting, and code debugging. Their use in data synthesis and evaluation is described in Section 3.2 and Appendix C. The authors take full responsibility for the accuracy, originality, and integrity of the final manuscript and research artifacts.

ETHICS STATEMENT

Our data pipeline and evaluations interact with live, publicly accessible websites. Exploration, probing, and rollouts perform ordinary browsing interactions and do not create accounts, make purchases, submit personal information, or otherwise durably alter site state. We have checked the collected data for personal information; the released data contains no personal information. The fact pool contains public facts such as prices, counts, and product attributes. Web agents that can both research and operate websites are dual-use; our release targets research reproducibility, and our operation-based mode training data derives from audited synthetic hybrid trajectories on public sites rather than from interactions with authenticated or private services.

REPRODUCIBILITY STATEMENT

The inference harness—mode-specific tool sets, output protocol, and segmented context management with exact folding watermarks—is specified in Section 3.1, with sampling parameters and seed derivation given in Appendix I.2. The data pipeline, including the switching patterns, naturalization validators, dual-blind phase control, audit criteria, and the export invariant, is specified in Section 3.2, and training details in Section 3.3 and Appendix G. Appendix C specifies benchmark coverage, judge prompts, and scoring; all of our reported results use full benchmark sets. Live websites can change, so identical seeds do not guarantee identical observations. We will release all code and research artifacts, including the framework, collection and evaluation pipelines, task blueprints, training data, fine-tuned checkpoints, and the human-written HYWEBENCH tasks and info/op checklists.

REFERENCES

Ahmed Awadallah, Yash Lara, Raghav Magazine, Hussein Mozannar, Akshay Nambi, et al. Fara-7B: An efficient agentic model for computer use. *arXiv preprint arXiv:2511.19663*, 2025.

Lei Bai, Zongsheng Cao, Yang Chen, Zhiyao Cui, Shangheng Du, Yue Fan, Shiyang Feng, Zijie Guo, Haonan He, Liang He, et al. Scaling the horizon, not the parameters: Reaching trillion-parameter performance with a 35B agent. *arXiv preprint arXiv:2606.30616*, 2026.

ByteDance UI-TARS Team. UI-TARS-1.5: Open-source multimodal agent. `https://github.com/bytedance/UI-TARS`, 2025. Model release, April 2025.

Kaiyuan Chen, Yixin Ren, Yang Liu, Xiaobo Hu, Haotong Tian, Tianbao Xie, Fangfu Liu, et al. xbench: Tracking agents productivity scaling with profession-aligned real-world evaluations. *arXiv preprint arXiv:2506.13651*, 2025.

Zheng Chu, Xiao Wang, Jack Hong, Huiming Fan, Yuqi Huang, Yue Yang, Guohai Xu, Chenxiao Zhao, Cheng Xiang, Shengchao Hu, Dongdong Kuang, Ming Liu, Bing Qin, and Xing Yu. RED-Searcher: A scalable and cost-efficient framework for long-horizon search agents. *arXiv preprint arXiv:2602.14234*, 2026.

Jia Deng, Yimeng Chen, Xiaoqing Xiang, Ziyang Zeng, Shuo Tang, Wayne Xin Zhao, Feng Chang, Chuan Hao, Yuan Wei, Ran Tao, Bryan Dai, and Ji-Rong Wen. FORT-Searcher: Synthesizing shortcut-resistant search tasks for training deep search agents. *arXiv preprint arXiv:2606.12087*, 2026.

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

Yuwen Du, Rui Ye, Shuo Tang, Xinyu Zhu, Yijun Lu, Yuzhu Cai, and Siheng Chen. OpenSeeker: Democratizing frontier search agents by fully open-sourcing training data. *arXiv preprint arXiv:2603.15594*, 2026.

Nikita Gupta, Riju Chatterjee, Lukas Haas, Connie Tao, Andrew Wang, Chang Liu, Hidekazu Oiwa, Elena Gribovskaya, Jan Ackermann, John Blitzer, Sasha Goldshtein, and Dipanjan Das. DeepSearchQA: Bridging the comprehensiveness gap for deep research agents. *arXiv preprint arXiv:2601.20975*, 2026a.

Tanmay Gupta, Piper Wolters, Zixian Ma, Peter Sushko, Rock Yuren Pang, et al. MolmoWeb: Open visual web agent and open data for the open web. *arXiv preprint arXiv:2604.08516*, 2026b.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. WebVoyager: Building an end-to-end web agent with large multimodal models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024. arXiv:2401.13919.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-R1: Training LLMs to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. *arXiv preprint arXiv:2501.05366*, 2025a.

Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yutao Zhu, Yongkang Wu, Ji-Rong Wen, and Zhicheng Dou. WebThinker: Empowering large reasoning models with deep research capability. *arXiv preprint arXiv:2504.21776*, 2025b.

Junteng Liu, Yunji Li, Chi Zhang, Jingyang Li, Aili Chen, Ke Ji, Weiyu Cheng, Zijia Wu, Chengyu Du, Qidi Xu, Jiayuan Song, Zhengmao Zhu, Wenhu Chen, Pengyu Zhao, and Junxian He. WebExplorer: Explore and evolve for training long-horizon web agents. *arXiv preprint arXiv:2509.06501*, 2025.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants. *arXiv preprint arXiv:2311.12983*, 2023.

MiroMind Team. MiroThinker-1.7 & H1: Towards heavy-duty research agents via verification. *arXiv preprint arXiv:2603.15726*, 2026.

Shikhar Murty, Hao Zhu, Dzmitry Bahdanau, and Christopher D Manning. NNetNav: Unsupervised learning of browser agents through environment interaction in the wild. *arXiv preprint arXiv:2410.02907*, 2024.

OpenAI. Introducing deep research. `https://openai.com/index/introducing-deep-research/`, 2025.

Thinh Pham, Nguyen Nguyen, Pratibha Zunjare, Weiyuan Chen, Yu-Min Tseng, and Tu Vu. SealQA: Raising the bar for reasoning in search-augmented language models. *arXiv preprint arXiv:2506.01062*, 2025.

Long Phan, Alice Gatti, et al. Humanity's last exam. *arXiv preprint arXiv:2501.14249*, 2025.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, et al. UI-TARS: Pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL `https://qwen.ai/blog?id=qwen3.6-35b-a3b`.

Tongyi DeepResearch Team. Tongyi DeepResearch technical report. *arXiv preprint arXiv:2510.24701*, 2025.

Brandon Trabucco, Gunnar Sigurdsson, Robinson Piramuthu, and Ruslan Salakhutdinov. InSTA: Towards internet-scale training for agents. *arXiv preprint arXiv:2502.06776*, 2025.

Haoming Wang, Haoyang Ye, Junjie Fang, et al. UI-TARS-2 technical report: Advancing GUI agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*, 2025a.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, et al. OpenCUA: Open foundations for computer-use agents. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025b. arXiv:2508.09123.

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

Jian Xie, Tianhe Lin, Zilu Wang, Yuting Ning, Yuekun Yao, Tianci Xue, Zhehao Zhang, Zhongyang Li, Kai Zhang, Yufan Wu, et al. QUEST: Training frontier deep research agents with fully synthetic tasks. *arXiv preprint arXiv:2605.24218*, 2026.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, et al. Mobile-Agent-v3.5: Multi-platform fundamental GUI agents. *arXiv preprint arXiv:2602.16855*, 2026.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents. *arXiv preprint arXiv:2504.01382*, 2025.

XYZ Agentic Team. AI4AI at scale: A full-pipeline system for enhancing LLM agentic capabilities. `https://xyz-lab.ai/blogs/ai4ai-at-scale/`, 2026. Technical report, July 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. GPT-4V(ision) is a generalist web agent, if grounded. In *International Conference on Machine Learning (ICML)*, 2024. arXiv:2401.01614.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. DeepResearcher: Scaling deep research via reinforcement learning in real-world environments. *arXiv preprint arXiv:2504.03160*, 2025.

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. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2307.13854.

A CASE STUDIES

A.1 HYWEBENCH

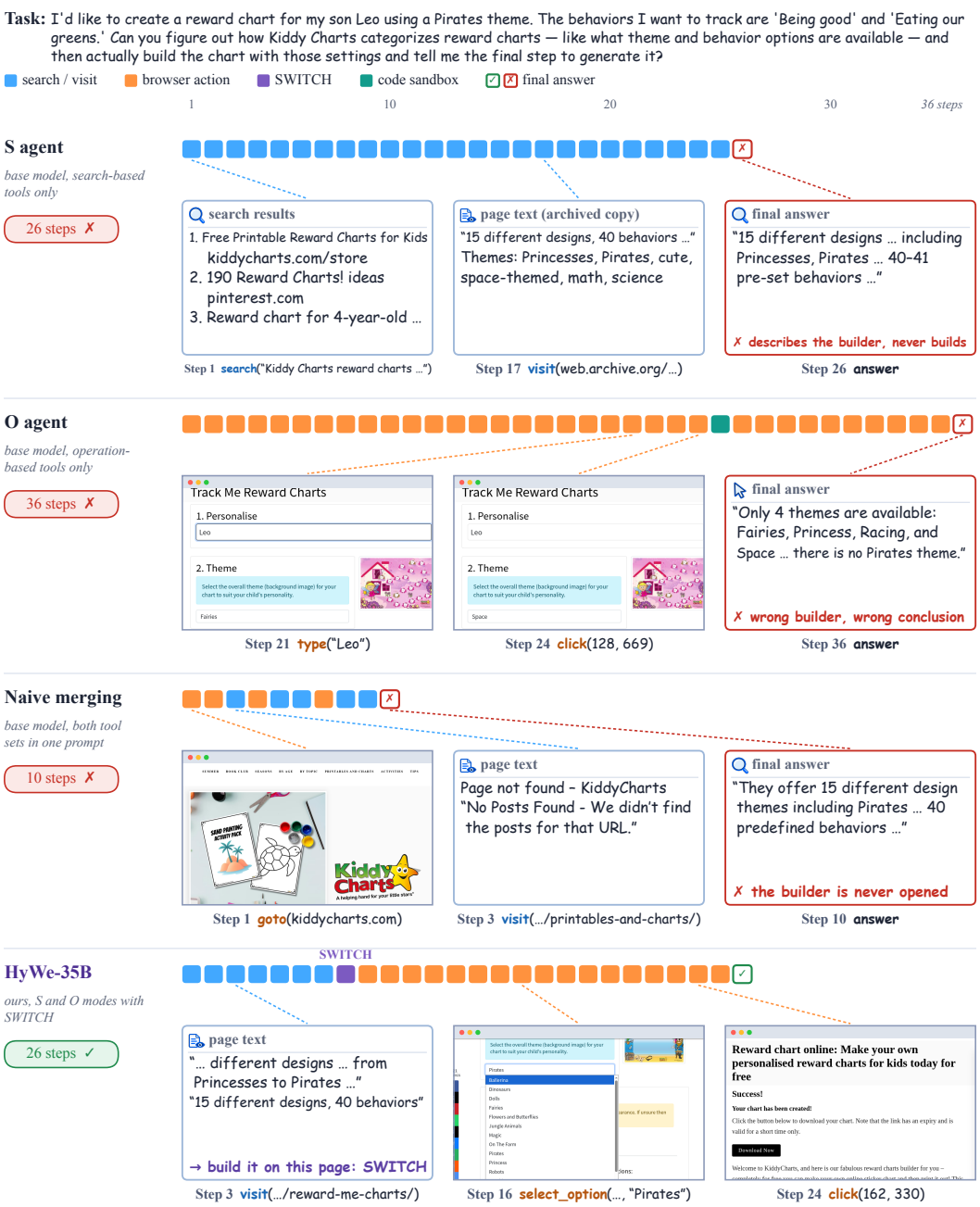

Figure 3: Case study on one HYWEBENCH task with seven sub-goals. Rows 1–3 use the same base model with search-based tools only, operation-based tools only, or both tool sets under one prompt (naive merging); row 4 is HYWE-35B. Each block of a strip is one step, and all strips share one step scale. Dashed lines link key steps to their tool calls (bottom) and results (top). Texts and screenshots are taken from the evaluation runs; quotations are shortened.

