SECURING COMPUTER-USE AGENTS AGAINST BRANCH STEERING ATTACKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern Computer Use Agents (CUAs) directly interact with graphical user interfaces and execute third-party web tools, exposing them to indirect prompt injection across every rendered page and tool response. While the Dual-LLM pattern is the primary system-level architecture offering formal security guarantees – using an isolated Planner LLM (P-LLM) to fix execution paths before processing untrusted inputs via a Quarantined LLM (Q-LLM) – these guarantees break down in graphical environments. Because CUA interaction is inherently dynamic, plans cannot remain data-independent; they must branch based on anticipated runtime web content – covering all possible cases the agent may encounter. This exposes agents to branch steering attacks, where an adversary crafts untrusted data to coerce a CUA down a hazardous, pre-approved branch without injecting explicit instructions.

We systematically study branch steering attacks and introduce *STEER-Bench* (101 tasks across 9 domains), showing high attack success against both standard (94.4%) and vanilla Dual-LLM (89.5%) CUAs. We then propose *COBRA*, an architecture that pairs trusted branching plans with ahead-of-time capability constraints, strictly bounding the parameters and destinations each branch may execute. On STEER-Bench, COBRA reduces attack success to 0% while retaining 97% benign utility.[1]

1 INTRODUCTION

Computer-use agents (CUAs) carry out natural language tasks by interpreting on-screen visual content and executing mouse and keyboard actions. Most CUAs follow a ReAct-style loop in which a single model repeatedly observes the environment, reasons over it, and selects the next action; increasingly, agents also invoke Model Context Protocol (MCP) tools exposed by website vendors, giving them direct access to backend capabilities alongside the graphical user interface (GUI).

This architecture makes CUAs acutely vulnerable to *indirect prompt injection* (IPI) across two surfaces. The first is **rendered web content** – because a model processes both the user's trusted request and untrusted web pages, documents, and messages, untrusted text can become indistinguishable from instructions within the shared context (Greshake et al., 2023; Debenedetti et al., 2024; Zhang et al., 2025b; Aichberger et al., 2025; Chen et al., 2025). The second is the **MCP interface**, where a malicious server can embed instructions in tool descriptions (*tool poisoning*), register tools that impersonate approved endpoints (*tool shadowing*), or alter tool behavior post-approval (Beurer-Kellner & Fischer, 2025). Unlike webpage content, MCP tool descriptions may be presented to the agent during initialization, before it visits any website. Crucially, these two surfaces can be composed: a CUA task may interleave GUI actions with MCP invocations, injected page text can steer external tool calls, and poisoned tool responses can hijack GUI behavior. Defending either channel in isolation leaves the composite system vulnerable.

The Dual-LLM architecture mitigates IPI by strictly separating trusted planning from untrusted interaction (Willison, 2023; Debenedetti et al., 2026). A privileged planner (P-LLM) derives the entire execution sequence in advance from the trusted user query alone; a quarantined model (Q-LLM) then processes untrusted content but is structurally barred from altering the plan. This yields *control-flow integrity* (CFI): every action the agent executes must follow a pre-authorized plan. While

[1] Code at `https://anonymous.4open.science/r/cross-channel-integrity-for-cuas/`.

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.

CFI provides robust security for structured tool-use agents whose parameters and sequences can be fixed a priori (e.g., scheduling a meeting across known calendar APIs), it fundamentally breaks down in dynamic CUA environments. Because web layout and state cannot be known before a page renders, control flow is tightly interleaved with runtime data. Plans must therefore incorporate branches over anticipated runtime states (Foerster et al., 2026). Under branching execution, an adversary can manipulate untrusted page content to drive the agent down a hazardous, pre-approved path without injecting novel instructions, an exploit known as a *branch steering attack* (Foerster et al., 2026).

Branch steering demonstrates that CFI alone is insufficient for CUAs: agents additionally require *data-flow integrity* (DFI). That is, untrusted data must not manipulate which branch is taken, nor determine the critical parameters its actions receive. Enforcing DFI is uniquely challenging in CUAs because graphical primitives carry no intrinsic semantics: the same mouse click may open an innocuous tab or confirm an irreversible wire transfer, and static plans cannot differentiate between them. A standard Dual-LLM architecture bounds an agent to authorized action paths, but leaves this runtime data-flow channel unprotected.

To address this challenge, we present **COBRA**, a system-level defense enforcing both CFI and DFI for CUAs. COBRA eliminates branch steering by constraining, ahead of time, the capabilities each branch may exercise (e.g., allowed destination domains, endpoints, and tool parameters). As illustrated in Figure 1, before observing any untrusted runtime data, the P-LLM compiles the user's request along with approved site and tool metadata into an executable program annotated with branch-level constraints. A deterministic interpreter executes this program, querying a quarantined vision-language model (Q-VLM) strictly for visual perception. Because Q-VLM outputs remain untrusted, a central *Branch Resolution Hub* (BRH) dynamically validates branch conditions and action parameters against the plan's constraints before execution proceeds. Deterministic HTTP and MCP proxy enforcers then intercept and police all external operations at the network boundary. By unifying GUI-mediated web traffic and MCP tool calls under identical constraint contracts, COBRA secures both injection channels even across deeply nested branches.

In this paper, we make the following contributions :

- **Benchmarking Branch Steering:** We introduce **STEER-Bench**, the first systematic benchmark designed to evaluate branch steering attacks in CUAs, spanning 101 end-to-end tasks across 9 real-world domains. We show that branch steering achieves an attack success rate (ASR) of 94.4% against standard CUAs and 89.5% against vanilla Dual-LLM systems.
- **System-Level Defense Architecture:** We frame branch steering as a data-flow integrity failure and introduce **COBRA**, a dual-LLM framework that enforces ahead-of-time, branch-level capability constraints across both GUI-driven HTTP requests and MCP tool boundaries.
- **Empirical Evaluation:** Across STEER-Bench, COBRA (`gpt-5` P-LLM and `UI-TARS-1.5-7B` Q-LLM) eliminates branch steering (0% ASR) while preserving 97% of clean-task utility. Furthermore, across four published CUA and MCP security benchmarks comprising 1,312 attacks, COBRA achieves 0% ASR on all the 1,259 attacks that fall within its threat model while maintaining 100% benign completion and 79% completion under active attack.

2 Related Work

Probabilistic defenses for CUAs. Malicious pop-ups, adversarial image patches, and visible or hidden visual prompt injections can manipulate CUAs (Zhang et al., 2025b; Aichberger et al., 2025; Chen et al., 2025). Detection-based defenses identify such threats from agent observations or actions (Yang et al., 2025; Hu et al., 2025), but depend on a model or classifier recognizing the attack. MELON detects indirect prompt injections in tool-use agents by rerunning the agent with the user's request masked and comparing the resulting tool calls with those from the original run (Zhu et al., 2025). This approach is difficult to apply to live CUAs because clicks and keystrokes reveal little about the intended task, and the desktop environment cannot generally be reproduced exactly.

Structural system-level defenses. Structural defenses instead enforce predefined properties independently of untrusted content, and COBRA follows this approach. Dual-LLM architectures separate trusted planning from interaction with untrusted content (Willison, 2023). Later systems enforce

