Trace4Tool: Tool-Use-Contract-Guided Tool Synthesis and Reuse for Web Agents

Anonymous ACL submission

Abstract

Web agents often address recurring browser tasks by repeatedly planning over large Document Object Model (DOM) trees and screenshot observations. This pattern makes similar tasks expensive and fragile, even when previous trajectories contain reusable execution patterns. We present **Trace4Tool**, a framework that synthesizes executable tools from web-agent trajectories and selects them using Tool-Use Contracts. Tool candidates and Tool-Use Contracts are synthesized by an LLM and promoted only after evidence validation. A Tool-Use Contract specifies when a synthesized tool may be safely reused and what evidence its result must provide. At runtime, the tool selector invokes a tool only when the task and observation uniquely match the Tool-Use Contract of a validated tool; otherwise, it abstains or falls back. On the 157-task shopping benchmark of **ContractWebBench**, Trace4Tool succeeds on 123/157 tasks, compared with 70/157 tasks for a no-tool WebAgent; a separately reported maintenance check reaches 155/157 tasks. The no-tool run uses 2,051 runtime LLM calls and 9.9M effective tokens, whereas Trace4Tool supports zero-LLM runtime execution under unique Tool-Use Contract matches. A 60-task boundary diagnostic further shows that Trace4Tool can reject incompatible tasks instead of forcing unsafe tool use.

1 Introduction

LLMs have enabled web agents for realistic shopping, knowledge, and enterprise-style websites (Yao et al., 2023; Zhou et al., 2023; Deng et al., 2023; He et al., 2024; Lu et al., 2024). A typical web agent observes a webpage, reasons over screenshots or Document Object Model (DOM) trees, and executes actions such as clicking, typing, selecting, or scrolling. However, these models are highly sensitive to the outcome of each perception and action step, making long-horizon tasks challenging. Moreover, many common repetitive tasks, such as searching for a product, filling out forms, or opening a dashboard, are repeatedly executed from scratch, which results in increased context length, more LLM calls, higher latency, and a greater risk of failures.

Recent work has reused historical trajectories as memory prompts (Wang et al., 2023; Shinn et al., 2023; Putta et al., 2024), replay signals (Cheng et al., 2025; Wang et al., 2026a), or reusable skills (Prabhu et al., 2025; Lu et al., 2026; Zheng et al., 2025). However, memory-prompt methods still depend on online model planning, trajectory replay works only when tasks match closely, and reusable skills often leave execution scope and evidence requirements underspecified. As illustrated in Figure 1, *fully exploiting historical trajectories for future reuse remains a challenging task.*

In this paper, we propose Trace4Tool, a Tool-Use-Contract-guided, LLM-based framework for tool synthesis and reuse. We shift the role of LLMs from a perception-and-action policy model to a tool-synthesis model. Specifically, an LLM automatically annotates historical trajectories, proposes reusable segments, and synthesizes candidate tool implementations without human tool engineering. This process is practical for web tasks because web trajectories expose structured cues: frontend observations provide DOM regions, forms, tables, product cards, and page state, while browser-visible backend information such as URLs, query parameters, rendered server state, and page-accessible responses helps the model infer stable execution mechanisms. Yet an executable tool implementation alone does not determine when the tool should be reused or what evidence should make its output valid.

Our key observation is that reusable web-agent tools need a contract for use, not only an implementation. We therefore introduce a Tool-Use Contract that specifies use conditions, tool arguments, Result type, metadata, lifecycle state, and Risk Level

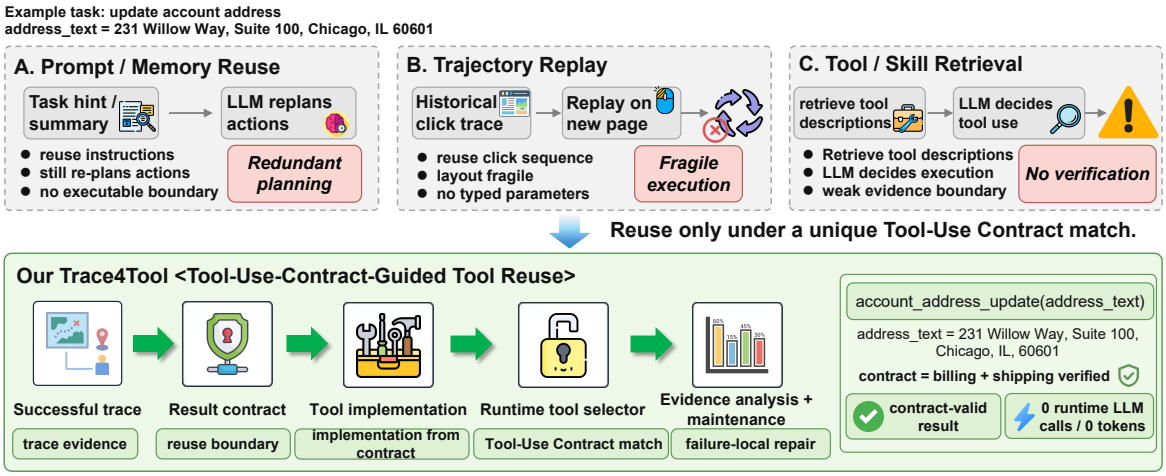

Figure 1: Trace4Tool contrasts prompt reuse, trajectory replay, and skill/tool retrieval with Tool-Use-Contract-guided tool reuse. Trace4Tool synthesizes tool implementations from successful traces, pairs them with Tool-Use Contracts, executes a tool only under a unique Tool-Use Contract match, and uses evidence analysis to guide maintenance.

for the synthesized tools. A Tool-Use Contract supports reuse beyond exact trajectory replay: a tool may generalize to new entities, queries, categories, and paraphrases when the task, observation, required arguments, Result type, and evidence remain compatible. We design a rule-based tool selector that matches new tasks and observations to synthesized tools using the Tool-Use Contract. If the selector uniquely matches a validated tool, the agent executes it at runtime with zero LLM calls and returns a contract-valid result. If the match fails, is ambiguous, or violates the contract, the selector falls back to LLM-assisted matching or the standard WebAgent loop.

To evaluate tool-use results and support tool library maintenance, we propose ContractWebBench along with a Tool-Use-Contract-aware evaluation protocol. ContractWebBench is a WebArena-style shopping benchmark that includes Tool-Use Contracts, difficulty levels, and verifiable evidence. The evaluation protocol covers task success, compatible tool success, correct rejection, runtime LLM calls, token costs, and failure modes. These signals help diagnose whether a failure comes from coverage, contract mismatch, argument extraction, tool execution, or validation.

Our contributions are threefold:

- We introduce **Tool-Use-Contract-guided trajectory-to-tool synthesis**, converting web-agent trajectories into executable tools whose Tool-Use Contracts define safe reuse and expected evidence.
- We design a **tool selector with abstention** that uses a validated tool only when the task and observation uniquely match its Tool-Use Contract before LLM planning.
- We develop a **Tool-Use-Contract-aware evaluation protocol** for tool reuse that reports task success, compatible tool success, correct rejection, runtime cost, evidence-backed failures, and maintenance checks separately.

2 Related Work

Web and GUI agents. LLM-based web and GUI agents typically follow an observe-plan-act loop, using language or multimodal models to interpret observations and choose browser actions. This paradigm has been studied in reasoning-action agents, multimodal web navigators, and GUI grounding agents (Yao et al., 2023; Gur et al., 2023; He et al., 2024; Lai et al., 2024; Cheng et al., 2024; Hong et al., 2024). Benchmarks such as WebArena and Mind2Web evaluate realistic web navigation, while related benchmarks cover shopping, dialogue-grounded browsing, and visual web tasks (Zhou et al., 2023; Deng et al., 2023; Yao et al., 2022; Lu et al., 2024; Koh et al., 2024). Recent work further studies open-web agents and long-horizon or generated web environments (Gupta et al., 2026; Jang et al., 2026; Yuan et al., 2026; Zhou, 2026; Hao et al., 2026). These agents remain general but often solve recurring procedures from scratch, incurring repeated planning costs. Trace4Tool is complementary to this line of work: it studies how recurring tasks can be converted from trajectories into reusable tools

with Tool-Use Contracts.