Figure 3 compares four agents on the task "*I'd like to create a reward chart for my son Leo using a Pirates theme ... build the chart with those settings and tell me the final step to generate it*". The

first three agents use the same base model, Qwen3.6-35B-A3B, and differ only in their tools and prompt; the fourth is HYWE-35B.

Search-based agent. The chart builder is rendered by page scripts, so page reading returns only its description ("15 different designs, 40 behaviors"). The agent spends 26 steps on searches, third-party reviews, and archived copies of the page, then answers with a description of the builder. It never configures a chart and achieves 4 of 7 sub-goals.

Operation-based agent. Without search-based tools, the agent navigates the site's menus and reaches a different builder, Track Me Reward Charts, which offers four themes. After entering the child's name and trying the theme menu, it concludes that no Pirates theme exists. It achieves 5 of 7 sub-goals in 36 steps.

Naive merging. With both tool sets under one prompt, the agent alternates between browser actions and search-based tools five times in ten steps. It opens the site in the browser, reads a missing page through `visit`, scrolls the browser page, and returns to search. After the first step, it outputs tool calls without any reasoning. It never opens the builder, answers with a description, and achieves 2 of 7 sub-goals.

HYWE-35B. Seven search-based steps establish that the Reward Me Charts builder offers a Pirates theme and 40 behaviors. The agent then calls SWITCH, stating that the remaining details require interacting with the page, and completes the chart in 17 browser actions: it enters the name, selects the Pirates theme, checks both behaviors, and generates the chart. It achieves all 7 sub-goals in 26 steps. Under the same framework, the base model collects the same information but never calls SWITCH and achieves 3 of 7 sub-goals in 25 steps. This suggests that the gain comes from learning when to switch, not from the tool interface alone.

A.2 OPERATION BENCHMARKS

Figures 4 and 5 show one task from WebVoyager and one from Online-Mind2Web. Both tasks start from a web page. Each figure compares HYWE-35B with GUI-Owl-32B and with the base model under HYWE.

A_1 on WebVoyager. The task asks for the Market Data section of BBC News and the company that supplies its data. The BBC discontinued this section in November 2024, so no sequence of page actions can reach it. GUI-Owl-32B clicks the cookie dialog 11 times in its first 14 steps. Its next 41 clicks target the Business tab, but none of them reaches the Business section, and it stops at the 100-step limit. The base model under HYWE browses the News and Business pages for 70 steps and opens a missing markets page. It never calls SWITCH and answers "Dow Jones", which no visited page supports. HYWE-35B calls SWITCH at its first step, stating that searching is faster than navigating the site. Its searches find the BBC notice that the feeds were discontinued and a Press Gazette report that the feed was licensed from Morningstar. It answers correctly in 8 steps.

A_1 and A_2 on Online-Mind2Web. The task asks to display the chart that compares women's unemployment trends in Illinois and Michigan on America's Health Rankings. GUI-Owl-32B scrolls 52 times and opens menus and ranking tables. It reaches the Compare States link only at its 100th step, when the step limit ends the episode. The base model reaches a chart of Illinois against the United States in 16 steps but does not add Michigan. After 41 browser steps, it reports that the site failed to render the chart. HYWE-35B first switches to search (A_1) and finds the state comparison page by step 7. Reading that page returns only the default comparison of Illinois with the United States, because page reading cannot operate the state selector. The agent then switches back to the browser (A_2), opens the comparison page, selects Michigan, and displays the chart, completing the task in 14 steps. In both tasks, the operation-based baselines spend most of their step budget on navigation. HYWE-35B uses search to locate the target and uses the browser only when the page must be operated.

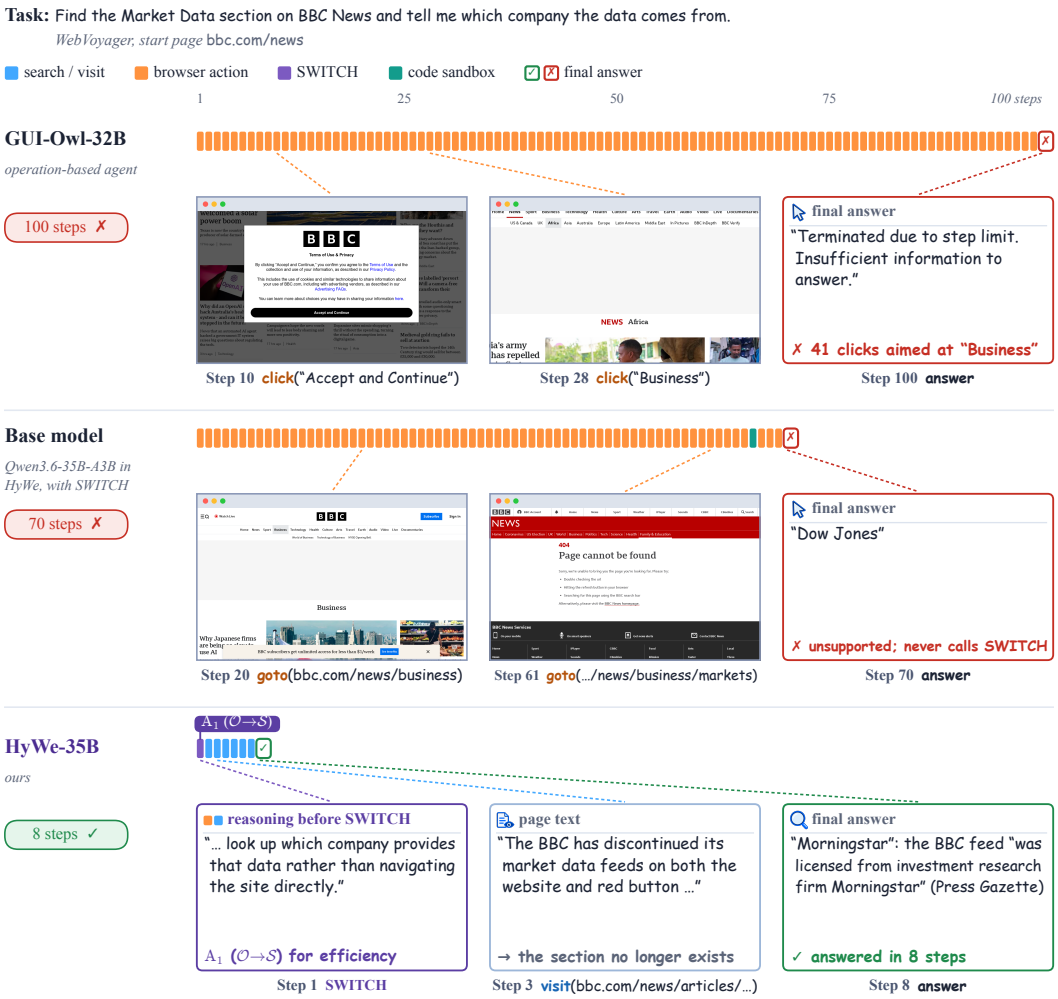

Figure 4: Case study on a WebVoyager task, showing switching pattern A_1 ($\mathcal{O} \to \mathcal{S}$ for efficiency). The rows compare GUI-Owl-32B, the base model under HyWe, and HyWe-35B. The layout follows Figure 3, and all strips share one step scale. The GUI-Owl-32B screenshots come from its raw rollout.

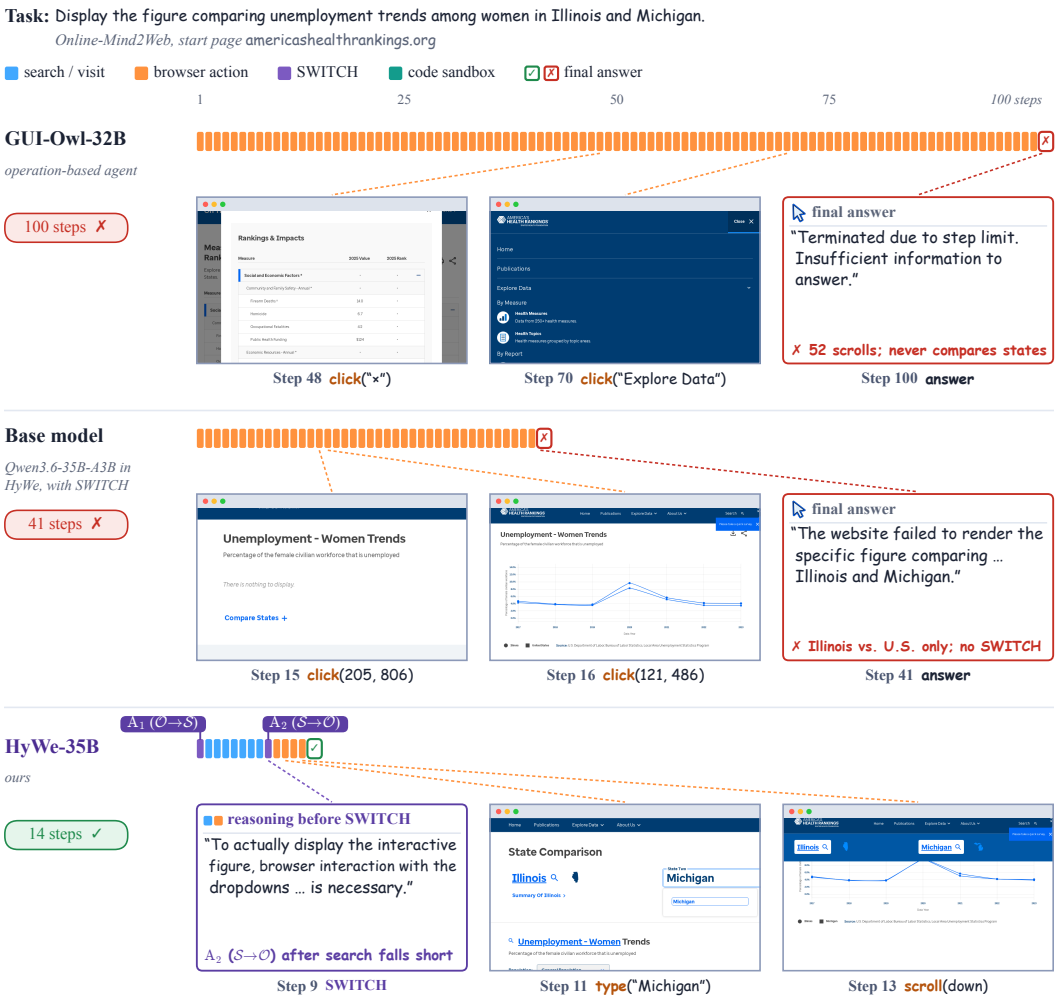

Figure 5: Case study on an Online-Mind2Web task, showing switching patterns A_1 and A_2 ($\mathcal{S} \to \mathcal{O}$ after a search attempt falls short). The rows and layout are the same as in Figure 4.

