MCPDISCO: SEESAW-MITIGATED TUNING FOR MCP-NATIVE AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

The Model Context Protocol (MCP) is rapidly emerging as the substrate of the agentic web: a unified client-server standard that decouples LLM agents from an open-ended, ever-growing space of external tools. As agents become MCP-native, they face a decision-making regime that is qualitatively different from ad hoc function calling—one that requires jointly reasoning over a vast tool ecosystem and producing syntactically precise tool invocations. In this paper, we present MCPDisCo, the first work to perform reinforcement learning (RL) tuning of MCP-native agents at ecosystem scale, spanning 70 heterogeneous MCP servers and 527 tools. We formalize the agent as a Thought-Markov Decision Process (Thought-MDP) that makes internal reasoning a first-class citizen alongside external actions, and we identify a *reasoning-tool seesaw effect*: under jointly optimized rewards, gains on reasoning tokens systematically trade off against tool-call correctness due to conflicting gradient directions. To resolve this, we propose 1) a disentangled mask reward mechanism that decouples reasoning quality from tool syntax and 2) reasoning-focused gradient surgery that densifies task-relevant reasoning. MCPDisCo achieves higher success with *fewer* chain-of-thought tokens. On the out-of-distribution MCP-Atlas benchmark, MCPDisCo maintains monotone gains, confirming that the learned behavior generalizes beyond the training server distribution. Model weight and implementation are available. [1].

1 INTRODUCTION

The rapid advancement of Large Language Models (LLMs) has catalyzed a paradigm shift from conversational assistants to autonomous agents that act on the external world. A critical enabler of this shift is tool use—retrieving information, querying databases, driving software—but traditional tool-augmented LLMs rely on model-specific function-calling formats or bespoke API glue. Each tool requires custom integration, agents remain tightly coupled to a fixed set of pre-declared capabilities, and the ecosystem fragments along vendor and framework lines.

The Model Context Protocol (MCP) (Anthropic, 2024) has emerged as a transformative open standard that promises to resolve this fragmentation. MCP defines a unified client-server architecture in which servers act as lightweight wrappers exposing tools through standardized interfaces, and an MCP client—such as an LLM agent—can *dynamically discover, query, and execute* tools across an unbounded set of servers without prior knowledge of their internal implementations. Much as HTTP standardized document exchange for the Web, MCP is on track to become the substrate of the *agentic web*: new tools can be added as MCP servers without touching agent logic (plug-and-play scalability), a single MCP-compliant agent can operate seamlessly across servers from any provider (cross-vendor interoperability), and integration effort collapses from hard-coding function schemas to routing over a standardized discovery layer.

Despite these architectural benefits, effectively *learning* to act inside a large, dynamic MCP ecosystem introduces novel challenges that are absent from static, hand-crafted tool sets. An MCP-native agent must reason about the user's query while navigating a potentially unbounded space of servers and tools. At each step, it must decide whether to search for a relevant tool (Route), invoke a tool with syntactically correct parameters (Execute), or directly respond to the user (Terminate). Chain-of-thought reasoning and tool invocation are entangled: they share the same token stream, are optimized

[1] https://anonymous.4open.science/r/MCPDisCo/README.md

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.

against overlapping reward signals, and—as we show in this work—push model parameters in *conflicting* directions. We observe that under naively integrated rewards, improvements in reasoning quality come at the cost of tool-call accuracy and vice versa, a phenomenon we call the *reasoning–tool seesaw effect*. This coupling means that scaling up MCP-native tool use is not merely a data problem; it is fundamentally an optimization and reward-design problem.

In this paper, we introduce MCPDisCo, an RL framework for training MCP-native agents at ecosystem scale. We formalize the agent's operation as a Thought-Markov Decision Process (Thought-MDP) that models the interplay between internal reasoning and external actions as separate but coupled decision streams. To break the reasoning–tool seesaw, we design a hierarchical, *disentangled* reward system that scores reasoning quality and tool syntax as first-class, independently attributable signals, thereby densifying task-relevant reasoning and cutting redundant chain-of-thought without sacrificing action correctness. Empirically, MCPDisCo substantially raises success rates on LiveMCPBench, uses *fewer* chain-of-thought tokens than the untuned baseline, and—trained on 70 servers—transfers to a disjoint out-of-distribution benchmark (MCP-Atlas).

