# MATCHED CAPABILITIES, DIFFERENT BEHAVIORS: A CONTROLLED STUDY OF MCP AND CLI INTERFACES FOR LANGUAGE AGENTS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Language agents increasingly act through external tools and interactive environments, making the interface between the model and its executable capabilities part of the decision process. Model Context Protocol (MCP) and command-line interfaces (CLI) can expose the same semantic operations through markedly different model-facing representations. We study whether these representational differences change agent behavior when executable capabilities are matched. We introduce Matched Interface Comparison and Analysis (MICA), which renders MCP and CLI from a shared semantic action contract and routes decoded actions through the same runtime. Across six models and three interactive environments, matched interfaces produce substantial differences in task performance and interaction cost. These effects extend beyond surface validity to semantic action selection, failure pathways, and recovery. Matched-state probes localize sensitivity to capability presentation, realization, and feedback. Paired natural trajectories reveal structured divergence in 94.3% of 2,004 episodes and show that early differences propagate through subsequent interaction. This analysis identifies interface realization as a direct intervention boundary. We therefore introduce Decoupled Action Realization and Targeted Repair (DART), which preserves validated semantic commitments while repairing interface-specific realization. DART improves benchmark score in all six evaluated settings by up to 9.0 points and reduces MCP token cost in all three environments by up to 27.1k tokens per task. Together, these results establish interface representation as part of the closed-loop decision environment shaping agent behavior. Code is available in the Supplementary Material.

## 1 INTRODUCTION

Modern language agents increasingly act through tools, APIs, and executable environments (Yao et al., 2023; Lu et al., 2025; Li et al., 2023; Liu et al., 2024). These external capabilities are accessed through interfaces that specify how available actions, arguments, response conventions, and execution feedback are presented to the model. In an interactive agent, however, an interface does more than expose what actions are possible: it becomes part of the context from which the next action is selected. As a result, different representations of the same executable capabilities can induce different local decisions and, through closed-loop interaction, different trajectories. Prior work has shown that action representations (Wang et al., 2024a), tool metadata (Yuan et al., 2025; Blankenstein et al., 2026), and agent–computer interfaces (Yang et al., 2024) materially shape tool use and task performance. This raises a fundamental question for interactive language agents: *When executable capabilities are held fixed, to what extent does their model-facing representation change the policy that emerges?*

This question is particularly important because both structured tool invocation and terminal-based interaction are increasingly common in modern agent systems. The Model Context Protocol (MCP) presents capabilities as named tools with schema-defined arguments (Anthropic, 2024; Model Context Protocol Contributors, 2026), whereas a command-line interface (CLI) expresses operations through commands, flags, and textual conventions. These interfaces can encode the same semantic operations while presenting systematically different cues for selecting and expressing them. Both MCP and CLI are now widely used in modern agent systems (Anthropic, 2025a;b; Mullen & Salva, 2025; GitHub, 2025; OpenAI, 2025), making a systematic study of how their representational differences



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.

affect agent behavior increasingly important. MCP and CLI therefore provide a natural setting for disentangling *what an agent can do* from *how those capabilities are represented to the model*.

To answer this question, we need a controlled comparison that holds executable semantics fixed while varying only the model-facing representation. To make such a comparison informative, however, we must solve two attribution problems. First, system-level MCP–CLI comparisons often change more than the interface. Tool granularity (Béchard et al., 2026), prompting, public information, validation, or execution scaffolding can change at the same time (Alier Forment et al., 2026; Xu et al., 2026). A performance difference under such conditions cannot be attributed to representation alone. Second, even after executable semantics are matched, endpoint metrics collapse distinct interaction pathways (Wang et al., 2026). The same final score can arise from different semantic choices, realization failures, and interaction costs, while an early local difference can alter the history and state conditioning all later decisions. Systematically comparing how MCP and CLI representations shape agent behavior therefore requires both *semantic matching* and *trajectory-level attribution*.

We introduce Matched Interface Comparison and Analysis (MICA) to systematically compare MCP and CLI under matched executable semantics. By holding the underlying executable capabilities fixed, MICA attributes behavioral differences to model-facing representation rather than differences in what the agent can do. It then analyzes interface effects at two complementary scales. At fixed decision contexts, *matched-state probes* vary capability presentation, realization, and feedback to localize where interface sensitivity enters the policy. Along *paired natural trajectories*, MICA identifies the first consequential divergence and traces how that local difference propagates through subsequent history, state, recovery, cost, and task outcome. This analysis further separates semantic action selection from interface-specific realization and identifies realization as a direct intervention boundary. A semantic action may already be valid even when its interface-specific expression fails. Guided by this analysis, we introduce Decoupled Action Realization and Targeted Repair (DART) to address interface-specific realization failures without discarding an already validated semantic decision. Once a semantic action is validated, DART preserves it as a fixed commitment. If its realization fails while the commitment remains valid, DART repairs only the interface-specific expression rather than selecting a new semantic action.

Our experiments reveal a consistent mechanistic picture. Across six models and three interactive environments, MCP and CLI with matched executable capabilities still produce substantial differences in task performance and interaction cost. These differences extend beyond surface validity. Interface representation changes semantic action selection, while matched-state analysis shows that presentation, realization, and feedback interact strongly in shaping local behavior. Paired trajectories further show that consequential divergence is widespread and often emerges early, after which interaction histories and costs separate. Acting on the realization boundary identified by MICA, DART provides a direct intervention on the identified realization mechanism, improving task performance across all six evaluated benchmark–interface settings. Under MCP, it also reduces interaction cost. Together, these results establish interface representation as an active part of the closed-loop decision environment through which agent policies are realized.

In summary, our contributions are as follows:

- We provide a study of MCP and CLI under matched executable capabilities, showing that interface representation can substantially alter agent behavior and interaction cost.
- We introduce Matched Interface Comparison and Analysis (MICA), which localizes interface sensitivity and traces how local differences propagate through closed-loop interaction.
- We introduce Decoupled Action Realization and Targeted Repair (DART), which repairs interface-specific realization while preserving validated semantic decisions, enabling more reliable recovery from the realization failures identified by MICA.

## 2 Related Work

**Tool Use and Interface Representation.** Language agents have progressed from interleaved reasoning and acting (Yao et al., 2023) toward stateful, multi-step interaction with external tools and environments (Patil et al., 2025; Lu et al., 2025; Wang et al., 2024b; Yao et al., 2025). A growing body of work shows that agent behavior depends on how available actions are represented to the model. Executable code, structured tool calls, and textual actions induce different interaction patterns

(Wang et al., 2024a). Agent–computer interface design also affects downstream behavior (Yang et al., 2024), while changes in tool names, descriptions, ordering, and other interface metadata alter action selection (Blankenstein et al., 2026; Yuan et al., 2025; Xu et al., 2026). Related work on constrained generation and conversational recovery further shows that valid action realization and post-failure correction are integral parts of tool-mediated interaction (Banerjee et al., 2025; Kim et al., 2026). Together, these studies establish interface representation as an important source of behavioral variation and motivate controlled attribution under matched executable semantics.

**MCP-based Tool Use.** The Model Context Protocol standardizes tool discovery and invocation through named tools with schema-defined arguments (Anthropic, 2024; Model Context Protocol Contributors, 2026). Recent benchmarks including MCP-Bench (Wang et al., 2026), MCPMark (Wu et al., 2026), MCP-AgentBench (Guo et al., 2026), and MCP-Universe (Luo et al., 2025) evaluate increasingly complex MCP interaction involving multi-step planning, tool discovery, and cross-tool coordination. These benchmarks characterize agent capability within MCP-native tool representations and provide broad coverage of realistic MCP use. System-level comparisons of MCP and CLI across multiple agent scaffoldings also report substantial variation across deployed configurations (Alier Forment et al., 2026). MICA studies a complementary question by holding the semantic action contract and post-decoding runtime fixed while varying the model-facing presentation, realization, and feedback components of the interface.

**CLI and Terminal Agents.** Terminal environments expose actions through textual commands and iterative execution feedback, as studied in InterCode (Yang et al., 2023) and Terminal-Bench (Merrill et al., 2026). These environments capture central properties of command-line interaction including broad command grammars, compositional actions, and long-horizon workflows. Recent enterprise-automation work directly compares terminal agents with MCP-based tool registries and shows that capability packaging explains a substantial part of the observed performance difference (Béchard et al., 2026). This finding makes capability matching a central control for studying interface representation. Our construction therefore aligns benchmark-specific semantic operations and downstream execution semantics before comparing MCP-style tool calls with CLI-style commands.

## 3 Problem Formulation

**Semantic actions and capability matching.** We consider a language agent $\pi_\theta$ interacting with a stateful environment. At decision step $t$, the environment is in state $s_t$, while the agent observes a policy-visible observation $o_t$ and public interaction history $h_t$. The benchmark defines a semantic action contract $\mathcal{C}_t$, under which the agent selects

$$a_t = (u_t, v_t) \in \mathcal{A}_{\mathcal{C}_t}(s_t),$$

where $u_t$ denotes an operation and $v_t$ its normalized typed arguments. We write $a \equiv_{\text{sem}} a'$ when two proposals specify the same registered operation with equivalent normalized arguments under the active contract. This normalized semantic action is the common unit of comparison across interfaces. MCP and CLI are constructed from the same benchmark-specific contract and therefore expose matched executable capabilities through different model-facing representations.

**Interface factorization and shared execution.** For $I \in \{\text{MCP}, \text{CLI}\}$, we represent the interface as $I = (\Phi_I, R_I, \Psi_I)$ with $R_I = (\rho_I, D_I)$. Here, $\Phi_I$ specifies capability presentation, $\rho_I$ the response protocol, $D_I$ the interface-specific decoder, and $\Psi_I$ the feedback representation. Given

$$x_t^I = \Phi_I(\mathcal{C}_t, o_t, h_t),$$

the model samples $y_t^I \sim \pi_\theta(\cdot \mid x_t^I)$ and produces a decoded proposal

$$\hat{a}_t^I = D_I(y_t^I) \in \mathcal{A}_{\mathcal{C}_t}(s_t) \cup \{\perp\},$$

where $\perp$ denotes a response that does not decode to a registered semantic action. Decoded proposals then enter the shared semantic validation, native-action compilation, and execution path

$$\hat{a}_t^I \xrightarrow{V,K} n_t \xrightarrow{E} (s_{t+1}, \omega_t), \qquad z_t^I = \Psi_I(\omega_t). \tag{1}$$

Here, $V$, $K$, and $E$ denote semantic validation, native-action compilation, and environment execution, respectively, and $n_t$ is the compiled native action. Model-facing variation enters through $(\Phi_I, \rho_I, D_I, \Psi_I)$, while both interfaces share these downstream runtime operators. This factorization distinguishes response decoding, semantic admission, and environment execution.

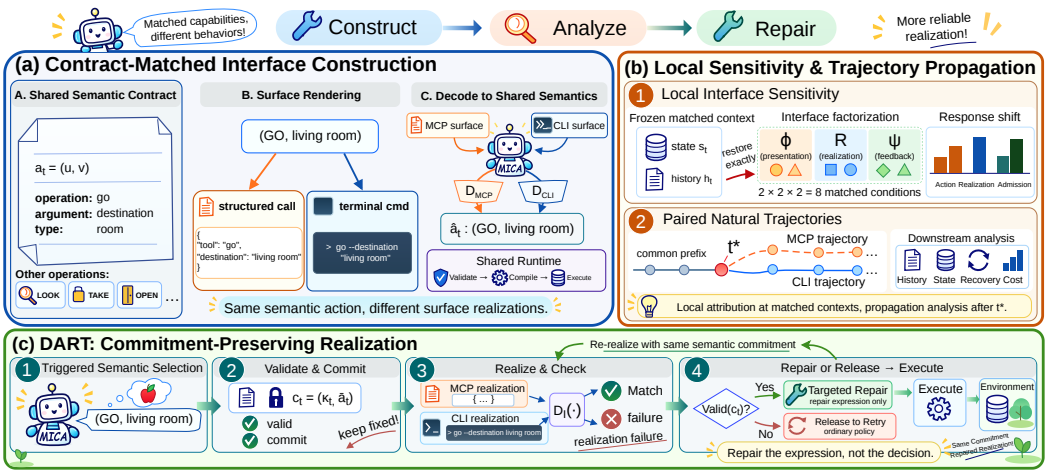



Figure 1: **The overview of MICA.** MCP and CLI render a shared semantic action contract through different model-facing interfaces while sharing downstream validation and execution. Matched-state probes localize interface sensitivity, paired trajectories trace its propagation, and DART preserves semantic commitments during realization repair.

**Interaction equivalence and consequential divergence.** Interface effects are analyzed at two complementary scales. At a matched decision context, fixing the environment state, public history, semantic contract, model, and generation configuration supports local comparison under different assignments of $\Phi$, $R$, and $\Psi$. Along closed-loop interaction, an early difference changes the observations, histories, and environment states that condition subsequent decisions.