B PILOT STUDY: CAPABILITY AND EFFICIENCY

Evidence for A_1 and A_2. To motivate the two mode-selection patterns, we analyze 100 tasks from InSTA (Trabucco et al., 2025), each run twice under the same harness and budget, once with a search-based agent and once with an operation-based agent (Figure 6).

The two agents solve *different tasks*. The search-based agent solves 60 tasks and the operation-based agent 62, but only 44 are solved by both: 16 are solved only by the search-based agent and 18 only by the operation-based agent (Figure 6a).

Stratifying by *task requirements* reveals different advantages. Retrospective annotation of the original task descriptions and reference steps identifies 52 information-lookup tasks and 43 interaction-oriented tasks, such as conditional queries, configured quotes, and form submissions. Five uncertain cases remain in the overlap analysis but are omitted from the stratified curves. These labels describe task demands, not mode exclusivity.

The search-based agent has an *early advantage on information lookup*: within four steps, it solves $26/52$ tasks (50.0%), compared with $6/52$ (11.5%) for the operation-based agent. Final success rates are 82.7% and 67.3%, respectively (Figure 6b).

The operation-based agent achieves *higher coverage on interaction-oriented tasks*, solving $24/43$ (55.8%), compared with $15/43$ (34.9%) for the search-based agent (Figure 6c). The search-based agent can still solve some of these tasks, so the task category alone does not determine which mode is needed.

These observations support two design choices: *prefer the search-based mode* when it suffices (A_1), and *fall back to the operation-based mode* when the search-based mode fails to obtain the required information (A_2). When reachability is uncertain, A_2 permits a bounded search attempt before switching. The pilot motivates these patterns; Section 4.3 evaluates learned switching.

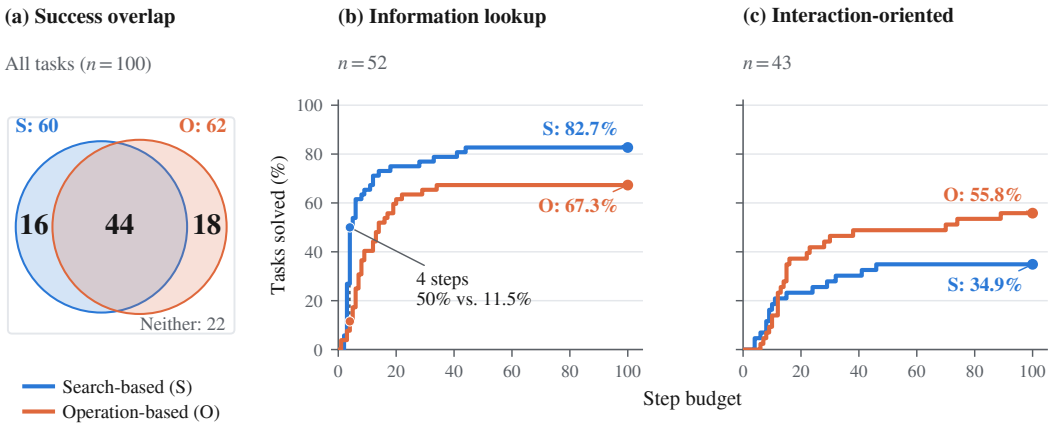

Figure 6: Pilot study on 100 InSTA tasks with search-based (S) and operation-based (O) runs. **(a)** Success overlap; circle and intersection areas are proportional to success counts. **(b,c)** Cumulative success rates for information-lookup and interaction-oriented tasks, classified from task descriptions and reference steps. Each denominator includes all tasks in its stratum, including failures. The five uncertain tasks appear only in (a). The curves use matching linear axes and count successful runs completed within the indicated number of steps; they summarize fixed-budget runs, not separate evaluations rerun at each budget.

B.1 COST AND FAILURE ANALYSIS

Paired costs. The pilot contains 100 tasks from 100 domains. Cost ratios are computed within each task and then summarized by their median; they are not ratios of the two modes' median costs. Steps and token usage are available for all 44 joint successes. Five of these operation-based runs lack wall-clock measurements, so the time comparison uses the remaining 39 pairs without imputation. Two-sided Wilcoxon signed-rank tests on the paired raw costs give $p = 0.0089$ for steps, $p = 5.15 \times 10^{-9}$ for tokens, and $p = 1.63 \times 10^{-9}$ for time. The median ratios of operation-based to search-based costs are 1.9, 9.0, and 6.3, respectively.

Tasks solved only by the search-based agent. Of the 16 tasks solved only by the search-based agent, failure annotations for the operation-based agent identify incorrect or incomplete extraction in 7, looping or giving up in 5, widget failures in 2, and page-load failures in 2. Thus 14 failures concern interpreting or interacting with pages rather than loading them. The failed operation-based runs take a median of 25 steps, compared with 10 for their successful search-based counterparts. In 12 of these 16 tasks, the search-based run bypasses on-site navigation through a deep link or a parameterized URL.

Tasks solved only by the operation-based agent. Among the 18 tasks solved only by the operation-based agent, failure annotations for the search-based agent identify inaccessible dynamic state in 7, answers sourced only from other sites in 5, incorrect extraction in 4, an incorrect claim of absence in 1, and another failure in 1. The failed search-based run uses a median of 0.22 times the tokens of its successful operation-based counterpart. This measures the overhead of these particular attempts; it does not determine an optimal search budget. Successful operation-based runs are annotated as interaction-dependent on $16/18$ tasks solved only by the operation-based agent and $26/44$ joint successes (two-sided Fisher exact $p = 0.0347$). Interaction dependence is a trajectory annotation: a browser run may use a form even when search can obtain the answer through a direct URL. The pilot's modes are locked, so these outcomes cannot measure whether the model would choose to switch. Likewise, termination with an answer action alone does not show whether its response acknowledges failure.

B.2 Task selection and annotation

Sample and task requirements. The 100 tasks span 100 domains and form a curated pilot rather than an unfiltered random sample of InSTA. The selection script retains 81 tasks from the first candidate pool after excluding cases whose operation failure is attributed primarily to anti-bot blocking, then samples 19 tasks from a second pool under the same exclusion rule (seed 20260915). The results therefore describe this filtered sample.

For Figure 6b,c, we retrospectively assign task-demand labels using the original task wording and dataset reference steps. This is an AI-assisted annotation, not an independently blinded or human-validated label set. The annotation input records contain no paired outcomes, costs, or rollout-level interaction attributions. Information lookup covers retrieving, comparing, or summarizing existing page content; using a search box or filters merely to locate a named product page does not suffice for an interaction label. Interaction-oriented tasks request conditional query results, filtered catalogue rankings or counts, configured prices, service-generated outputs, or state-changing actions. These categories describe requested outputs and actions; they do not establish that browser operation is necessary. Search may reach a conditional result through a parameterized URL or another permitted route.

Five tasks remain uncertain because the reference material does not resolve whether the requested information is directly readable or depends on an interface state: enquiry-form options, bike-station availability, shipment availability for a part, size availability for a boot, and chapter-specific audio languages. They are retained in all aggregate analyses and excluded only from the two stratified panels. No fresh website reachability checks are used to resolve these cases. Table 5 reports all three groups; success counts use every task in each group, including operation-based failures. The task-level annotations retain the original wording, reference steps, source identifiers, and a rationale for each label.

Table 5: Pilot results by task requirement. Successes are counts, with success rates in parentheses. Each rate uses all tasks in its row as the denominator, including failed runs.

Task demand	Tasks	Search-based successes	Operation-based successes
Information lookup	52	43 (82.7%)	35 (67.3%)
Interaction-oriented	43	15 (34.9%)	24 (55.8%)
Uncertain	5	2 (40.0%)	3 (60.0%)
All tasks	100	60 (60.0%)	62 (62.0%)

C Evaluation and ablation details

C.1 HyWeBench tasks and evaluation

The benchmark's 100 tasks and their checklists are written by humans. Each checklist contains 2–11 items (5.6 on average), including 1–6 *op* items per task. The 556 items comprise 309 *info* and 247 *op* sub-goals. Checklists are fixed before evaluation and shared across all systems.

Each system runs every task once with a budget of 100 steps. Tasks with a start page begin on that page; the others start without a URL. Kimi-K3 reads the complete execution record, including text tool outputs and browser screenshots with their URLs and page titles. For each item, it reports whether the requested outcome was achieved, the supporting step, and the evidence. Labels describe the requested outcome and do not prescribe tools or modes. External agents retain their native prompts and action spaces; their execution frameworks are described in Appendix C.3.

Task success requires every checklist item to be achieved. Information and operation completion rates aggregate achieved items of each type across all 100 tasks. Switching statistics report executed switches per task and the fraction of tasks with at least one switch. Successful-run length averages steps over successful tasks.

C.2 Base-model and framework controls

Base model. Qwen3.6-35B-A3B (Qwen Team, 2026) is the checkpoint from which HyWe-35B is fine-tuned. It has approximately 35B total parameters and 3B active parameters. We evaluate it within HyWe before our training and compare it with HyWe-35B under matched settings to measure the effect of the training procedure in Section 3.3.

Framework and mode ablations. Using the same base model, naive merging exposes both tool sets under one prompt without explicit modes or a switch action. The full HyWe configuration adds mode-specific instructions, tool isolation, explicit switching, and shared history. We also evaluate search-based-only and operation-based-only configurations within HyWe, removing SWITCH and its prompt instructions in each single-mode variant. Together, these controls distinguish the effect of organizing tools into modes from the contribution of having access to each tool set (Table 3). The framework-design comparison uses the same base model for both configurations on all 100 tasks. Mode ablations use the same model on all 100 tasks; the HyWe row in panel (a) and the $\mathcal{S}+\mathcal{O}$ row in panel (b) are the same run.