Our contributions are fourfold:

- **Positioning MCP as the substrate of the agentic web.** We articulate MCP's role as an open standard that decouples LLM agents from tools, discuss its use cases and orthogonality to existing agentic frameworks, and frame MCP-native agents as a distinct learning problem—not a variant of static function calling.
- **Thought-MDP and the first RL tuning of MCP-native agents at ecosystem scale.** We formalize the MCP tool-calling pipeline as a Thought-MDP that makes internal reasoning a first-class state variable. To the best of our knowledge, MCPDisCo is the first work to perform RL fine-tuning of MCP-native agents on real, heterogeneous MCP infrastructure (70 servers, 527 tools, 6 domains), and the trained policy transfers to the out-of-distribution MCP-Atlas benchmark with monotone gains.
- **Reasoning–tool seesaw and disentangled rewarding.** We surface the reasoning–tool seesaw effect as a concrete failure mode of jointly optimized rewards in MCP-native RL, and propose a hierarchical, disentangled reward mechanism that separately attributes credit to reasoning quality and tool syntax. This design densifies task-relevant reasoning, achieving higher task success with *fewer* chain-of-thought tokens than the untuned baseline (1663 vs. 1762 CoT tokens; 15.79%→26.32% success rate).
- **Open resources for the community.** We release (i) a public training dataset of 8,000 reasoning-augmented samples curated from over 200,000 candidates via a synthesize-and-recheck pipeline, (ii) the training code, and (iii) the trained model weights, to lower the barrier for future research in MCP-native agents.

2 Related Work

Tool-augmented LLMs and MCP-native agents. Early tool-augmented LLMs teach models to invoke a small, hand-curated set of APIs, either via supervised fine-tuning on annotated tool traces (Schick et al., 2023) or via prompt-based orchestration such as ReAct (Yao et al., 2023). Scaling up the tool set, ToolLLM (Qin et al., 2024) demonstrated that instruction-tuned models can handle 16,000+ real-world APIs, but the tool schemas remain part of the training distribution. The Model Context Protocol (Anthropic, 2024) moves this contract out of the model: any server complying with the protocol can be discovered and invoked at inference time, independent of the model's pretraining exposure. This shift has spawned a family of MCP-oriented benchmarks: LiveMCPBench (Mo et al., 2025) stresses agents against 70 heterogeneous real servers with 527 tools across six domains, while MCP-Atlas (Bandi et al., 2026) extends the tool population to test out-of-distribution competency. Recent work has begun to close the training-data gap. MCP-Flow (Wang et al., 2026) automatically discovers 1,166 servers and 11,536 tools and constructs 68,733 instruction–function-call pairs plus 6,439 trajectories for supervised fine-tuning and retrieval augmentation. Its scale makes it an important complementary reference point, but its main objective is tool selection and function-call generation from synthesized data. In contrast, MCPDisCo studies reinforcement learning over long-horizon, live-server trajectories and isolates optimization interference between

reasoning tokens and tool-call tokens. To our knowledge, it is the first ecosystem-scale RL study of an MCP-native policy to diagnose this reasoning–tool conflict and test transfer to a disjoint server benchmark.

Reinforcement learning for tool use. Applying RL to tool-augmented LLMs has been explored in the direction of end-task reward: ToolRL (Qian et al., 2025) shows that a carefully designed outcome-plus-format reward suffices to teach general tool competency; RLHF-style training pipelines (Ouyang et al., 2022) more broadly optimize a preference model over full trajectories. These approaches use a single scalar reward whose gradient flows uniformly across reasoning and action tokens. In the MCP-native setting, we argue and empirically show that this uniform treatment is precisely what produces the reasoning–tool seesaw. Concurrent work by Li et al. (2026) also observes reasoning–tool interference in agentic RL and advocates a disentangled tuning protocol; our work differs in (i) grounding the phenomenon in a Thought-MDP formulation, (ii) offering a mechanistic gradient-cosine diagnosis, and (iii) validating on real MCP servers with an OOD transfer test.

Reward disentanglement and reasoning length control. Disentangling reward into orthogonal signals has recently gained traction as an antidote to reward hacking. ODIN (Chen et al., 2024) decomposes an RLHF reward into content vs. length components to prevent length exploitation; DuPO (She et al., 2025) constructs dual preferences to verify a model's own outputs; CoLD (Zheng et al., 2025) counterfactually debiases process reward models against reasoning length. A parallel line of work directly manipulates reasoning verbosity: Thinkless (Fang et al., 2025) lets the model decide when to think, and DeepCompress (Liang et al., 2025) pairs an exploration reward with a compression reward to dynamically shorten chains-of-thought. MCPDisCo is complementary: rather than disentangling *within* the reasoning signal or *within* the reward-model output, it disentangles across *decision streams*—reasoning quality vs. tool syntax—and shows that the resulting policy shortens CoT as a side-effect of eliminating the seesaw, without an explicit length penalty.