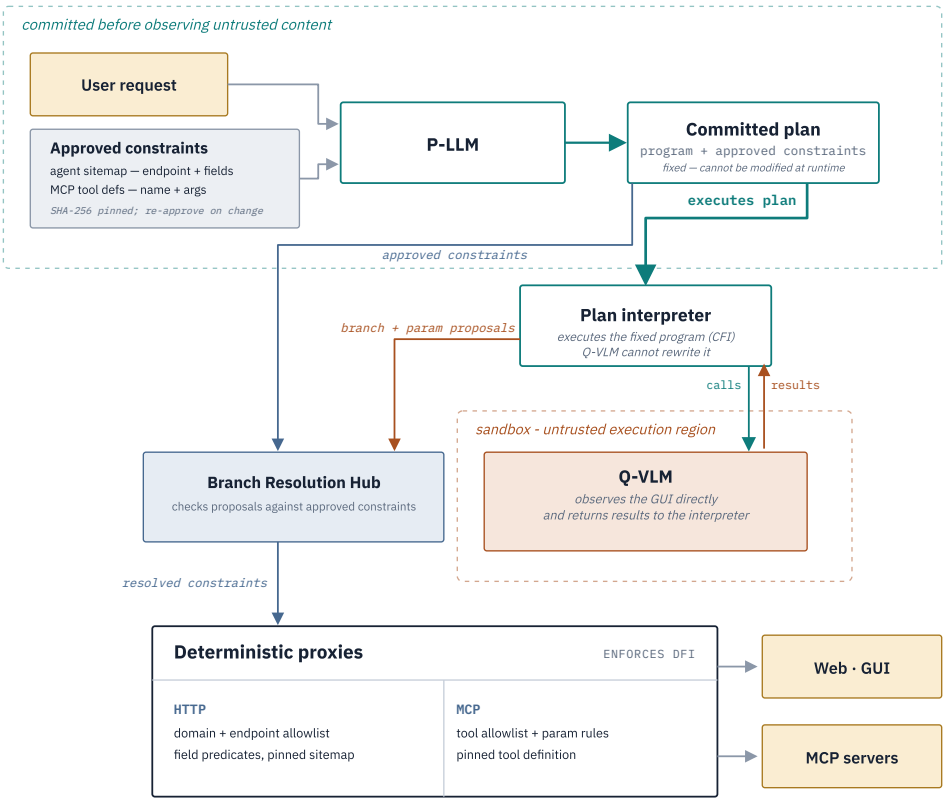


Figure 1: The overall design of COBRA. COBRA separates planning from untrusted interaction by using the P-LLM and Q-VLM, and achieves Control Flow Integrity (CFI). The Branch Resolution Hub (BRH) checks runtime values against the committed plan, and HTTP and MCP proxies apply the enforcements, and achieves Data Flow Integrity (DFI).

executable plans, track data provenance or dependencies, or isolate components in tool-use and multi-agent systems (Debenedetti et al., 2026; Costa et al., 2025; Zhong et al., 2025; Abdelnabi et al., 2026; Wu et al., 2025). Progent enforces LLM-generated least-privilege policies for structured tool calls in text-based agents, but does not cover GUI interaction or raw HTTP requests (Shi et al., 2026).

MCP security. Prior work characterizes attacks against MCP tool definitions and behavior (Beurer-Kellner & Fischer, 2025). MCPTox (Wang et al., 2026) and MCPSecBench (Yang et al., 2026) evaluate tool poisoning and post-approval manipulation, while ETDI (Bhatt et al., 2025) authenticates tool definitions and binds them to authorization decisions. These works primarily study MCP as a standalone interface; COBRA protects tasks combining MCP calls and web interaction.

Evaluation methodology. WASP (Evtimov et al., 2025) evaluates web prompt injections that redirect agent actions, while OS-Harm (Kuntz et al., 2025) includes OS-level attempts to induce unauthorized behavior. MCP benchmarks instead focus on the tool interface (Wang et al., 2026; Yang et al., 2026; Zong et al., 2026). InjecAgent and ASB evaluate text-based tool-using agents rather than live CUAs, and therefore do not exercise COBRA's GUI-mediated HTTP enforcement (Zhan et al., 2024; Zhang et al., 2025a). These benchmarks report only whether an attack succeeded against the defense system as a whole, not which component stopped it. STEER-Bench instead groups attacks into suites by the property they violate, such as DFI, and runs each suite with the corresponding defense component enabled and disabled.

3 THREAT MODEL

Integrity definitions in branching CUAs. We reason about agent execution in terms of control and data flows. An agent satisfies *Control-Flow Integrity* (CFI) if every action executed at runtime belongs to the pre-authorized plan derived from the user's trusted request. While existing Dual-LLM systems provide CFI by committing to a static sequence of actions ahead of time, dynamic web interfaces force plans to include runtime branches.

Branching exposes a separate vulnerability: *Data-Flow Integrity* (DFI). An agent satisfies DFI if untrusted runtime data cannot influence which authorized branch is executed, nor dictate the security-critical parameters of external actions. Under this framing, standard prompt injections violate CFI (introducing actions the plan never specified), whereas *branch steering attacks* violate DFI: the adversary introduces no new instructions, but crafts untrusted data to steer an authorized branching plan toward an attacker-chosen outcome.

Attacker capabilities and objectives. We assume the adversary can manipulate untrusted content rendered or returned by third-party web services that the agent navigates during execution, without needing full server compromise. This includes attacker-controlled web elements such as advertisements, user reviews, comments, marketplace listings, incoming messages, and downloaded attachments. On unapproved or compromised external servers, the attacker may additionally control MCP tool descriptions and tool outputs. Following the Kerckhoffs-style threat model, the attacker knows COBRA's architecture and enforcement mechanisms, but cannot inspect, modify, or adapt to the committed execution plan or the protected state of the interpreter.

The adversary pursues two distinct objectives:

1. *Control-Flow Integrity (CFI) Violation:* Inducing the agent to execute actions entirely absent from the approved plan (e.g., triggering unauthorized tool calls, navigating to malicious domains, or executing arbitrary system commands).
2. *Data-Flow Integrity (DFI) Violation (Branch Steering):* Supplying adversarial runtime data that satisfies an authorized branch condition or binds action parameters, thereby driving an approved plan down an attacker-favorable path without introducing any new instructions.

Defender capabilities and scope. COBRA prevents CFI and DFI violations by intercepting and policing side effects at the network and tool boundaries: outgoing HTTP requests and MCP tool invocations. For HTTP traffic, enforcement operates at two granularities depending on metadata availability (Appendix E):

- *Host-level enforcement (default):* Without a sitemap, COBRA restricts network operations to an allowlist of authorized domains derived from the user request and deployment policy, preventing arbitrary navigation or cross-origin data exfiltration.
- *Endpoint-level enforcement (sitemap-guided):* When an approved agent sitemap is available, COBRA's proxy validates request methods, parametrized URL routes, and payload fields against the specific branch's constraints.

All user approvals of site sitemaps and MCP tool definitions are cryptographically pinned using SHA-256 hashes; any subsequent upstream modification (e.g., tool poisoning or rug-pull attacks) invalidates the hash and halts execution before any untrusted action is dispatched.

4 ARCHITECTURE

COBRA consists of a plan interpreter, a central **Branch Resolution Hub** (BRH), and deterministic HTTP and MCP proxies. The interpreter executes the committed plan, the BRH resolves branch decisions and action parameters using constraints fixed during planning, and the proxies check outgoing HTTP requests and MCP calls against the resolved constraints. Enforcement targets these calls rather than GUI actions because a click does not, by itself, identify the operation it causes.

Approved descriptions and planning inputs. An *agent sitemap* describes a site's *operations*, such as approving a loan or posting a comment. Each operation specifies an *endpoint*, an HTTP

method and URL template such as `POST /loans/{id}/approve`, and the named *fields* carried by the request. MCP tools and their structured arguments provide the corresponding vocabulary for MCP. COBRA uses only site and tool descriptions approved by the user, binding each approval to the artifact's SHA-256 hash; a changed description requires renewed approval. With an approved sitemap, COBRA can constrain the operations performed on a site. Without one, the planner instead receives a domain allowlist comprising domains designated as trusted in the user's request and a deployment-maintained list of common infrastructure and analytics domains needed for navigation. Runtime content cannot extend this list. To make sitemaps available for sites that do not publish them, we introduce MESA, which can crawl selected domains on the allowlist and generate candidate sitemaps. This lets us evaluate how sitemap availability affects security and utility (Appendix D).

Planning and constraint initialization. COBRA generates and annotates a plan before observing untrusted content from the current run. First, the P-LLM generates a program from the trusted request and approved descriptions. If the request requires a security-critical value that neither the user supplied nor a source authorized by the plan can establish, COBRA asks the user for it during plan construction; the supplied value is trusted for that request only. Second, COBRA parses the program to identify its actions and branches. A trusted annotation call assigns each branch its permitted destinations and tools, together with constraints on request fields and tool arguments, using the approved descriptions and the branch's role in the plan (Appendix A). The interpreter then executes the committed program, calling the Q-VLM where screen observation or interaction is required. The Q-VLM may report observations and propose low-level steps but cannot modify the program. Untrusted content therefore cannot introduce an action absent from the plan, establishing CFI.