C.3 Execution frameworks on HyWeBench

XYZ-Aquila-mini uses its released AxisAgentic search framework, and Agents-A1 uses the search evaluation implementation in its released repository. Fara-7B runs its official agent and browser environment; GUI-Owl-1.5 uses the Mobile-Agent-v3.5 web evaluation implementation; MolmoWeb-8B uses its official agent and benchmark runner. For UI-TARS-1.5-7B, we implement the browser execution loop with Playwright, using the released computer-use prompt and action parser. Trajectories are converted to a common record format for the HyWeBench judge. The base model, HyWe-35B, frontier models, and internal mode ablations use HyWe, with the changes for each ablation described above.

C.4 Operation-based evaluation protocol

Our agents start in operation-based mode and are evaluated on all 300 Online-Mind2Web tasks and all 643 WebVoyager tasks. We adapt WebJudge (Xue et al., 2025) and the WebVoyager auto-evaluator (He et al., 2024) to include evidence from both modes, retaining o4-mini for WebJudge and GPT-4o for the WebVoyager evaluator.

For the open-source operation-based baselines in Table 1, success rates are quoted from the original reports, while step counts are measured on our re-runs judged with Kimi-K3. Published baselines may use different evaluation settings. Steps/SR is the mean action count over all retained task runs, including failures, divided by the success rate of those same runs under Kimi-K3. Actions in both modes and switch calls are counted. This denominator is separate from the published or official-judge SR column. Switch reports the fraction of tasks with at least one executed mode switch.

For example, Fara-7B uses only 12.2 and 10.0 actions on the tasks it solves on Online-Mind2Web and WebVoyager. Counting actions from failed runs as well raises its Steps/SR to 61.5 and 34.9, respectively, about 2.0 and 1.7 times the base model's 30.9 and 20.8.

Adapting the operation-based benchmark judges to hybrid trajectories. The official judges of Online-Mind2Web (WebJudge) and WebVoyager assess a trajectory from its browser screenshots and final response. A hybrid trajectory also contains steps in the search-based mode whose evidence is text—search results, page contents read by `visit`, code outputs—and a screenshot-only judge cannot see it. We make two changes and keep the official prompts otherwise verbatim. (i) *WebJudge.* Each text observation is rated against the task's key points on the same 1–5 relevance scale and with the same threshold (3) as screenshots; the selected observations enter the final verdict next to the selected screenshots, introduced as page-content evidence with the same status. (ii) *WebVoyager auto-evaluator.* Text observations and screenshots are merged into one chronological timeline, from which the last $k = 15$ observations are passed to the judge, and one paragraph is added to the system prompt stating that text observations carry the same status as screenshots. Both adapted prompts additionally ask the judge to report, separately from success, whether the site blocked access (CAPTCHA, bot detection, region block); this flag does not enter the success label.

C.5 Search-based evaluation protocol

Our agents start in search-based mode. Table 4 reports results for the base model and HyWe-35B on HLE, xbench-DeepSearch-2510, and BrowseComp-EN, whose full benchmark sets contain 2,500, 100, and 1,266 tasks, respectively.

Published baselines in Table 4 retain their reported evaluation settings. HLE results reported as text-only are marked with ^{t}; unmarked HLE scores use the full set. QUEST and REDSearcher use their reported context-management results on BrowseComp-EN. We report xbench-DeepSearch-2510 scores only when that benchmark version is explicitly identified.

Search-based answers are graded against reference answers. Exact matches are accepted directly; other answers receive five Kimi-K3 judge votes at temperature 0.6, with the majority deciding correctness. Each vote uses the prompt in Appendix K.2 and returns a JSON verdict with the extracted final answer and a binary label.

C.6 Shared settings and benchmark coverage

Shared execution settings. The base model and HyWe-35B run in the same HyWe Framework with the same tools, prompts, and sampling parameters: a 100-step budget per attempt on operation-based tasks and 300 steps on search-based tasks, reasoning enabled, 8,192 output tokens per call, per-call decorrelated sampling seeds, and the 128K/64K context compression schedule. The switch action remains available unless explicitly disabled for an ablation. The HyWeBench comparison uses a 100-step budget for all systems, as detailed in Appendix C.1.

Benchmarks. Table 6 lists the full evaluation sets. Online-Mind2Web and WebVoyager measure success and efficiency on operation-based tasks; HyWeBench measures hybrid task completion. HLE, xbench-DeepSearch-2510, and BrowseComp-EN assess retention of deep search capabilities. Every system in a controlled comparison is evaluated on the same tasks.

Benchmark	n	Protocol
Online-Mind2Web (Xue et al., 2025)	300	adapted WebJudge; o4-mini
WebVoyager (He et al., 2024)	643	adapted auto-evaluator; GPT-4o
HyWeBench	100	human-written tasks and info/op checklists; at least one op sub-goal per task
HLE (Phan et al., 2025)	2,500	reference-answer judge
xbench-DeepSearch-2510 (Chen et al., 2025)	100	full 2510 release; reference-answer judge
BrowseComp-EN (Wei et al., 2025)	1,266	reference-answer judge

Table 6: Benchmark sizes and scoring protocols. n is the full benchmark size.

D SWITCHING PATTERNS AND EXAMPLE TASKS

Table 7 summarizes the four patterns and their difficulty profiles; one naturalized task from the design run is shown for each. Difficulty is realized by the number of bound sites and the number of cross-site *bridge* rounds (a bridge is an $\mathcal{S}$ prototype that consumes an $\mathcal{O}$ output from one site and produces the input for an $\mathcal{O}$ prototype on another). Medium and hard $\mathrm{A}_3/\mathrm{A}_4$ tasks chain phases across sites with one or two bridge rounds, yielding two to five executed switches.

Pattern	Start	Working modes	Difficulty	Switching motivation
A_1	$\mathcal{O}$	$\mathcal{S}$	standard (1 site)	The opening decision corrects from $\mathcal{O}$ to $\mathcal{S}$ before any browser work: the task is plainly search-answerable.
A_2	$\mathcal{S}$	$\mathcal{S}, \mathcal{O}$	standard (1 site)	A bounded, good-faith $\mathcal{S}$ attempt (two materially different exact queries plus one result inspection), then fall back to $\mathcal{O}$ once structural insufficiency is established.
A_3	$\mathcal{S}$	$\mathcal{S}, \mathcal{O}$	easy / medium / hard (1 / 2 / 3 sites)	$\mathcal{S}$ produces an entity, constraint, identifier, threshold, or shortlist that $\mathcal{O}$ requires.
A_4	$\mathcal{O}$	$\mathcal{O}, \mathcal{S}$	easy / medium / hard (1 / 2 / 3 sites)	$\mathcal{O}$ surfaces an entity, offer, item, metric, or status that binds the subsequent $\mathcal{S}$ question.

Table 7: Switching patterns and difficulty levels. $\mathcal{S}$/$\mathcal{O}$ denote search-based/operation-based modes. Start is the initial mode; Working modes lists the modes used to perform the task.

A_1 **(standard).** "*I'm looking at three stocks for a possible portfolio addition: SoundHound AI (SOUN), BigBear.ai (BBAI), and C3.ai (AI). I want to compare their current market data side by side—current price, market cap, P/E ratio, 52-week high and low, and average volume—so I can evaluate which one looks most promising.*" The task starts on a finance site, but every requested value is published on indexed pages; the intended decision is to switch to $\mathcal{S}$ immediately.

A_2 **(standard).** "*I need to check the voting status of Uniswap governance proposals that are still active. Find any proposals whose voting deadline falls on or after August 15, and report the proposal title, status, current For and Against vote counts, deadline, and quorum requirement.*" Search surfaces proposal titles and discussion threads but not live vote tallies; after a bounded attempt the intended decision is to switch to the governance dashboard.

A_3 **(medium, two sites).** "*I'm planning a trip in October and need to get from a small town near Lyon to the nearest major TGV hub. First, find which major TGV station is closest to Vienne, France. Then use Wanderlog to map the driving route from Vienne to that station. After that, look up the transit fare policy from that station to Geneva, and finally use Transit App to check the actual public-transit route and fare.*" $\mathcal{S}$ resolves the station and the fare policy; each $\mathcal{O}$ phase consumes an $\mathcal{S}$ output.

A_4 **(medium, two sites).** "*I'm a senior machine-learning engineer looking at both Apple and Airbnb. Find the senior ML engineer opening at Apple that lists PyTorch and TensorFlow as required skills; once you have that role and its job ID, look up how Airbnb's career site labels the equivalent role family so you can search for the matching opening on Airbnb's careers page and compare the two listings.*" $\mathcal{O}$ surfaces the Apple listing; $\mathcal{S}$ bridges the role family; $\mathcal{O}$ consumes it on the second site.

E TRAJECTORY REVIEW AND REFINEMENT

The reviewers and refiners in the multi-agent synthesis framework verify and refine the executor's trajectories. Table 8 lists their checks, inputs, and outputs. Every quotation a judge makes is mechanically verified against the trajectory; a quotation that cannot be located voids the judgement.

Judgement	**Input**	**Consumption**
Unified audit (two stages)	rollout, contract, final answer; screenshots for $\mathcal{O}$ phases	per-phase progress account; leakage steps $\rightarrow$ rewrite; task-success and route verdicts $\rightarrow$ filter
Leakage rewrite	leakage steps and their reasoning	reasoning replaced in place under fact/decision/coherence constraints, then re-verified
Residue detection	controller's feedback log (which message reached which step)	second rewrite pass; trajectories with pervasive leakage dropped
Site fidelity (A_1)	final values, observations	drop if the answer was not obtained from the named site (judged value by value)
Switch rationale ($\mathrm{A}_1/\mathrm{A}_2$)	rewritten reasoning at the switch step	drop if the switch was not triggered by the pattern's intended condition
Answer coverage	answer schema, all observations	drop if a required field is unobtainable; otherwise merge fields obtained in earlier phases (numeric values re-verified against observations)
$\mathcal{O}$-segment purity	$\mathcal{O}$ segments with screenshots	drop the segment if no browser-only interaction was performed

Table 8: Trajectory review checks.