Tool use and reusable procedures. Tool-use work studies how LLMs invoke, create, or coordinate external tools (Schick et al., 2023; Cai et al., 2023). Web and GUI-agent work extends this idea to API-based web agents, reusable routines, contracted skills, typed web functions, tool tokenization, process automation, and programmatic GUI control (Song et al., 2025; Li et al., 2024; Wang et al., 2024; Lou et al., 2026; Prabhu et al., 2025; Chen et al., 2026b; Lu et al., 2026; Jiang et al., 2026; Wang et al., 2026b; Zhao et al., 2026; Zhong et al., 2026). Such abstractions are useful, but a retrieved skill, verb, or routine may still lack output and evidence rules, abstention conditions, or lifecycle state. Trace4Tool uses Tool-Use Contracts as a common representation for tool synthesis, tool selection, and evaluation. Skill-style descriptions become metadata inside the Tool-Use Contract; the reusable object is an executable tool that can replace online planning when the task and observation uniquely match its Tool-Use Contract.

Memory and trajectory reuse. Memory-based agents store prior experience as skills, reflections, or trajectory summaries to guide future decisions (Wang et al., 2023; Shinn et al., 2023; Putta et al., 2024). Recent web and GUI-agent work adds hierarchical memory, trajectory summaries, experience replay, and pruning (Tan et al., 2026; Yan et al., 2026; Xiao et al., 2026; Shi et al., 2026; Cheng et al., 2025; Nekoei et al., 2025; Wang et al., 2026a; Lin et al., 2026; Zhang et al., 2025). These methods show that trajectories are useful, but reuse often remains advisory or prompt-level. Trace4Tool instead converts selected trajectory segments into executable tools bounded by Tool-Use Contracts, so reused experience is executed and analyzed rather than only retrieved as planner context.

Verification and reliable reuse. Many web-agent evaluations emphasize end-to-end success, but reusable tools require more fine-grained correctness criteria. Recent work verifies computer-use agents through action-effect checks, intent-conditioned evaluation, verifiers, diagnosis tools, and safety guardrails (Zhang et al., 2026b; Chen et al., 2026a; Rosset et al., 2026; Zhai et al., 2026; Aghzal et al., 2026; Zhang et al., 2026a; Cui et al., 2026; Chen et al., 2026c; Ding et al., 2026). Verification is often applied after or around action execution; Trace4Tool moves part of it into the reusable decision unit. Tool-Use Contracts and the tool selector decide whether a tool should be used before execution, and evidence analysis records the supporting evidence.

Table 1: Main components of a Tool-Use Contract.

Component	Function
Use conditions	Determine whether a task and observation are in scope and whether incompatible requests must be rejected.
Arguments	Determine whether required tool arguments can be extracted.
Result type	Determine what type of result must be produced, what evidence must support it, and how it is validated.
Metadata	Supports retrieval, inspection, and maintenance.
Lifecycle state	Determines whether the tool is eligible for automatic use.
Risk Level	Determines whether automatic use is allowed under the current conditions.

3 Method

Trace4Tool converts historical web-agent trajectories into *Tool-Use-Contract-guided tools*: executable tools selected through Tool-Use Contracts. The Tool-Use Contract is the main decision unit for the tool selector. It specifies use conditions, required arguments, Result type, metadata, lifecycle state, and Risk Level. Figure 2 illustrates trajectory-to-tool synthesis, the online tool selector, and evidence analysis with maintenance. This section defines the representation, synthesis pipeline, tool selector, and evaluation protocol.

3.1 Tasks, Tools, and Tool-Use Contracts

Web-agent trajectories. We consider a web agent that receives an instruction, observes DOM trees or screenshots, emits browser actions, and returns an answer or terminal state. A logged run contains the instruction, observations, actions, final output, and validation evidence. Standard WebAgents solve each instruction as a fresh planning problem; Trace4Tool treats successful or repairable trajectories as reusable-tool candidates subject to validation.

Tool-Use-Contract-guided tools. We use *tool* for the reusable callable capability exposed to the agent, and *tool implementation* for its executable site-specific implementation. In Trace4Tool, each tool pairs a tool implementation with a Tool-Use Contract. The tool implementation performs site-specific execution, while the Tool-Use Contract serves as the reusable decision unit. Table 1 sum-

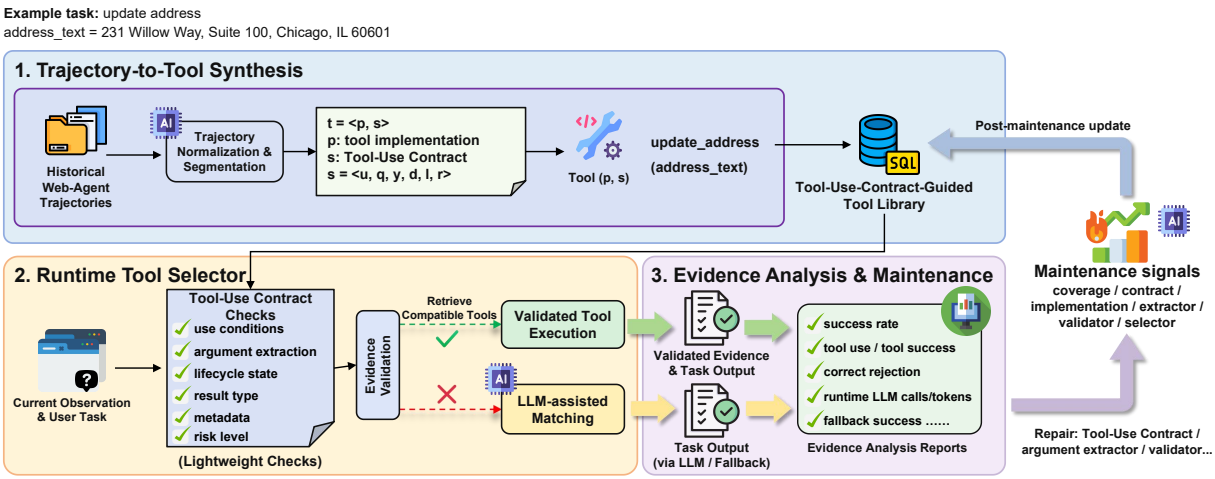

Figure 2: Overview of Trace4Tool. Trajectory-to-tool synthesis normalizes historical web-agent trajectories, identifies reusable segments, and synthesizes tool implementations paired with Tool-Use Contracts. The online tool selector executes a validated tool only when the task and observation uniquely match its Tool-Use Contract; otherwise, it abstains or falls back to LLM-assisted matching or WebAgent control. Evidence analysis reports task success, tool success, rejection, runtime cost, and failure signals, then guides maintenance of contracts, tool implementations, argument extractors, validators, and the selector.

marizes the Tool-Use Contract components and their functions. Automatic use and reporting are determined by the Tool-Use Contract rather than by a separate metadata or state object.

Formally, we represent a tool library entry as

$$t = \langle p, s \rangle, \tag{1}$$

where p is the tool implementation and s is the Tool-Use Contract. The tool selector treats s as the authority for safe reuse. Skill-style descriptions, when present, are folded into the metadata inside s; they are not separate reusable objects. Thus a tool library $\mathcal{T}$ is a set of tools whose selection and execution are mediated by Tool-Use Contracts, the tool selector, and validation evidence.

Tool-Use Contract. The Tool-Use Contract is the decision unit for safe reuse. Unlike a tool-call schema, which describes invocation, a Tool-Use Contract specifies use conditions, arguments, Result type, metadata, lifecycle state, and Risk Level. For example, tasks on the same search-results page may ask for a price range, product names, the highest-priced item, or a cart update. They should not share a tool unless their Tool-Use Contracts are compatible.

Tool-Use Contracts define reuse at the level of task intent, observation pattern, required arguments, Result type, and evidence, rather than by exact replay of a source trajectory. This lets a tool generalize from a source task to held-out variants with new entities, queries, categories, or paraphrases, while still rejecting tasks that require a different result or evidence on a similar page.