Let

$$b_t^I = (\hat{a}_t^I, f_t^I, \omega_t^I, r_t^I)$$

denote the interaction signature at step $t$, consisting of the semantic proposal, failure stage, execution outcome, and recovery status. We write $b_t^I \equiv b_t^J$ when the proposals are semantically equivalent and the remaining components agree. The first consequential divergence is

$$t^\star = \min\left\{t \mid b_t^{\mathrm{MCP}} \not\equiv b_t^{\mathrm{CLI}}\right\}. \tag{2}$$

Pairs with no such step are recorded as having no observed divergence. Surface variation remains within the common prefix whenever both responses decode to the same semantic action and induce the same interaction outcome. Before $t^\star$, the aligned contexts support matched interface comparison. After $t^\star$, each episode evolves under its own history and environment state, and subsequent differences characterize downstream propagation. This separation provides the basis for the matched-state and trajectory analyses in Section 4.2.

## 4 Matched Interface Comparison and Analysis

Matched Interface Comparison and Analysis (MICA) isolates the behavioral effect of interface representation while keeping executable capabilities fixed. It combines contract-matched MCP–CLI construction, matched-state and trajectory analysis, and commitment-preserving realization with DART (Figure 1).

### 4.1 Contract-Matched Interface Construction

**Shared semantic contract.** MICA constructs MCP and CLI from a common benchmark-specific semantic contract, fixing the executable action space while varying its model-facing representation. At decision step $t$, the contract is

$$\mathcal{C}_t = \left\{\left(u, \Sigma_u, \mathcal{V}_u(s_t), \Gamma_u\right) \mid u \in \mathcal{U}_t\right\}, \tag{3}$$

where $\Sigma_u$ specifies the typed arguments of operation $u$, $\mathcal{V}_u(s_t)$ its state-local argument domain, and $\Gamma_u$ its native execution binding.

**Interface rendering and shared execution.** For $I \in \{\mathrm{MCP}, \mathrm{CLI}\}$, MICA renders the contract as $\mathcal{S}_t^I = \mathrm{Render}_I(\mathcal{C}_t)$. MCP exposes named tools with structured arguments, whereas CLI exposes registered commands with textual argument bindings. Both interfaces decode responses into the same normalized semantic action space and then enter the shared validation, compilation, and execution path in Equation 1. Consequently, surface syntax can vary while executable semantics remain fixed. Appendices C.2 and C.3 give the rendering rules, benchmark-specific mappings, and conformance checks.

### 4.2 Local Interface Sensitivity and Trajectory Propagation

MICA separates local interface sensitivity from its downstream propagation. Matched-state probes vary interface components at a fixed decision context, while paired natural trajectories trace what follows after the first consequential divergence.

**Matched-state probes.** At a registered decision context, let $\xi = (\xi_\Phi, \xi_R, \xi_\Psi) \in \{\mathrm{MCP}, \mathrm{CLI}\}^3$ denote the presentation, realization, and feedback forms assigned to the model. Holding the environment state, public history, semantic contract, model, and generation configuration fixed, each assignment induces a local response distribution $P_\xi^t(b) = P(b_t = b \mid s_t, h_t, \mathcal{C}_t; \xi)$. The two homogeneous assignments recover the primary MCP and CLI interfaces. The remaining six exchange individual components under the same semantic contract and downstream runtime.

For $k \in \{\Phi, R, \Psi\}$, let $\xi^{k \leftarrow I}$ replace component $k$ with its form under interface $I$. We summarize the component main effect for a registered contrast $d$ as

$$\Delta_k(d) = \mathbb{E}_{\xi_{-k}} \left[ d\Big( P^t_{\xi^{k \leftarrow \mathrm{MCP}}}, P^t_{\xi^{k \leftarrow \mathrm{CLI}}} \Big) \right]. \tag{4}$$

This averages the MCP–CLI contrast for component $k$ over assignments of the remaining components. The same $2 \times 2 \times 2$ factorial coding yields the pairwise and three-way interaction contrasts used in the local analysis. Sampling and estimation details are given in Appendix B.4.

**Natural trajectory propagation.** Paired MCP and CLI episodes begin from aligned initial conditions and are compared through the first consequential divergence $t^\star$ defined in Equation 2. Before $t^\star$, the aligned interaction supports matched comparison. After $t^\star$, each episode follows its own history and environment state. We characterize this natural suffix through semantic-action agreement, failure and recovery, interaction cost, and terminal outcome, thereby separating local attribution from downstream propagation. Alignment rules and suffix-level estimators are detailed in Appendix B.3.

### 4.3 DART: Commitment-Preserving Realization

DART (**D**ecoupled **A**ction **R**ealization and **T**argeted Repair) targets the realization boundary exposed by MICA. It separates semantic action selection from interface-specific expression so that a validated decision remains fixed while its realization is repaired.

**Semantic commitment and realization.** DART first elicits a normalized semantic action $\bar{a}_t = (u_t, v_t)$ under the active public contract, binds it to the contract identity $\kappa_t$, and then realizes the resulting commitment through the assigned interface,

$$\bar{a}_t \sim \pi^{\mathrm{sel}}_{\theta,I}\big(\cdot \mid x_t^I\big), \qquad c_t = (\kappa_t, \bar{a}_t), \qquad y_t^I \sim \pi^{\mathrm{real}}_{\theta,I}\big(\cdot \mid x_t^I, c_t\big). \tag{5}$$

Both phases use the same underlying model with phase-specific response contracts. Selection fixes the semantic operation and typed arguments. The realization phase determines only their MCP or CLI expression. A realization is admitted when it decodes to the committed action and the commitment remains valid under the active contract. The complete preservation gate is specified in Appendix D.

**Commitment-preserving repair.** If the initial realization fails while $c_t$ remains valid, DART keeps the commitment fixed and issues one preserving correction conditioned on the failed expression and its public feedback. The correction may change the interface-specific expression while preserving the committed operation and every normalized typed argument. An admitted correction proceeds through the shared runtime. Otherwise, control returns to the ordinary retry policy. This intervention isolates

realization repair from semantic reselection, providing a direct test of the realization mechanism identified by MICA. Trigger scopes, controller logic, and phase-specific prompts are detailed in Appendices D and E.

# 5 Experimental Setup

**Benchmarks and models.** We evaluate agents in three interactive environments with complementary demands, namely ALFWorld (Shridhar et al., 2021) for grounded household interaction, WebShop (Yao et al., 2022) for goal-directed web shopping, and ScienceWorld (Wang et al., 2022) for multi-step scientific reasoning. Our model suite includes Qwen3-4B/8B/14B/32B (Yang et al., 2025), DeepSeek-v4-Flash (DeepSeek-AI, 2026), and Gemini-2.5-Flash (Comanici et al., 2025). Native benchmark interfaces are retained as calibration references.

**Controlled evaluation.** For each task, MCP and CLI are rendered from the same semantic action contract (Section 4.1) while holding fixed the model, initial task state, public information, semantic action space, validation and execution pipeline, generation settings, and interaction budget. Primary metrics are benchmark score or success and total charged model tokens per task. We further measure semantic action selection, failure and recovery behavior, matched-state component sensitivity, first consequential divergence, and post-divergence trajectory behavior. For RQ4, Gemini-2.5-Flash is evaluated in all six benchmark–interface settings against a protocol-matched Base. DART changes only the commitment-preserving controller and registered trigger configuration. Full task sets, budgets, estimators, and DART protocol details are given in Appendices B and D.

**Research questions.** We organize the evaluation around four research questions. **RQ1.** How does interface representation affect task performance and interaction cost under matched executable capabilities? **RQ2.** How does interface representation alter semantic decisions and recovery behavior? **RQ3.** Where do interface-induced differences originate, and how do they propagate through subsequent interaction? **RQ4.** How does commitment-preserving realization affect task performance and interaction cost after realization failures?

# 6 Results

We report the results in the order of the four research questions, moving from task-level performance and interaction cost to failure and recovery, local interface effects and trajectory propagation, and finally the DART intervention.

## 6.1 Task Performance and Interaction Cost

**Matched capabilities produce substantial endpoint differences.** Table 1 shows that both the direction and magnitude of the interface effect vary across model–environment pairs. For Qwen3-14B, MCP trails CLI by 36.6 score points on ALFWorld (20.1 vs. 56.7) but exceeds it by 4.9 points on WebShop (64.4 vs. 59.5). The interaction-cost effect reverses at the same time. MCP uses 54.9k more tokens per ALFWorld task but 35.0k fewer on WebShop. Comparable reversals occur across the model suite, showing that task performance and interaction cost respond differently to the same interface change.

Interface representation therefore acts as a model- and environment-dependent factor in the interaction loop, redistributing difficulty across operation selection, argument grounding, and action realization. The following analyses trace these endpoint differences to changes in semantic decisions, failures, and recovery.

## 6.2 Failure and Recovery

**Interface representation changes semantic action selection after successful decoding.** The effect therefore extends beyond surface realization. For WebShop/Qwen3-14B, MCP shifts decoded action frequency by $+6$ percentage points toward `click` and $-6$ points from `search`. ALFWorld and ScienceWorld exhibit corresponding redistributions across their larger operation sets (Appendix A.2). The shared semantic action space is thus exercised differently under the two interface representations.

| Model | Score (%) | | | Tokens (k / task) | | |
|---|---|---|---|---|---|---|
| | Native | CLI (Δ) | MCP (Δ) | Native | CLI (Δ%) | MCP (Δ%) |
| **ALFWorld** | | | | | | |
| Qwen3-4B | 50.0 | 44.8$_{-5.2}$ | 41.8$_{-8.2}$ | 295.5 | 384.2$_{+30.0\%}$ | 387.1$_{+31.0\%}$ |
| Qwen3-8B | 31.3 | 44.8$_{+13.4}$ | 35.1$_{+3.7}$ | 273.5 | 292.4$_{+6.9\%}$ | 356.5$_{+30.3\%}$ |
| Qwen3-14B | 62.7 | 56.7$_{-6.0}$ | 20.1$_{-42.5}$ | 176.9 | 241.9$_{+36.7\%}$ | 296.8$_{+67.8\%}$ |
| Qwen3-32B | 85.1 | 73.9$_{-11.2}$ | 78.4$_{-6.7}$ | 72.4 | 114.7$_{+58.3\%}$ | 106.5$_{+46.9\%}$ |
| DeepSeek-v4-Flash | 97.0 | 99.3$_{+2.2}$ | 95.5$_{-1.5}$ | 50.5 | 58.0$_{+14.8\%}$ | 56.7$_{+12.4\%}$ |
| Gemini-2.5-Flash | 67.2 | 56.7$_{-10.4}$ | 67.2$_{0.0}$ | 86.7 | 120.0$_{+38.3\%}$ | 113.8$_{+31.2\%}$ |
| **WebShop** | | | | | | |
| Qwen3-4B | 62.9 | 58.3$_{-4.6}$ | 64.1$_{+1.2}$ | 32.5 | 37.6$_{+15.9\%}$ | 36.3$_{+11.7\%}$ |
| Qwen3-8B | 57.7 | 59.5$_{+1.8}$ | 56.8$_{-0.9}$ | 66.2 | 89.6$_{+35.4\%}$ | 78.8$_{+19.0\%}$ |
| Qwen3-14B | 54.9 | 59.5$_{+4.6}$ | 64.4$_{+9.5}$ | 92.9 | 83.1$_{-10.5\%}$ | 48.1$_{-48.3\%}$ |
| Qwen3-32B | 64.0 | 61.6$_{-2.4}$ | 65.7$_{+1.7}$ | 85.8 | 80.4$_{-6.3\%}$ | 88.6$_{+3.3\%}$ |
| DeepSeek-v4-Flash | 86.0 | 83.0$_{-3.0}$ | 83.5$_{-2.5}$ | 63.8 | 69.1$_{+8.3\%}$ | 65.1$_{+2.1\%}$ |
| Gemini-2.5-Flash | 82.2 | 84.5$_{+2.3}$ | 83.3$_{+1.0}$ | 54.9 | 42.8$_{-22.1\%}$ | 41.9$_{-23.8\%}$ |
| **ScienceWorld** | | | | | | |
| Qwen3-4B | 37.5 | 41.0$_{+3.5}$ | 37.9$_{+0.4}$ | 100.8 | 106.9$_{+6.1\%}$ | 108.7$_{+7.9\%}$ |
| Qwen3-8B | 51.1 | 46.9$_{-4.2}$ | 53.0$_{+1.9}$ | 82.6 | 82.5$_{-0.2\%}$ | 101.1$_{+22.5\%}$ |
| Qwen3-14B | 35.8 | 39.5$_{+3.7}$ | 35.1$_{-0.7}$ | 60.6 | 73.4$_{+21.0\%}$ | 76.0$_{+25.3\%}$ |
| Qwen3-32B | 63.9 | 61.3$_{-2.6}$ | 60.7$_{-3.2}$ | 73.7 | 80.9$_{+9.9\%}$ | 83.8$_{+13.8\%}$ |
| DeepSeek-v4-Flash | 81.3 | 77.5$_{-3.7}$ | 75.3$_{-6.0}$ | 77.2 | 97.4$_{+26.1\%}$ | 93.1$_{+20.5\%}$ |
| Gemini-2.5-Flash | 52.4 | 51.2$_{-1.2}$ | 54.0$_{+1.6}$ | 61.0 | 65.2$_{+7.0\%}$ | 59.2$_{-2.8\%}$ |

Table 1: **Task performance and interaction cost under matched interfaces.** MCP and CLI expose the same executable capabilities through different model-facing representations. Native is included as a calibration reference.