Multi-task optimization and gradient conflict. The multi-task learning literature has long recognized that summing task losses can be pathological when their gradients disagree. PCGrad (Yu et al., 2020) projects each task gradient onto the normal plane of the others when they conflict; GradNorm (Chen et al., 2018) rescales gradients to balance learning rates; CAGrad (Liu et al., 2021) solves a min-norm subproblem for a conflict-averse update direction. These methods operate at the gradient level after the joint objective has been formed. MCPDisCo takes a structural view: we identify that the reasoning and tool-syntax rewards act on *disjoint* token spans, and route credit accordingly at the reward level. This eliminates token-level gradient conflict by construction; any residual parameter-level tension in the shared trunk is well-behaved and does not require post-hoc surgery. Whether combining our reward-level disentanglement with gradient-level surgery yields further gains is an open question we leave to future work.

3 FORMULATION

We formalize the MCPDisCo agent as a *Thought-MDP* (Markov Decision Process with explicit thinking states), defined by $\langle \mathcal{S}, \mathcal{A}, p, r, \gamma, \mathcal{T}, \mathcal{C}, p_{\mathcal{T}} \rangle$.

Specifically, $\mathcal{S}$ denotes the set of environment states, representing the accumulated context comprising the user query q, conversation history, and tool execution observations. $\mathcal{A}$ denotes the action space, comprising three mutually exclusive choices at each step: (1) **Route**: invoke a routing tool to search for and retrieve relevant MCP servers or tools; (2) **Execute**: call an execution tool with specific JSON-formatted parameters; (3) **Terminate**: output a direct response to conclude the session. $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ denotes the environment transition function, in which tool execution produces observations that update the state. $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ denotes the reward function, providing hierarchical feedback on both reasoning quality and tool syntax correctness. $\gamma \in [0, 1)$ represents the discount factor for cumulative reward optimization. $\mathcal{T}$ denotes the set of thought states capturing the agent's internal reasoning trajectory. $\mathcal{C}$ denotes the set of thought actions (i.e., reasoning tokens) that do not directly affect environment states but guide subsequent action selection. Finally, $p_{\mathcal{T}} : \mathcal{S} \times \mathcal{T} \times \mathcal{C} \to \Delta(\mathcal{T})$ denotes the thought transition function governing the evolution of internal reasoning.

Given a user query q, the agent generates a trajectory $\tau = [\langle s_1, t_1, a_1, o_1\rangle, \langle s_2, t_2, a_2, o_2\rangle, \ldots, \langle s_L, t_L, a_L, o_L\rangle]$ of maximum length L, where $t_i = [z_1, z_2, \ldots, z_{L_i}]$ denotes the reasoning tokens generated at step i.

4 METHODOLOGY

4.1 THOUGHT-MDP AGENT LOGIC

The overall pipeline of the MCP-Agent is illustrated in Figure 3. The lower server-to-tool decision corresponds to conventional tool-call intelligence—selecting a schema, producing valid JSON, and filling arguments—which can benefit from general tool-format data. MCP adds the upper query-to-server layer: servers act as flexible wrappers around heterogeneous or composite functions, so a single MCP-native agent can discover the appropriate wrapper, invoke its tools, absorb observations, and continue composing calls. At each Thought-MDP state, the agent selects one of three actions: invoking the *Route* tool for server and tool discovery, invoking the *Execute* tool for task execution, or issuing a direct response to terminate the session.

4.2 REWARD COMPONENTS FORMULATION

We formalize a hierarchical reward structure for the MCP tool-calling pipeline, in which each level addresses a distinct aspect of agent behavior.

L1: Reasoning Quality Reward. The agent's reasoning output is first parsed for explicit planning structures (preambles) following the Harmony format (Agarwal et al., 2025):

$$r_1^{\text{preamble}} = \mathbb{I}\big[\text{STRUCT}(\mathbf{z}) = \langle \text{preamble}, \text{analysis}, \text{action}\rangle\big], \tag{1}$$

where $\mathbf{z} = [z_1, z_2, \ldots, z_L]$ denotes the reasoning tokens generated for query q.

The reasoning reward is formulated as an intrinsic reward to encourage task-aligned and logically coherent intermediate thought processes. It comprises two components: Relevance Density (D_{rel}) and Coherence Score (S_{coh}).

- **Relevance Density (D_{rel}):** This component measures the degree to which the agent's reasoning aligns with the user's task. Task-specific keywords are extracted from the input prompt using KeyBERT, forming a relevant vocabulary set V_{rel}. The density is computed as the proportion of tokens in the reasoning text $T_{\text{reasoning}}$ that belong to this vocabulary:

$$D_{rel} = \frac{\sum_{t \in T_{\text{reasoning}}} \mathbb{I}(t \in V_{rel})}{|T_{\text{reasoning}}|}, \tag{2}$$

  where $\mathbb{I}(\cdot)$ is the indicator function.