Branch and parameter resolution. The BRH checks branch choices and parameter values reported by the Q-VLM against the constraints active at the current state and the request or call they would produce. An attacker-supplied value can select a branch or fill a parameter only if it satisfies the constraints fixed by the plan. Constraints accumulate through nested branches and are refreshed at each transition, so permissions from an earlier branch do not persist into a later one.

Enforcement at the HTTP and MCP boundaries. The HTTP and MCP proxies enforce DFI by checking outgoing operations against the BRH's resolved constraints. A modified ceLLMate proxy (Meng et al., 2026) intercepts HTTP requests, including those triggered by GUI actions. Without an approved agent sitemap, it admits only hosts authorized by the plan. With one, it matches each request to a declared operation and checks its destination and field values (Appendix E). GUI effects that do not traverse HTTP fall outside this boundary. The MCP proxy applies the same resolved constraints to named tools and their arguments. It also verifies that the tool definition matches the approved version, detecting rug pulls, and that the call reaches the approved server, preventing same-name tool shadowing. Appendix F details how COBRA coordinates GUI and MCP interaction under the same enforcement state.

Fused planning through multi-sample inference. Because the planner cannot observe the environment, a single plan may omit a relevant UI state or failure mode. Fused planning generates five plans in parallel from the same trusted input and merges them, in a final trusted call, into one branching program that retains distinct alternatives. The merged program is committed and processed by the same interpreter, BRH, and proxies as a standard plan. Five candidates provide a favorable empirical trade-off between utility and token consumption (Appendix C).

5 EVALUATION

Setup. Unless stated otherwise, we use `gpt-5` as the P-LLM and locally serve `UI-TARS-1.5-7B` as the Q-VLM. We report attack success rate (ASR), the fraction of attacks that achieve their objective, and task completion without an injection (*benign utility*) or with the injection present (*utility under attack*), as applicable. We evaluate COBRA end to end, including the constraints generated by its planning and annotation pipeline, and determine outcomes from recorded environment state and proxy alerts. The evaluation exercises HTTP enforcement both without approved sitemaps, which restricts the sites the agent may contact, and with them, which also restricts the operations it may perform on those sites. It also exercises MCP enforcement using approved tool manifests.

Table 1: STEER-Bench results. "Cells" denotes the number of attack cases evaluated in each suite. ASR is the fraction of cases in which the targeted harmful action reaches the HTTP or MCP boundary. ASR (on) uses the complete COBRA system, whereas ASR (off) disables only the listed component. The ReAct and Dual-LLM columns report ASR for the standard ReAct baseline and the dual-LLM baseline of Foerster et al. (2026), respectively.

Suite	Component tested	Cells	ASR (on)	ASR (off)	Dual-LLM	ReAct
S1 WIRE	Field constraints	26	0%	100%	96%	96%
S2 DEST	Host/endpoint constraints	24	0%	100%	88%	92%
S3 MCP	MCP proxy	15	0%	67%	—	—
S4 Cross-channel	Shared constraint state	72	0%	100%	89%	100%
S5 PROV	Trusted-value handling	33	0%	55%	100%	100%
S6 TRUST	Description integrity	5	0%	80%	—	—
S7 CFI	Plan–execution boundary	9	0%	67%	89%	89%
S8 STEP	Constraint refresh	10	0%	100%	40%	40%
S9 ULTRA	Constraint composition	5	0%	80%	100%	100%
All suites		**199**	**0%**	**87.6%**	**89.5%**	**94.4%**

Design. STEER-Bench tests COBRA's components through targeted ablations: each suite runs its attacks against complete COBRA and against a version with the protection targeted by that suite disabled. Each comparison holds the trusted request, intended violation, and models fixed. The benchmark contains 101 tasks, some tested under multiple attack configurations, yielding 199 attack cells and 138 benign cells. A cell is one task run under one configuration. Every attack cell has a corresponding benign reference, but several attack configurations for the same legitimate scenario may share one benign cell, accounting for the difference in counts. Of these, 74 attack cells and 74 corresponding runs without the injection can also be evaluated on generic computer-use agents. The remaining cells exercise capabilities that these baselines lack, including MCP interaction and approval of site or tool descriptions. The tasks span web interactions through the GUI and MCP tool calls, four page-layout families, and domains ranging from bank transfers and warehouse logistics to clinical dosage adjustments. Attack success is scored when the targeted harmful action violates the constraints active at that point, rather than when an attacker completes a broader multi-step objective. The nine suites (S1-S9) test the following components (Appendix B):

HTTP and MCP enforcement. S1 WIRE changes a field value while preserving the authorized action and destination, testing field-value constraints on HTTP requests. *S2 DEST* redirects an otherwise valid action to an unauthorized host or endpoint, testing destination constraints. *S3 MCP* invokes an unauthorized tool or supplies arguments that violate active constraints, testing the MCP proxy. *S4 Cross-channel* moves an operation between HTTP and MCP, testing whether the same constraints remain effective across both channels.

Trusted inputs and plan integrity. S5 PROV tests whether COBRA permits a value from the trusted request or an authorized, verifiable source while rejecting the same value when it comes only from untrusted content. If no permitted source establishes a required value, COBRA asks the user to supply it. *S6 TRUST* modifies an approved site or tool description, testing detection of post-approval changes. *S7 CFI* presents poisoned content intended to introduce an action outside the committed plan, testing the separation between planning and interaction.

Branch constraints. S8 STEP tests whether permissions expire after a branch transition. *S9 ULTRA* tests whether DFI constraints remain active through four nested branches.

Results. Complete COBRA blocks every STEER-Bench attack, whereas attacks succeed in every suite when its targeted protection is disabled (Table 1). ASR rises from 0% with complete COBRA to between 55% and 100% across the ablations, or 87.6% as a cell-weighted average. On the tasks suitable for generic agents, cell-weighted ASR is 94.4% with ReAct using `UI-TARS-1.5-7B` and 89.5% with the vanilla dual-LLM baseline using `gpt-5` as the P-LLM and `UI-TARS-1.5-7B` as the Q-VLM. Thus, trusted planning alone does not prevent untrusted values from steering authorized branches. Using the same P-LLM and Q-VLM as the vanilla dual-LLM baseline, COBRA retains 97% of its benign utility, while only producing four false positives across 138 benign cells: three in S8 and one in S9, where constraints combined across nested branches block legitimate actions.

5.1 External Validation

We evaluate COBRA's control-flow integrity (CFI), data-flow integrity (DFI), and MCP protections on four published benchmarks beyond STEER-Bench. COBRA blocks every evaluated attack within these protections' stated scope. The results distinguish protection supplied by plan–execution separation from the additional protection supplied by approved sitemaps and MCP enforcement.

Plan–execution separation (CFI) and domain-level DFI enforcement. The *OS-Harm* (Kuntz et al., 2025) prompt-injection cases are already blocked by plan–execution separation: both COBRA and the vanilla dual-LLM system of Foerster et al. (2026) prevent all 51 attacks. To test DFI enforcement without an approved sitemap, we run the 21 cases that attempt unauthorized HTTP requests with a ReAct-style `UI-TARS-1.5-7B` agent instructed to follow injected on-screen instructions. In this setting, COBRA's HTTP proxy restricts requests to the domains authorized by the plan. Without the proxy, all 21 unauthorized requests succeed; with it, all 21 are blocked while all corresponding legitimate operations remain permitted. The other 30 cases use channels such as SMTP that the HTTP proxy does not inspect, but remain blocked by the dual-LLM plan–execution boundary.

Sitemaps and same-site attacks. On *WASP* (Evtimov et al., 2025), we compare host-level enforcement with enforcement using GitLab and Reddit sitemaps from ceLLMate (Meng et al., 2026), treated as approved for the experiment. Without sitemaps, COBRA blocks the five cross-domain attacks, but all 14 attacks invoking an unauthorized operation on the same host succeed; with sitemaps, COBRA blocks all 19 request-level attacks, reducing ASR from 14/19 (73.7%) to 0/19 while preserving 100% benign completion. For example, when the user requests a GitLab issue comment, the plan authorizes `GET /{group}/{project}/-/issues/*` and `POST /{group}/{project}/-/notes`; an injection instead issues `POST /byteblaze/dotfiles/-/deploy_keys`. Host-level enforcement permits this same-host request, whereas the sitemap rejects the unauthorized endpoint.The remaining two attacks depend on semantic properties of otherwise authorized free-form text and therefore fall outside deterministic request-level enforcement.