We write the Tool-Use Contract as

$$s = \langle u, q, y, d, l, r \rangle, \tag{2}$$

where u is the use conditions, q the argument schema and argument extractor, y the Result type, d the metadata, l the lifecycle state, and r the Risk Level. A tool is selectable only when l and r permit execution, the task satisfies u, all arguments required by q have been extracted, and the requested result can satisfy y.

Lifecycle state prevents tools from being treated as permanently valid. A tool can be assigned a candidate, active, hidden, diagnostic, or archived lifecycle state according to validation results, replacement status, and failure history. Risk Level separately determines whether automatic use is allowed under the current task and environment. Together, these fields determine default loading, eligibility for the tool selector, and whether results count as deployable tool use or diagnostic evidence.

3.2 Tool Synthesis from Trajectories

Tool synthesis is automated by an LLM and deterministic validators. Given a historical trajectory, the LLM annotates typed events, proposes reusable segments, drafts the Tool-Use Contract, and synthesizes a candidate tool implementation. Human-written task-specific tools or held-out answers are

not inserted into the tool library. A candidate becomes active only after it passes evidence validation; otherwise it remains a candidate or diagnostic artifact for later repair.

Trajectory normalization. Synthesis first normalizes trajectories into typed events such as navigation, click, input, extraction, state check, and terminal answer. URL rules canonicalize direct and proxied URLs, normalize query parameters, and identify stable route patterns. DOM and page fingerprints preserve reusable regions such as search boxes, filters, product cards, review blocks, account forms, and admin grids. Browser-visible backend information, including URL parameters, rendered server state, form schemas, and page-accessible responses, further helps the LLM infer stable execution mechanisms from the trajectory. These web-specific signals make trajectory annotation more constrained than open-ended trace interpretation and turn raw logs into comparable segments without relying on transient DOM nodes or product IDs.

Segment selection. Trace4Tool does not need to convert an entire run into a tool. A failed long-horizon run may contain a correct subroutine, while a successful run may include exploratory actions that should not be reused. The LLM therefore selects compact trajectory segments whose action pattern, source page, target evidence, and final result can be expressed as a Tool-Use Contract. The reuse unit is a verifiable subtask, not an entire trajectory or a low-level action.

Fixed before parameterized. For each candidate routine, Trace4Tool first creates a fixed tool seed that preserves source-trajectory values. This seed validates the action pattern, evidence, and Tool-Use Contract before generalization. Trace4Tool then parameterizes safe fields such as query text, category URL, price direction, product name, star bucket, address fields, coupon text, or refund text. It does not parameterize timing decisions or arbitrary click targets, which limits the impact of an incorrect contract. If fixed validation or later parameterized validation fails, the tool is not promoted for automatic use.

Argument extraction and rejection rules. A parameterized tool is useful only when arguments can be extracted reliably. Trace4Tool uses argument extractors for constrained values such as category phrases, product names, price directions, capacities, star ranges, address fields, and form messages. Each contract also stores rejection rules for incompatible requests. For example, a price-range tool should reject review, wishlist, cart, order-status, and ranked-product requests even when the instruction mentions products and prices.

Table 2: Algorithmic view of the tool selector.

Step	Operation
1	Identify the observation and retrieve compatible tools.
2	Check use conditions, arguments, lifecycle state, Risk Level, and Tool-Use Contract compatibility.
3	If exactly one validated tool passes, use zero-LLM runtime execution; otherwise abstain or fall back.
4	Validate returned evidence against the Tool-Use Contract.
5	Return a valid result, or record the failure category for later maintenance.

Metadata. Synthesis also records metadata inside the Tool-Use Contract: intent, examples, keywords, and analysis notes derived from use conditions, arguments, and Result type. Metadata supports retrieval, inspection, and maintenance, while the full Tool-Use Contract remains the authority on safe reuse and the tool implementation remains the executable component. Tools can be hidden, deprecated, marked as diagnostic, or blocked through Tool-Use Contract lifecycle state or Risk Level without deleting their evidence.

3.3 Tool Selector

Tool selector. At runtime, Trace4Tool identifies the observation, loads compatible tools, and applies five checks before invoking the general LLM planner. Use-condition checking determines whether the task and observation are in scope and rejects incompatible requests. Argument extraction fills tool arguments and rejects tools with unresolved arguments. Lifecycle checks require the Tool-Use Contract to permit automatic use. Risk-Level checks require the Tool-Use Contract to allow execution under the current conditions. Finally, Tool-Use Contract compatibility checks whether the tool can produce the requested output.

For an instruction x, page observation o, and tool t, the gate is a Boolean predicate:

$$\begin{aligned}\mathrm{Pass}(x,o,t) \iff & U(x,o,t) \wedge S(x,t) \wedge L(t) \\ & \wedge R(x,o,t) \wedge C(x,o,t).\end{aligned} \tag{3}$$

Here U denotes use-condition checking, S argument extraction, L lifecycle checking from the Tool-Use Contract, R Risk-Level checking from the Tool-Use Contract, and C Tool-Use Contract

Table 3: Main results on the 157-task ContractWebBench shopping benchmark. The success-rate and tool-use columns are percentages over 157 tasks; calls/tokens count runtime critical-path LLM calls only.

Setting	Success rate (%)	Tool used (%)	Calls / tokens
No-tool WebAgent	44.6	–	2,051 / 9,926,857
MSR-Web-Verbs (Jiang et al., 2026)	48.4	–	83 / 842,597
ContractSkill (Lu et al., 2026)	62.4	–	520 / 1,769,526
SkillWeaver (Zheng et al., 2025)	68.8	–	724 / 5,890,400
Trace4Tool	78.3	91.1	331 / 1,578,470

compatibility. Zero-LLM runtime execution is used only when exactly one candidate satisfies $\mathrm{Pass}(x, o, t)$; otherwise, the tool selector abstains or falls back. For unsupported Tool-Use Contracts, rejecting all incompatible tools is correct and is reported as a rejection decision, not tool success.

The tool selector uses lightweight deterministic string, schema, and status checks before any LLM planner call. This selection step avoids costly planner calls and browser exploration when a validated tool is available.

Controlled LLM use and fallback. Trace4Tool does not remove LLMs from web agents; it uses them at controlled decision points. If deterministic checks are insufficient, LLM-assisted matching or the WebAgent planner may canonicalize the task, rank candidate tools, extract arguments, or continue browsing. LLM outputs remain constrained by schema validation and Tool-Use Contract compatibility, so a rejected tool cannot re-enter through another route. If no compatible tool exists, a correct fallback answer counts toward success rate but not tool success.

Failure write-back. Runtime failures are written back as retrieval, use-condition, argument extraction, tool-implementation execution, validation, or upstream-site errors. These categories localize maintenance without task-id-specific patches: contract mismatch suggests refining the Tool-Use Contract or rejection rules, unresolved arguments suggest improving argument extraction or requiring earlier fallback, and execution failures suggest updating the site-specific tool implementation.

3.4 ContractWebBench and Evidence Analysis

Benchmark construction. To evaluate synthesis and runtime reuse, we construct **ContractWebBench**, a WebArena-style benchmark for Tool-Use-Contract-guided tool reuse. Starting from shopping tasks (Zhou et al., 2023), we organize tasks by Tool-Use Contracts, difficulty, and evidence. Source tasks validate fixed tool seeds and Tool-Use Contracts; held-out variants support first-pass and post-maintenance evaluation. Reference answers and held-out task identifiers are not runtime tool inputs. The benchmark covers navigation, live extraction, form filling, bounded state changes, adjusted login/static tasks with live probes, and small compositions, matching the focus of the method on contract-valid evidence.

Difficulty levels. ContractWebBench uses L0–L4 levels. L0 validates seed or source tools. L1 covers close same-contract variants with argument, entity, category, or paraphrase changes. L2 contains semantically distant or harder parameterized variants. L3 requires richer outputs or longer chains within a task type, and L4 composes multiple Tool-Use Contracts. Drift and negative probes test fail-closed behavior but are excluded from the positive denominator.

Evidence analysis. The analysis separates live page evidence, browser-state evidence, live-probed adjusted tasks, snapshot diagnostics, static capability diagnostics, and auth-confounded cases. Diagnostic artifacts are reported separately from live or browser-state evidence, keeping evidence explicit.