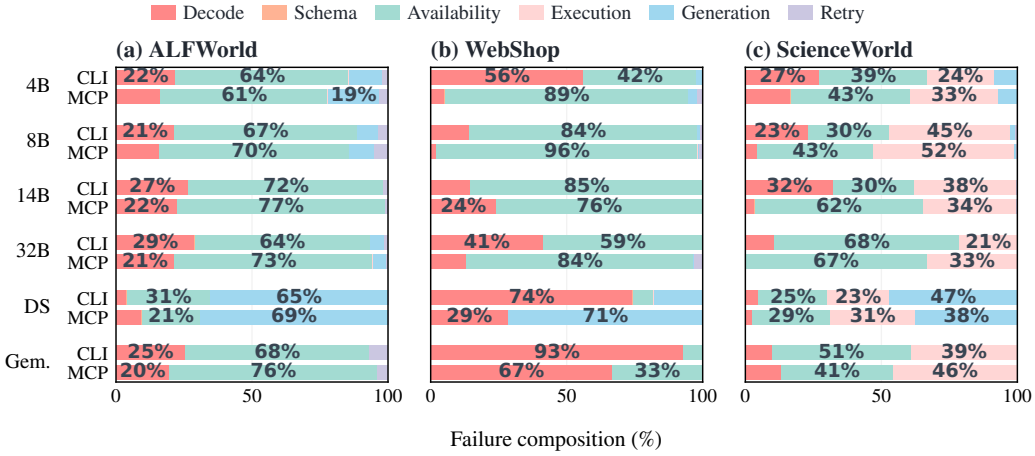



Figure 2: **Failure pathways under MCP and CLI.** Bars show the composition of failure stages among unsuccessful attempts. The dominant failure pathway changes with both the interface and environment.

**Interfaces reshape both failure incidence and failure location.** Figure 2 shows that unsuccessful attempts enter different stages of the action pipeline under MCP and CLI. On WebShop, MCP lowers the attempt-level failure rate for all six models by 1.1–8.9 percentage points, while the remaining failures shift from the stronger surface-decoding component under CLI toward later availability and execution stages. ScienceWorld exhibits a different failure composition, with availability and execution accounting for a larger share of unsuccessful attempts. Interface representation therefore determines both the frequency and stage of interaction failure.

**Recovery behavior carries interface effects into subsequent decisions.** After failure feedback, the next response can preserve the semantic decision and repair its realization, select a different action, repeat the error, or enter another failure mode. On WebShop/Qwen3-4B, retry acceptance is $50\%$ under CLI and $77\%$ under MCP, while later task success is $18\%$ and $34\%$, respectively (Appendix A.2). These interface-specific recovery paths update the visible interaction history and thereby change the context conditioning later decisions.

### 6.3 Local Effects and Trajectory Propagation

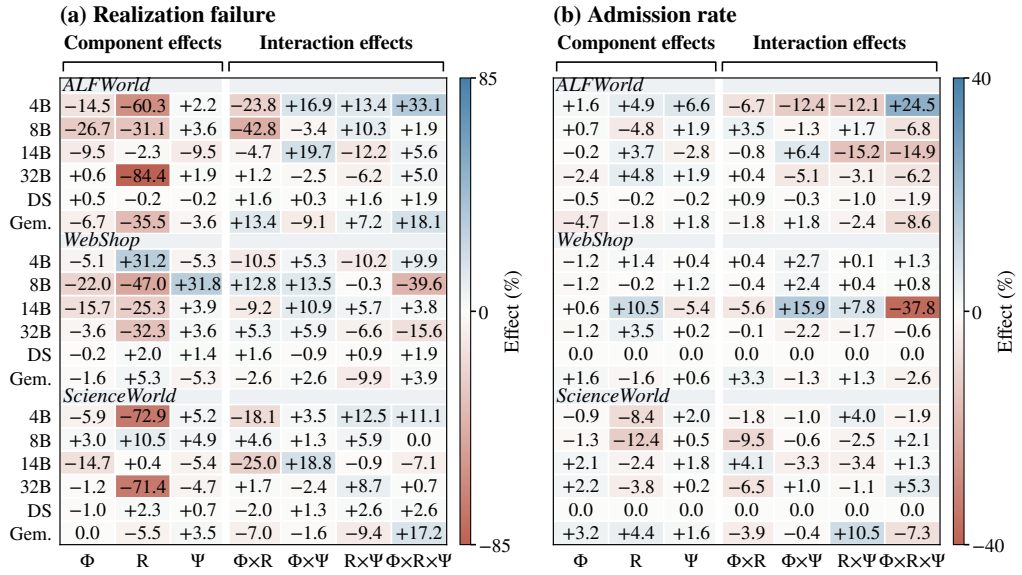



Figure 3: **Component and interaction effects on local response behavior.** Effects on (a) realization failure and (b) admission given decoding are shown for presentation $\Phi$, realization $R$, and feedback $\Psi$. Main effects compare CLI with MCP while averaging over the remaining blocks. Interaction terms capture non-additivity under the factorial coding. Each panel uses a symmetric percentage-point scale centered at zero.

**Local interface effects are large and strongly compositional.** Figure 3 separates presentation, realization, and feedback at matched decision states. Realization produces the largest local shifts in several strata. Its realization-failure effect reaches 84.4 percentage points for ALFWorld/Qwen3-32B and 72.9 points for ScienceWorld/Qwen3-4B. Interaction terms are also substantial. For example, the three-way interaction in WebShop/Qwen3-14B changes admission by 37.8 points. The behavioral effect of each interface component therefore depends on the surrounding interface configuration.

**Local divergence propagates through the interaction trajectory.** Across 2,004 paired MCP–CLI episodes, 1,889 ($94.3\%$) exhibit a structured divergence, with first divergence occurring near the beginning of interaction across model–environment strata (Appendix A.3). Semantic-action differences account for the dominant share of divergent pairs, with realization differences providing a recurring second source. After divergence, the paired trajectories accumulate different token costs and attempt counts and reach distinct terminal outcomes across settings. MICA thereby connects matched-state interface sensitivity to the trajectory-level performance and cost differences observed in Section 6.1.

### 6.4 DART Evaluation

**DART improves task performance in every evaluated setting.** Figure 4 reports gains of $+1.5/+9.0$ points on ALFWorld, $+0.8/+4.4$ on WebShop, and $+6.7/+5.3$ on ScienceWorld for CLI/MCP, respectively. Under MCP, these gains are accompanied by token reductions of 27.1k, 11.2k, and 4.3k per task across the three environments. Under CLI, token cost changes by $-0.8$k, $+4.8$k, and $+1.6$k.

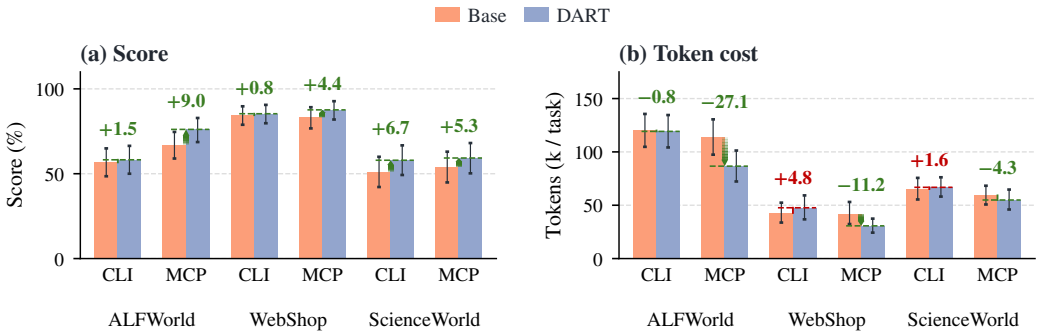



Figure 4: **Effect of DART on task performance and interaction cost.** Each Gemini-2.5-Flash Base–DART pair matches the benchmark, interface, task set, generation settings, and episode budget. Labels show the DART-minus-Base change.

**Commitment preservation concentrates recovery at the realization boundary.** DART keeps the validated semantic commitment fixed and changes only its interface-specific realization. The event-level analysis in Appendix A.4 shows that 71–100% of triggered interventions are admitted on the first realization, and three of the four preserving-correction opportunities are admitted after a single correction. The preserving branch therefore resolves interface-specific realization failures while keeping semantic selection fixed, directly linking the mechanism identified by MICA to the score improvements in Figure 4.

## 7 Discussion

**Interface representation is part of the decision process.** Under matched executable capabilities, MCP and CLI share the same semantic action contract and downstream runtime yet induce different policies because they present, realize, and return feedback about those capabilities differently. Interface design should therefore be treated as part of the agent's decision environment rather than as a passive access layer.

**Trajectory-level behavior emerges from local interface effects.** MICA links local sensitivity to downstream behavior by combining matched-state probes with paired natural trajectories. Presentation, realization, and feedback can change a local response, after which the resulting history and state condition subsequent decisions. Later semantic realignment can occur after the trajectories have already followed different histories, making the trajectory the appropriate unit for analyzing interactive agent behavior.

**Semantic commitment separates decision from realization during recovery.** DART turns this distinction into an explicit controller boundary. Once a semantic action is validated, recovery preserves that commitment and repairs only its interface-specific realization. Language-agent controllers can improve recovery by preserving established semantic decisions and restricting repair to the representation-specific component that requires correction.

## 8 Conclusion

Our study establishes interface representation as a consequential component of interactive language-agent behavior under matched executable capabilities. MICA combines contract-matched interfaces, matched-state probes, and paired natural trajectories to connect local representational sensitivity with downstream differences in semantic decisions, recovery, interaction cost, and task outcomes. Building on this mechanism, DART preserves validated semantic commitments while repairing interface-specific realization and improves benchmark score in all six evaluated benchmark–interface settings. Together, these results show that interface design shapes the effective decision environment through which an agent acts. For interactive language agents, how capabilities are represented is part of how the resulting policy is realized through interaction.

## ETHICS STATEMENT

This work operates within existing action, validation, and execution boundaries and does not expand the executable capabilities available to the agent. MICA varies only the model-facing representation of a shared semantic action contract while preserving the underlying capability set and downstream runtime. DART likewise operates within the same contract and execution pipeline by preserving validated semantic commitments during interface-specific realization repair. The study therefore focuses on understanding and improving the reliability of tool-mediated agent interaction without introducing additional action authority or altering the safeguards defined by the evaluation environment.

## REPRODUCIBILITY STATEMENT

We provide the experimental details required to reproduce the study throughout the main paper and appendices. Section 3 defines the semantic comparison boundary, interface factorization, and consequential divergence. Section 4 specifies MICA and DART, while Section 5 summarizes the controlled evaluation setting. Appendix B provides the task sets, generation settings, interaction budgets, evaluation protocols, metrics, and statistical analysis. Appendix C specifies the contract-matched MCP and CLI construction and conformance checks. Appendix D gives the DART controller, trigger configuration, and evaluation protocol. Appendix E provides the ordinary policy and phase-specific response contracts used in the experiments. Code is available in the Supplementary Material.

## AI USE STATEMENT

Generative AI tools were used as assistive tools during the preparation of this work, including language polishing, code implementation assistance, and the preparation of figures. All AI-assisted material was reviewed and incorporated by the authors, who retain full responsibility for the technical content and final presentation of the paper.

## REFERENCES


Marc Alier Forment, María José Casañ Guerrero, Francisco José García-Peñalvo, and Juanan Pereira. The scaffolding matters more than the interface: A controlled comparison of MCP and CLI tool use across seven agent scaffoldings, five language models, and one software task. *arXiv preprint arXiv:2608.08654*, 2026.

Anthropic. Introducing the Model Context Protocol. Anthropic announcement, 2024. URL https://www.anthropic.com/news/model-context-protocol. Accessed 2026-08-26.

Anthropic. Claude 3.7 Sonnet and Claude Code. Anthropic announcement, 2025a. URL https://www.anthropic.com/news/claude-3-7-sonnet. Accessed 2026-08-26.

Anthropic. Remote MCP support in Claude Code. Anthropic announcement, 2025b. URL https://claude.com/blog/claude-code-remote-mcp. Accessed 2026-08-26.

Debangshu Banerjee, Tarun Suresh, Shubham Ugare, Sasa Misailovic, and Gagandeep Singh. CRANE: Reasoning with constrained LLM generation. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 2836–2857, 2025.

Patrice Béchard, Orlando Marquez Ayala, Emily Chen, Jordan Skelton, Sagar Davasam, Srinivas Sunkara, Vikas Yadav, and Sai Rajeswar. Terminal agents suffice for enterprise automation. *arXiv preprint arXiv:2604.00073*, 2026.

Thierry Blankenstein, Jialin Yu, Zixuan Li, Vassilis Plachouras, Sunando Sengupta, Philip Torr, Yarin Gal, Alasdair Paren, and Adel Bibi. BiasBusters: Uncovering and mitigating tool selection bias in large language models. In *International Conference on Learning Representations*, 2026.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Bradley Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1): 1–26, 1979.