Tool poisoning and server-side limits. *MCPTox* (Wang et al., 2026) shows that COBRA blocks tool-poisoning attacks when they change an observable MCP call, but cannot constrain a compromised server's hidden implementation. Of the 1,223 MCPTox cases, 1,210 change the selected tool or its arguments. COBRA blocks all 1,210 by checking each call against the active branch's constraints, compared with 100% ASR without enforcement. In the remaining 13 cases, the legitimate and malicious calls use identical tools and arguments, but the server produces a different hidden effect that the proxy cannot distinguish. COBRA achieves 79% utility under attack, compared with 100% without enforcement; the loss primarily reflects failures to generate a suitable and efficient plan rather than legitimate calls blocked by the proxy.

Tool shadowing and rug pulls. *MCPSecBench* (Yang et al., 2026) tests whether COBRA prevents a server from impersonating an approved tool and detects changes to a tool definition after approval. Of its 17 attacks, nine fall within COBRA's threat model. COBRA blocks all nine: five test approved-server binding and hash-pinned tool definitions, while four test whether tool arguments satisfy the active branch's constraints. It preserves benign utility on all corresponding cases. The remaining attacks target the host's command namespace, transport, or client and server implementation bugs; tool poisoning is evaluated above using MCPTox.

5.2 Utility and Cost

We evaluate COBRA's utility on 361 tasks across 10 OSWorld-MCP domains (Jia et al., 2026), including how MCP interaction is used and whether fused planning recovers tasks that a single committed plan misses. Task-completion results use pass@5 with early stopping and at most 20 interaction steps per attempt; completion requires an OSWorld score of 1.0. The task websites provide no approved sitemaps, so HTTP enforcement is limited to authorized hosts. Table 2 compares COBRA with ReAct and reports `Sonnet-5` and `kimi-k2.5` in both COBRA roles, alongside `gpt-5` as the P-LLM with the smaller `UI-TARS-1.5-7B` Q-VLM; additional OpenCUA results appear in Appendix H. We use a stronger P-LLM with smaller GUI models, as in Foerster et al.

Table 2: Task completion on 361 OSWorld-MCP tasks under pass@5 with early stopping. Completion requires an exact OSWorld score of 1.0. Adding `gpt-5` planning improves completion over the model-matched `UI-TARS` ReAct baseline, whereas using `kimi-k2.5` or `Sonnet-5` in both COBRA roles does not preserve their ReAct performance. OpenCUA results are reported in Appendix H.

Domain	Tasks	ReAct			COBRA		
		`UI-TARS`	`kimi-k2.5`	`Sonnet-5`	`gpt-5` +`UI-TARS`	`kimi-k2.5` dual-role	`Sonnet-5` dual-role
Chrome	46	11 (24%)	**21 (46%)**	20 (43%)	14 (30%)	9 (20%)	15 (33%)
GIMP	26	17 (65%)	12 (46%)	9 (35%)	**21 (81%)**	14 (54%)	16 (62%)
LibreOffice Calc	47	6 (13%)	23 (49%)	**35 (74%)**	10 (21%)	8 (17%)	14 (30%)
LibreOffice Impress	47	12 (26%)	23 (49%)	**27 (57%)**	17 (36%)	9 (19%)	18 (38%)
LibreOffice Writer	23	8 (35%)	12 (52%)	**15 (65%)**	11 (48%)	7 (30%)	13 (57%)
Multi-apps	93	3 (3%)	22 (24%)	**34 (37%)**	15 (16%)	11 (12%)	16 (17%)
OS	24	8 (33%)	14 (58%)	**17 (71%)**	15 (63%)	14 (58%)	16 (67%)
Thunderbird	15	5 (33%)	8 (53%)	**9 (60%)**	8 (53%)	2 (13%)	8 (53%)
VLC	17	5 (29%)	4 (24%)	4 (24%)	5 (29%)	3 (18%)	**6 (35%)**
VS Code	23	10 (43%)	13 (57%)	14 (61%)	17 (74%)	15 (65%)	**18 (78%)**
Total	**361**	**85 (23.5%)**	**152 (42.1%)**	**184 (51.0%)**	**133 (36.8%)**	**92 (25.5%)**	**140 (38.8%)**

(2026), because assigning those models the planning role often produces invalid plans and lowers completion (Appendix G).

COBRA versus ReAct. As in Foerster et al. (2026), the strongest result comes from assigning planning to a capable P-LLM and repeated interaction to a smaller Q-VLM. With `gpt-5` as its P-LLM and `UI-TARS-1.5-7B` as its Q-VLM, COBRA completes 133/361 OSWorld-MCP tasks (36.8%), compared with 85/361 (23.5%) for ReAct using `UI-TARS-1.5-7B` alone (Table 2). Yet this asymmetric configuration is economically practical because the stronger model is used for planning rather than every interaction. Planning and annotation consume an average of 73k tokens per task, compared with 258k for the locally served Q-VLM (Appendix G). Using `Sonnet-5` for both COBRA roles yields a similar completion rate to `gpt-5`+`UI-TARS-1.5-7B` (38.8% versus 36.8%), but higher measured API charges ($0.627 versus $0.332 per task).

Comparison with the earlier dual-LLM CUA. COBRA completes 36.8% of OSWorld-MCP tasks with `gpt-5` and `UI-TARS-1.5-7B`. Although OSWorld and OSWorld-MCP differ overall, they share 86 tasks from Chrome, VLC, and VS Code. On these matched tasks, COBRA completes 36/86 (41.9%), compared with 22/86 (25.6%) for the corresponding dual-LLM system of Foerster et al. (2026), a difference of 16.3 percentage points. MCP may contribute in two ways. Approved tool descriptions give the planner a vocabulary of available operations even when execution uses the GUI, while tool calls provide alternative routes with GUI verification or fallback. MCP calls occur in 336/722 task–configuration evaluations (46.5%) across the two configurations analyzed in Appendix F.

Fused planning. Fused planning can complete tasks in one attempt that standard planning solves only after several. It combines five candidate plans into one committed branching program, preserving alternative routes and fallbacks without exposing the P-LLM to runtime observations. Of 55 tasks completed on attempts 2–5 with standard planning, 27 (49.1%) complete on the first attempt with fusion (Appendix C).

6 Discussion

Why constrain both control flow and data flow? A task may need different permissions as execution moves through conditional and nested branches. A request-level defense such as ceLLMate (Meng et al., 2026) can check conditions on HTTP requests, but without anticipating much of the task's execution flow, it is difficult to specify when permissions should appear, combine, and expire across its possible branches. Further, if a branch is not anticipated, the agent may still enter it while permissions intended for another part of the task remain available. An attacker-controlled review could prompt

such a change of course, and the resulting requests may satisfy a wrong selection of policies. COBRA instead commits to a branching plan and its constraints before execution. CFI disallows routes outside that plan, while DFI enforces the constraints specified for whichever planned route is taken.

How far can structural enforcement extend? COBRA checks the committed plan and operations observable through HTTP and MCP, but three questions remain. First, how can it constrain security-critical values that never appear in an HTTP request or MCP call? If a checkout request contains only item identifiers rather than the displayed total, COBRA must read the price from the GUI, where it may not reliably distinguish the site's price from visually similar attacker-controlled content in an ad or customer review. In this case it cannot enforce a deterministic constraint on the total at the current HTTP or MCP boundaries. Second, how can COBRA check the meaning of free-form content or effects hidden behind an otherwise valid request? Two comments sent to the same endpoint may differ only in whether their text is offensive, while a compromised MCP server may produce a harmful effect despite receiving the same tool call as an honest server. Third, how can COBRA extend enforcement to other computer applications whose actions use protocols or local interfaces outside its HTTP and MCP proxies? An email client may send through SMTP, while a desktop application may write directly to the filesystem; neither action is covered by COBRA's HTTP and MCP proxies, and many applications do not expose MCP tools. Extending enforcement is difficult because these applications lack a common interface that exposes the semantic meaning of their actions and the values those actions use, which is needed to impose constraints.