Evaluation modes. We report first-pass and post-maintenance results separately. First-pass evaluation fixes the tool library before held-out tasks are run. Under post-maintenance evaluation, failures may update only the responsible Tool-Use Contract, tool implementation, tool selector, argument extractor, or validator, and the results are reported separately. This prevents maintenance after failure from being counted as first-pass generalization. We also separate task success from tool success: a fallback answer can succeed without demonstrating reusable tool use.

Maintenance protocol. Maintenance is local and evidence-based. After first-pass, each failure is assigned to coverage, contract mismatch, unresolved arguments, tool-implementation execution, validator/finalizer mismatch, or site/reference drift. Re-

pairs may update a Tool-Use Contract, adjust rejection rules, expose or hide a tool, update the tool selector, improve an argument extractor, update a parser/finalizer, or relabel evidence. Maintenance cannot insert held-out answers, task-id-specific cached outputs, or shortcuts that bypass the Tool-Use Contract. Repaired tasks or contract groups are rerun only in the post-maintenance condition; Appendix D.2 gives an end-to-end example.

4 Experiments

4.1 Setup

Benchmark. We evaluate Trace4Tool on a 157-task ContractWebBench shopping benchmark organized by Tool-Use Contracts: 40 L0 seed/source tasks, 98 L1 same-contract variants, 14 L2 harder variants, one L3 richer-output task, two L4 cross-contract or long-horizon tasks, and two clean live-interaction tasks. Drift and negative probes are reserved for fail-closed diagnostics and excluded from the positive denominator.

Systems and metrics. The main runtime agent model is GPT-5.3-Codex accessed via API, while tool synthesis uses GPT-5.5. The no-tool WebAgent baseline uses the same site access mode, task configurations, authentication state, evaluators, and 15-action budget, but has no access to Trace4Tool tools. We also adapt MSR-Web-Verbs (Jiang et al., 2026), ContractSkill (Lu et al., 2026), and SkillWeaver (Zheng et al., 2025) under the same setting. We report success rate, tool use, runtime LLM calls/tokens, compatible tool success, correct rejection, evidence, and failure categories; zero-LLM runtime execution requires no planner, decision, or tool-selector model calls at runtime.

4.2 Main Results

Table 3 summarizes the 157-task shopping benchmark. The external reuse baselines improve over the no-tool WebAgent, with MSR-Web-Verbs reaching 76/157, ContractSkill reaching 98/157, and SkillWeaver reaching 108/157 successful tasks. Trace4Tool further reaches 123/157 successful tasks while using 331 runtime LLM calls and 1.58M effective tokens, substantially below the 2,051 calls and 9.93M tokens of the no-tool WebAgent. This suggests that Tool-Use Contract matching and evidence validation improve both correctness and runtime exposure beyond reusable skill retrieval alone. Calls/tokens count runtime critical-path LLM use only; tool synthesis, contract drafting, metadata refinement, candidate ranking, and post-hoc evaluator checks are excluded.

Table 4: Rejection diagnostic on a 60-task site-independent Shopping set. Values are percentages over all diagnostic tasks.

Outcome	Share (%)
Correct rejection	51.7
Coverage/runtime gap	28.3
Wrong/questionable tool use	3.3

Table 5: Non-shopping site checks. Success rates are reported separately from the maintenance checks in Table 6.

Site	Tasks	Success rate (%)
Admin	45	80.0
GitLab	48	72.9
Postmill	30	70.0
Non-shopping total	123	74.8

Table 6: Separately reported maintenance checks. These results are not merged into the primary tables.

Task set	Tasks	Success rate (%)
ContractWebBench shopping	157	98.7
Admin	45	100.0
GitLab	48	100.0
Postmill	30	100.0
Non-shopping total	123	100.0

Figure 3 highlights the same efficiency trend visually: under unique Tool-Use Contract matches, Trace4Tool avoids runtime planner calls, while fallback or non-matched cases account for the remaining runtime LLM use.

Table 4 reports Shopping60 as a rejection diagnostic, not a primary success-rate benchmark. Trace4Tool rejects incompatible requests instead of forcing unsafe tool use; Appendix C.4 gives details.

Table 5 tests whether the Tool-Use-Contract-guided tool selector extends beyond shopping across Admin, GitLab, and Postmill tasks.

4.3 Analysis

Difficulty and failures. Trace4Tool solves all L0 seed/source and L3/L4 tasks, missing one L1 task and one L2 task; Appendix C.3 gives the breakdown. The remaining errors occur in cases where the requested result is close to an existing contract but still requires a different evidence boundary or argument interpretation. The evidence analysis keeps references as validation targets, not runtime

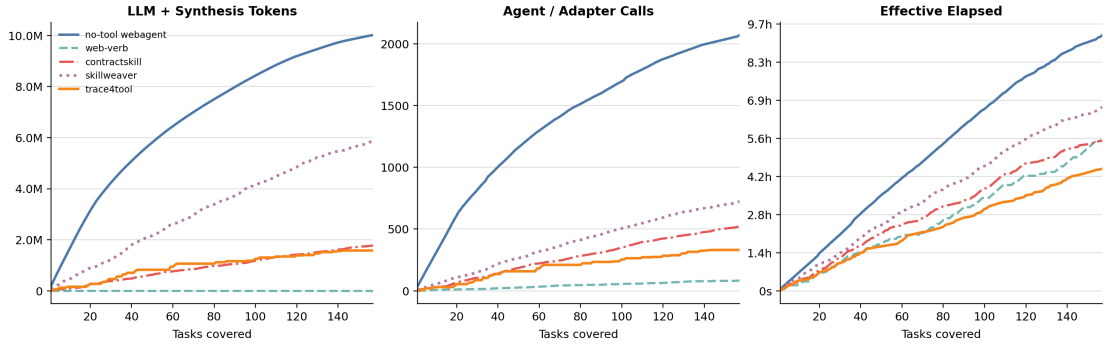

Figure 3: Runtime efficiency on the 157-task shopping benchmark. Calls and tokens count only runtime critical-path LLM usage.

Table 7: Tool-Use-Contract mechanism ablation on a 64-task diagnostic subset. Each ablation removes one component from Full Trace4Tool (maintained); percentages are over completed runs except for the Completed column.

Setting	Completed (%)	Success rate (%)	Tool success (%)	LLM calls
Full Trace4Tool	100.0	100.0	100.0	0
No direct contract matching	45.3	31.0	0.0	263
No argument binding	90.6	31.0	0.0	514
No parameterized tools	4.7	0.0	0.0	36

arguments, and localizes failures to coverage, selection, argument extraction, execution, evidence mismatch, or evaluator formatting.

Runtime exposure. No-tool runs repeatedly expose the agent to long observations, planner calls, retries, and browser-state drift. Trace4Tool reduces this exposure when a validated Tool-Use Contract uniquely matches the task and observation. This effect is most visible on repetitive navigation and extraction tasks, where the same page structure can be reused without repeatedly sending large DOM or screenshot contexts to the planner.

Mechanism ablation. Table 7 ablates Tool-Use-Contract components on a 64-task subset already covered by the maintained tool library, so the 100.0% row is a diagnostic reference point rather than a primary result. Removing direct contract matching reintroduces planner calls; removing argument binding leaves contracts without reliable arguments; fixed tools remove parameterized reuse. These results show that the gains come from contract matching, argument binding, and validated execution together.

Qualitative coverage. The benchmark covers navigation, live extraction, bounded form/state actions, and small compositions. These task groups test whether Tool-Use Contracts support reuse across both read-only extraction and controlled state-changing workflows. The state-changing tasks are intentionally bounded by account state and validators, so their results are interpreted as evidence for controlled reuse rather than unrestricted browser automation.

5 Conclusion

We introduced Trace4Tool, a Tool-Use-Contract-guided framework for converting web-agent trajectories into reusable tools. A Trace4Tool tool pairs an implementation with a verifiable Tool-Use Contract, so reuse depends on task and observation compatibility rather than task wording, page similarity, or replayed actions.