GitHub. GitHub Copilot CLI is now in public preview. GitHub Changelog, 2025. URL https://github.blog/changelog/2025-09-25-github-copilot-cli-is-now-in-public-preview/. Accessed 2026-08-26.

Zikang Guo, Benfeng Xu, Chiwei Zhu, Wentao Hong, Xiaorui Wang, and Zhendong Mao. MCP-AgentBench: Evaluating real-world language agent performance with MCP-mediated tools. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(37):30888–30896, 2026.

Takyoung Kim, Jinseok Nam, Chandrayee Basu, Xing Fan, Chengyuan Ma, Heng Ji, Gokhan Tur, and Dilek Hakkani-Tür. ReIn: Conversational error recovery with reasoning inception. In *International Conference on Learning Representations*, 2026.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3102–3116, 2023.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, 2025.

Ziyang Luo, Zhiqi Shen, Wenzhuo Yang, Zirui Zhao, Prathyusha Jwalapuram, Amrita Saha, Doyen Sahoo, Silvio Savarese, Caiming Xiong, and Junnan Li. MCP-Universe: Benchmarking large language models with real-world Model Context Protocol servers. *arXiv preprint arXiv:2508.14704*, 2025.

Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, et al. Terminal-Bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, 2026.

Model Context Protocol Contributors. Model Context Protocol specification: Tools. Protocol revision 2026-07-28, 2026. URL https://modelcontextprotocol.io/specification/2026-07-28/server/tools. Accessed 2026-08-26.

Taylor Mullen and Ryan J. Salva. Gemini CLI: Your open-source AI agent. Google announcement, 2025. URL https://blog.google/technology/developers/introducing-gemini-cli-open-source-ai-agent/. Accessed 2026-08-26.

OpenAI. Codex CLI: Frontier reasoning in the terminal. OpenAI announcement, 2025. URL https://openai.com/index/introducing-o3-and-o4-mini/. Accessed 2026-08-26.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The Berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 48371–48392, 2025.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Côté, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 11279–11298, 2022.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 50208–50232, 2024a.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. MINT: Evaluating LLMs in multi-turn interaction with tools and language feedback. In *International Conference on Learning Representations*, 2024b.

Zhenting Wang, Qi Chang, Hemani Patel, Shashank Biju, Cheng-En Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Parag Shah, Yujia Bao, and Eugene Siow. MCP-Bench: Benchmarking tool-using LLM agents with complex real-world tasks via MCP servers. In *International Conference on Learning Representations*, 2026.

Zijian Wu, Xiangyan Liu, Xinyuan Zhang, Lingjun Chen, Fanqing Meng, Lingxiao Du, Yiran Zhao, Fanshi Zhang, Yaoqi Ye, Jiawei Wang, Zirui Wang, Jinjie Ni, Yufan Yang, Arvin Xu, and Michael Qizhe Shieh. MCPMark: A benchmark for stress-testing realistic and comprehensive MCP use. In *International Conference on Learning Representations*, 2026.

Xiangzhe Xu, Hamidreza Saghir, Qianhui Wu, Marc-Alexandre Côté, Tong Wang, Kiran Lakkaraju, Kexin Pei, and Xiangyu Zhang. The devil is in the interface: Evaluating how tool architecture shapes coding agent behavior. *arXiv preprint arXiv:2608.11386*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

John Yang, Akshara Prabhakar, Karthik Narasimhan, and Shunyu Yao. InterCode: Standardizing and benchmarking interactive coding with execution feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

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

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

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. $\tau$-bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, 2025.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Kan Ren, Dongsheng Li, and Deqing Yang. EASYTOOL: Enhancing LLM-based agents with concise tool instruction. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 951–972. Association for Computational Linguistics, 2025.

# Appendix

## Appendix Contents

**A Supplementary Results by Research Question** **15**
A.1 RQ1: Endpoint Performance and Interaction Cost . . . . . 15
A.2 RQ2: Semantic Decisions, Failures, and Recovery . . . . . 16
A.3 RQ3: Local Origins and Downstream Propagation . . . . . 17
A.4 RQ4: Commitment-Preserving Realization with DART . . . . . 21

**B Experimental Protocol and Statistical Analysis** **24**
B.1 Evaluation Suite and Controlled Comparison . . . . . 24
B.2 Generation, Context, and Interaction Budgets . . . . . 24
B.3 Paired Episode Evaluation Protocol . . . . . 24
B.4 Matched-State Probe Protocol . . . . . 25
B.5 Metrics and Statistical Analysis . . . . . 25

**C Contract-Matched Interface Construction** **26**
C.1 Shared Semantic Boundary . . . . . 26
C.2 Surface Rendering and Conformance Checks . . . . . 26
C.3 Benchmark-Specific Contract Realizations . . . . . 27
C.4 Worked Round-Trip Example . . . . . 28

**D DART: Controller, Triggers, and Evaluation Protocol** **28**
D.1 Commitment-Preserving Realization . . . . . 29
D.2 Error-Triggered Controller . . . . . 29
D.3 Trigger Scope and Benchmark Instantiation . . . . . 29
D.4 RQ4 Paired Evaluation . . . . . 30

**E Prompt Templates and Response Contracts** **30**
E.1 Prompt Assembly and Ordinary Policy . . . . . 30
E.2 Semantic Selection Contract . . . . . 31
E.3 Realization and Preserving Correction . . . . . 32

# A Supplementary Results by Research Question

This appendix extends the main-text results with finer-grained evidence for the four research questions. RQ1 resolves endpoint performance and interaction cost to individual tasks and resource components. RQ2 follows the interaction pipeline from semantic action selection through failure and recovery. RQ3 localizes interface sensitivity at matched decision states and traces how the first consequential divergence propagates through natural trajectories. RQ4 then examines how DART acts at the realization boundary identified by this analysis.

## A.1 RQ1: Endpoint Performance and Interaction Cost

**Interface representation reshapes token use at the task level.** Figure S1 resolves the aggregate token costs in Table 1 into paired task-level effects. The distributions show substantial within-stratum variation. MCP uses more tokens on some tasks and fewer on others even for the same model and environment. The aggregate cost differences in Table 1 therefore arise from systematic shifts in the distribution of task-level interaction rather than from a uniform interface overhead.

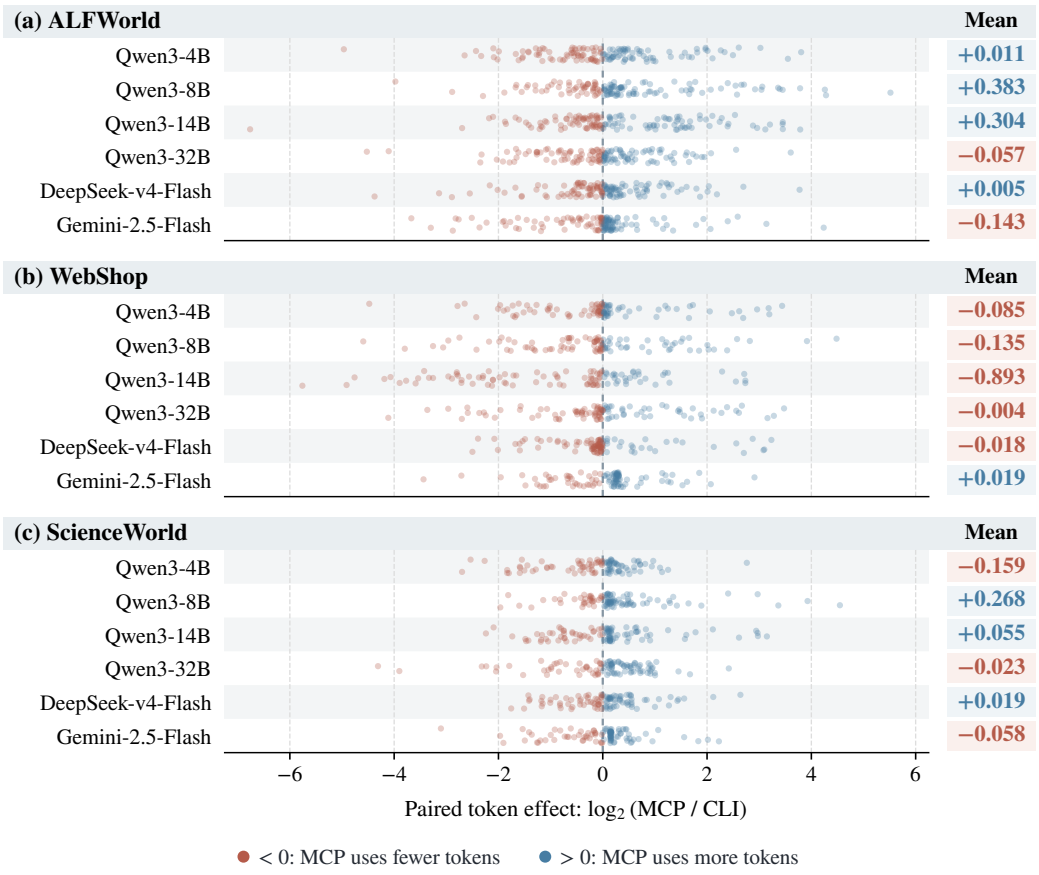



**Figure S1.** Paired task-level token effects for the CLI/MCP configurations in Tables 1 and S1. Each point reports $\log_2(T_{\mathrm{MCP}}/T_{\mathrm{CLI}})$ for one task after within-task replicate averaging. Negative values indicate lower token use under MCP and positive values indicate higher token use. The right column reports the model-level mean within each environment.

**Interaction cost is shaped jointly by call frequency and token use per call.** Table S1 decomposes total token use into the number of model calls and the tokens consumed per call, revealing distinct mechanisms behind the endpoint cost shifts. On ALFWorld, Qwen3-14B under MCP increases from 46.9 to 57.1 calls per task while tokens per call remain nearly unchanged (4.46k versus 4.50k), showing that its higher cost is driven primarily by a longer interaction trajectory. On WebShop, the same model shows the opposite pattern. MCP reduces both calls per task (14.2 → 9.6) and tokens per call (4.26k → 3.23k), producing a substantially lighter interaction profile. These results show

that interface representation changes not only how much computation an agent consumes, but also how that computation is distributed across the trajectory.

**Table S1.** Secondary endpoint and resource statistics for the CLI/MCP settings in Table 1. We report task success, model calls per task, and tokens per call. Parenthesized values give the MCP-minus-CLI difference for each metric.

| Model | Success (%) | | Calls / task | | Tokens / call (k) | |
|---|---|---|---|---|---|---|
| | CLI | MCP (Δ) | CLI | MCP (Δ) | CLI | MCP (Δ) |
| **ALFWorld** | | | | | | |
| Qwen3-4B | 44.8 | 41.8 (−3.0) | 53.1 | 51.5 (−1.6) | 5.96 | 6.23 (+0.27) |
| Qwen3-8B | 44.8 | 35.1 (−9.7) | 47.3 | 53.2 (+5.9) | 5.34 | 5.89 (+0.56) |
| Qwen3-14B | 56.7 | 20.1 (−36.6) | 46.9 | 57.1 (+10.2) | 4.46 | 4.50 (+0.04) |
| Qwen3-32B | 73.9 | 78.4 (+4.5) | 27.8 | 26.6 (−1.2) | 3.54 | 3.50 (−0.04) |
| DeepSeek-v4-Flash | 99.3 | 95.5 (−3.7) | 15.2 | 15.3 (+0.1) | 3.10 | 3.08 (−0.02) |
| Gemini-2.5-Flash | 56.7 | 67.2 (+10.4) | 30.2 | 27.1 (−3.1) | 3.48 | 3.55 (+0.07) |
| **WebShop** | | | | | | |
| Qwen3-4B | 17.0 | 22.0 (+5.0) | 8.6 | 7.9 (−0.7) | 3.49 | 3.48 (−0.01) |
| Qwen3-8B | 23.0 | 21.0 (−2.0) | 14.4 | 12.7 (−1.7) | 4.48 | 4.48 (−0.01) |
| Qwen3-14B | 22.0 | 35.0 (+13.0) | 14.2 | 9.6 (−4.6) | 4.26 | 3.23 (−1.03) |
| Qwen3-32B | 35.0 | 39.0 (+4.0) | 13.8 | 14.0 (+0.2) | 4.33 | 4.53 (+0.19) |
| DeepSeek-v4-Flash | 79.0 | 82.0 (+3.0) | 11.2 | 10.9 (−0.3) | 4.13 | 4.10 (−0.03) |
| Gemini-2.5-Flash | 77.0 | 74.0 (−3.0) | 9.1 | 8.4 (−0.7) | 3.57 | 3.92 (+0.35) |
| **ScienceWorld** | | | | | | |
| Qwen3-4B | 26.0 | 24.0 (−2.0) | 17.0 | 16.3 (−0.8) | 6.00 | 6.21 (+0.21) |
| Qwen3-8B | 35.0 | 37.0 (+2.0) | 14.0 | 15.8 (+1.9) | 5.48 | 6.01 (+0.53) |
| Qwen3-14B | 28.0 | 24.0 (−4.0) | 14.0 | 13.8 (−0.2) | 4.78 | 5.06 (+0.28) |
| Qwen3-32B | 46.0 | 47.0 (+1.0) | 14.7 | 13.9 (−0.8) | 5.31 | 5.76 (+0.44) |
| DeepSeek-v4-Flash | 64.0 | 66.0 (+2.0) | 13.6 | 12.7 (−0.8) | 6.51 | 6.74 (+0.23) |
| Gemini-2.5-Flash | 42.0 | 48.0 (+6.0) | 10.5 | 9.1 (−1.4) | 5.87 | 6.31 (+0.44) |

## A.2 RQ2: Semantic Decisions, Failures, and Recovery

**Interface representation changes semantic action selection beyond surface decoding.** Figure S2 compares operation frequencies among responses that decode into typed proposals, isolating semantic choice from surface-format failure. The resulting shifts are structured but model- and environment-dependent. On WebShop, for example, Qwen3-14B moves by +6 percentage points toward `click` and −6 points from `search` under MCP. ALFWorld and ScienceWorld show broader redistributions across navigation, inspection, manipulation, and other operations.