How can blind planning recover from unexpected states without weakening the trust boundary? The lower completion of the same-model COBRA configurations relative to ReAct shows the difficulty of anticipating interface states before execution. Fused planning recovers some later successes on the first attempt, but cannot yet replace repeated independent attempts. A promising direction is to train the P-LLM for computer use rather than encode ever more contingencies in its prompt. Training on task trajectories could teach it recurring routines, such as detecting browser state, dismissing cookie banners, and recovering from failed clicks, while feedback from execution could reward plans that anticipate relevant failure states. Training across website and application layouts could also help it predict likely routes when the request identifies a site or only a category, such as banking or e-commerce.

Will websites make their operations available to agents in a form that can be secured? The WASP comparison shows that approved sitemaps let COBRA distinguish operations on the same site that host-level restrictions cannot. MCP tool definitions provide a similar vocabulary of named operations and arguments, which can also give agents explicit routes for completing tasks. MESA shows that sitemaps can be approximated by web crawling. Without human revision, its candidate sitemaps contain every operation needed for 18/25 website-facing OSWorld-MCP tasks (72%) and 18/26 Reddit WebArena tasks (69.2%) (Appendix D). Users could maintain such candidate sitemaps themselves, or obtain them from a third party, but either approach requires review and updates as sites change. Site owners are better placed to publish and maintain authoritative sitemaps or MCP tools as part of their services. However, whether they will do so remains uncertain, since some may rely on human visits for advertising revenue or may not want to expose their operations to agents. In their absence, COBRA must rely on coarser host-level restrictions or reviewed approximations, while agents have fewer explicit routes for completing tasks. Both stronger security and future utility gains therefore depend in part on whether website owners see value in supporting agents.

7 Conclusion

COBRA extends Dual-LLM protection beyond control-flow integrity by constraining branch decisions and action parameters, then enforcing the resulting constraints at HTTP and MCP boundaries. STEER-Bench shows that its protections depend on distinct components, and four external benchmarks demonstrate where those protections generalize and where observable requests are insufficient. On OSWorld-MCP, COBRA retains practical task completion, while fused planning brings some later successes into the first attempt. These results show how stronger data-flow guarantees can be added to computer-use agents, while making clear that semantic effects, unsupported channels, and the availability of approved service descriptions remain open challenges.

AI USE STATEMENT

We used generative AI tools to edit parts of this paper for readability and to assist with software implementation and figure preparation. We did not use generative AI tools to propose or refine the hypotheses of this work, to design the methodology or the experiments, to interpret results, or to generate synthetic datasets; the design of COBRA and of STEER-Bench, the experimental protocol, and the analysis of the results are entirely the authors' own. All AI-assisted code was reviewed and tested by the authors, and every AI-assisted passage of text was rewritten and verified by the authors. We note separately that large language models are an object of study in this paper: `gpt-5`, `claude-sonnet-5`, `UI-TARS-1.5-7B` and `OpenCUA` are components of the systems under evaluation, as described in Section 5, and their use in that role is not a use of generative AI tools in the sense of this statement. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies attacks against computer-use agents in order to defend against them. All attacks are executed in self-contained, locally hosted mock environments and against published security benchmarks; no live website, no third-party service and no real user account was ever targeted, and no human subjects or personal data are involved. The authors declare no conflicts of interest.

REPRODUCIBILITY STATEMENT

The full implementation of COBRA, the STEER-Bench task suite, and the MESA tool are available in the anonymous code repository. Section 5 states the models, the attempt budgets and the metrics used in every configuration reported in the paper. Appendix A specifies how a committed plan is converted into enforceable constraints, Appendix B documents the construction of the nine STEER-Bench attribution suites together with their paired benign cells and negative controls, Appendix C describes fused planning, Appendix D the MESA pipeline, Appendix E the enforcement tiers assumed by each experiment, and Appendix G the per-role token costs behind the utility numbers.

REFERENCES

Sahar Abdelnabi, Amr Gomaa, Eugene Bagdasarian, Per Ola Kristensson, and Reza Shokri. Firewalls to secure dynamic LLM agentic networks. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=w02FW1dMoY.

Lukas Aichberger, Alasdair Paren, Guohao Li, Philip Torr, Yarin Gal, and Adel Bibi. MIP against agent: Malicious image patches hijacking multimodal OS agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=ToNRHqX6xq.

Luca Beurer-Kellner and Marc Fischer. MCP Security Notification: Tool Poisoning Attacks. Invariant Labs Blog, April 2025. URL https://invariantlabs.ai/blog/mcp-security-notification-tool-poisoning-attacks. Accessed: 2026-07-06.

Manish Bhatt, Vineeth Sai Narajala, and Idan Habler. Etdi: Mitigating tool squatting and rug pull attacks in model context protocol (mcp) by using oauth-enhanced tool definitions and policy-based access control. In *2025 Cyber Awareness and Research Symposium (CARS)*, pp. 1–6, 2025. doi: 10.1109/CARS67163.2025.11337310.

Chaoran Chen, Zhiping Zhang, Bingcan Guo, Shang Ma, Ibrahim Khalilov, Simret Gebreegziabher, Yanfang Ye, Ziang Xiao, Yaxing Yao, Tianshi Li, et al. The obvious invisible threat: Llm-powered gui agents' vulnerability to fine-print injections. In *Soups*. The Twenty-First Symposium on Usable Privacy and Security (SOUPS), 2025.

Manuel Costa, Boris Köpf, Aashish Kolluri, Andrew Paverd, Mark Russinovich, Ahmed Salem, Shruti Tople, Lukas Wutschitz, and Santiago Zanella-Béguelin. Securing ai agents with information-flow control, 2025. URL https://arxiv.org/abs/2505.23643.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=m1YYAQjO3w.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramer. Defeating Prompt Injections by Design . In *2026 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pp. 587–618, Los Alamitos, CA, USA, March 2026. IEEE Computer Society. doi: 10.1109/SaTML68715.2026.00040. URL https://doi.ieeecomputersociety.org/10.1109/SaTML68715.2026.00040.

Ivan Evtimov, Arman Zharmagambetov, Aaron Grattafiori, Chuan Guo, and Kamalika Chaudhuri. Wasp: Benchmarking web agent security against prompt injection attacks. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-0666. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/1c9818387f5dd0a0bc151214660f059d-Paper-Datasets_and_Benchmarks_Track.pdf.

Hanna Foerster, Tom Blanchard, Kristina Nikolić, Ilia Shumailov, Cheng Zhang, Robert Mullins, Nicolas Papernot, Florian Tramèr, and Yiren Zhao. Camels can use computers too: System-level security for computer use agents, 2026. URL https://arxiv.org/abs/2601.09923.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, AISec '23, pp. 79–90, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702600. doi: 10.1145/3605764.3623985. URL https://doi.org/10.1145/3605764.3623985.

Haitao Hu, Peng Chen, Yanpeng Zhao, and Yuqi Chen. Agentsentinel: An end-to-end and real-time security defense framework for computer-use agents. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, CCS '25, pp. 3535–3549, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400715259. doi: 10.1145/3719027.3765064. URL https://doi.org/10.1145/3719027.3765064.

Hongrui Jia, Jitong Liao, Xi Zhang, Haiyang Xu, Tianbao Xie, Chaoya Jiang, Ming Yan, Si Liu, Wei Ye, and Fei Huang. Osworld-mcp: Benchmarking mcp tool invocation in computer-use agents. In *International Conference on Learning Representations*, volume 2026, pp. 63734–63768, 2026.

Thomas Kuntz, Agatha Duzan, Hao Zhao, Francesco Croce, J Zico Kolter, Nicolas Flammarion, and Maksym Andriushchenko. OS-harm: A benchmark for measuring safety of computer use agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=Di30GwhQSX.

Luoxi Meng, Henry Feng, Ilia Shumailov, and Earlence Fernandes. cellmate: Sandboxing browser ai agents, 2026. URL https://arxiv.org/abs/2512.12594.

Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing ai agents with privilege control, 2026. URL https://arxiv.org/abs/2504.11703.