Algorithm 2 HYWECOLLECTOR trajectory synthesis pipeline

Require: site set W; pattern and difficulty ratios; probe budget B; task synthesizer; executor π_E; controller ν; reviewers and refiners

Stage 1 — task prototypes from capability probing
for all sites $w \in W$ **do**
  $F_w \leftarrow \mathrm{Explore}(w)$ ▷ operator drives the browser; extractor records answer-shaped facts
end for
$F \leftarrow \mathrm{ValueFilter}\big(\bigcup_w F_w\big)$ ▷ keep concrete values; drop scaffolding and duplicates
for all facts $f \in F$ **do**
  $\ell(f) \leftarrow \mathbf{1}[\mathrm{Probe}_{\mathcal{S}}(f; B) \text{ recovers } f]$ ▷ Eq. (4); transcript retained
  label f as search-reachable / intrinsically operation-bound / prober-limited / volatile
end for
remove unstable or unreliable facts; form $F_{\mathcal{S}}$ and $F_{\mathcal{O}}$
abstract prototype vocabularies $P_{\mathcal{S}}, P_{\mathcal{O}}$; retain website and entity bindings
Stage 2 — switching pattern design
define $\Phi = \{\mathrm{A}_1, \mathrm{A}_2, \mathrm{A}_3, \mathrm{A}_4\}$ ▷ efficiency, search failure, and information transfer in both directions
Stage 3 — multi-agent trajectory synthesis
for all blueprints to generate **do**
  sample pattern ϕ, sites w, entity e, difficulty d; sample composition c over admissible pairs $\mathrm{out}(p_{\mathcal{S}}) \in \mathrm{in}(p_{\mathcal{O}})$ ▷ Eq. (5); deterministic given the seed
  build contract $C(b) = ((g_k, \delta_k, m_k))_{k=1}^{K}$ with produce/consume edges ▷ Eq. (5)
  repeat
    $x \leftarrow \mathrm{Naturalize}(b)$ ▷ question, per-phase goals, switch triggers, answer schema
  until $\mathrm{Validate}(x, b)$ passes or the retry limit is reached
end for
for all tasks $(x, C(b))$ **do**
  $\tau \leftarrow$ COORDINATEDEXECUTION$(x, C(b))$ ▷ Algorithm 3
  if τ did not end in ANSWER **then continue**
  (progress, L, verdicts) $\leftarrow \mathrm{Audit}(\tau, C(b))$ ▷ two stages; every quotation verified verbatim
  if task not successful **then** discard τ
  for all leakage steps $i \in L$ **do**
    $r_i \leftarrow \mathrm{Rewrite}(r_i)$; re-verify ▷ facts and decision unchanged
  end for
  detect residual leakage from the controller's feedback log; rewrite again or discard if pervasive
  drop rejected intermediate ANSWER steps
  if site fidelity, switch rationale, or answer coverage fails **then** discard τ (or merge fields obtained in earlier phases)
  for all $\mathcal{O}$ segments ς of τ **do**
    if ς performed no browser-only interaction **then** drop ς
  end for
  $\mathcal{D} \leftarrow \mathcal{D} \cup \mathrm{Export}(\tau)$ ▷ Eq. (8): $n_{\mathrm{sw}} + 1$ samples
end for
return $\mathcal{D}$

F PIPELINE ALGORITHMS

Algorithm 2 summarizes the synthesis pipeline of Section 3.2 end to end, and Algorithm 3 expands the collaboration between the executor and controller in Stage 3. This stage also includes task synthesis and trajectory review, as shown in Algorithm 2. Data collection models and task validation checks are described in Appendix J.

Algorithm 3 COORDINATEDEXECUTION: collaboration between the executor and controller

Require: task x; contract $C = ((g_k, \delta_k, m_k))_{k=1}^{K}$; executor π_E; controller ν; sanitizer σ; per-phase action budgets; step budget $T_{\max}$

$k \leftarrow 1$; $m \leftarrow m_1$; $\beta \leftarrow \emptyset$; $u \leftarrow \emptyset$ ▷ phase, mode, verified bindings, last control-agent message
for $t = 1, \ldots, T_{\max}$ **do**
  $a_t \sim \pi_E\big(\cdot \mid x, g_k, \beta, \sigma(u), \tau_{<t}\big)$ rendered through $\mathcal{R}$ ▷ Eq. (6); plan details hidden
  if $a_t = \text{ANSWER}(y)$ **then**
    if $k = K$ and $\nu(\delta_K, \tau_K) = 1$ **then return** τ **else** $u \leftarrow$ rejection (answer step recorded, later dropped); **continue**
  else if $a_t = \text{SWITCH}$ **then**
    if $\nu(\delta_k, \tau_k) = 1$ **then**
      $\beta \leftarrow \beta \cup \text{bindings}(\tau_k)$; pin cited evidence steps against folding; $k \leftarrow k{+}1$; $m \leftarrow \bar{m}$; open a new segment
    else
      $u \leftarrow$ rejection; record the step with `switch_executed`=false ▷ no new segment
    end if
  else
    $o_t \leftarrow \text{Env}_m(a_t)$; append (m, a_t, o_t)
    if a human-verification gate is detected **then** terminate
    if phase budget exhausted, or no progress for three actions, or eight actions since the last check **then**
      $u \leftarrow \sigma\big(\nu\text{'s assessment of } (\delta_k, \tau_k)\big)$ ▷ contract vocabulary stripped before it reaches π_E
    end if
  end if
end for
return τ ▷ terminated by budget

G TRAINING DETAILS

Both data sources are rendered into multi-turn SFT samples using the same code paths as inference. System prompts, history blocks, context compression, and screenshot placement follow Section 3.1. Samples that exceed the context budget even after folding are split into two or three parts by greedy bisection rather than dropped, preserving long-horizon supervision. Table 9 gives the training settings.

Setting	Value
Backbone	Qwen3.6-35B-A3B (Qwen Team, 2026)
Trainable parameters	language model; vision encoder and visual projector frozen
Objective	next-token cross-entropy
Sample unit	one segment (context rendered by the harness; targets = the segment's assistant turns)
Context length	131,072 tokens (folded to the 64K watermark when building samples)
Reasoning tokens	retained and supervised
Data mixture	hybrid samples and 7,000 public search-based trajectories, stored in one shuffled dataset
Optimizer	Adam; $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$
Learning rate	peak 2×10^{-5}; minimum 2×10^{-6}
Schedule	5% warmup from zero, followed by cosine decay
Weight decay	0.1 (constant)
Gradient clipping	1.0
Training epochs	1
Effective batch size	32 samples per parameter update
Precision	BF16
Packing / padding-free mode	disabled / enabled
Random / data seed	42 / 42
Data shuffling / validation split	enabled / 0
Activation recomputation	full, uniform, one layer per recomputation group
Training framework	Megatron via ms-swift `4.5.0.dev0`

Table 9: Training configuration.

H BASELINE DETAILS

We describe the baselines in Tables 1, 2, 3, and 4. Published benchmark scores serve as external references; our evaluations on HYWEBENCH compare agents on the same tasks and completion checklists. Evaluation protocols and differences in benchmark coverage are given in Appendix C; base-model and framework controls are in Appendix C.2.

H.1 OPEN-SOURCE OPERATION-BASED AGENTS

These agents choose browser actions from visual observations. All four families appear in both the traditional operation-based evaluation and the comparison on HYWEBENCH; the GUI-Owl model size differs between these settings.

Fara-7B. Fara-7B (Awadallah et al., 2025) is a 7B computer-use agent that reads screenshots and predicts mouse and keyboard actions. Its FaraGen data pipeline generates tasks on real websites, collects multiple execution attempts, and filters trajectories using several verifiers. It provides a compact operation-based baseline trained on synthetic web interactions. We use the Fara-7B release.

UI-TARS-1.5-7B. UI-TARS-1.5 (Qin et al., 2025; ByteDance UI-TARS Team, 2025) integrates visual perception, reasoning, and action prediction in a vision-language model. It reasons before acting and uses reinforcement learning to improve GUI interaction. Actions include coordinate-based clicks, typing, and scrolling. Our baseline is the publicly released 7B model, which tests whether a general GUI agent can solve the web tasks through browser interaction.

GUI-Owl-1.5. GUI-Owl-1.5 (Xu et al., 2026) supports browser, desktop, and mobile interaction. Its training combines synthetic interaction data with reinforcement learning across environments, covering visual grounding, planning, and action execution. We include GUI-Owl-1.5-8B-Thinking

on Online-Mind2Web and WebVoyager (Table 1), and the 32B variant on WebVoyager (Table 1) and HYWEBENCH (Table 2). The 32B variant has no published Online-Mind2Web result, so that benchmark is omitted for it in Table 1.

MolmoWeb-8B. MolmoWeb (Gupta et al., 2026b) predicts browser actions from task instructions and screenshots. Its MolmoWebMix training data includes synthetic trajectories, human demonstrations, and examples of individual web skills and visual grounding. It interacts through the rendered page without requiring HTML or an accessibility tree. We use the 8B model as an open visual web-agent baseline.

H.2 OPEN-SOURCE SEARCH-BASED AGENTS

All eight agents below appear in the deep search comparison (Table 4). We additionally evaluate XYZ-Aquila-mini and Agents-A1 with search-based tools on HYWEBENCH (Table 2). This classification describes their role in our evaluation; some models also support other agentic tasks.

Tongyi DeepResearch. Tongyi DeepResearch (Tongyi DeepResearch Team, 2025) is a 30B-class mixture-of-experts research agent with roughly 3B active parameters. It learns long search-and-reading trajectories through agentic mid-training, supervised fine-tuning, and reinforcement learning. Its training uses automatically generated tasks and trajectories, while context management supports extended information gathering. We include the published scores of its standard open-model setting.

OpenSeeker-v1. OpenSeeker (Du et al., 2026) builds on Qwen3-30B-A3B-Thinking-2507 and uses supervised fine-tuning on about 11.7K synthetic trajectories. Its task generation links facts across web pages and hides identifying clues to require multi-step search. This provides a search-based baseline trained with an open data pipeline and SFT alone. We report the first release, OpenSeeker-v1.

REDSearcher. REDSearcher (Chu et al., 2026) constructs difficult search tasks by controlling how evidence is connected and distributed across sources. It combines mid-training for evidence grounding, planning, and tool use with supervised fine-tuning and reinforcement learning. A simulated search environment reduces the cost of training rollouts. We include the published 30B search-agent results.

MiroThinker-1.7-mini. MiroThinker-1.7 (MiroMind Team, 2026) targets long research tasks with agentic mid-training focused on planning, reasoning over accumulated context, and tool interaction. These objectives help the agent sustain information gathering across many steps. We include the open 30B MiroThinker-1.7-mini model.

FORT-Searcher. FORT-Searcher (Deng et al., 2026) focuses on training tasks that cannot be solved through easy shortcuts. Its data pipeline controls entity selection, evidence connections, and question wording, then refines questions to remove alternative routes that bypass the intended search. It trains on the resulting trajectories using supervised fine-tuning alone. Our comparison uses the published 30B model.

QUEST-35B. QUEST (Xie et al., 2026) trains deep research agents with fully synthetic tasks. It organizes evaluation criteria into rubric trees to provide rewards for both factual questions and open-ended research, and combines mid-training, supervised fine-tuning, and reinforcement learning. Its tasks cover evidence gathering and report writing. We include the 35B model's published search benchmark results.

Agents-A1. Agents-A1 (Bai et al., 2026) is a 35B mixture-of-experts model trained for long-horizon tasks across several agentic domains. Its post-training combines broad supervised data with domain-specific expert knowledge and long interaction trajectories. We include its published deep search results and evaluate its search-based configuration on HYWEBENCH, providing a baseline with the same total parameter scale as HYWE-35B.