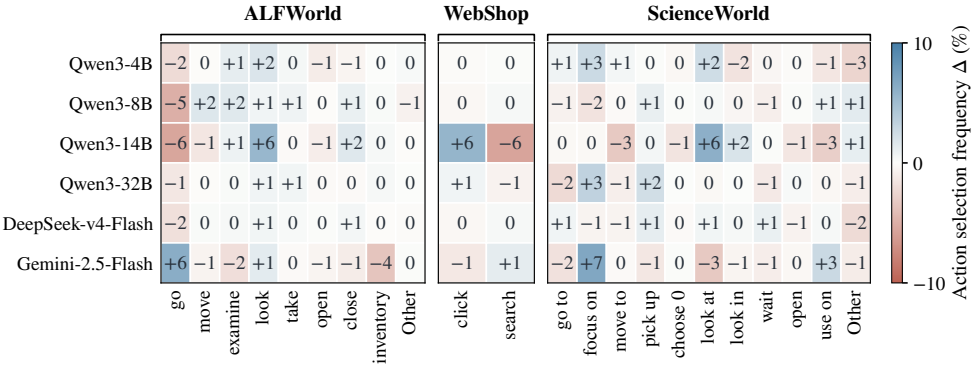



**Figure S2.** MCP-minus-CLI changes in semantic action-selection frequency (%), conditional on successful decoding. Models are shown by row and operations by environment. The Other category aggregates the remaining operations.

**Interfaces reshape both failure frequency and failure pathway.** Table S2 complements Figure 2 by measuring how often attempts fail, while the main-text figure resolves where those failures occur in the action pipeline. WebShop provides a clear example. MCP has lower attempt-level failure rates for all six models, with absolute reductions ranging from 1.1 to 8.9 percentage points. At the

same time, the composition of the remaining failures differs markedly across models, shifting among decoding, availability, execution, generation, and retry stages.

**Table S2.** Attempt-level failure rates (%) under CLI and MCP, including retry attempts. Δ reports the MCP-minus-CLI difference.

| Model | ALFWorld | | | WebShop | | | ScienceWorld | | |
|---|---|---|---|---|---|---|---|---|---|
| | CLI | MCP | Δ | CLI | MCP | Δ | CLI | MCP | Δ |
| Qwen3-4B | 31.4 | 29.6 | −1.9 | 20.8 | 12.1 | −8.8 | 22.3 | 23.8 | +1.5 |
| Qwen3-8B | 20.3 | 23.2 | +2.9 | 31.2 | 22.4 | −8.9 | 22.2 | 20.2 | −2.0 |
| Qwen3-14B | 22.6 | 22.6 | 0.0 | 12.0 | 10.8 | −1.1 | 20.9 | 20.7 | −0.2 |
| Qwen3-32B | 5.7 | 4.6 | −1.2 | 14.2 | 10.9 | −3.2 | 6.4 | 7.7 | +1.4 |
| DeepSeek-v4-Flash | 3.7 | 2.1 | −1.6 | 3.5 | 1.3 | −2.2 | 7.4 | 6.3 | −1.1 |
| Gemini-2.5-Flash | 1.5 | 2.1 | +0.6 | 1.5 | 0.4 | −1.2 | 10.1 | 9.1 | −1.1 |

**Recovery dynamics separate immediate correction from downstream task progress.** Figures S3 and S4 trace what happens after an eligible failure. The next response can preserve the decision and repair its realization, change the semantic action, repeat the error, or move into a different failure mode. On WebShop/Qwen3-4B, for example, retry acceptance is 50% under CLI and 77% under MCP, while later task success is 18% and 34%, respectively. Across settings, the two rates move on different scales, showing that recovery unfolds in stages, first restoring an admissible action, then converting that local repair into useful trajectory progress.

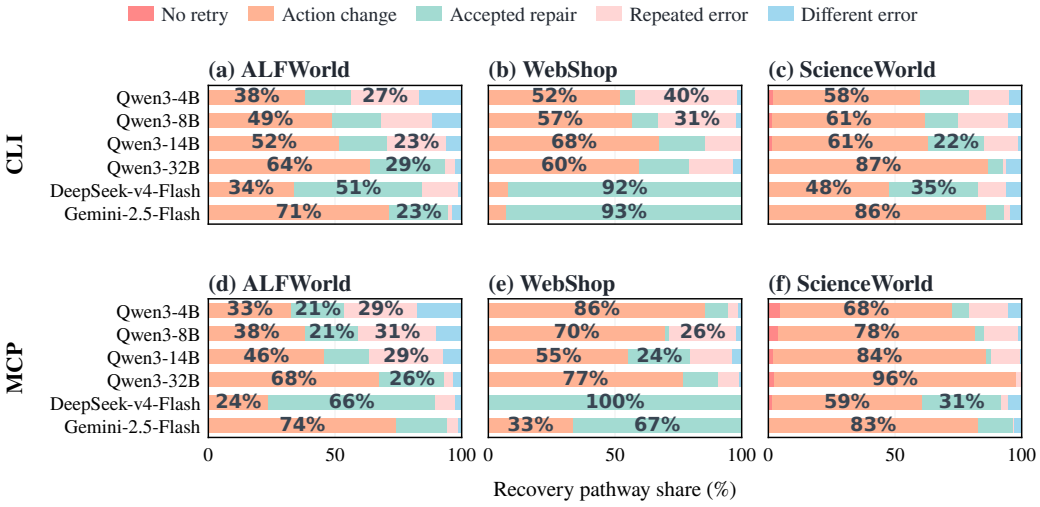



**Figure S3.** Recovery pathways following eligible failures under CLI and MCP. Bars report the share of failures followed by no retry, semantic action change, accepted repair, repeated error, or a different error.

### A.3 RQ3: Local Origins and Downstream Propagation

**Interface components interact strongly at matched decision states.** The main-text factorial analysis summarizes the component and interaction effects, while Figure S5 exposes the eight matched-state response profiles from which those contrasts arise. Realization is especially sensitive to its surrounding interface context. For Qwen3-32B on ALFWorld, switching only the realization protocol from MCP to CLI changes realization failure from 1% to 87% under MCP presentation and from 0% to 89% under CLI presentation. Across models and environments, admission and semantic-action entropy shift alongside realization failure, showing that presentation, realization, and feedback jointly shape both whether an action is successfully expressed and how decoded semantic actions are distributed. The behavioral consequence of one interface component depends on the configuration around it.

**Consequential divergence is widespread and emerges early.** Across the 2,004 paired MCP–CLI trajectories, 1,889 (94.3%) exhibit an observed structured divergence. Figure S7 further shows that the first divergence appears near the beginning of interaction across model–environment strata, with semantic-action differences accounting for most divergent pairs and realization differences forming a smaller but recurrent component.

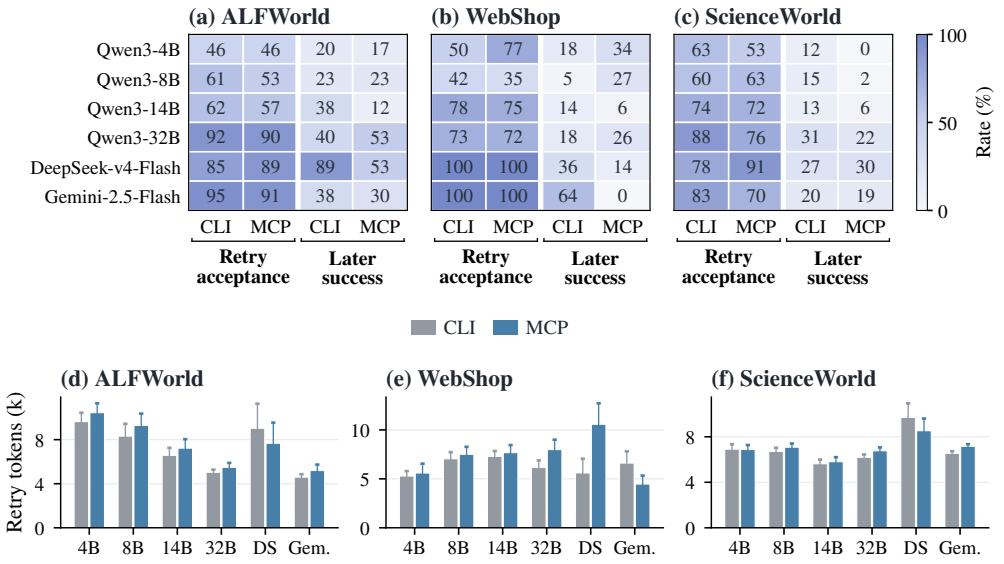



**Figure S4.** Immediate retry acceptance, later task success, and mean retry token cost following eligible failures. Whiskers denote 95% task-cluster bootstrap intervals.

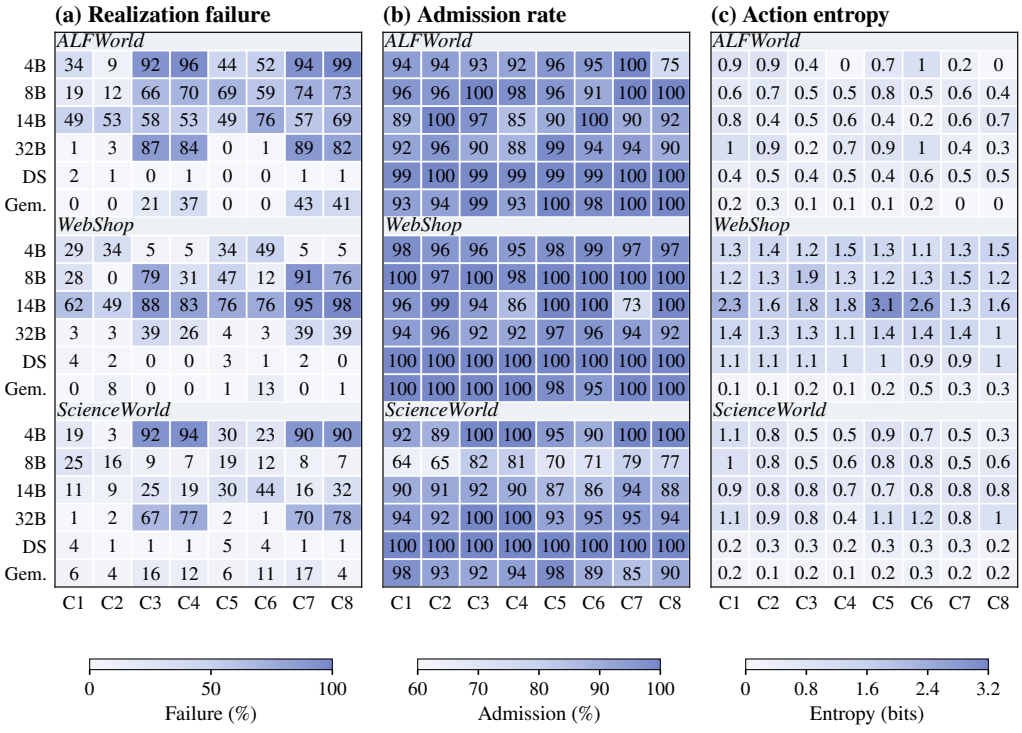



**Figure S5.** Matched-state response profiles for the eight combinations of presentation $\Phi$, realization $R$, and feedback $\Psi$. Panels report realization failure, admission after decoding, and semantic-action entropy. C1–C8 correspond to MMM, MMC, MCM, MCC, CMM, CMC, CCM, and CCC, where M/C denotes MCP/CLI.

**Early divergence reshapes the remainder of the interaction trajectory.** Figure S6 tracks the natural suffix after the first consequential difference. Once the paired runs separate, they accumulate different token costs and attempt counts and can reach different terminal outcomes. The magnitude and direction of these changes vary across models and environments. Figure S7(c) adds a complementary view. Semantic actions can align again later, but the extent of that later agreement differs substantially across strata. Together, these results show how a local interface difference changes the history conditioning subsequent decisions, producing persistent, path-dependent differences in interaction cost and task outcome.