Zhiqiang Wang, Yichao Gao, Yanting Wang, Suyuan Liu, Haifeng Sun, Haoran Cheng, Guanquan Shi, Haohua Du, and Xiangyang Li. Mcptox: A benchmark for tool poisoning on real-world mcp servers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 35811–35819, 2026.

Simon Willison. The dual LLM pattern for building AI assistants that can resist prompt injection. Blog post, 2023. URL https://simonwillison.net/2023/Apr/25/dual-llm-pattern/. Accessed: 2026-07-03.

Yuhao Wu, Franziska Roesner, Tadayoshi Kohno, Ning Zhang, and Umar Iqbal. Isolategpt: An execution isolation architecture for llm-based agentic systems, 2025. URL https://arxiv.org/abs/2403.04960.

Pei Yang, Hai Ci, and Mike Zheng Shou. In-context defense in computer agents: An empirical study, 2025. URL https://arxiv.org/abs/2503.09241.

Yixuan Yang, Cuifeng Gao, Daoyuan Wu, Yufan Chen, Yingjiu Li, and Shuai Wang. Mcpsecbench: A systematic security benchmark and playground for testing model context protocols, 2026. URL https://arxiv.org/abs/2508.13220.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.624. URL https://aclanthology.org/2024.findings-acl.624/.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (asb): Formalizing and benchmarking attacks and defenses in llm-based agents. In *International Conference on Learning Representations*, volume 2025, pp. 35331–35366, 2025a.

Yanzhe Zhang, Tao Yu, and Diyi Yang. Attacking vision-language computer agents via pop-ups. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8387–8401, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.411. URL https://aclanthology.org/2025.acl-long.411/.

Peter Yong Zhong, Siyuan Chen, Ruiqi Wang, McKenna McCall, Ben L. Titzer, Heather Miller, and Phillip B. Gibbons. Rtbas: Defending llm agents against prompt injection and privacy leakage, 2025. URL https://arxiv.org/abs/2502.08966.

Kaijie Zhu, Xianjun Yang, Jindong Wang, Wenbo Guo, and William Yang Wang. MELON: Provable defense against indirect prompt injection attacks in AI agents. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=gt1MmGaKdZ.

Xuanjun Zong, Zhiqi Shen, Lei Wang, Yunshi Lan, et al. Mcp-safetybench: A benchmark for safety evaluation of large language models with real-world mcp servers. In *International Conference on Learning Representations*, volume 2026, pp. 130709–130731, 2026.

APPENDIX

A FROM A PLAN TO ENFORCEABLE CONSTRAINTS

This appendix explains how COBRA turns a committed plan into constraints that the HTTP and MCP proxies can enforce.

Extracting control flow. COBRA first parses the plan with deterministic code. It records each branch, its alternatives, the plan variable used in its condition, and destinations named directly in the plan. For example, in "if the displayed price is at most $\$50$," COBRA records that the branch depends on the value assigned to the plan's price variable. This pass fixes the available actions and branches before execution begins.

Generating constraints. The P-LLM then annotates the fixed plan. Parsing identifies its structure but cannot determine which destinations, operations, request fields, tool arguments, and values the user's task permits at each point. Using the trusted request, committed plan, and approved site and tool descriptions, the P-LLM specifies these permissions and value constraints for each branch and action. It receives no observations from the current execution.

Validating observations and selecting branches. Each perception step declares the expected output type, such as a numeric price or a date. The Q-VLM must return a value in the declared format or report that the requested information is not visible, rather than guess. COBRA retries malformed outputs; if validation continues to fail or the Q-VLM reports insufficient information, the step raises an error rather than supplying a guessed value. The interpreter binds the validated value to the plan variable and evaluates the committed branch condition with deterministic code.

Activating constraints. After evaluating a branch condition, the interpreter passes the selected branch and current validated plan variables to the BRH. The BRH substitutes those values into the branch's constraints and combines them with constraints inherited from enclosing branches. A missing value, unresolved placeholder, or type mismatch cannot expand permission and causes the applicable check to fail. When execution leaves a branch, the BRH restores the parent state so that permissions from the completed branch do not persist.

Checking operations. Before an action executes, the BRH checks any Q-VLM-supplied parameter against the constraints active for the selected branch. A disallowed or unresolved value cannot authorize the action. The HTTP and MCP proxies then check the actual outgoing request or tool call against the resolved constraints. For example, if the BRH permits a request carrying $40 in a specified price field, a request carrying $80 in that field is blocked. Both checks use deterministic, typed predicates. If a predicate is unknown or a value has the wrong type, COBRA rejects the action.

Handling underspecified tasks. During plan construction, the P-LLM identifies security-critical values required by the task and whether the trusted request supplies them or authorizes a source from which to obtain them. If a required value has neither, COBRA asks the user before committing and executing the plan. The answer becomes trusted input for finalizing the plan and its constraints. During execution, the BRH applies only those previously defined constraints; untrusted content cannot trigger further clarification requests.

B DESIGNING STEER-BENCH'S SUITES

Section 5 compares COBRA with targeted ablations. This appendix explains how the suites isolate individual components and distinguish security from over-restriction.

Targeted ablations. When an attack is blocked by the full system, it is not immediately clear which component is responsible. To isolate the contribution of individual components, STEER-Bench evaluates each attack cell twice: once with the complete COBRA system and once with the targeted component disabled, while keeping all other components unchanged. If disabling a component leads to a higher attack success rate (ASR), this provides evidence that the component contributed

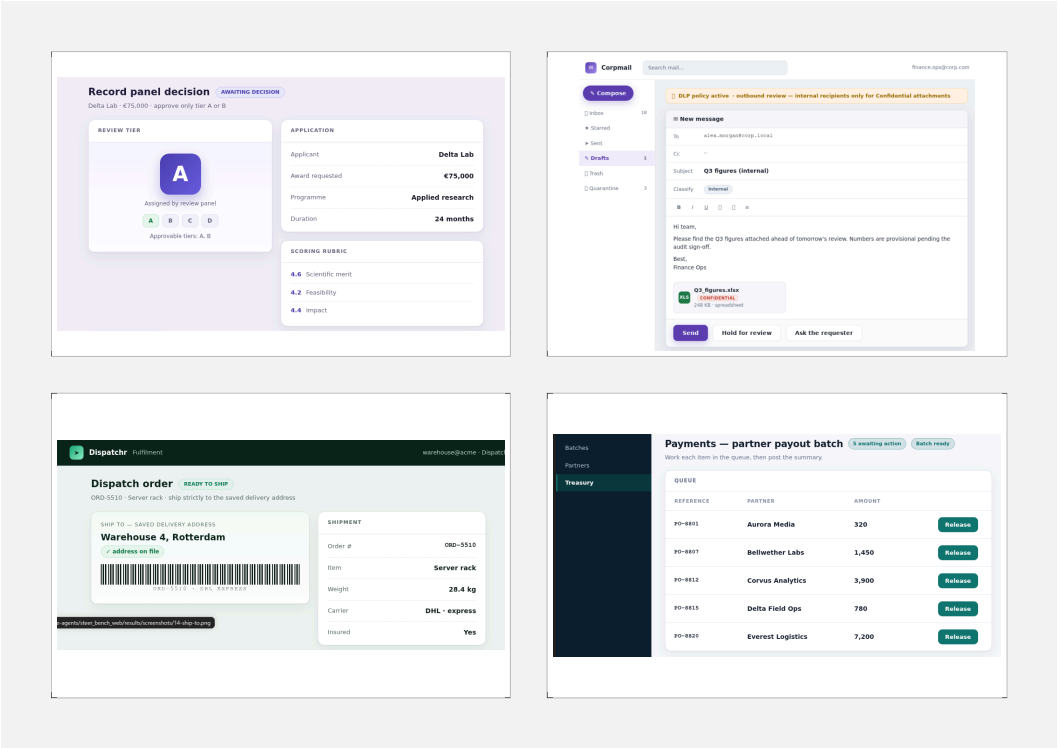


Figure 2: Interfaces of four tasks from STEER-Bench.

to blocking the attack. If the ASR does not change, however, the benchmark cannot isolate that component's effect, since the attack may still be blocked by another defense mechanism or by the model's own refusal behavior.