- **Coherence Score (S_{coh}):** To ensure that the reasoning process is logically sequential rather than disjointed, the thoughts are segmented into N individual steps (sentences). Each sentence is embedded into a vector space using a pre-trained sentence embedding model. To ground the embeddings in the task context, the matched query keywords are appended to each sentence prior to embedding (Reimers & Gurevych, 2019). The coherence score is defined as the mean cosine similarity between consecutive reasoning steps, normalized to $[0, 1]$:

$$S_{coh} = \frac{1}{N-1} \sum_{i=1}^{N-1} \frac{\cos(e_i, e_{i+1}) + 1}{2}, \tag{3}$$

  where e_i denotes the embedding of the i-th reasoning step.

The final reasoning reward is computed as:

$$R_{\text{reasoning}} = D_{rel} + \lambda \cdot S_{coh}, \tag{4}$$

where λ is a hyperparameter controlling the relative weight of coherence in the overall reasoning reward.

L2: Tool Syntax Reward. The tool syntax reward evaluates the formatting and parameter correctness of the agent's generated tool calls. It is decomposed as $R_{\text{tool}} = R_{\text{format}} + R_{\text{correct}}$, where R_{format} is a base score awarded for generating a syntactically valid JSON tool call.

The correctness score R_{correct} is defined separately for the two types of tool invocations: *execution* tools and *routing* tools, both evaluated against ground-truth tool call parameters.

Execution Tool Reward: For direct tool executions, the agent must correctly predict the target server, tool name, and specific arguments. An accumulated heuristic score H_{exec} is defined as:

$$H_{\text{exec}} = R_{\text{format}} + s_{\text{base}} + s_{\text{req}} + s_{\text{srv}} + s_{\text{tool}} + \text{IoU}_{\text{keys}} + N_{\text{val_match}}, \tag{5}$$

where s_{base} and s_{req} are bonus scores for providing a parameter dictionary and all required keys, respectively; s_{srv} and s_{tool} reward correct server and tool name predictions; IoU_{keys} is the Intersection over Union of the predicted and ground-truth argument keys; and $N_{\text{val_match}}$ is the count of exactly matched argument values.

To normalize this score across tools with varying numbers of arguments, R_{correct} is scaled dynamically as:

$$R_{\text{correct}}^{\text{exec}} = 6 \times \left(\frac{H_{\text{exec}}}{2 + |A_{\text{gt}}|} \right) - 3, \tag{6}$$

where $|A_{\text{gt}}|$ denotes the number of arguments in the ground-truth function call.

Routing Tool Reward: For routing actions, the reward is determined by whether the routing query successfully retrieves the target tool required for the subsequent execution step. A heuristic score H_{route} is defined as:

$$\begin{aligned} H_{\text{route}} = R_{\text{format}} + s_{\text{base}} + s_{\text{query}} \\ + s_{\text{srv_match}} + s_{\text{tool_match}}, \end{aligned} \tag{7}$$

where s_{query} rewards the presence of a well-formed search query, and $s_{\text{srv_match}}$ and $s_{\text{tool_match}}$ are binary indicators of whether the ground-truth server and tool names were successfully recalled in the router's output response.

The correctness score is linearly scaled as:

$$R_{\text{correct}}^{\text{route}} = 6 \times \left(\frac{H_{\text{route}}}{3} \right) - 2. \tag{8}$$

In cases where the agent produces severe formatting errors or omits required parameters, R_{correct} incurs a substantial penalty (e.g., -2.0 or -3.0). Conversely, if the agent successfully concludes a task with a well-formed conversational answer, a modest positive shaping reward is provided.

4.3 Token-Masked Disentangled Reward

Having defined the two reward families $R_{\text{reasoning}}$ and R_{tool}, a natural design choice is to optimize their scalar sum $R_{\text{joint}} = R_{\text{reasoning}} + R_{\text{tool}}$ with policy gradient RL. We refer to this baseline as *integrated rewarding*. Integrated rewarding treats a trajectory-level scalar as if every generated token were responsible for both reasoning quality and tool syntax, even though reasoning tokens and tool-call tokens serve different decision streams.

MCPDisCo instead uses token-masked reward attribution. Let $y_{1:M}$ denote the generated token sequence for one trajectory, and let $m_t^R, m_t^T \in \{0, 1\}$ be disjoint masks satisfying $m_t^R m_t^T = 0$. The reasoning mask m_t^R selects tokens inside the explicit reasoning span $\mathbf{z}$, while the tool mask m_t^T selects tokens that encode external actions, including ROUTE, EXECUTE, TERMINATE, JSON arguments, and final-answer tokens that receive terminal shaping. The integrated baseline assigns the summed advantage to every token,

$$\nabla_\theta J_{\text{int}} = \mathbb{E}\left[\sum_{t=1}^{M} \nabla_\theta \log \pi_\theta(y_t \mid h_t) \left(A_t^{R_{\text{reasoning}}} + A_t^{R_{\text{tool}}} \right) \right], \tag{9}$$

where h_t is the decoding history and the two advantages are estimated from the corresponding reward families.