XYZ-Aquila-mini. XYZ-Aquila-mini (XYZ Agentic Team, 2026) builds on Qwen3.6-35B-A3B and targets deep search through improvements to training data and the execution framework. Its pipeline generates tasks from connected web evidence, preserves tool observations in supervised trajectories, and refines context management using verified evaluation feedback. We include its pub-

lished deep search scores and evaluate its search-based configuration on HYWEBENCH. It shares the backbone family of HYWE-35B.

H.3 FRONTIER LLM SYSTEMS

Operation-based references. Table 1 includes published results for GPT-5.5, GPT-5 (SoM), o3 (SoM), Claude-Opus-4.6, Seed2.1-Pro, GLM-5V-Turbo, Kimi-K2.6, and Qwen3.8-Max. These rows represent models paired with the browser interfaces used in their respective evaluations. SoM denotes a set-of-marks interface that labels actionable screen elements. We also include Gemini 2.5 Computer Use, OpenAI Operator, and Yutori Navigator (n1), which are systems designed for computer or browser interaction. These results provide frontier references under their reported settings.

Deep search references. Table 4 includes GPT-5.6 Sol, Claude Fable 5, Qwen3.8-Max, Kimi-K3, Kimi-K2.6, GLM-5.2, and DeepSeek-V4-Pro (Preview). Their published scores describe performance with tool-assisted information gathering and reasoning. These rows provide context for the open-source search agents; differences in evaluation sets are retained in the table annotations and described in Appendix C.

Frontier models within HYWE. We evaluate Kimi-K3, Qwen3.8-Max, and Qwen3.7-Plus zero-shot within HYWE on HYWEBENCH. Each has access to both search-based and operation-based modes through the framework. This comparison tests how much hybrid task performance can be obtained from frontier models without our hybrid training. These are our evaluations, separate from the published reference rows above.

I FRAMEWORK IMPLEMENTATION DETAILS

I.1 TOOL FUNCTIONS AND SIGNATURES

Table 10 lists tool functions and arguments, grouped into search-based tools, operation-based tools, and tools shared by both modes. Search-based APIs return text; browser actions return a screenshot and status after execution. The reason for `switch` appears in the preceding reasoning. Coordinates use a 0–1000 logical grid over a 1280×720 viewport and are converted to pixels by the runtime. This convention is learned from data and is not described in the prompt.

The `search` and `scholar` tools use the Serper API. The `visit` tool uses the Jina API to fetch page content and `deepseek-flash-0731` to summarize it according to the extraction goal.

I.2 RUNTIME PARAMETERS

Table 11 lists the runtime parameters shared by the base and fine-tuned rows of Table 1.

I.3 CONTEXT COMPRESSION

Compression is triggered by two watermarks on the token count $T(\mathcal{R})$ of the rendered context (text tokens, image tokens, and chat-template overhead):

$$\text{if } T(\mathcal{R}) > \theta_{\text{hi}} \text{ then fold until } T(\mathcal{R}) \le \theta_{\text{lo}}, \qquad \theta_{\text{hi}} = 131{,}072, \;\; \theta_{\text{lo}} = 65{,}536, \tag{9}$$

where "fold" means truncating a step's observation to its first $c = 600$ characters and dropping its reasoning and screenshot. The trigger actually used reserves room for generation, $\theta_{\text{hi}}^{\text{eff}} = \min(\theta_{\text{hi}}, L_{\max}) - (n_{\text{out}} + 256)$, with $L_{\max}$ the serving window and n_{out} the maximum output length. Folding proceeds in a fixed order: first the steps already in the history block, oldest first; only when all of them are folded is the segment start $s(t)$ advanced, which moves the oldest steps of the current segment into the history block, entering it already folded. The most recent step is never folded, tool calls are never altered, and steps cited as evidence by the controller during data collection are exempt. The gap between θ_{hi} and θ_{lo} exists for the key–value cache: a fold rewrites the prefix, so folding "just enough" at every step would invalidate the cache at every step, whereas folding down to θ_{lo} once buys a long run of pure appends. Screenshots follow the same rule—kept for unfolded steps, dropped once a step is folded—and observations in the search-based mode never carry images.

Tool	Arguments	Function
Search-based tools ($\mathcal{S}$)		
`search`	**query** (string or list)	Search the web; return ranked snippets with titles, URLs, sources, and dates.
`scholar`	**query** (string or list)	Search academic sources; return publication details, citation counts, and PDF links.
`visit`	url (string or list), **goal**, parse_type	Extract goal-relevant text from pages or PDFs; defaults to the current URL and does not change browser state.
Operation-based tools ($\mathcal{O}$)		
`goto` / `new_tab`	**url**	Open a URL in the current or a new browser tab.
`go_back`	none	Return to the previous page in the current tab.
`switch_tab` / `close_tab`	**index**	Select or close the specified browser tab.
`click` / `double_click` / `hover`	**x**, **y**	Click, double-click, or hover at the specified coordinates.
`type`	**text**, submit	Enter text, optionally submitting it.
`press_key`	**keys**	Press a key or key combination.
`select_option`	**x**, **y**, **value**	Select a value in a dropdown at the specified coordinates.
`scroll`	**direction**, x, y	Scroll at a position, or scroll the page if no position is supplied.
`drag`	**from_x**, **from_y**, **to_x**, **to_y**, hold_ms	Drag between two positions, optionally holding before release.
`wait`	time	Wait for the specified duration.
Shared tools ($\mathcal{S}$ and $\mathcal{O}$)		
`code_sandbox`	**code**, language, stdin, run_timeout, compile_timeout, files, fetch_files	Run isolated code for calculation, parsing, and validation.
`switch`	none	Change mode, replace its tool interface, and rebuild context; return a hand-off notice without executing an environment action.

Table 10: HyWe tool functions and arguments, grouped by mode. Bold arguments are required; all other arguments are optional.

Parameter	Value
Viewport	1280×720
Coordinate grid	0–1000 (converted to pixels by the runtime)
Step budget $T_{\max}$ per attempt (operation-based)	100
Max output tokens per turn	8,192
Temperature / top-p / top-k	1.0 / 0.95 / 20
Repetition penalty	1.05
Fold watermarks $\theta_{\text{hi}}/\theta_{\text{lo}}$	131,072 / 65,536 tokens
Folded-observation prefix c	600 characters
Screenshots retained in context	all unfolded $\mathcal{O}$ steps
Search results per query	top 10
`visit` extraction	goal-conditioned summary, up to 32,000 tokens per chunk
Format retries per turn	5 (fresh seed each)
Error/format-failure attempts	up to 2
Attempts ending at the step limit	up to 5, restarting with a fresh context
Per-call timeouts (agent / summarizer)	1,800 s / 1,800 s
Seed derivation	base seed + crc32(task, attempt) $+7919 \cdot t$

Table 11: HyWe runtime parameters for the base model and HyWe-35B in Table 1.

J Data collection implementation details

We use Qwen3.8-Max and Kimi-K3 for the LLM-based components of HyWeCollector.

J.1 NATURALIZATION VALIDATORS

A naturalized task is accepted only if all of the following hold; a failure returns the reason to the generator, which re-samples.

1. The question is non-trivial in length and contains no meta-language about agent routing or mode switching.
2. The site bindings in the generated text preserve the blueprint's fixed site domains exactly.
3. The per-component goals and completion conditions cover the blueprint's components exactly, with no empty entries.
4. No component's goal or completion condition names a site outside its declared site scope, except the site of a producer it directly consumes from.
5. For medium and hard $\mathrm{A}_3/\mathrm{A}_4$ tasks, the question explicitly names every target site.
6. The switch triggers cover every switch policy in the blueprint, with no empty entries.
7. A coherence rationale and a non-empty answer schema are present.
8. The dependency bindings match the blueprint's dependency edges exactly.
9. Producer answers remain unknown in the question text until their phase completes (no leakage of downstream values).

K SYSTEM AND EVALUATION PROMPTS

K.1 MODE-SPECIFIC SYSTEM PROMPTS

The two system prompts share the output-format block; the mode-specific parts are reproduced below verbatim. The answer-shape sentence differs by mode: in $\mathcal{S}$ it asks for a concise entity, number, or list; in $\mathcal{O}$ it additionally allows stating the end state reached when the task was an operation. Tool schemas (Table 10) are appended after the prompt.

Search-based mode ($\mathcal{S}$).

```
You are a general-purpose agent on the SEARCH-BASED path.

Your tools this turn: search, scholar, visit, code_sandbox, switch.
- `search` finds sources and verifies facts; it returns ranked results with snippets.
- `scholar` is for academic papers, citation-rich evidence and PDFs.
- `visit` opens a specific URL and extracts the facts you ask for via `goal`.
- `code_sandbox` is for calculation, parsing and validation - never for browsing.
- `switch` hands control to the operation-based (browser) path. Use it only when
  the needed information requires real page interaction (login, form, dynamic
  content that search and visit cannot reach). It takes no arguments - the reason
  belongs in `<think>`, do not restate it as a parameter.

# Avoiding Dead Ends (read this before every action)

Your earlier steps are recorded in the task message below - every query you ran
is listed there, though older observations may be truncated to save space.
Use it. Before acting, check that history:

1. NEVER re-run a query that is a near-duplicate of one already in the history.
   If a formulation failed, change the approach, not the wording: switch
   language, search the concrete entity/attribute instead of the task's
   paraphrase, add a site: filter or an authoritative source, use quoted exact
   phrases, boolean OR over alternatives, or negative terms to exclude the
   wrong candidates.

2. NEVER stay on one candidate for more than ~5 consecutive actions. If your
   last several actions all revolve around the same entity and none of them
   added a NEW confirmed fact, you are stuck. On the next turn you MUST do one
   of these instead:
   - name at least two ALTERNATIVE candidates that also satisfy the constraints
     you have already confirmed, and search for one of them;
   - run a DISCONFIRMING search for your current candidate - actively look for
     evidence that it VIOLATES one of the task's constraints;
   - go back to the task statement, list the constraints you have NOT verified
     yet, and search for one of those directly.

3. Search snippets are LEADS, not evidence. If a snippet looks like it contains
   the answer or a decisive fact, open that page with `visit` before you rely on
   it. Do not build a final answer out of snippets alone.

4. Work the constraints in parallel, not strictly in order. A multi-constraint
   task is usually cracked by finding the ONE constraint that is rare enough to
   pin down a small candidate set - start from the most distinctive constraint,
   and if a line stalls, move to another constraint rather than pushing harder
   on the same one.

5. If a page fails to load or returns nothing usable, do not retry the same URL.
   Find another source for the same fact.
```