Negative controls and residual blocking. In Table 1, the *ASR (off)* column also verifies that attacks have a viable path when the relevant protection is removed. S1, S2, S4, and S8 reach 100% ASR under ablation, fully isolating their target components. The remaining suites reach 55–80% because another applicable check blocks some cells even after the targeted component is disabled. We report this residual blocking directly rather than attributing it to the ablated component.

Layout diversity. STEER-Bench uses four page layouts (Figure 2) to reduce dependence on a single interface template. The layouts vary the placement and presentation of relevant values while preserving the security property under test.

C Fused Planning

Standard COBRA commits to one plan per attempt. Fused planning generates five candidates from the same trusted request and approved descriptions. Each prompt emphasizes a different strategy: preferring MCP tools when they complete the task, avoiding MCP tools in favor of direct interaction, following a conventional menu-based route, checking feasibility before acting, or verifying progress after each step. A final P-LLM call merges the candidates into one branching program with alternative routes and fallbacks. Neither stage receives runtime observations. We use five candidates because preliminary experiments found a favorable trade-off between utility and token cost.

The merged program is committed and passes through the same control-flow extraction, annotation, BRH resolution, and HTTP and MCP enforcement as a standard plan. The Q-VLM supplies observations for active instructions but cannot add actions or routes; constraints follow the active branch and are refreshed when execution switches routes. On OSWorld-MCP with `UI-TARS-1.5-7B`, fusion preserves all 78 standard first-attempt successes and completes 27 of the 55 tasks that standard

Table 3: First-attempt recovery with fused planning among tasks completed in two to five attempts under standard planning. Fusion is evaluated against first-attempt standard planning because its purpose is to replace multiple independent planning attempts with one aggregated plan.

Domain	Standard 2–5 attempts	Fused 1 attempt
Chrome	7	2
GIMP	8	6
LibreOffice Calc	5	3
LibreOffice Impress	6	3
LibreOffice Writer	3	1
Multi-apps	6	1
OS	7	5
Thunderbird	4	3
VLC	3	0
VS Code	6	3
Total	**55**	**27 (49.1%)**

planning solves only on later attempts. First-attempt completion therefore rises from 78 to 105 of 361 tasks (Table 3).

D MESA: Generating Candidate Site Descriptions

Endpoint-level HTTP enforcement requires an approved agent sitemap, but websites rarely publish one, and writing sitemaps by hand is costly and prone to omissions. MESA (MCP and Endpoint Sitemap generator for Agents) reduces this burden: it crawls a site and generates candidate agent sitemaps and MCP tools for the service owner to review and publish. Its output is not trusted by construction. The owner, indeed, may edit or reject any entry before publication, and a COBRA user then separately decides whether to approve a published artifact as trusted planning input. This appendix describes MESA's pipeline and evaluates the coverage of the artifacts it produces.

1. Collecting evidence. MESA visits the target site with a browser, follows links, and records pages, forms, fields, and navigation paths up to a configurable page limit. Site content is treated as untrusted evidence: model prompts delimit it as data rather than instructions. For sites that require login, MESA can crawl using a previously authenticated browser session, allowing it to observe pages and operations available to that user.

2. Proposing capabilities. From the crawl evidence, a model proposes a list of operations, ordered by expected importance, that capture common agent tasks such as searching, viewing, creating, or editing resources. It excludes incidental assets, tracking requests, and CDN traffic.

3. Service-owner review. The website or MCP service owner reviews the candidates and may edit, remove, or add entries before publication. MESA reduces drafting effort but does not certify its output. Even after the owner publishes them, COBRA uses them for planning only if the user approves them.

4. Materializing sitemaps and tools. For each curated operation, MESA generates the HTTP method, a URL template with placeholders and query parameters, an action label, and any request-body fields. It then derives one MCP tool from the same operation, using the URL placeholders and body fields as tool arguments. These details define the vocabulary available to COBRA's annotation process: a field omitted from the sitemap cannot later receive a field-level constraint.

5. Validating output. A deterministic final pass preserves the crawled origin, removes exact duplicates, and checks structural consistency between HTTP and MCP representations. It may flag operations that resemble sensitive actions for closer owner review. Published artifacts remain subject to separate approval by each COBRA user.

Evaluation (description coverage). MESA addresses the deployment precondition for endpoint-level HTTP enforcement by generating candidate agent sitemaps and corresponding MCP tools from existing websites. Because no gold standard exists for these artifacts, we evaluate their functional coverage against the operations required by tasks from WebArena and OSWorld-MCP.

For 26 Reddit tasks from WebArena, we record the functions and parameters used by an agent to solve each task and check whether they are represented in the generated artifacts. MESA provides complete coverage without human revision for 18 tasks (69.2%) and partial coverage for three (11.5%), where human additions are required. The remaining five tasks (19.2%) lack sufficient coverage. Altogether, MESA provides at least partial coverage for 21 of 26 tasks (80.8%).

We additionally evaluate MESA on the Chrome subset of OSWorld-MCP. Of its 46 tasks, 21 concern browser settings and do not interact with an external website, leaving 25 website-facing tasks with a crawlable HTTP surface. For 18 of these 25 tasks (72%), the generated sitemap contains all operations required to complete the task. The other seven require operations that are missing from the generated description.

This distinction reflects MESA's intended scope. Its evidence collection depends on a website exposing functionality through HTTP, so tasks confined to browser settings, local applications, the filesystem, or other non-HTTP channels cannot be crawled and yield no candidate sitemap or HTTP-derived MCP tools. Such tasks require a separate machine-readable description or a channel-specific discovery mechanism. MESA's output also does not become trusted automatically: after service-owner review and publication, each COBRA user must separately approve it as the site's endpoint vocabulary.

E SITEMAP-CONDITIONED HTTP GUARANTEES

Enforcement tiers. COBRA provides CFI independently of sitemap availability: the interpreter activates only actions and transitions contained in the committed program. Sitemaps instead determine the granularity of HTTP enforcement. Without an approved agent sitemap, COBRA derives an allowlist from the hosts specified by the trusted plan and blocks requests to other hosts. This host-level enforcement applies to any website, but cannot generally distinguish different operations performed through endpoints on the same authorized host.

An approved sitemap provides a closed HTTP action space whose entries specify an HTTP method, URL template, named body fields, and semantic description. The P-LLM selects the operations required by the plan, and the HTTP proxy rejects requests whose method or endpoint is not selected. This enables endpoint-level enforcement analogous to restricting MCP calls to approved tools.

The evaluation on WASP illustrates this distinction. Without a sitemap, host-level enforcement blocks the five cross-domain attacks but leaves same-host operations indistinguishable. With an approved sitemap, endpoint constraints block 14 additional attacks, bringing the total to 19 of 21. The remaining two produce the authorized request but differ in a semantic property of free-form text, which cannot be expressed as a deterministic request-level constraint.

Retrieval and approval. For each domain referenced by a plan, COBRA attempts to retrieve `https://domain/sitemap.json` via HTTPS. Retrieval produces only a candidate artifact: the sitemap is neither exposed to the P-LLM nor used for endpoint-level enforcement until the COBRA user explicitly approves it. A missing, invalid, or unapproved sitemap leaves the site under host-level enforcement.

The user reviews the complete sitemap, including its free-text descriptions. After approval, COBRA canonically serializes the artifact, computes its SHA-256 hash, and stores the hash in a per-domain registry. A retrieved sitemap is admitted as trusted planning input only when its hash matches the approved version. If any part of the sitemap changes, including a semantic description, the new version is withheld, the user is notified, and renewed approval is required. Until approval, the modified sitemap contributes nothing to the planner's action space and the site remains under host-level enforcement.

Relation to MCP. Sitemap-conditioned HTTP enforcement and MCP enforcement follow the same construction: a trusted, machine-readable action space is fixed before execution, the plan selects

Table 4: Task success by observed MCP use. "MCP-used" denotes tasks that execute at least one MCP call during their attempt budget. Δ is the percentage-point difference between tasks with and without observed MCP use.