At runtime, the selector uses zero-LLM execution only under a unique match to a validated Tool-Use Contract; otherwise, it abstains or falls back to standard agent control. ContractWebBench results show that this design improves success rate, reduces runtime LLM calls on matched tasks, and makes failures local to contracts, implementations, argument extractors, selectors, or validators. Trace4Tool therefore offers a path from trajectory memory to executable, evidence-bounded tool reuse.

Limitations

Trace4Tool is designed for web tasks with recurring interaction patterns, identifiable parameters, and verifiable outputs. Open-ended browsing, subjective judgment, highly personalized workflows, or rapidly changing pages may rely more heavily on fallback planning than on direct tool reuse. The method also depends on Tool-Use Contracts with observable evidence; tasks with ambiguous success criteria or evidence that is not visible from the page are less suitable for direct tool reuse.

State-changing tasks require isolated accounts, reset protocols, and stricter validators to avoid contamination across runs. Cross-site reuse should be understood as the transfer of task intent, arguments, and Tool-Use Contracts, not as direct replay of the same browser actions; a new website may require a new tool implementation even when the task intent and Tool-Use Contract transfer. Zero-LLM runtime execution is therefore available for unique Tool-Use Contract matches, not as a constraint on fallback behavior. For online tasks without an external verifier, Trace4Tool can synthesize candidate tools with evidence, but those candidates should not be counted as benchmark-valid until they are validated by live evidence, held-out tasks, or human analysis.

Ethics and Responsible Research

Trace4Tool studies web-agent automation, which can introduce risks if used outside controlled research settings, especially for authenticated or state-changing workflows. We mitigate these risks by evaluating on benchmark and isolated accounts, using reset protocols for state-changing tasks, requiring Tool-Use Contract validation before tool use, and reporting fallback or abstention when no compatible tool is available. The intended use of the artifacts is research on reliable web-agent reuse, not production automation on third-party websites without authorization.

The work uses and creates scientific artifacts. Existing benchmark and web-agent artifacts are cited where introduced, and ContractWebBench documents task groups, evidence, and evaluation modes in the appendix. We do not intentionally collect personal data from real users, and the benchmark tasks use synthetic or controlled site/account state. Logs and derived artifacts should exclude credentials, private account details, and raw site snapshots whose redistribution is not permitted. AI assistants were used for coding and writing support; all experimental claims, citations, and final text remain the responsibility of the authors.

References

Mohamed Aghzal, Gregory J. Stein, and Ziyu Yao. 2026. Why do llm-based web agents fail? a hierarchical planning perspective. *Preprint*, arXiv:2603.14248.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2023. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*.

Rongqian Chen, Yu Li, Zeyu Fang, Sizhe Tang, Weidong Cao, and Tian Lan. 2026a. Intentscore: Intent-conditioned action evaluation for computer-use agents. *Preprint*, arXiv:2604.05157.

Tianyi Chen, Yinheng Li, Michael Solodko, Sen Wang, Nan Jiang, Tingyuan Cui, Junheng Hao, Jongwoo Ko, Sara Abdali, Leon Xu, Suzhen Zheng, Hao Fan, Pashmina Cameron, Justin Wagle, and Kazuhito Koishida. 2026b. Cua-skill: Develop skills for computer using agent. *Preprint*, arXiv:2601.21123.

Yurun Chen, Zeyi Liao, Ping Yin, Taotao Xie, Keting Yin, and Shengyu Zhang. 2026c. Safepred: A predictive guardrail for computer-using agents via world models. *Preprint*, arXiv:2602.01725.

Jiali Cheng, Anjishnu Kumar, Roshan Lal, Rishi Rajasekaran, Hani Ramezani, Omar Zia Khan, Oleg Rokhlenko, Sunny Chiu-Webster, Gang Hua, and Hadi Amiri. 2025. Webatlas: An llm agent with experience-driven memory and action simulation. In *NeurIPS 2025 Workshop on Language Agents and World Models*.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. 2024. SeeClick: Harnessing GUI grounding for advanced visual GUI agents. *arXiv preprint arXiv:2401.10935*.

Chaoqun Cui, Jing Huang, Shijing Wang, Liming Zheng, Qingchao Kong, and Zhixiong Zeng. 2026. Agentic reward modeling: Verifying gui agent via online proactive interaction. *Preprint*, arXiv:2602.00575.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2Web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114.

Xuwei Ding, Skylar Zhai, Linxin Song, Jiate Li, Taiwei Shi, Nicholas Meade, Siva Reddy, Jian Kang, and Jieyu Zhao. 2026. The blind spot of agent safety: How benign user instructions expose critical vulnerabilities in computer-use agents. *Preprint*, arXiv:2604.10577.

Tanmay Gupta, Piper Wolters, Zixian Ma, Peter Sushko, Rock Yuren Pang, Diego Llanes, Yue Yang, Taira

Anderson, Boyuan Zheng, Zhongzheng Ren, Harsh Trivedi, Taylor Blanton, Caleb Ouellette, Winson Han, Ali Farhadi, and Ranjay Krishna. 2026. Molmoweb: Open visual web agent and open data for the open web. *Preprint*, arXiv:2604.08516.

Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2023. A real-world webagent with planning, long context understanding, and program synthesis. *arXiv preprint arXiv:2307.12856*.

Shibo Hao, Zhining Zhang, Zhiqi Liang, Tianyang Liu, Yuheng Zha, Qiyue Gao, Jixuan Chen, Zilong Wang, Zhoujun Cheng, Haoxiang Zhang, Junli Wang, Hexi Jin, Boyuan Zheng, Kun Zhou, Yu Wang, Feng Yao, Licheng Liu, Yijiang Li, Zhifei Li, and 12 others. 2026. Cocoabench: Evaluating unified digital agents in the wild. *Preprint*, arXiv:2604.11201.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. WebVoyager: Building an end-to-end web agent with large multimodal models. *arXiv preprint arXiv:2401.13919*.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and 1 others. 2024. CogAgent: A visual language model for GUI agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14281–14290.

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

Linxi Jiang, Rui Xi, Zhijie Liu, Shuo Chen, Zhiqiang Lin, and Suman Nath. 2026. Web verbs: Typed abstractions for reliable task composition on the agentic web. *Preprint*, arXiv:2602.17245.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. 2024. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*.

Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and 1 others. 2024. AutoWebGLM: A large language model-based web navigating agent. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5295–5306.

Yanda Li, Chi Zhang, Wanqi Yang, Bin Fu, Pei Cheng, Xin Chen, Ling Chen, and Yunchao Wei. 2024. AppAgent v2: Advanced agent for flexible mobile interactions. *arXiv preprint arXiv:2408.11824*.

Zichuan Lin, Feiyu Liu, Yijun Yang, Jiafei Lyu, Yiming Gao, Yicheng Liu, Zhicong Lu, Yangbin Yu, Mingyu Yang, Junyou Li, Deheng Ye, and Jie Jiang. 2026. Ui-voyager: A self-evolving gui agent learning via failed experience. *Preprint*, arXiv:2603.24533.

Renze Lou, Baolin Peng, Wenlin Yao, Qianhui Wu, Hao Cheng, Suman Nath, Wenpeng Yin, and Jianfeng Gao. 2026. The tool illusion: Rethinking tool use in web agents. *Preprint*, arXiv:2604.03465.

Xing Han Lu, Zdenek Kasner, and Siva Reddy. 2024. WebLINX: Real-world website navigation with multi-turn dialogue. *arXiv preprint arXiv:2402.05930*.

Zijian Lu, Yiping Zuo, Yupeng Nie, Xin He, Weibei Fan, Lianyong Qi, and Shi Jin. 2026. Contractskill: Repairable contract-based skills for multimodal web agents. *Preprint*, arXiv:2603.20340.

Hadi Nekoei, Aman Jaiswal, Patrice Bechard, Oleh Shliazhko, Orlando Marquez Ayala, Mathieu Reymond, Massimo Caccia, Alexandre Drouin, Sarath Chandar, and Alexandre Lacoste. 2025. Jef-hinter: Leveraging offline knowledge for improving web agents adaptation. *Preprint*, arXiv:2510.04373.