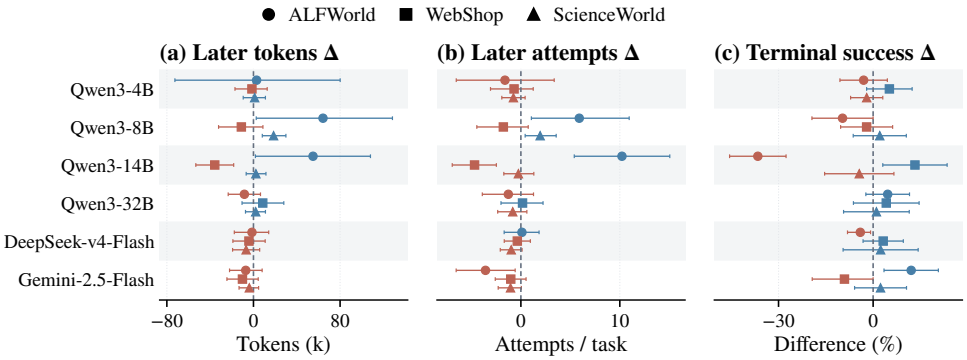



**Figure S6.** Post-divergence MCP-minus-CLI differences in charged tokens, interaction attempts, and terminal success. Shapes identify environments and whiskers denote 95% paired-task bootstrap intervals.

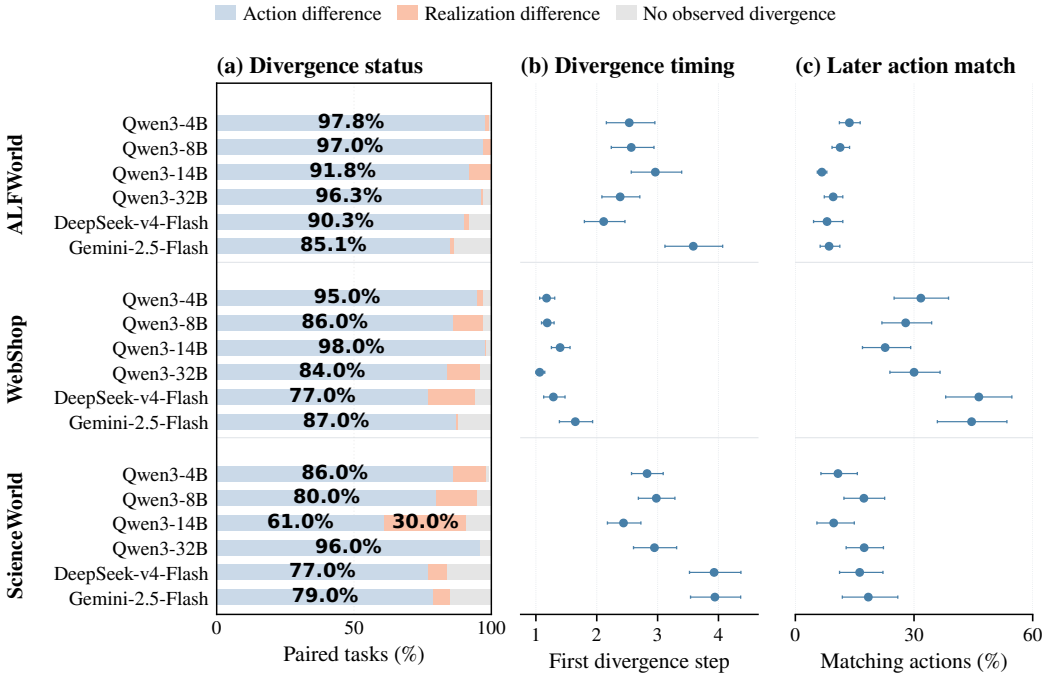



**Figure S7.** Structure of natural MCP–CLI divergence across 2,004 paired trajectories. Panels show (a) divergence type, (b) mean first-divergence step, and (c) later semantic-action agreement. Whiskers denote 95% task-cluster bootstrap intervals.

**Paired trajectories make the propagation mechanism concrete.** Example S1 shows a WebShop CLI quoting failure that forces a retry and redirects subsequent interaction, while the paired MCP trajectory proceeds to task success. Example S2 shows the complementary pattern. An earlier ScienceWorld divergence is followed by the same later `choose-0` semantic action, and both trajectories ultimately succeed. Together, the examples illustrate two complementary propagation patterns—divergence that persists into the endpoint and divergence that is partially absorbed by later interaction.

**Example S1. WebShop — a decode failure and retry**

| | |
|---|---|
| **Task** | Find me machine wash men's dress shirts with cotton spandex, classic fit, short sleeve with color: delft, and size: x-large, and price lower than 70.00 dollars |
| **MCP search** | `<tool_call>`<br>`{"name": "webshop__search",`<br>`"arguments": {"query": "machine wash men's dress shirt cotton spandex classic fit short sleeve delft x-large"}}`<br>`</tool_call>`<br>**Observation:**<br>[...]<br>Page 1 (Total results: 50)<br>[...] |
| **CLI failure** | `<command>`<br>`bench webshop search --query 'men's dress shirts cotton spandex classic fit short sleeve delft x-large'`<br>`</command>` |
| **CLI feedback** | CLI validation error [surface.invalid_argv] at surface_decode. Constraint: surface_response_contract. invalid argv quoting: No closing quotation State transition: unchanged. Retryable: true. Allowed values: search, click. Correction: Current semantic operations: search, click.; Use exactly one current operation and all required arguments.; Return exactly one <command>argv command</command> and no other text; each flag must have one value, and quote every multiword string value with POSIX single quotes. |
| **CLI retry** | `<command>`<br>`bench webshop search --query 'mens dress shirts cotton spandex classic fit short sleeve delft x-large'`<br>`</command>`<br>**Observation:**<br>[...]<br>Page 1 (Total results: 50)<br>[...] |
| **MCP next action** | `<tool_call>`<br>`{"name": "webshop__click",`<br>`"arguments": {"target": "b085wqkrrj"}}`<br>`</tool_call>`<br>**Observation:**<br>[...]<br>CLANMILUMS Men's Classic Comfort Soft Regular Fit Short Sleeve Henley T-Shirt Tee<br>Price: $17.99 to $19.99<br>[...] |
| **Outcome** | **MCP:** task success.<br>**CLI:** no task success; the environment-step limit is reached. |

**Example S2. ScienceWorld — later action agreement**

| | |
|---|---|
| **Task** | Your task is to grow a orange plant from seed. Seeds can be found in the greenhouse. First, focus on a seed. Then, make changes to the environment that grow the plant until it reaches the reproduction life stage. |
| **Shared prefix** | **MCP:**<br>`<tool_call>`<br>`{"name": "scienceworld__go_to", "arguments": {"destination": "greenhouse"}}`<br>`</tool_call>`<br>**CLI:**<br>`<command>`<br>`bench scienceworld go-to --destination 'greenhouse'`<br>`</command>`<br>**Observation:**<br>You move to the greenhouse. |
| **MCP target** | `<tool_call>`<br>`{"name": "scienceworld__focus_on", "arguments": {"target": "orange seed in the seed stage"}}`<br>`</tool_call>` |
| **CLI target** | `<command>`<br>`bench scienceworld focus-on --target 'orange seed in seed jar containing orange seed'`<br>`</command>` |
| **Environment** | Ambiguous request: Please enter the number for the action you intended (or blank to cancel):<br>0: focus on orange seed (in seed jar, in greenhouse)<br>1: focus on orange seed (in seed jar, in greenhouse)<br>2: focus on orange seed (in seed jar, in greenhouse)<br>3: focus on orange seed (in seed jar, in greenhouse)<br>4: focus on orange seed (in seed jar, in greenhouse) |
| **Later action** | **MCP:**<br>`<tool_call>`<br>`{"name": "scienceworld__choose_0", "arguments": {}}`<br>`</tool_call>`<br>**CLI:**<br>`<command>`<br>`bench scienceworld choose-0`<br>`</command>` |
| **Environment** | You focus on the orange seed. |
| **Outcome** | Both trajectories eventually succeed. |

### A.4 RQ4: Commitment-Preserving Realization with DART

**DART concentrates recovery at the realization boundary while preserving the validated semantic commitment.** Across the six benchmark–interface settings, 71–100% of triggered interventions are admitted on the first realization. Among the four interventions that proceed to a preserving-correction opportunity, three are admitted after that single correction. These results show that DART resolves triggered failures primarily by repairing the interface-specific expression while keeping the committed operation and typed arguments fixed.

**Recorded trajectories instantiate both successful DART paths.** Example S3 shows direct realization. After an ALFWorld/MCP failure, DART commits to `alfworld.move`, re-expresses the same typed action in a valid tool call, and executes it successfully. Example S4 shows preserving correction. The first WebShop/CLI realization loses the literal apostrophes in `18''x18''`, while the

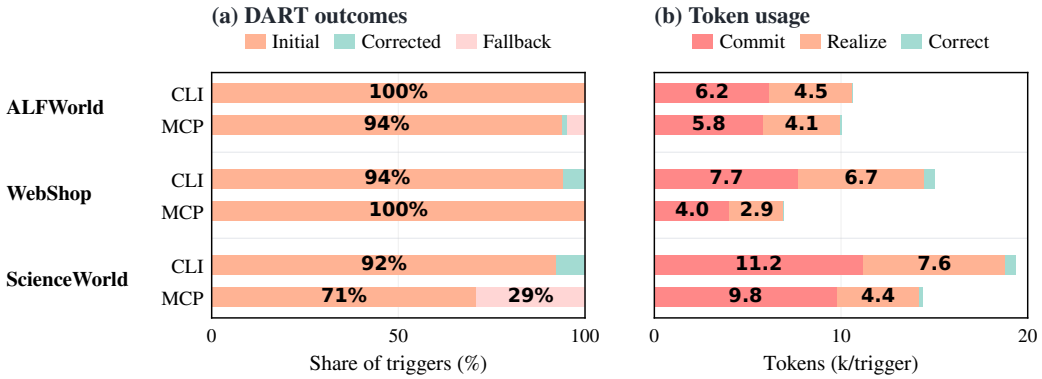



**Figure S8.** DART trigger outcomes and token use across the six benchmark–interface settings evaluated in Figure 4. (a) Share of triggers admitted on the initial realization, after preserving correction, or released to standard retry. (b) Mean tokens per trigger spent on semantic commitment, realization, and correction.

correction changes only the command expression, preserves `webshop.click` and the full target value, and is admitted before the agent completes the purchase. These trajectories provide concrete executions of the two success branches summarized in Figure S8.

**Example S3. ALFWorld / MCP — realizing a validated action**

| | |
|---|---|
| **Task** | cool some mug and put it in coffeemachine. |
| **Recent context** | **Executed:** `cool mug 1 with fridge 1`<br>**Observation:** You cool the mug 1 using the fridge 1.<br>**Executed:** `go to coffeemachine 1`<br>**Observation:** You arrive at coffeemachine 1. On the coffeemachine 1, you see nothing. |
| **Failed output** | `<tool_call>`<br>`{"name": "alfworld__move",`<br>`"destination": "coffeemachine 1",`<br>`"object": "mug 1"}`<br>`</tool_call>` |
| **Feedback** | tool call must contain exactly name and arguments<br>The proposal is rejected before environment execution. |
| **Commitment** | `{"schema_id": "alfworld.move",`<br>`"arguments": {"object": "mug 1",`<br>`"destination": "coffeemachine 1"}}` |
| **Realization** | `<tool_call>`<br>`{"name": "alfworld__move",`<br>`"arguments": {"destination": "coffeemachine 1",`<br>`"object": "mug 1"}}`<br>`</tool_call>`<br>Admitted on the first DART realization; no correction call is needed. |
| **Execution** | **Executed:** `move mug 1 to coffeemachine 1`<br>**Observation:** You move the mug 1 to the coffeemachine 1.<br>**Recorded outcome:** task success, score 1.0. |

**Example S4. WebShop / CLI — preserving a quoted option value**

| | |
|---|---|
| **Task** | Find me super soft decorative pillows for living room with color: lavender, and size: 18"x18", and price lower than 50.00 dollars |
| **Current page** | You have clicked lavender.<br>[...]<br>size<br>[button] 18"x18" [button_]<br>[button] 20"x20" [button_]<br>[...]<br>Price: $13.99<br>[...]<br>[button] Buy Now [button_] |
| **Failed output** | `<command>`<br>`bench webshop click --target '18''x18'''`<br>`</command>` |
| **Feedback** | target is not uniquely available in the current state<br>The parsed target is `18x18`; its literal apostrophes have been lost. |
| **Commitment** | `{"schema_id": "webshop.click",`<br>`"arguments": {"target": "18''x18'''}}` |
| **First realization** | `<command>`<br>`bench webshop click --target '18''x18'''`<br>`</command>`<br>**Feedback:** target is not uniquely available in the current state<br>The target is still `18x18`, so this realization is not admitted. |
| **Correction** | `<command>`<br>`bench webshop click`<br>`--target '18'''''''''x18'''''''''`<br>`</command>`<br>Admitted: the parsed target now matches the committed `18''x18''`. |
| **Environment** | You have clicked 18"x18".<br>You have clicked lavender.<br>[...] |
| **Next action** | `<command>`<br>`bench webshop click --target 'buy now'`<br>`</command>`<br>**Observation:**<br>Thank you for shopping with us!<br>[...]<br>Your score (min 0.0, max 1.0)<br>1.0<br>[...] |

**DART localizes its additional computation to commitment and realization.** The token decomposition in Figure S8(b) shows that commitment formation is the larger component of DART token use in every evaluated stratum, followed by realization, while correction is invoked only for the small subset of triggers that require a second surface attempt. This targeted accounting complements the episode-level results in Figure 4. Under MCP, DART uses fewer total tokens in all three environments. DART improves benchmark score across every evaluated benchmark–interface setting.