Operation-based mode ($\mathcal{O}$).

```
You are a general-purpose agent on the OPERATION-BASED path (real browser).

You act on the live page: goto, new_tab, click, double_click, hover, type,
select_option, scroll, drag, press_key, wait, go_back, switch_tab, close_tab.
`code_sandbox` is available for calculation/parsing. `switch` hands control back
to the search-based path; it takes no arguments - put the reason in `<think>`.

Once a page is open you get its screenshot each turn - ground every action in
what is actually visible there. If no page is open yet, `goto` a URL first;
there is nothing on screen to click.

# Avoiding Dead Ends (read this before every action)

Your earlier actions and the resulting pages are recorded in the task message
below (older observations may be truncated). Use it.

1. NEVER click the same coordinate twice. If a click produced no visible change
```

```
   (same page, same layout), that element is not what you thought it was - do
   NOT click it again, and do NOT click a point a few pixels away hoping for a
   different result. Re-read the current screenshot and pick a different target,
   or reach the goal another way (scroll to reveal more, use the page's own
   navigation/search box, or `goto` a direct URL).

2. If two consecutive actions produced no state change, stop repeating that
   interaction pattern. Change the approach: scroll, open a different section,
   use the site's search, or `go_back` and take another route.

3. Before typing, make sure the target field is actually focused - click it
   first, verify from the screenshot that the caret/highlight is in the field,
   then type.

4. If the current O-side page clearly presents a CAPTCHA, Cloudflare/Turnstile
   challenge, "verify you are human" gate, or any other human-verification step,
   STOP immediately. Do not click, wait, refresh, retry, work around it, or
   `switch` to another path. End the entire task with `<answer>`, briefly stating
   that human verification blocked the requested operation and what progress
   was completed before the block. Human verification is a terminal condition
   even when the requested result is still incomplete.

5. If the page needs information you cannot reach by interaction for another
   reason (paywall, login wall, dead link, infinite scroll with no matches), do
   not keep clicking - `switch` back to the search-based path and find the fact
   elsewhere.

6. Verify state changes from the screenshot before assuming success. Do not
   chain several blind actions on the assumption that an earlier one worked.
```

Shared output format (both modes).

```
# Output Format

Every turn, output exactly:

<think>
Your reasoning for this turn: what the last observation told you, what is still
unverified, and why this specific next action.
</think>
<tool_call>
{"name": "<tool name>", "arguments": {...}}
</tool_call>

Call at most ONE tool per turn. The next message contains that tool's result.
[...]
When you have enough verified evidence, stop calling tools and output:

<think>
Why the evidence is now sufficient - map each constraint in the task to the fact
that confirms it.
</think>
<answer>
your final answer
</answer>

# Tool call OR answer - never both

Each turn ends in EXACTLY ONE of two ways: a `<tool_call>`, or an `<answer>`.
Emitting both in the same turn is an error. [...]
```

K.2 SEARCH-BASED ANSWER JUDGING

The reference-answer judge in Appendix C.5 uses the following prompt for each vote.

```
You are a strict benchmark evaluator.

Your task is to judge whether an agent's final answer matches the provided
reference answer for the given question.

Rules:
1. First extract the agent's final committed answer from the agent output.
2. Judge only whether that extracted final answer matches the reference answer.
3. Ignore extra explanation, evidence, or reasoning unless it changes or
   contradicts the final answer.
4. Accept semantically equivalent answers even if they differ in formatting,
   capitalization, punctuation, articles, common abbreviations, or obvious
   unit/currency/date formatting.
5. For numbers, accept equivalent numeric values after straightforward
   normalization, but reject materially different values, incompatible units,
```

```
   or unsupported rounding.
6. For list or set answers, all required items must be present and no
   materially incorrect extra items should be introduced.
7. If the agent gives multiple conflicting answers, or never clearly commits
   to a final answer, mark it incorrect.
8. If there is no answer, set extracted_final_answer to null and
   predicted_label to 0.

Return JSON only, with exactly these keys:
{
  "extracted_final_answer": "string or null",
  "reasoning": "brief explanation",
  "predicted_label": 0
}
```

L FULL TASK PROTOTYPE VOCABULARIES

Stage 1 of the pipeline (Section 3.2) abstracts two task prototype vocabularies from the probed fact pool; the task synthesizer in Stage 3 composes them into tasks. Each operation prototype names a recurring kind of interaction that produces operation-bound facts and declares a typed interface; each search prototype names a recurring kind of information need satisfiable from the indexed web. The interface types are: for $\mathcal{S}$ outputs, `entity`, `candidate_set`, `location`, `date_window`, `constraint_set`, `identifier`, `threshold`, `classification`, and `fact_set`; for $\mathcal{O}$ outputs, `live_offer`, `quote`, `ranked_item`, `configured_item`, `computed_result`, `route`, `itinerary`, `cart_total`, `metric`, `live_status`, `protocol_quote`, `template`, `visual_item`, `record`, `service_quote`, `build`, `plan`, and `artifact`. Tables 12 and 13 list the full vocabularies.

A pair $(p_{\mathcal{S}}, p_{\mathcal{O}})$ is admissible when $\mathrm{out}(p_{\mathcal{S}}) \in \mathrm{in}(p_{\mathcal{O}})$.

Table 12: Operation-based task prototypes $P_{\mathcal{O}}$ (48 total). *out*: the prototype's output type; *in*: the search-based output types it accepts as inputs.

Prototype	Interaction it captures	out	in
`flight_live_fare`	Live flight fare and itinerary: run a date-, route-, cabin-, and stop-constrained flight search and read the currently displayed itinerary and fare	`live_offer`	`entity`, `candidate_set`, `location`, `date_window`, `constraint_set`, `identifier`
`stay_availability`	Live lodging availability: apply destination, dates, party, amenity, and budget constraints and inspect the currently available properties	`ranked_item`	`entity`, `candidate_set`, `location`, `date_window`, `constraint_set`
`tour_slot`	Tour or experience slot: select an experience, date, time, language, and group size and read the available slot and price	`live_offer`	`entity`, `candidate_set`, `location`, `date_window`, `constraint_set`
`car_rental_quote`	Car-rental quote: enter pickup and return details plus vehicle constraints and read the available rental and full quote	`quote`	`entity`, `location`, `date_window`, `constraint_set`, `identifier`
`event_ticket`	Event ticket inventory: locate an event and inspect live date, section, quantity, and price inventory	`live_offer`	`entity`, `candidate_set`, `location`, `date_window`, `constraint_set`, `identifier`
`insurance_quote`	Insurance quote wizard: complete a profile-, asset-, location-, and coverage-specific quote flow and read the generated estimate	`quote`	`entity`, `constraint_set`, `identifier`, `threshold`, `classification`
`finance_calculator`	Finance calculator result: enter amount, term, rate, contribution, and risk assumptions and read the calculated financial result	`computed_result`	`entity`, `constraint_set`, `threshold`, `classification`
`product_configurator`	Product configuration: configure a product on its page for a concrete use case and read the resulting model, compatibility, price, and availability	`configured_item`	`entity`, `candidate_set`, `constraint_set`, `identifier`, `threshold`, `classification`
`auto_configurator`	Vehicle configuration: select model, powertrain, trim, packages, and appearance options and read the resulting configuration and price	`configured_item`	`entity`, `candidate_set`, `constraint_set`, `identifier`, `threshold`

Prototype	Interaction it captures	out	in
marketplace_filtered_item	Marketplace filtered result: apply category, delivery, seller, condition, rating, and budget constraints and inspect the best displayed result	ranked_item	entity, candidate_set, constraint_set, identifier, threshold, classification
auction_live_bid	Auction state: filter auction lots and inspect the current bid, estimate, time remaining, and lot status	live_status	entity, candidate_set, constraint_set, identifier, threshold, classification
map_route	Interactive map route: set endpoints, travel mode, and route constraints and read the rendered route result	route	entity, candidate_set, location, constraint_set, identifier
transit_itinerary	Public-transit itinerary: set stops, departure or arrival time, accessibility, and transfer constraints and read the live itinerary	itinerary	entity, candidate_set, location, date_window, constraint_set, identifier
food_delivery_cart	Food delivery selection: filter restaurants or dishes, configure an order, and read availability, substitutions, fees, and total	cart_total	entity, candidate_set, location, constraint_set, threshold, classification
real_estate_filter	Real-estate filtered property: apply location, price, property, amenity, and commute constraints and inspect matching listings	ranked_item	entity, candidate_set, location, constraint_set, threshold, classification
job_filter	Job-board filtered opening: apply role, location, seniority, compensation, skill, and work-mode constraints and inspect matching openings	ranked_item	entity, candidate_set, location, constraint_set, identifier, threshold, classification
course_filter	Course or test-prep selection: apply subject, level, schedule, delivery, prerequisite, and price constraints and inspect the matching course	ranked_item	entity, candidate_set, constraint_set, identifier, threshold, classification
sports_dashboard	Sports dashboard state: set league, season, position, scoring, and split controls and read the requested ranking or metric	metric	entity, candidate_set, date_window, constraint_set, identifier, threshold
trading_screener	Trading screener state: set market, asset, indicator, horizon, and threshold controls and inspect the resulting asset or chart value	metric	entity, candidate_set, date_window, constraint_set, identifier, threshold, classification
crypto_protocol_quote	Crypto or DeFi protocol quote: select network, assets, amount, route, and risk constraints and inspect the live swap, bridge, or yield result	protocol_quote	entity, candidate_set, constraint_set, identifier, threshold, classification
template_filter	Template gallery selection: filter templates by workflow, audience, features, format, and price and inspect the best matching template	template	entity, candidate_set, constraint_set, identifier, classification
creative_visual_selection	Creative visual selection: apply style, medium, subject, color, and licensing constraints and inspect a visually matching work	visual_item	entity, candidate_set, constraint_set, identifier, classification

Prototype	Interaction it captures	out	in
gov_record_lookup	Government interactive record lookup: submit jurisdiction, entity, record type, and date constraints and read the returned filing or status	record	entity, candidate_set, location, date_window, constraint_set, identifier
science_interactive_result	Scientific interactive tool result: set dataset, variables, region, transformation, and display controls and read the computed or visualized result	computed_result	entity, candidate_set, location, date_window, constraint_set, identifier, threshold, classification
health_provider_filter	Health-provider finder: apply location, specialty, insurance, availability, and patient constraints and inspect matching providers	ranked_item	entity, candidate_set, location, date_window, constraint_set, identifier, classification
home_service_quote	Home-service or utility quote: enter property, location, service, usage, and schedule details and read the generated option or quote	service_quote	entity, candidate_set, location, constraint_set, identifier, threshold, classification
streaming_catalog	Streaming catalog state: apply region, service, genre, language, and maturity constraints and inspect currently available titles	ranked_item	entity, candidate_set, location, constraint_set, identifier, classification
game_interactive_build	Interactive game build or state: set character, mode, equipment, map, and performance constraints and read the resulting build or game state	build	entity, candidate_set, constraint_set, identifier, threshold, classification
fitness_plan_result	Fitness or wellness plan result: enter profile, goal, schedule, equipment, and limitation constraints and read the generated class or plan	plan	entity, candidate_set, constraint_set, identifier, threshold, classification
saas_data_artifact	SaaS data transformation artifact: configure a data source, transformation, grouping, visualization, and export and inspect the generated artifact	artifact	entity, candidate_set, constraint_set, identifier, threshold, classification
flight_aggregator_search	Flight-metasearch itinerary and fare: run a route-, date-, passenger-, cabin-, stop-, and baggage-constrained flight search and inspect a displayed itinerary and fare	live_offer	entity, candidate_set, location, date_window, constraint_set, identifier
general_car_rental	Car-rental availability and quote: enter pickup and return locations, dates, driver profile, and vehicle constraints and inspect the available rental and complete quote	quote	entity, location, date_window, constraint_set, identifier, classification
booking_tour_slot	Travel experience availability: filter travel experiences by destination, activity type, date, time, language, accessibility, and group size and inspect a matching live slot	live_offer	entity, candidate_set, location, date_window, constraint_set, identifier