Viraj Prabhu, Yutong Dai, Matthew Fernandez, Jing Gu, Krithika Ramakrishnan, Yanqi Luo, Silvio Savarese, Caiming Xiong, Junnan Li, Zeyuan Chen, and Ran Xu. 2025. Walt: Web agents that learn tools. In *ICLR 2026*.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. 2024. Agent Q: Advanced reasoning and learning for autonomous AI agents. *arXiv preprint arXiv:2408.07199*.

Corby Rosset, Pratyusha Sharma, Andrew Zhao, Miguel Gonzalez-Fernandez, and Ahmed Awadallah. 2026. The art of building verifiers for computer use agents. *Preprint*, arXiv:2604.06240.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36.

Yibo Shi, Jungang Li, Linghao Zhang, Zihao Dongfang, Biao Wu, Sicheng Tao, Yibo Yan, Chenxi Qin, Weiting Liu, Zhixin Lin, Hanqian Li, Yu Huang, Song Dai, Yonghua Hei, Yue Ding, Xiang Li, Shikang Wang, Chengdong Xu, Jingqi Liu, and 9 others. 2026. Androtmem: From interaction trajectories to anchored memory in long-horizon gui agents. *Preprint*, arXiv:2603.18429.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*.

Yueqi Song, Frank F. Xu, Shuyan Zhou, and Graham Neubig. 2025. Beyond browsing: API-based web agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 11066–11085.

Yunteng Tan, Zhi Gao, and Xinxiao Wu. 2026. Enhancing web agents with a hierarchical memory tree. *Preprint*, arXiv:2603.07024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Junjie Wang, Zequn Xie, Dan Yang, Jie Feng, Yue Shen, Duolin Sun, Meixiu Long, Yihan Jiao, Zhehao Tan, Jian Wang, Peng Wei, and Jinjie Gu. 2026a. Webclipper: Efficient evolution of web agents with graph-based trajectory pruning. *Preprint*, arXiv:2602.12852.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158*.

Xiaoce Wang, Guibin Zhang, Junzhe Li, Jinzhe Tu, Chun Li, and Ming Li. 2026b. Tooltok: Tool tokenization for efficient and generalizable gui agents. *Preprint*, arXiv:2602.02548.

Han Xiao, Guozhi Wang, Hao Wang, Shilong Liu, Yuxiang Chai, Yue Pan, Yufeng Zhou, Xiaoxin Chen, Yafei Wen, and Hongsheng Li. 2026. Ui-mem: Self-evolving experience memory for online reinforcement learning in mobile gui agents. *Preprint*, arXiv:2602.05832.

Dawei Yan, Haokui Zhang, Guangda Huzhang, Yang Li, Yibo Wang, Qing-Guo Chen, Zhao Xu, Weihua Luo, Ying Li, Wei Dong, and Chunhua Shen. 2026. M^2: Dual-memory augmentation for long-horizon web agents via trajectory summarization and insight retrieval. *Preprint*, arXiv:2603.00503.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*.

Peng Yuan, Yuyang Yin, Yuxuan Cai, and Zheng Wei. 2026. Webforge: Breaking the realism-reproducibility-scalability trilemma in browser agent benchmark. *Preprint*, arXiv:2604.10988.

Yuwen Zhai, Runze Li, Liang Wang, Nian Shi, Liwu Xu, Wei Zhang, Ran Lin, Bo Xu, and Benlei Cui. 2026. Guide: Interpretable gui agent evaluation via hierarchical diagnosis. *Preprint*, arXiv:2604.04399.

Jiayuan Zhang, Kaiquan Chen, Zhihao Lu, Enshen Zhou, Qian Yu, and Jing Zhang. 2025. Prune4web: Dom tree pruning programming for web agent. In *AAAI 2026*.

Yao Zhang, Shijie Tang, Zeyu Li, Zhen Han, and Volker Tresp. 2026a. Webarbiter: A principle-guided reasoning process reward model for web agents. In *ICLR 2026*.

Yuzhe Zhang, Xianwei Xue, Xingyong Wu, Mengke Chen, Chen Liu, Xinran He, Run Shao, Feiran Liu, Huanmin Xu, Qiutong Pan, and Haiwei Wang. 2026b. Don't act blindly: Robust gui automation via action-effect verification and self-correction. In *ACL 2026*.

Zirui Zhao, Jun Hao Liew, Yan Yang, Wenzhuo Yang, Ziyang Luo, Doyen Sahoo, Silvio Savarese, and Junnan Li. 2026. Gpa: Learning gui process automation from demonstrations. *Preprint*, arXiv:2604.01676.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. 2025. Skillweaver: Web agents can self-improve by discovering and honing skills. *Preprint*, arXiv:2504.07079.

Hongbin Zhong, Fazle Faisal, Luis França, Tanakorn Leesatapornwongsa, Adriana Szekeres, Kexin Rong, and Suman Nath. 2026. Actionengine: From reactive to programmatic gui agents via state machine memory. *Preprint*, arXiv:2602.20502.

Shuyan Zhou. 2026. Webarena-infinity: Generating browser environments with verifiable tasks at scale. In *Blog Post*.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2023. WebArena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

Appendices

The appendices provide supplementary definitions, diagnostic results, representative cases, artifact records, and responsible research details. Appendix A defines benchmark levels and evidence. Appendix B defines evaluation modes and tool-success metrics. Appendix C reports diagnostic evaluations. Appendix D provides representative cases, an end-to-end price-range example, and artifact records. Appendix E documents artifact use, computational reporting, human-subject status, and AI-assistant use.

A Benchmark and Evidence Definitions

A.1 Benchmark Levels

ContractWebBench uses levels to separate source tasks, close generalization, harder semantic variants, and composition. The positive shopping benchmark contains 157 tasks; drift and negative probes are used for fail-closed diagnostics and are not included in the positive denominator.

A.2 Source and Evaluation Separation

ContractWebBench is built from WebArena-style shopping tasks and controlled task variants organized by Tool-Use Contract. L0 tasks serve as seed/source tasks for validating fixed tools and source contracts. L1–L4 tasks and the additional live tasks are used to evaluate close variants, harder semantic variants, richer outputs, and small compositions. References are drawn from original benchmark configurations, live page or browser-state probes, programmatic evidence checks, and evidence analysis. During synthesis, historical trajectories and task configurations may be used to propose contracts and tools, but held-out reference answers and task identifiers are not used as runtime tool arguments. Reference answers are reserved for post-hoc validation, semantic adjudication, and failure categorization. Ambiguous reference-conflict cases are retained rather than being folded into tools. This source/evaluation separation is intended to prevent leakage from benchmark labels into tool execution.

Table 8: ContractWebBench level definitions in the positive split.

Level	Tasks	Definition
L0	40	Seed or source tasks used to validate fixed tools and source contracts.
L1	98	Same-contract variants, including argument, entity, category, status, number, date-window, or close paraphrase changes.
L2	14	Semantically distant or harder parameterized tasks that stress unique Tool-Use Contract matching, argument extraction, and rejection rules.
L3	1	Longer or richer-output tasks within a primary contract type.
L4	2	Cross-type or long-horizon tasks that compose multiple Tool-Use Contracts.
Additional live tasks	2	Additional live-interaction tasks included in the 157-task benchmark.

A.3 Evidence and Validity

The evidence analysis records whether results are supported by live pages, browser state, snapshot diagnostics, or account-dependent state. Success rate is the main metric, while strict string success is retained for diagnosis. Live evidence comes from the current website state; the evaluator is the benchmark scoring mechanism. When live evidence and an old reference disagree, we report drift or evidence mismatch rather than forcing the tool to reproduce the stale reference.

B Evaluation Modes and Tool-Success Metrics

B.1 Evaluation Modes

Trace4Tool separates deployable generalization from maintenance. This distinction is important because a tool library can improve after failures are inspected, but that improvement should not be counted as if it had succeeded on the first-pass run. Post-maintenance results are not pass@2: the denominator is fixed, failures are classified, and only the responsible Tool-Use Contract, adapter, tool selector, argument extractor, validator, or evidence label is patched before rerun.