Under token-masked disentanglement, each reward only supervises the token span that generated it:

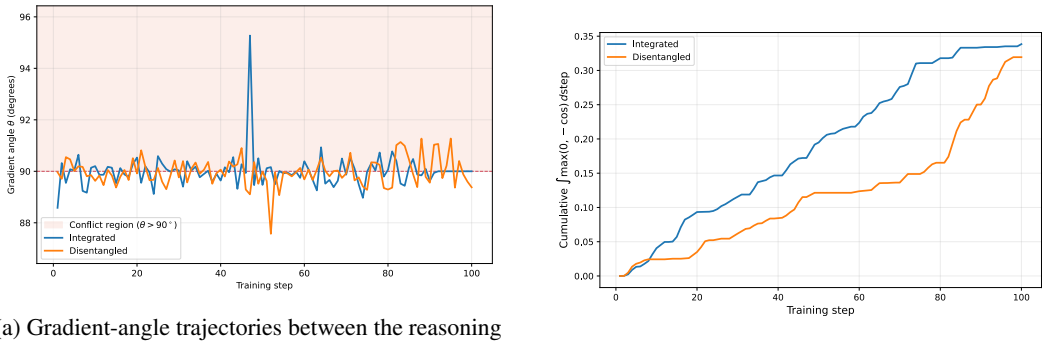
$$\nabla_\theta J_{\text{mask}} = \mathbb{E}\left[\sum_{t=1}^{M} m_t^R \nabla_\theta \log \pi_\theta(y_t \mid h_t)\, A_t^{R_{\text{reasoning}}}\right] + \mathbb{E}\left[\sum_{t=1}^{M} m_t^T \nabla_\theta \log \pi_\theta(y_t \mid h_t)\, A_t^{R_{\text{tool}}}\right]. \quad (10)$$

This is the core disentangled mask reward mechanism: $R_{\text{reasoning}}$ cannot directly reward or penalize JSON syntax tokens, and R_{tool} cannot directly reward or penalize reasoning verbosity. Since the two terms act on disjoint token spans, token-level credit conflict is removed by construction; any residual conflict can only arise through shared parameters, which we handle with the reasoning-focused surgery below.

4.4 Reasoning-Focused Gradient Surgery

Token masking resolves misattribution at the reward level, but reasoning and tool objectives are still optimized through the same model parameters. We therefore diagnose the remaining shared-parameter interference and apply an asymmetric surgery that protects tool execution while making reasoning updates denser and more task-aligned.

Empirical observation: the seesaw effect.



(a) Gradient-angle trajectories between the reasoning and tool policy gradients over 100 training steps.

(b) The cumulative negative-cosine integral.

Figure 1: Illustrations of the seesaw effect.

The gradient-angle trajectories provide a direct geometric view of the mechanism underlying the reasoning–tool seesaw. An angle above 90° implies a negative inner product between the reasoning and tool policy gradients, such that a local update that improves one objective can impair the other. Over the first 100 optimization steps, both conditions fluctuate narrowly around the orthogonality boundary, with mean angles of approximately 90° and 44% of recorded steps lying marginally in the conflict region. The integrated-reward condition, however, exhibits substantially larger excursions, reaching approximately 95.3°, whereas the disentangled condition remains bounded near 91.3°. This pattern indicates that reward disentanglement primarily attenuates the severity of destructive interference rather than eliminating all conflicts. The threshold crossings are mostly short-lived rather than forming a prolonged conflict interval; accordingly, the figure should be interpreted as evidence of intermittent parameter-level conditions that can induce a seesaw.

Mechanistic diagnosis: gradient conflict. The seesaw is not an artifact of noisy estimation. Let $g_R = \nabla_\theta \mathbb{E}[R_{\text{reasoning}}]$ and $g_T = \nabla_\theta \mathbb{E}[R_{\text{tool}}]$ denote the per-step policy gradients induced by each reward family after token masking. Figure 1a plots the gradient angle between g_R and g_T across training steps. Under integrated rewarding, $g_R^\top g_T < 0$ on a substantial fraction of steps, i.e., a positive update on one objective is on average a *descent* direction on the other. This is exactly the *gradient conflict* regime characterized by Yu et al. (2020) in the multi-task setting: naive gradient summation causes the two updates to partially cancel, so training reduces to an implicit oscillation between the two objectives.