Prototype	Interaction it captures	out	in
cruise_sailing	Cruise sailing and cabin availability: search a cruise line by region, embarkation port, sailing window, duration, party, and cabin constraints and inspect a matching live sailing	live_offer	entity, candidate_set, location, date_window, constraint_set, identifier
intercity_transport	Intercity train or bus itinerary: search an intercity rail or coach service using exact stations, date, time, passenger, transfer, and accessibility constraints	itinerary	entity, candidate_set, location, date_window, constraint_set, identifier
ride_fare_estimate	Ride-hail or micromobility estimate: enter pickup, destination, timing, party, and vehicle or accessibility constraints and inspect the displayed ride estimate	quote	entity, candidate_set, location, date_window, constraint_set, identifier, threshold
urban_food_delivery	Address-specific food delivery cart: set a delivery address, filter restaurants and dishes, configure a cart, and read address-specific availability, fees, and total	cart_total	entity, candidate_set, location, constraint_set, threshold, classification
game_catalog_filter	Game or digital-asset catalog selection: filter a game or digital-asset catalog by platform, genre, mechanics, rating, license, compatibility, and budget and inspect matching listings	ranked_item	entity, candidate_set, constraint_set, identifier, threshold, classification
mortgage_calculator	Mortgage payment or rate estimate: enter purchase price, down payment, location, loan term, rate assumptions, taxes, and insurance and inspect the generated mortgage estimate	computed_result	entity, location, constraint_set, identifier, threshold, classification
credit_card_filter	Credit-card offer filter: filter credit-card offers by credit profile, fee, reward type, spending pattern, and welcome-offer constraints and inspect matching offers	ranked_item	entity, candidate_set, constraint_set, identifier, threshold, classification
visual_asset_filter	Licensed visual-asset selection: search and visually inspect images by subject, composition, palette, orientation, people, and licensing constraints	visual_item	entity, candidate_set, constraint_set, identifier, classification
park_admission	Attraction admission and add-on inventory: select an attraction, visit date, party, admission duration, and add-ons and inspect live ticket options and totals	live_offer	entity, candidate_set, location, date_window, constraint_set, identifier
restaurant_reservation	Restaurant table availability: filter restaurants by location, cuisine, dietary support, date, party size, and price and inspect a live reservation slot	live_offer	entity, candidate_set, location, date_window, constraint_set, identifier, classification
science_data_export	Scientific dataset subset and export: choose a dataset, variable, spatial and temporal subset, aggregation, units, and export format and inspect the resulting data product	artifact	entity, candidate_set, location, date_window, constraint_set, identifier, threshold, classification

Prototype	Interaction it captures	out	in
defi_governance_dashboard	DeFi governance proposal or gauge state: filter governance proposals, gauges, or votes by protocol, status, round, date, and threshold and inspect the current voting state	live_status	entity, candidate_set, date_window, constraint_set, identifier, threshold, classification
collectible_market_listing	Collectible marketplace visual listing: filter collectible listings by object type, period, maker, visual condition, authenticity, seller, auction or fixed-price format, and budget	visual_item	entity, candidate_set, constraint_set, identifier, threshold, classification
game_item_catalog	Official game item or mode catalog: filter the official game catalog or current item and mode listings by game mode, item class, role, compatibility, rotation, and cost	ranked_item	entity, candidate_set, constraint_set, identifier, threshold, classification
fantasy_lineup	Fantasy lineup or roster builder: set contest, slate, roster, salary, position, and projection constraints and inspect the resulting lineup or eligible player set	build	entity, candidate_set, date_window, constraint_set, identifier, threshold, classification

Table 13: Search-based task prototypes $P_{\mathcal{S}}$ (36 total). *out*: the prototype's output type. A *producer* supplies an operation-based input; a *consumer* uses an operation-based output; cross-site bridges serve both roles.

Prototype	Information need it captures	out	role
organization_background	Organization identity and background: establish who operates the relevant site or entity, where it is based, and the key dated background fact needed by the user	fact_set	consumer
entity_by_constraints	Entity identification by public constraints: identify the named entity that satisfies the public, stable constraints in the scenario	entity	producer
candidate_shortlist	Public candidate shortlist: build a short candidate set satisfying stable public criteria before live comparison	candidate_set	producer
nearest_relevant_location	Resolve a relevant location: identify the location, hub, venue, district, or waypoint that best satisfies the stated geographic constraint	location	producer
official_date_window	Resolve an official date window: determine the authoritative event, season, deadline, or eligibility dates needed to parameterize the live task	date_window	producer
requirement_derivation	Derive requirement set: derive the stable policy, technical, accessibility, dietary, or eligibility requirements that the live result must satisfy	constraint_set	producer
public_identifier	Resolve public identifier: resolve the canonical identifier, ticker, filing number, model code, station code, or standardized name needed for an exact lookup	identifier	producer

Prototype	Information need it captures	out	role
public_threshold	Find a decision threshold: find the authoritative benchmark, limit, target, or recommended threshold to apply in the interactive tool	threshold	producer
public_classification	Resolve classification: determine the official category, risk class, genre, role family, or product class needed to choose the correct interface controls	classification	producer
official_policy_check	Official policy or eligibility check: use the surfaced entity or result to determine the applicable official policy, eligibility rule, restriction, or obligation	fact_set	consumer
external_benchmark	External benchmark comparison: compare the surfaced price, quote, metric, route, or calculated value with an authoritative public benchmark on the same basis	fact_set	consumer
creator_owner_lookup	Creator, owner, or operator lookup: identify the creator, owner, operator, manufacturer, or issuing authority behind the surfaced item	fact_set	consumer
reputation_and_reviews	Independent reputation comparison: use independent public sources to compare reputation, reviews, awards, or reliability for the surfaced option or shortlist	fact_set	consumer
historical_context	Historical context: place the surfaced entity, metric, status, or result in the relevant historical sequence or prior-period context	fact_set	consumer
technical_compatibility	Technical compatibility verification: verify from authoritative documentation whether the surfaced configuration, artifact, build, or item is compatible with the user's external system or requirement	fact_set	consumer
safety_accessibility	Safety and accessibility follow-up: determine the relevant safety, accessibility, health, or operational constraints for the surfaced route, venue, service, or plan	fact_set	consumer
availability_alternative	Public alternative discovery: find a publicly documented alternative when the surfaced live result is unavailable, over budget, incompatible, or otherwise unsuitable	fact_set	consumer
method_and_unit_check	Method, definition, and unit verification: verify the public definition, method, units, coverage, and limitations needed to interpret the surfaced metric or computed result	fact_set	consumer
public_specification	Official specification lookup: retrieve the authoritative specifications and limits for the surfaced product, asset, record, or output	fact_set	consumer
multi_source_fact_join	Multi-source enrichment join: join the surfaced entities with the same public attributes from multiple sources and return a comparable table	fact_set	consumer
deep_conjunctive_entity	Conjunctive entity identification: identify an initially unknown entity that uniquely satisfies at least four independent clues distributed across multiple sources	entity	producer

Prototype	Information need it captures	out	role
deep_exhaustive_shortlist	Exhaustive constrained shortlist: enumerate a candidate universe, intersect several independently sourced constraints, deduplicate aliases, and return the complete qualifying shortlist	candidate_set	producer
deep_location_optimization	Cross-source location resolution: resolve the exact location that best satisfies geographic, jurisdictional, schedule, accessibility, and proximity constraints drawn from multiple sources	location	producer
deep_temporal_window	Temporal provenance and date-window derivation: reconstruct a dated sequence across authoritative records and derive the exact future window in which all eligibility, event, season, or policy constraints overlap	date_window	producer
deep_requirement_synthesis	Multi-authority requirement synthesis: combine policy, technical, legal, accessibility, or compatibility requirements from several authorities into one operational constraint set	constraint_set	producer
deep_identifier_chain	Canonical identifier relationship chain: follow a multi-hop relationship chain to resolve the canonical code, registry ID, ticker, model number, station code, or normalized name required for an exact lookup	identifier	producer
deep_threshold_aggregation	Normalized multi-source threshold: collect comparable numeric evidence from multiple sources, normalize units and time bases, calculate the requested benchmark, and return the decision threshold	threshold	producer
deep_classification_reconciliation	Cross-authority classification reconciliation: reconcile conflicting taxonomies, definitions, editions, or jurisdictions and determine the one classification applicable to the user's scenario	classification	producer
deep_result_table_join	Deep result enrichment table: take every entity surfaced by the operation, resolve stable join keys, collect the same public attributes from multiple sources, and produce a comparable qualified table	fact_set	consumer
deep_source_reconciliation	Conflicting-source result reconciliation: use the surfaced entity or value to gather conflicting public claims, compare definitions, dates, scope, units, and authority, and resolve which interpretation applies	fact_set	consumer
deep_historical_provenance	Historical provenance of surfaced result: trace the surfaced entity, item, status, or metric through a dated origin, ownership, release, renaming, or prior-period chain and explain the verified transitions	fact_set	consumer

Prototype	Information need it captures	out	role
deep_evidence_synthesis	Scholarly or technical evidence synthesis: formulate a focused question from the surfaced result, locate multiple primary or review sources, and synthesize methods, findings, disagreement, and limitations	fact_set	consumer
cross_site_constraint_bridge	Cross-site result-to-constraint bridge: use the exact result surfaced on one site to look up the policy, compatibility, eligibility, or comparison rule that becomes a concrete constraint for the next site's operation	constraint_set	consumer / producer
cross_site_identifier_bridge	Cross-site canonical-identifier bridge: resolve the surfaced result to the canonical name, code, model, listing identity, or normalized join key required for an exact operation on the next site	identifier	consumer / producer
deep_cross_site_constraint_bridge	Deep cross-site requirement reconciliation: take the prior site's exact result, reconcile multi-source policy, technical, legal, or compatibility requirements, and produce one operational constraint set for the next site	constraint_set	consumer / producer
deep_cross_site_candidate_bridge	Deep cross-site candidate transfer: expand and verify the exact entity surfaced on one site against several public sources, reconcile aliases and exclusions, and produce a qualified candidate set for the next site's operation	candidate_set	consumer / producer