B.2 Tool Success and Contract Metrics

The main tables report tool-path and tool-success behavior. Additional contract-level fields analyze whether the selected tool belongs to the expected capability group or contract type. Metadata helps retrieve candidates, while the full Tool-Use Contract determines valid reuse. Table 11 defines the extended reporting metrics.

Table 9: Evidence used in ContractWebBench analysis.

Evidence	Meaning	Reporting role
Live/browser evidence	Result is supported by a live page, DOM, URL, programmatic HTML, or verified browser state.	Reported separately as a live-interaction result.
Adjusted login/static	Admin, order, static capability, or formerly disputed tasks after live probes or live extraction were added.	Included in the 157-task positive benchmark, with evidence reporting.
Snapshot diagnostic	A result is supported by stored snapshot evidence rather than fresh live page evidence.	Reported as diagnostic unless promoted by live verification.
Auth-confounded diagnostic	The result depends on mismatched login or account state.	Rerun with corrected auth state or excluded from live claims.
Strict-string diagnostic	Raw evaluator/string equality without semantic equivalence.	Used to inspect formatting and reference issues, not as the main success-rate metric.

Table 10: Evaluation modes used for reporting Trace4Tool results.

Mode	Definition	Reporting rule
No-tool baseline	The WebAgent runs without Trace4Tool tools.	Paired denominator for measuring correctness and cost gains.
With-tool first-pass	The tool library is fixed before held-out tasks are run.	Evidence for deployable generalization.
With-tool post-maintenance	Failures may update contracts, tool implementations, the tool selector, argument extractors, validators, or evidence labels before rerun.	Reported as maintenance lift, not retroactively merged into first-pass.
Additional post-maintenance checks	The post-maintenance tool library is evaluated on additional task groups or sites.	Evidence of coverage potential, reported separately from first-pass results.

Table 11: Tool success and contract-level metrics.

Metric	Definition
tool_success	The task succeeds, uses a Trace4Tool tool, and the selected tool is compatible with the expected Tool-Use Contract or task group.
contract_selection_accuracy	The selected Tool-Use Contract matches the annotated expected contract type, regardless of final answer validity.
contract_reuse_count	Number of held-out tasks successfully solved with the same Tool-Use Contract after its seed or source task.
fallback_success	The task succeeds without tool success.
no_metadata_success	Success when metadata is removed or disabled, used to test whether verbose metadata contributes beyond the tool implementation and other contract fields.

B.3 Failure Signals and Maintenance Actions

The analysis metrics are designed to make failures actionable. Table 12 maps common failure signals to maintenance actions. This mapping is diagnostic; first-pass results are not changed retroactively after maintenance.

C Diagnostic Results

C.1 Shopping Primary and Maintenance Results

The 157-task shopping benchmark separates the primary result from the separately reported maintenance check. The maintenance result is not merged back into the primary table. The benchmark-level scores are reported in Table 3; the appendix focuses on diagnostic decompositions and artifact records below.

C.2 Completed and Diagnostic Baseline Rows

The completed baseline rows are included in Table 3, including MSR-Web-Verbs, ContractSkill, and SkillWeaver. The additional no-tool Qwen3-VL diagnostic row obtains a 0.0% success rate over the same 157-task shopping denominator. External systems that did not produce completed benchmark rows under comparable conditions are not included.

Table 12: Failure signals and corresponding maintenance actions.

Failure signal	Likely cause	Maintenance action
Tool not used, fallback succeeds	Coverage or tool selector gap	Update metadata, use conditions, or the tool selector.
Wrong tool selected	Contract mismatch	Add rejection rules or split the Tool-Use Contract.
Required argument missing	Argument extraction failure	Improve the argument extractor or require fallback.
Tool used, task fails	Execution, parser, or evidence error	Update tool implementation, parser, finalizer, or validator.
Strict-string fail, success-rate pass	Formatting or evaluator mismatch	Keep strict-string success as a diagnostic metric.
Live evidence differs from reference	Site state drift or reference conflict	Report as a boundary case and preserve evidence.
Snapshot-only success	Diagnostic evidence	Report separately from live tool success.

Table 13: Maintenance success rate by ContractWebBench subset. Scores are percentages within each subset.

Subset	Tasks	Trace4Tool (%)
L0	40	100.0
L1	98	99.0
L2	14	92.9
L3	1	100.0
L4	2	100.0
Additional live tasks	2	100.0

C.3 Maintenance Success Rate by Benchmark Subset

Table 13 decomposes the separately reported shopping maintenance check by benchmark level. The split is intended to diagnose where reuse is reliable rather than to introduce a separate benchmark or duplicate the primary result. Trace4Tool is stable on source tasks, close variants, and the small higher-level slices, with the remaining errors concentrated in one L1 task and one harder L2 task. The L3 and L4 rows are small, so they should be read as case-level coverage evidence rather than broad claims about richer or composed tasks.

C.4 Site-Independent Shopping Rejection Diagnostic

The 60-task site-independent Shopping set is a rejection and coverage diagnostic, not a primary first-pass success-rate benchmark. The denominator is complete, but the aggregate combines historical/resume evidence, corrected storefront reruns, maximum-action rejection-rule runs, and auth-refreshed runs. It also does not include a complete paired no-tool baseline. We therefore report clean tool use and correct rejection separately, and we do not pool them into a single success rate. Table 4 reports the main rejection and failure shares; the complementary tool-use share under unique Tool-Use Contract matches is 16.7%, so the four outcome classes sum to 100.0%.

The 17 coverage/contract/runtime gaps comprise 12 covered rows without an exposed tool, 3 selected-but-unsupported argument/runtime cases, and 2 selected-tool reference-mismatch cases. The post-hoc labels "covered existing contract" and "incompatible existing contract" are design labels for what the diagnostic intended to probe, not proof that a working tool was available. Correct rejection is rejection-rule correctness rather than end-task success; conversely, wrong or questionable tool use is the tool-selector failure mode that this diagnostic is designed to surface.

C.5 Live Task-Group Coverage

The live/browser set contains 117 tasks. Table 14 reports the 115-task task-group coverage view used for family analysis; the remaining two additional live tasks are included in the 157-task benchmark totals but are not assigned to the original 25-group coverage table. This set is useful because every task has live or browser-state evidence.

C.6 Diagnostic Tool Library Snapshot Evaluation

Tool library snapshot evaluations are useful for checking the tool selector, reporting, and bookkeeping. They are diagnostic because their evidence differs from live page or browser-state evidence. Table 15 therefore should not be read as a live or fresh tool-use result.

C.7 Reduced Mechanism Ablation

Table 16 reports a reduced diagnostic ablation over 64 non-shopping tool library snapshot rows: 35 Admin, 20 GitLab, and 9 Postmill. Each ablation removes one Tool-Use-Contract-related component

Table 14: Task-group coverage in the live-interaction set.

Type	Tasks	Task groups
Navigation	40	category_browse_navigation; category_filter_extreme; category_price_filter_navigation; category_sorted_listing_navigation; search_query_navigation; search_sorted_listing_navigation; switch_game_card_capacity_navigation; capacity_constrained_lowest_price_navigation.
Live extraction	33	price_range; product_names_price_range_extraction; product_rating_lookup; product_criticism_extraction; reviewer_by_star_filter; reviewer_keyword_filter; reviewer_review_signal_filter.
State/form	40	wishlist_current_product_addition; wishlist_exact_product_addition; wishlist_generic_product_addition; cart_lowest_unit_price_open_tabs_addition; account_address_update; product_review_submission; contact_coupon_draft; contact_refund_draft.
Composition	2	long_horizon_composed; cross_family_price_range_rating.

Table 15: Tool library snapshot diagnostic evaluation. Scores are percentages within each site; diagnostic evidence differs from live/browser-state evidence.

Site	Tasks	With-tool success rate (%)	Tool success (%)	Zero-LLM runtime execution (%)	No-tool success rate (%)
Admin	86	98.8	98.8	100.0	51.9
GitLab	40	100.0	97.5	97.5	47.5
Postmill	9	100.0	100.0	100.0	66.7
Total	135	99.3	98.5	99.3	51.5

from Full Trace4Tool. The completed column is the percentage of runs that produced a result summary under the same reporting format. Runs outside the completed denominator are completion or reporting failures. The row without metadata matches the full system, supporting the interpretation that metadata mainly supports retrieval, analysis, and maintenance rather than contract validity itself.