# B Experimental Protocol and Statistical Analysis

This appendix specifies the controlled evaluation protocol underlying MICA.

## B.1 Evaluation Suite and Controlled Comparison

**Evaluation suite.** The baseline study evaluates six models—Qwen3-4B, Qwen3-8B, Qwen3-14B, Qwen3-32B (Yang et al., 2025), DeepSeek-v4-Flash, and Gemini-2.5-Flash—on three interactive environments. Each model is evaluated on 134 ALFWorld tasks, 100 WebShop tasks, and 100 ScienceWorld tasks, yielding 2,004 paired MCP–CLI episodes. ALFWorld is evaluated by task success. WebShop uses its native reward and ScienceWorld its native benchmark score, with the latter two reported on a 0–100 scale in the paper. Interaction cost is the total recorded input-plus-output model tokens consumed along a trajectory, including retry and intervention requests. Native benchmark interfaces are retained as calibration references for endpoint behavior.

**Controlled interface comparison.** For every paired task, MCP and CLI use the same model checkpoint, initial task state, public task information, semantic action contract, shared validation and execution path, generation configuration, and interaction budget. The controlled variation lies in capability presentation, action realization, and feedback representation.

## B.2 Generation, Context, and Interaction Budgets

**Generation settings.** Natural episode evaluation uses temperature-zero decoding, whereas matched-state probes use $T = 0.6$ to estimate local response distributions from repeated samples. All requests use top-$p = 1$, a 24,576-token input ceiling, and a 16,384-token output ceiling. The four Qwen3 models are evaluated from local checkpoints. DeepSeek-v4-Flash and Gemini-2.5-Flash use their corresponding API bindings under the same registered protocol.

**Context and episode budgets.** Interaction histories are retained as complete events and truncated oldest-first to fixed benchmark-specific limits of 100 events / 24,000 tokens for ALFWorld, 30 / 12,000 for WebShop, and 8 / 8,000 for ScienceWorld. The immediately preceding failed expression and its public feedback remain in context for a correction request. Episode-level environment-action / decision / model-call ceilings are 50 / 75 / 150 for ALFWorld, 30 / 45 / 90 for WebShop, and 32 / 32 / 32 for ScienceWorld. The same ceilings apply to MCP and CLI within each paired comparison.

Every dispatched model request contributes to interaction cost. A proposal rejected before execution counts as an action attempt but does not consume an environment transition. Under the Base policy, a decision receives at most one correction opportunity. When DART is activated, semantic selection, realization, and at most one preserving correction all operate within the remaining episode-level call budget.

## B.3 Paired Episode Evaluation Protocol

**Matched episode construction.** MCP and CLI episodes are paired by task identifier and initial state under the same checkpoint, public task information, generation configuration, registered randomness policy, and episode budget. Complete replicates are averaged within task before paired aggregation. An episode terminates on native task completion, an explicit terminal response, or the corresponding registered interaction budget.

**Common prefix and first divergence.** For each pair, we align the longest semantic prefix over which the normalized proposal, validation result, execution outcome, public feedback, and recovery record agree. Surface strings may differ within this prefix when they decode to the same semantic action and produce the same interaction outcome. The first step at which this interaction signature differs defines the first consequential divergence $t^\star$ used throughout the trajectory analysis.

**Natural propagation.** After $t^\star$, each episode continues under its own interaction history and environment state. We characterize the resulting suffix through subsequent semantic-action agreement, failure and recovery events, charged tokens, interaction attempts, and terminal outcome. Paired natural trajectories thus connect a common initial condition to the downstream effects summarized in Appendix A.3.

### B.4 Matched-State Probe Protocol

**Probe design.** The matched-state experiment contains 338 source states and 21,632 sampled responses across all six models and three environments. Each source state is evaluated under the eight $(\Phi, R, \Psi)$ assignments in Table S3, with eight independent draws per condition. Every draw restarts from the same registered source context, so repeated sampling estimates the local response distribution without allowing one draw to alter the state observed by the next.

**Matched decision contexts.** Source states and public histories are fixed before sampling using the registered category/hash selection rule. Within a source state, the environment state, policy-visible history, semantic contract, model, and generation configuration are held fixed while capability presentation $\Phi$, realization protocol $R$, and feedback representation $\Psi$ are varied. For ScienceWorld, the matched context is the recorded policy-visible source state and public interaction history supplied to the model.

**Factorial interface assignment.** Presentation specifies operation descriptions, argument types, and public constraints. Realization specifies the response container, operation binding, argument serialization, and decoder. Feedback specifies how normalized public results and failure facts are rendered. Mixed assignments use a canonical semantic ledger to render prior decoded actions and public results under the assigned components, preserving the semantic interaction history while changing the representation exposed to the model.

Table S3: **Factorial interface assignments used in the matched-state probes.**

| Condition | Presentation $\Phi$ | Realization $R$ | Feedback $\Psi$ |
|---|---|---|---|
| C1 | MCP | MCP | MCP |
| C2 | MCP | MCP | CLI |
| C3 | MCP | CLI | MCP |
| C4 | MCP | CLI | CLI |
| C5 | CLI | MCP | MCP |
| C6 | CLI | MCP | CLI |
| C7 | CLI | CLI | MCP |
| C8 | CLI | CLI | CLI |

C1 and C8 recover the homogeneous MCP and CLI interfaces, while C2–C7 expose component substitutions at the same decision context. These eight conditions form the response profiles underlying the component and interaction effects in Figure 3.

### B.5 Metrics and Statistical Analysis

#### B.5.1 Paired Episode Estimands

**Paired effect.** Tasks are the unit of paired episode comparison. For metric $m$, $N$ tasks in a model–environment stratum, and $K_i$ registered replicate pairs for task $i$, the MCP-minus-CLI effect is

$$\Delta_m = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{K_i}\sum_{r=1}^{K_i}\left(m_{ir}^{\mathrm{MCP}} - m_{ir}^{\mathrm{CLI}}\right). \tag{6}$$

Replicates are averaged within task before aggregation, so each task receives equal weight within a stratum. We report pointwise percentile 95% intervals from 10,000 paired task-cluster bootstrap resamples (Efron, 1979). RQ4 uses the same task-clustered procedure within each protocol-matched Base–DART comparison. Positive outcome contrasts denote larger MCP outcomes. Positive token, call, attempt, or step contrasts denote larger MCP resource use.

#### B.5.2 Failure and Recovery Metrics

**Failure metrics.** Failure incidence is measured over all action attempts, including correction attempts. Failure composition is measured over failed attempts and assigns each attempt to its earliest registered failure boundary in the action pipeline. Semantic-choice profiles are computed

from decoded normalized proposals, separating changes in action selection from failures of surface realization.

**Recovery metrics.** Recovery analysis begins from an eligible first-failure root, a retryable, policy-owned failure whose environment state remains unchanged. The immediate successor is classified by whether the model changes its semantic proposal, produces an admitted repair, repeats the same error, produces a different failure, or does not retry. Retry acceptance measures whether that successor is admitted, while later task success records the eventual episode outcome. Retry cost is the charged token use of the successor response. Bootstrap resampling keeps all failure roots from the same task together, preserving the task as the statistical resampling unit.

#### B.5.3 Matched-State Response Statistics

**Semantic response distribution.** At a matched decision context, each draw yields either a realization failure or a decoded semantic proposal with a shared-validation outcome. Let $n_t^I(a)$ denote the number of decoded draws proposing semantic action $a$, $n_{t,\mathrm{dec}}^I = \sum_a n_t^I(a)$ the number of decoded draws, and $n_t^I$ the total number of draws. The empirical semantic-action distribution and its entropy are

$$\widehat{p}_t^I(a \mid \mathrm{dec}) = \frac{n_t^I(a)}{n_{t,\mathrm{dec}}^I}, \qquad H_t^I = -\sum_a \widehat{p}_t^I(a \mid \mathrm{dec}) \log_2 \widehat{p}_t^I(a \mid \mathrm{dec}). \tag{7}$$

Realization failure is the complement of decoding coverage, while admission is the fraction of decoded proposals accepted by the shared semantic validator. Action entropy summarizes the dispersion of decoded semantic choices and is reported for conditions with at least two decoded draws.

**Probe aggregation.** Realization-failure and entropy summaries weight source tasks equally. Admission pools admitted and decoded draws within each condition. Bootstrap resamples source tasks jointly across C1–C8, preserving all factorial assignments from a source state in the same resample. Component main effects compare the MCP and CLI levels of one interface block while averaging over the assignments of the remaining blocks, matching the factorial contrasts reported in the main text.

## C Contract-Matched Interface Construction

This appendix details how MCP and CLI are constructed from a common semantic action contract. The construction separates the shared executable semantics from the model-facing representation. Both interfaces expose the same state-local operations and typed arguments, decode back to the same normalized action space, and enter the same validation, compilation, and execution path.

### C.1 Shared Semantic Boundary

**Semantic actions define the common unit of comparison.** A semantic action $a = (u, v)$ consists of an operation $u$ and its complete tuple of normalized typed arguments $v$. At environment state $s_t$, the public contract $\mathcal{C}_t$ specifies the operations currently exposed to the policy, their argument domains, and their native execution bindings. MCP and CLI render this same contract through different surface forms, but both decode model responses back to the contract's semantic action space before shared validation. Two surface responses are therefore treated as semantically equivalent at this boundary whenever they specify the same registered operation and equivalent normalized arguments.

**Interface components alter representation, not execution semantics.** Following the factorization in the main text, presentation $\Phi_I$ determines how available capabilities and constraints are shown to the model, realization $R_I = (\rho_I, D_I)$ determines the response grammar and decoder, and feedback $\Psi_I$ determines how public execution results and failure information are rendered. These components can change what the model sees and how it expresses an action, while the semantic validator, native compiler, and environment executor remain shared across interfaces.

### C.2 Surface Rendering and Conformance Checks

**Shared contract, distinct surfaces.** For every semantic operation, the MCP renderer produces a named tool with a structured argument object, whereas the CLI renderer produces a registered

command with explicit flags and argument values. The CLI decoder parses the registered argument vector directly. Quoted multiword values remain a single typed argument. Both renderers are generated from $\mathcal{C}_t$, so differences in naming, serialization, quoting, and response containers change the surface representation without introducing new executable capabilities.

**Round-trip semantic equivalence.** Conformance first checks that a registered semantic action survives rendering and decoding under either interface,

$$D_{\mathrm{MCP}}(\mathrm{Render}_{\mathrm{MCP}}(a)) \equiv_{\mathrm{sem}} a \equiv_{\mathrm{sem}} D_{\mathrm{CLI}}(\mathrm{Render}_{\mathrm{CLI}}(a)) \,. \tag{8}$$

Semantic equivalence requires agreement on the operation and every normalized typed argument. Surface-level differences such as whitespace, JSON field order, or valid CLI quoting therefore leave the decoded action unchanged.

**Common native compilation.** A decoded proposal that passes the shared semantic validator is compiled by the same benchmark-specific compiler $K_{\mathcal{C}_t}$. For semantically matched MCP and CLI responses,

$$K_{\mathcal{C}_t}(D_{\mathrm{MCP}}(y_{\mathrm{MCP}})) = K_{\mathcal{C}_t}(a) = K_{\mathcal{C}_t}(D_{\mathrm{CLI}}(y_{\mathrm{CLI}})) \,. \tag{9}$$

Conformance checks combine four tests, namely coverage of the registered semantic actions, round-trip decoding, common native compilation, and consistency of the policy-visible contract facts. Finite state-local action catalogs are checked over their registered members. Open string arguments are checked over the declared benchmark domain and schema-valid test cases.

### C.3 Benchmark-Specific Contract Realizations

**ALFWorld.** For ALFWorld (Shridhar et al., 2021), the state-local contract is built from the currently admissible native action instances. Operations such as navigation, pickup, placement, opening, closing, heating, cooling, and cleaning retain their complete object and destination names as typed arguments. MCP and CLI therefore expose different realizations of the same currently admissible action instances.

**WebShop.** WebShop (Yao et al., 2022) is represented by two semantic operations, namely `search(query)` and `click(target)`. The search argument is an open string satisfying the shared schema, whereas click targets are drawn from the controls currently visible to the agent. Product options, pagination controls, and purchase actions are represented as values of `click(target)` rather than as additional interface-specific operations.