From Figure 1b, we can see that the cumulative negative-cosine integral, defined as $\int \max(0, -\cos)\, d\text{step}$, aggregates the frequency, duration, and magnitude of gradient conflicts throughout optimization. Across the first 100 training steps, the integrated-reward run accumulates

conflict more rapidly for most of training and reaches a final value of approximately 0.338, compared with approximately 0.319 for the disentangled run. This corresponds to a reduction of about 5.6% in total conflict exposure. During the whole training process, the disentangled trajectory remains below the integrated one for almost all the time except the beginning. These results suggest that disentangled reward attribution improves gradient compatibility over the course of training, while also showing that non-negligible interference can still emerge at later optimization stages.

Asymmetric reasoning-focused projection. To remove the residual parameter-level conflict, MCPDisCo applies surgery only to the reasoning gradient and treats the tool gradient as the anchor, because syntactically valid and semantically correct tool execution is the non-negotiable interface constraint of MCP agents. Given minibatch gradients g_R and g_T from the masked losses, we compute

$$\tilde{g}_R = \begin{cases} g_R - \dfrac{g_R^\top g_T}{\|g_T\|_2^2 + \epsilon} g_T, & \text{if } g_R^\top g_T < 0, \\ g_R, & \text{otherwise,} \end{cases} \qquad g_{\text{update}} = g_T + \alpha \tilde{g}_R, \tag{11}$$

where ϵ is a small numerical stabilizer and α controls the strength of the reasoning update. Unlike symmetric PCGrad-style projection, this rule never projects away the tool gradient; it only removes the component of the reasoning update that would reduce tool-call correctness. The resulting update preserves the syntactic reliability required by MCP execution while still allowing reasoning-specific gradients to improve relevance density and coherence. This design predicts, and our experiments confirm (Section 5.2), that MCPDisCo simultaneously (i) attenuates the seesaw, (ii) densifies task-relevant reasoning and thus *shortens* chain-of-thought, and (iii) improves task success rate.

5 Experiments

5.1 Setup

Benchmark. We evaluate our method on LiveMCPBench (Mo et al., 2025), a comprehensive evaluation framework designed to assess MCP-native agents in real-world scenarios, encompassing 70 MCP servers and 527 tools. The benchmark covers 95 diverse daily tasks across six domains: Office, Leisure, Travel, LifeStyle, Finance, and Shopping. For OOD test, we test the trained model on MCP-Atlas (Bandi et al., 2026).

Training Data. We use the $8{,}000$-sample reasoning-augmented corpus $\mathcal{D}_{\text{train}}$ produced by the synthesize-and-recheck pipeline. The same pipeline and the $200{,}000$+ raw candidate pool are released for community reuse.

Evaluation Metrics. We adopt three metrics to measure both reasoning behavior and task-level performance: (1) **Avg. CoT Tokens**: the average number of chain-of-thought tokens generated per query, reflecting reasoning verbosity; (2) **Avg. Chain**: the average number of tool-calling steps per trajectory, reflecting planning depth; and (3) **Success Rate**: the proportion of tasks successfully completed, as the primary performance indicator.

5.2 Main Results

Table 1: Comparison of task-level performance. (Qwen3-4B)

Domain	Base			MCPDisCo (-surgery)			MCPDisCo (-mask)			MCPDisCo		
	Avg. CoT Tokens	Avg. Chain	Success Rate (%)	Avg. CoT Tokens	Avg. Chain	Success Rate (%)	Avg. CoT Tokens	Avg. Chain	Success Rate (%)	Avg. CoT Tokens	Avg. Chain	Success Rate (%)
Office	2048.26	10	22.58	2389.58	12.32	25.81	2369.65	11.26	16.13	1803.39	14.52	25.81
Leisure	933.64	5.07	14.29	1686.07	9.93	35.71	1585.14	9.21	35.71	1349.21	8.93	21.43
Travel	2815.92	9.67	8.33	2690.33	13	8.33	2948.92	11.17	16.67	2715.58	17.17	50
LifeStyle	1616.4	8.6	13.33	1812.73	10.67	26.67	1868.2	10.6	40	1331.17	12.13	13.33
Finance	1733.37	9.14	7.14	2212.07	13.79	7.14	2626.71	13.29	21.43	1665.29	16.29	21.43
Shopping	945.89	5.67	22.22	973.44	7.22	22.22	985.44	6.33	22.22	818.67	8.11	33.33
Overall	1761.97	8.47	15.79	2072.49	11.53	22.11	2154.78	10.67	24.21	1663.53	13.31	26.32

- **Main Result** Table 1 reports the task-level performance of the base agent the agents trained with different rewarding. Overall, reinforcement learning with MCPDisCo improves the success

Table 2: Reasoning On/Off (Base)

Method	#Chain	Pass Rate
Reasoning Off	11.85	14.74
Reasoning On	6.95	15.79

Table 3: Reasoning On/Off (w/ MCPDisCo)

Method	#Chain	Pass Rate
Reasoning Off	8.29	16.84
Reasoning On	13.31	26.32

rate from 15.79% to 26.32%, corresponding to an absolute gain of 10.53 percentage points compared to the base model without tuning. This result indicates that MCP-native tool use benefits substantially from the explicit optimization and disentanglement of routing, execution, and termination behaviors.