D Representative Cases and Artifacts

D.1 Representative Cases

Table 19 gives examples that illustrate why Tool-Use Contracts, evidence, rejection rules, and validators are necessary. The cases are not intended to replace aggregate results; they make the rejection rules concrete.

D.2 End-to-End Example: Price-Range Tool Lifecycle

The price-range family illustrates the full Trace4Tool lifecycle. A representative task asks for the price range of a product query, such as Canon photo printers. The Tool-Use Contract is read-only: it applies to search-result product cards, requires query and relevance arguments, returns minimum and maximum prices with supporting product evidence, and rejects review, wishlist, cart, order, account, and category-extreme requests.

Offline synthesis first creates fixed tool seeds from source tasks. A fixed seed preserves the concrete query and action pattern, such as extracting prices from search-result pages for a specific product query. After validation, Trace4Tool promotes the seed into a parameterized tool with query and relevance arguments. At runtime, the task and observation uniquely match the price-range Tool-Use Contract, so Trace4Tool selects the tool, extracts live evidence, and uses zero-LLM runtime execution.

Maintenance is also contract-level rather than answer-level. Table 18 shows task125, where an overly broad query-terms contract returned the wrong range. Maintenance blocked the broad contract for that case, introduced a narrower title-anchor price-range contract, and preserved zero-LLM runtime execution. The final answer is not cached; the tool still extracts and validates evidence under the refined contract.

D.3 Artifact Records

The evaluation summary combines fresh baseline reruns with retained live/browser artifacts. The machine-readable summary records per-task success, evidence, and level metadata. The benchmark manifest records task group, level, and evidence information. Runtime behavior is implemented by the WebAgent runtime, tool library loader, candi-

Table 16: Tool-Use-Contract mechanism ablation on 64 diagnostic rows. Completed is over all tasks; success rate, tool success, and zero-LLM runtime execution are over completed runs.

Setting	Tasks	Completed (%)	Success rate (%)	Strict-string success (%)	Tool success (%)	Zero-LLM runtime execution (%)	Calls	Tokens
Full Trace4Tool	64	100.0	100.0	100.0	100.0	100.0	0	0
Parameterized only	64	100.0	100.0	100.0	100.0	100.0	0	0
No metadata	64	100.0	100.0	100.0	100.0	100.0	0	0
No direct contract matching	64	45.3	31.0	31.0	0.0	0.0	263	1,001,841
No argument binding	64	90.6	31.0	31.0	0.0	0.0	514	1,876,566
No parameterized tools	64	4.7	0.0	0.0	0.0	0.0	36	138,840

Table 17: Remaining with-tool invalid cases in the post-maintenance shopping evaluation.

Task	Group	Case explanation
51	fulfilled_order_spend_by_window	Live evidence and the WebArena reference disagree on the order-count and spend window; the tool reports the live contract result rather than forcing the old reference.
204	order_attribute_by_status	The current admin order detail no longer matches the historical reference; the no-tool run also reads the current state and fails under the success-rate evaluator.

Table 18: Price-range maintenance example for task125. Runtime calls and tokens are counts.

Stage	Final answer	Selected contract/tool	LLM calls	Tokens
Before post-maintenance repair	$7.99 to $89	Broad query-terms price-range contract	0	0
After post-maintenance repair	$1.46 to $85	Narrow title-anchor price-range contract	0	0

Table 19: Representative cases for interpreting Trace4Tool behavior.

Case	Mechanism	Why it matters
category_filter_extreme	A fixed category-price trajectory is promoted into a parameterized tool with category and price-direction arguments.	Shows fixed-before-parameterized synthesis and why category extreme navigation differs from price-range extraction.
account_address_update	Address arguments drive a form-filling tool whose output is accepted only after account-page state evidence passes validation.	Demonstrates that the method covers bounded state changes, not only read-only navigation.
admin_review_status_count	The adjusted tool clears persisted admin-grid filters, applies the requested status/detail filter, and reads live record counts.	Shows how a disputed answer-style tool is converted into live evidence rather than tool-library lookup.
product_review_absence	Target-product relevance gates prevent nearby products or accessories from being counted as evidence for an absence claim.	Illustrates rejection rules for N/A and absence contracts.
task51 and task204	The remaining with-tool invalid cases expose a reference count/spend inconsistency and current live order-state drift.	Shows that Trace4Tool preserves evidence instead of forcing cached references into the answer.

date packaging, parameterization, contract inference, tool selector, and metadata modules. These artifacts allow the main tables to be traced back to concrete task-level runs and to the tool payloads used at runtime.

E Responsible Research and Reproducibility Details

E.1 Artifact Use, Terms, and Intended Use

Trace4Tool uses existing web-agent benchmark settings and locally instantiated websites for research evaluation, including WebArena-style shopping tasks and related web-agent baselines cited in the main paper. We create ContractWebBench task manifests, evaluation summaries, evidence anno-

Table 20: Key artifacts for reproduction and analysis.

Artifact	Role
Evaluation summary JSON	Per-task results, success, evidence, level, and source statistics.
Benchmark manifest	Canonical task list with L0–L4 levels, task-group labels, and evidence.
Runtime implementation	Template location, tool selector, argument extractors, live extraction, and contract-valid answer.
Synthesis implementation	Fixed-to-parameterized tool generation, constrained argument binding, contract inference, and metadata generation.
Evidence reports	Historical evidence analysis, auth repair, live reruns, and reference-conflict explanations.

tations, tool libraries, and evidence reports. These artifacts are intended for research on reliable web-agent tool reuse and should be used only with authorized benchmark deployments or websites whose access terms permit such evaluation. At submission time, the public redistribution license for our newly created code and benchmark artifacts has not been finalized. Any release should state the final license selected by the authors, include citation and license notices for upstream artifacts, omit credentials and private account state, and avoid redistributing raw third-party site content when the original terms do not allow it.

E.2 Privacy, Safety, and Content Review

The benchmark tasks use synthetic or controlled website accounts rather than data collected from human participants. We do not intentionally collect information that names or uniquely identifies real individuals. Authenticated and state-changing tasks are evaluated with isolated accounts, reset protocols, and validators to avoid contaminating shared state. The evidence analysis separates live evidence, snapshot diagnostics, auth-confounded cases, and drift so that stale references or cached artifacts are not presented as fresh live evidence. We did not target offensive content; logs and release artifacts should still be screened to remove credentials, private account details, and any unexpected sensitive page content.

E.3 Computational Experiment Reporting

We do not train or fine-tune language or vision-language models. The main API-based agent runs use GPT-5.3-Codex through a hosted API. The additional no-tool vision-language baseline uses Qwen3-VL-4B-Instruct, a 4B-parameter model, on a server with an NVIDIA RTX 4090 GPU. Reported computational cost is inference-time cost: LLM calls and effective tokens for the no-tool and with-tool conditions, with zero-LLM runtime execution rows requiring no planner, decision, or tool-selector model calls at runtime. Provider-side parameter counts and GPU hours for the hosted API model are not available to us, and GPU/CPU wall-clock hours for the local server runs were not recorded, so we do not report a complete training-style compute budget. The no-tool WebAgent uses the same 15-action budget as the Trace4Tool comparison. Unless explicitly marked as diagnostic, results are reported as paired benchmark counts on fixed denominators, not as a maximum over multiple trials. We did not perform hyperparameter search; the main implementation choices are the action budget, Tool-Use Contract validators, tool selector, argument extractors, evidence, lifecycle state, Risk Level, and WebArena-compatible string, URL, DOM, and programmatic-HTML evaluation checks described in Sections 4.1 and 3.4.

E.4 Human Subjects and AI Assistance

This work does not use crowdworkers, human-subject experiments, or data collected from participants. Author inspection is used for evidence analysis and reference-conflict cases, but no participant data are collected. AI assistants were used for coding and writing support, including drafting, editing, and consistency checks. The authors reviewed and are responsible for the final claims, citations, analyses, and artifact descriptions.