**ScienceWorld.** ScienceWorld (Wang et al., 2022) lifts the benchmark's official action templates into typed semantic operations and argument tuples. Public object referents, aliases, and tuple structure determine the state-local arguments available to the policy. MCP and CLI render these typed operations differently but compile admitted proposals back to the same native ScienceWorld action.

The following compact mappings illustrate the construction used across the three environments. In each block, the MCP and CLI surfaces decode to the displayed semantic action and compile to the same native command.

**ALFWorld**

| Layer | Representation |
|---|---|
| Semantic action | `alfworld.go(destination="desk 1")` |
| MCP surface | `<tool_call>`<br>`{"name":"alfworld_go",`<br>`  "arguments":{"destination":"desk 1"}}`<br>`</tool_call>` |
| CLI surface | `<command>`<br>`bench alfworld go --destination 'desk 1'`<br>`</command>` |
| Native action | `go to desk 1` |

**WebShop**

| Layer | Representation |
|---|---|
| Semantic action | `webshop.click(target="next >")` |
| MCP surface | `<tool_call>`<br>`{"name":"webshop__click",`<br>`"arguments":{"target":"next >"}}`<br>`</tool_call>` |
| CLI surface | `<command>`<br>`bench webshop click --target 'next >'`<br>`</command>` |
| Native action | `click[next >]` |

**ScienceWorld**

| Layer | Representation |
|---|---|
| Semantic action | `scienceworld.t2.go_to(destination="hallway")` |
| MCP surface | `<tool_call>`<br>`{"name":"scienceworld__go_to",`<br>`"arguments":{"destination":"hallway"}}`<br>`</tool_call>` |
| CLI surface | `<command>`<br>`bench scienceworld go-to --destination hallway`<br>`</command>` |
| Native action | `go to hallway` |

### C.4 Worked Round-Trip Example

**A single semantic action admits two surface realizations.** Consider an ALFWorld state whose contract includes `alfworld.go(destination="sofa 1")`. The two interfaces present and realize this action differently, but both responses round-trip to the same typed proposal and therefore enter the same shared validation and execution path.

**Example S5. Round-trip equivalence for an ALFWorld action**

| | |
|---|---|
| **Semantic action** | `alfworld.go(destination="sofa 1")` |
| **MCP realization** | `<tool_call>`<br>`{"name":"alfworld__go",`<br>`"arguments":{"destination":"sofa 1"}}`<br>`</tool_call>` |
| **CLI realization** | `<command>`<br>`bench alfworld go --destination 'sofa 1'`<br>`</command>` |
| **Decoded action** | Both decoders return `alfworld.go(destination="sofa 1")`. |
| **Compilation** | Both proposals compile to `go to sofa 1`. |

The two surface forms decode to the same semantic action and compile to the same native command. Appendix B.4 uses this shared boundary to exchange interface components at matched decision contexts, while Appendix D.4 uses the same semantic identity criterion to preserve validated actions during targeted realization repair.

## D DART: Controller, Triggers, and Evaluation Protocol

This appendix specifies the controller used in RQ4. DART separates semantic selection from interface-specific realization, binds a validated semantic action as an explicit commitment, and restricts subsequent repair to surface realization while that commitment remains valid.

### D.1 Commitment-Preserving Realization

**Commitment formation.** After an eligible failure, DART asks the same underlying model to select a canonical semantic action under the current public contract. A decoded selection that passes the shared semantic validator establishes $c_t = (\kappa_t, \bar{a}_t)$, where $\kappa_t$ identifies the contract and $\bar{a}_t = (\bar{u}_t, \bar{v}_t)$ contains the normalized operation and all typed arguments. This explicit commitment becomes the semantic target for the interface-specific realization stage.

**Commitment-preserving realization.** A realization is admitted only when the active contract is unchanged, the surface response decodes to the committed semantic action, and that action remains valid in the current state,

$$\kappa_{\text{current}} = \kappa_t, \qquad D_I(y_t^I) \equiv_{\kappa_t} \bar{a}_t, \qquad V(s_t, \bar{a}_t) = 1. \tag{10}$$

The semantic-equivalence test requires the same operation and every normalized typed argument. Thus, DART permits the interface expression to change while holding the selected action fixed.

**Repair boundary.** If the initial realization fails and the commitment remains valid, DART issues one preserving-correction request containing the validated commitment, the failed surface expression, and its public failure feedback. The correction may change quoting, serialization, container structure, or other interface-specific expression details, but not the committed operation or arguments. If the contract changes, the commitment becomes invalid, or the repair budget is exhausted, control returns to the ordinary retry policy.

### D.2 Error-Triggered Controller

**Controller flow.** Algorithm S1 gives the commitment-preserving branch used after a registered trigger. Semantic selection and initial realization require two model calls. A preserving correction is attempted only when a third episode-level call slot remains available. The $\mathrm{Invoke}$ routine enforces the active episode-level call budget and per-request limits before every dispatch.

**Algorithm S1** Error-triggered commitment-preserving repair

**Input:** Public retry context $x$; failure $f$; contract $\mathcal{C}$; interface $I$; remaining call budget $B_{\text{call}}$.
**Output:** An admitted proposal, or $\mathrm{Release}$ to the ordinary retry policy.

**if** $\neg\,\mathrm{Eligible}(f)$ or $B_{\text{call}} < 2$ **then return** $\mathrm{Release}$
$z \leftarrow \mathrm{Invoke}(\text{selection}, x, f, \mathcal{C})$
**if** $z = \bot$ **then return** $\mathrm{Release}$
$\bar{a} \leftarrow \mathrm{DecodeSelection}(z, \mathcal{C})$
**if** $\bar{a} = \bot$ or $\neg V_{\mathcal{C}}(\bar{a})$ **then return** $\mathrm{Release}$
$c \leftarrow (\kappa(\mathcal{C}), \bar{a})$
$y \leftarrow \mathrm{Invoke}(\text{realization}, x, c, I)$
**if** $y \neq \bot$ and $\mathrm{Preserves}(y, c, \mathcal{C}, I)$ **then return** $D_I(y)$
**if** $y = \bot$ or $\neg\,\mathrm{Valid}(c, \mathcal{C})$ **then return** $\mathrm{Release}$
$f' \leftarrow \mathrm{PublicFailure}(y, D_I(y), c)$
$y' \leftarrow \mathrm{Invoke}(\text{correction}, x, c, y, f', I)$
**if** $y' \neq \bot$ and $\mathrm{Preserves}(y', c, \mathcal{C}, I)$ **then return** $D_I(y')$
**return** $\mathrm{Release}$

Here $\mathrm{Preserves}(y, c, \mathcal{C}, I)$ implements the gate in Equation 10. The decoded realization must match the committed operation and every normalized typed argument and remain valid under the active contract. Native execution follows the shared runtime, and $\mathrm{Release}$ returns control to the ordinary retry policy.

### D.3 Trigger Scope and Benchmark Instantiation

**Trigger scope.** Across all benchmark–interface settings, DART is activated only for retryable, policy-owned failures under a confirmed unchanged environment state. Eligible triggers capture cases in which the current action cannot be successfully executed or does not produce the intended

progress, including interface-level decoding or validation failures and backend execution rejections. This scope ensures that DART intervenes only when the current commitment remains semantically valid but requires a corrected realization. The exact benchmark-specific trigger definitions and phase instructions are provided in Appendix E.

**Benchmark instantiation.** All RQ4 settings use the same commitment-preserving controller and semantic preservation criterion. DART operates exclusively on policy-visible task information and public retry context, with concise task or failure reminders included when required by the benchmark. After a commitment is selected and validated, it is realized through the interface contract assigned to the corresponding setting. Benchmark-specific prompt realizations may differ in presentation, but they follow the same selection, validation, and correction procedure described above.

### D.4 RQ4 Paired Evaluation

**Protocol-matched comparison.** RQ4 evaluates Gemini-2.5-Flash in all six benchmark–interface settings. Each DART condition is paired with a Base condition on the same task set. Within a pair, the model, interface renderer, public task information, ordinary policy, generation settings, semantic validator, execution pipeline, and episode-level budgets are shared. DART adds only the error-triggered commitment-preserving controller and the registered trigger/selection configuration in Appendix D.3. Endpoint score and token cost are aggregated with the paired task-level procedure defined in Appendix B.5.

**ScienceWorld protocol.** Both ScienceWorld DART and Base runs use the same stable domain-order-v2 public-input protocol. The RQ4 comparison is therefore entirely within this shared protocol, with DART as the controlled intervention.

**Cost accounting.** All RQ4 runs use temperature zero, top-$p = 1$, and the request limits specified in Appendix B.2. Endpoint token cost includes every dispatched ordinary, semantic-selection, realization, preserving-correction, and fallback response. These requests consume the same episode-level model-call budget used by the protocol-matched Base.

## E Prompt Templates and Response Contracts

This appendix specifies the response contracts used by the ordinary policy and the phase-specific DART requests. Ordinary interaction follows a shared policy instruction together with the currently rendered interface contract. When DART activates, the same model first selects a canonical semantic action, then realizes the validated commitment through the assigned interface, and, when needed, performs one commitment-preserving correction. The prompts below map directly to the controller stages defined in Appendix D.

### E.1 Prompt Assembly and Ordinary Policy

Ordinary policy requests use the same message structure across MCP and CLI. The shared system instruction in Prompt S1 is combined with the task instruction, current observation, rendered interface contract, visible interaction history, and final action request, in that order. The system-level policy and message ordering remain fixed. Interface-specific information enters through the rendered contract, required response surface, and visible interaction context. Outside DART trigger events, this ordinary policy governs the trajectory unchanged.

**Prompt S1. Shared ordinary-policy instruction**

*You are the only policy acting in an interactive benchmark. Complete the task using only the task instruction, current observation, visible interaction history, and the current interface contract.*

*The interface contract in the user message is authoritative. Select only the single best next action for the current state; do not wait for future actions to appear, because the contract is regenerated after every accepted environment transition. Keep any internal deliberation concise and prioritize returning a complete action before the output limit. Produce exactly one response container in the requested Native, controlled CLI-style, or controlled MCP-style syntax. Do not add explanations, Markdown, aliases, extra keys, guessed parameters, or text outside that container.*

*An accepted action can still make no progress. If the visible response is unchanged and a controller notice identifies an immediately repeated accepted action, choose a different currently grounded action; do not repeat that action again.*

*Treat different complete entity names as distinct. Never substitute a shorter, longer, or merely overlapping visible name for an entity explicitly named by the task. If the exact entity is not currently visible, choose a grounded action from the current contract to navigate, inspect, or otherwise gather visible state.*

*If the previous proposal was rejected, use the visible validation feedback to make the single allowed correction attempt. Never assume that a rejected pre-execution proposal changed the environment.*

### E.2 Semantic Selection Contract

When DART activates, semantic selection replaces the interface-specific output rule with the canonical selection contract in Prompt S2, while preserving the same policy-visible task and retry context. Prompt S3 then requests one canonical semantic action and is followed by the canonical output schema, triggering response, and public failure feedback. The model must return exactly one JSON object containing `schema_id` and `arguments`. This object is decoded and validated against the current semantic contract, and the commitment is created only after successful validation. Interface-specific realization begins only after this semantic commitment has been established. The same public-schema prompting and local validation path is used for both local and API-backed models, without guided decoding.

**Prompt S2. Selection-only system output rule**

*This call is the semantic recommitment phase, not the interface-realization phase. Return exactly one JSON object with exactly schema_id and arguments, conforming to the Canonical output schema in the user message. Do not use a Native, CLI, or MCP response container, command line, tool call, Markdown, or explanatory text for this call. This phase-specific output rule supersedes surface-format instructions and examples in the current interface contract and historical failure feedback. The current public semantic contract remains authoritative for valid operations and grounded argument values. Select one valid semantic action only; do not execute or realize it. The separate realization call will use the assigned interface syntax.*

**Prompt S3. Explicit semantic recommitment**

*For this retryable realization failure only, make an explicit semantic recommitment for the current task and unchanged environment state. The failed text does not establish an original latent intent. Choose one currently valid action using the same public capabilities above. For this call only, replace the interface output format by a JSON object with exactly* `schema_id` *and* `arguments`*, obeying the supplied canonical JSON schema. Do not execute or realize the action yet.*

### E.3 Realization and Preserving Correction

Once semantic selection has been validated, DART stores the semantic action together with its contract identity as the commitment. The initial realization request supplies this fixed commitment together with the assigned interface response contract and asks the model to express exactly that action through the required surface form. If realization fails while the commitment remains valid, the preserving-correction request additionally supplies the failed realization and its public failure feedback while leaving the commitment unchanged. Prompt S4 gives the exact instructions for both branches. Appendix A.4 provides recorded examples of successful first-pass realization and preserving correction.

**Prompt S4. Realization and preserving-correction instructions**

***Initial realization.*** *Realize the validated semantic commitment below using the assigned interface response contract above. Preserve the operation and every typed argument exactly. Do not select a different action. Return only the required interface expression.*

***Preserving correction.*** *Correct the failed realization using the assigned interface response contract above. Preserve the committed operation and every typed argument exactly. Correct only its interface expression. Return only the required interface expression.*