- **Reasoning Density and Token Efficiency**

  A central prediction of the seesaw analysis in Section 4.4 is that disentangled rewarding should not merely trade quality for quantity in the reasoning stream, but should raise reasoning *density*—the amount of task-relevant content per token—thereby enabling shorter chain-of-thought at equal or higher success. The results in Table 1 confirm this prediction along three orthogonal axes:

  – **Shorter CoT than the untuned baseline.** Disentangled rewarding reduces average CoT tokens to 1663.53, *below* the 1761.97 produced by the untuned base agent. This is unusual: RL fine-tuning for reasoning typically inflates CoT length (as does our own integrated-rewarding baseline, which reaches 2154.78). MCPDisCo is the only configuration that improves success while shrinking the reasoning budget.

  – **Deeper tool interaction.** Simultaneously, the average tool-call chain grows from 8.47 to 13.31. Tokens are being reallocated from redundant reasoning into productive tool invocations—consistent with the mechanism proposed in Section 4.4: separating credit for $R_{\text{reasoning}}$ and R_{tool} lets the policy stop over-verbalizing in service of a shared scalar.

  – **Efficiency $\times$ effectiveness.** Combining the two, MCPDisCo delivers a 10.53-point success-rate lift (15.79% $\rightarrow$ 26.32%) using 5.6% fewer CoT tokens than the base model, giving a strictly better operating point on the (tokens, success) frontier than either the base or the integrated-rewarding variant.

  Figure 2 decomposes this behavior across the six LiveMCPBench domains. The clear majority of domains lie in the "fewer tokens, higher success" quadrant under disentanglement, corroborating that the aggregate efficiency gain is not driven by a single outlier domain.

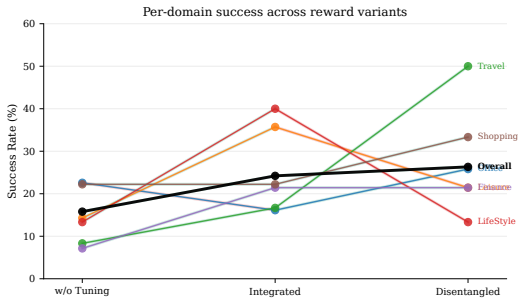


Figure 2: Per-domain change in average CoT tokens vs. change in success rate.

5.3 Out-of-Distribution Generalization

To probe whether the learned reasoning—tool interaction policy—transfers beyond the training server distribution, we evaluate on MCP-Atlas (Bandi et al., 2026), a benchmark whose server population is disjoint from LiveMCPBench, and a general agent benchmark GAiA (Mialon et al., 2024). Results are summarized in Table 6. Both integrated rewarding and disentangled rewarding produce monotone improvements over the untuned base on ACC and Mean Coverage, and disentanglement retains its advantage in the OOD regime, indicating that the learned policy generalizes rather than overfits to the training servers.

Table 4: Leaderboard performance on LiveMCP.

Model	Avg. CoT Tokens	Avg. Chain	Success Rate (%)
Base	1761.97	8.47	15.79
ToolACE-8B	0	9.08	8.42
Hammer-7B	255.67	12.05	12.63
MCPDisCo	1663.53	13.31	26.32

Table 5: OOD test on MCP-Atlas

Model	ACC	Mean Coverage
Base	0.0186	0.1165
ToolACE-8B	0.0268	0.1258
Hammer-7B	0.0205	0.1194
MCPDisCo	0.0404	0.1360

Table 6: OOD test on GAIA

Model	L1	L2
Base	0.1320	0.1050
ToolACE-8B	0.0430	0.0230
Hammer-7B	0.0750	0.0470
MCPDisCo	0.1510	0.0700

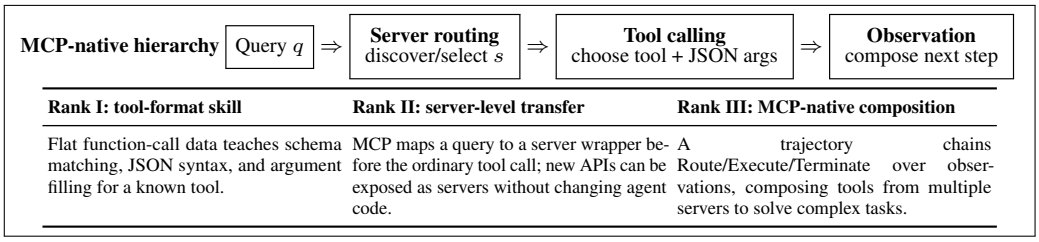


Figure 3: Hierarchy behind MCP-native generalization. Ordinary tool-call tuning covers the lower tool-format layer; MCP adds a server-routing and wrapping layer, enabling flexible, composable functions and a higher generalization rank than flat function calling.