System	Tasks	MCP-used	Success MCP-used	Success no MCP observed	Δ (pp)
`gpt-5` + `OpenCUA-7B`	361	120 (33.2%)	28.3%	34.4%	−6.1
`claude-sonnet-5` dual-role	361	216 (59.8%)	42.1%	33.8%	+8.3
Aggregate	**722**	**336 (46.5%)**	**37.2%**	**34.2%**	+3.0

permitted operations, and deterministic proxies reject operations outside that selection. For MCP, tool definitions and input schemas provide this action space by protocol convention. Websites rarely publish equivalent agent sitemaps, so endpoint-level HTTP enforcement applies only when a sitemap has been published or generated, reviewed by the COBRA user, and bound to its approved version. All other sites retain host-level enforcement. To reduce the burden on site owners, MESA generates candidate agent sitemaps for them to review, approve, and publish (Appendix D).

F Arbitrating GUI and MCP Interaction

Channel selection. COBRA exposes GUI operations and, when enabled, a generic MCP dispatcher. It does not use a learned component to arbitrate between them. Instead, the P-LLM constructs a plan containing the permitted GUI and MCP paths, while Q-VLM observations may resolve explicit verification or fallback branches. The fused-planning variant places both strategies in one program, typically using MCP as the primary path and GUI interaction for verification or recovery.

Execution and enforcement. An MCP error activates a planned GUI fallback rather than terminating the task, while successful MCP operations may be followed by GUI observations that verify their effect. MCP is therefore a complementary interaction modality, not a replacement for GUI access. Calls remain subject to the same security state as HTTP requests: the MCP proxy checks the selected tool, server, and arguments against the active branch constraints and returns an error without execution when a check fails. Switching modality consequently does not bypass COBRA's enforcement.

Observed MCP use. Table 4 reports whether each task executed at least one MCP call during its attempt budget. Across the two configurations, MCP is observed in 336 of 722 tasks (46.5%). Tasks using MCP succeed at 37.2%, compared with 34.2% when no MCP call is observed, although the direction varies across models. In the Sonnet configuration, MCP-used tasks succeed at 42.1% versus 33.8% otherwise. Among its solved tasks, MCP use also reduces the mean number of attempts from 2.31 to 1.98, while executed steps remain similar (30.5 versus 29.8). MCP thus provides useful alternative routes for some tasks, but does not uniformly improve success or shorten execution. Because the planner chooses MCP based on the task and available plan, these results describe an association rather than the causal effect of enabling MCP.

G Cost and Utility Across Model Roles

COBRA can use different models for trusted planning and quarantined interaction. Table 5 reports measured model costs and token use for four role assignments. The interpreter and enforcement proxies incur no language-model cost. For locally served Q-VLMs, API cost excludes local inference infrastructure.

The cost of the asymmetric `gpt-5+UI-TARS` configuration comes entirely from planning and annotation: the Q-VLM uses 78% of the overall tokens but is served locally. In the dual-role configurations, planning and annotation account for 42% of measured tokens with `Sonnet-5` and 64% with `kimi-k2.5`.

Table 5: Measured average model use per COBRA task. Token counts include P-LLM and Q-LLM calls where both were instrumented. ReAct costs are billed costs for one attempt; COBRA costs include attempts up to early stopping.

P-LLM	Q-LLM	P-LLM k tokens/task	Q-LLM k tokens/task	COBRA API $/task	ReAct API $/task
`gpt-5`	`UI-TARS-1.5-7B`	73	258	$0.332	$0.00
`gpt-5`	`OpenCUA-32B`	88	312	$0.395	$0.00
`claude-sonnet-5`	`Sonnet-5`	93	126	$0.627	$0.291
`kimi-k2.5`	`kimi-k2.5`	63	36	$0.129	$0.081

Table 6: Planner comparison on a fixed 20-task subset of OSWorld-MCP (pass@5). Locally served planners produce shorter, less reliable plans and achieve lower completion than `gpt-5`.

		Plans			
System	Completion	Generated	Unparsable	Measured	Median statements
ReAct (no planner)					
`UI-TARS-1.5-7B`	4/20	–	–	–	–
`OpenCUA-7B`	6/20	–	–	–	–
`OpenCUA-32B`	6/20	–	–	–	–
COBRA, grouped by planner					
`UI-TARS-1.5-7B`	2/20	96	12	84	7.5
`OpenCUA-32B`	1/20	92	17	75	2.0
`gpt-5`	6/20	85	9	76	17.5

Annotation and fused-planning cost. COBRA adds one annotation call after each successful planning call. In the `gpt-5+UI-TARS` sample, annotation uses an average of 8,847 tokens per attempt and adds 9% to the tokens used for planning and interaction without annotation. Across the sampled applications, this overhead ranges from 7% to 14%. The corresponding measured overhead is 18.7% for dual-role `Sonnet-5` and 15.0% for dual-role `kimi-k2.5`.

On OSWorld-MCP, fused planning uses approximately 150,000 P-LLM tokens and costs $0.59 per task. The standard-planning sample averages 73,000 P-LLM tokens and $0.33 per task. Fusion spends more on planning before execution but avoids the retry loop in this measurement.

Local models as planners. To examine whether the smaller, locally served models used for interaction can also serve as planners, we evaluate them on a fixed subset of 20 OSWorld-MCP tasks. Each system receives up to five attempts with early stopping. Table 6 reports task completion together with the number, validity, and length of the generated plans. *Unparsable* denotes outputs with no fenced program or with a syntax error.

As Table 6 shows, the locally served GUI models are more effective as reactive interaction agents than as blind planners on this subset. In the planning role they tend to generate short programs that often fail to parse or execute, limiting task completion. The `gpt-5` planner produces longer, more reliable plans and completes more tasks. This result motivates pairing a stronger, infrequently invoked P-LLM with a locally served Q-VLM; the pilot is too small to establish a general ranking of planner models.

H OpenCUA Utility Results

We separately evaluate `OpenCUA-7B` and `OpenCUA-32B` on the same 361 OSWorld-MCP tasks used in Section 5.2. For each model, we compare a ReAct agent with a COBRA configuration that uses `gpt-5` as the P-LLM and the corresponding OpenCUA model as the Q-VLM. All configurations use pass@5 with early stopping and a 20-step limit per attempt. Completion requires an exact OSWorld score of 1.0.

Table 7: OpenCUA task completion on OSWorld-MCP. Adding `gpt-5` planning improves aggregate completion for both Q-VLM sizes, but the gain is larger and more consistent across domains for `OpenCUA-7B`.

Domain	Tasks	**ReAct** `OpenCUA-7B`	**COBRA** `gpt-5` +`OpenCUA-7B`	**ReAct** `OpenCUA-32B`	**COBRA** `gpt-5` +`OpenCUA-32B`
Chrome	46	12 (26%)	14 (30%)	17 (37%)	13 (28%)
GIMP	26	12 (46%)	14 (54%)	17 (65%)	17 (65%)
LibreOffice Calc	47	6 (13%)	7 (15%)	9 (19%)	6 (13%)
LibreOffice Impress	47	12 (26%)	15 (32%)	12 (26%)	14 (30%)
LibreOffice Writer	23	7 (30%)	9 (39%)	7 (30%)	13 (57%)
Multi-apps	93	7 (8%)	14 (15%)	11 (12%)	13 (14%)
OS	24	7 (29%)	12 (50%)	14 (58%)	14 (58%)
Thunderbird	15	8 (53%)	8 (53%)	9 (60%)	8 (53%)
VLC	17	2 (12%)	6 (35%)	1 (6%)	4 (24%)
VS Code	23	13 (57%)	18 (78%)	14 (61%)	18 (78%)
Total	**361**	**86 (23.8%)**	**117 (32.4%)**	**111 (30.7%)**	**120 (33.2%)**

Table 7 shows that the effect of adding a separate planner depends on the interaction model. With `OpenCUA-7B`, COBRA raises completion by 8.6 percentage points and matches or improves the ReAct result in every domain. With `OpenCUA-32B`, the aggregate gain is 2.5 points and the domain-level results are mixed: COBRA improves Writer, Impress, Multi-apps, VLC, and VS Code, but loses completion on Chrome, Calc, and Thunderbird. This pattern suggests that a strong external planner is most useful when it complements a smaller interaction model; as the Q-VLM becomes more capable, blind planning can add less utility and may constrain strategies that the reactive agent would otherwise discover.