Overall, our real-world, hierarchical (query -> server -> tool), long-horizon (>10 interaction turns) agent data, together with our disentangled and reasoning-focused tuning algorithm, makes the resulting tuned model generalizable for wide agent tasks. OOD tests on common and untuned benchmarks validate the conclusion.

6 Conclusion

In this paper, we have argued that the Model Context Protocol is fast becoming the substrate of the agentic web, and that training *MCP-native* agents at ecosystem scale is a distinct learning problem—not a rebrand of function calling. We introduced MCPDisCo, the first RL framework to tune MCP-native agents on 70 heterogeneous MCP servers and 527 tools. We formalized the agent as a Thought-MDP that makes reasoning a first-class state variable, identified the reasoning–tool seesaw effect that jointly optimized rewards induce, and proposed a hierarchical disentangled reward mechanism that decouples reasoning quality from tool syntax. Empirically, MCPDisCo raises the LiveMCPBench success rate from 15.79% to 26.32% while producing shorter chain-of-thought than the untuned baseline, and it maintains monotone gains on the out-of-distribution MCP-Atlas benchmark, evidence that the learned behavior generalizes beyond the training server distribution. To catalyze further research in MCP-native agents, we release the 8,000-sample training dataset, the training code, and the trained model weights.

Reproducibility statement

The data-construction pipeline is described in Section **??**, the reward functions and attribution rule are specified in Section 4.4, and complete optimization, rollout, infrastructure, and evaluation settings are provided in Appendix A. We also provide anonymous code and model artifacts in the supplementary material. Because the reported live-server runs are expensive and server behavior can change over time, we explicitly identify single-run reporting and live-infrastructure variability as limitations.

Ethics statement

Training and evaluating agents against live MCP servers creates privacy, security, and unintended-side-effect risks. Our experiments use benchmark tasks and vetted servers, avoid private user data, cap interaction length, and treat tool outputs as untrusted observations. Released trajectories should be screened for credentials and personally identifiable information, and deployment should use least-privilege credentials, sandboxing, human confirmation for consequential actions, and auditable execution logs.

AI USE STATEMENT

Generative AI tools were used to assist with language editing, literature discovery, and LaTeX-format migration. All technical claims, citations, equations, experimental values, and generated edits were reviewed by the authors, who take responsibility for the final content. No AI-generated experimental result is presented as an observed result.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Anthropic. Introducing the model context protocol, 2024. URL https://www.anthropic.com/news/model-context-protocol.

Chaithanya Bandi, Ben Hertzberg, Geobio Boo, Tejas Polakam, Jeff Da, Sami Hassaan, Manasi Sharma, Andrew Park, Ernesto Hernandez, Dan Rambado, et al. Mcp-atlas: A large-scale benchmark for tool-use competency with real mcp servers. *arXiv preprint arXiv:2602.00933*, 2026.

Lichang Chen, Chen Zhu, Davit Soselia, Jiuhai Chen, Tianyi Zhou, Tom Goldstein, Heng Huang, Mohammad Shoeybi, and Bryan Catanzaro. Odin: Disentangled reward mitigates hacking in rlhf, 2024. URL https://arxiv.org/abs/2402.07319.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning (ICML)*, pp. 794–803, 2018.

Gongfan Fang, Xinyin Ma, and Xinchao Wang. Thinkless: Llm learns when to think. *arXiv preprint arXiv:2505.13379*, 2025.

Yu Li, Mingyang Yi, Xiuyu Li, Ju Fan, Fuxin Jiang, Binbin Chen, Peng Li, Jie Song, and Tieying Zhang. Reasoning and tool-use compete in agentic rl: From quantifying interference to disentangled tuning. *arXiv preprint arXiv:2602.00994*, 2026.

Tian Liang, Wenxiang Jiao, Zhiwei He, Jiahao Xu, Haitao Mi, and Dong Yu. Deepcompress: A dual reward strategy for dynamically exploring and compressing reasoning chains. *arXiv preprint arXiv:2510.27419*, 2025.

Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. Conflict-averse gradient descent for multi-task learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Grégoire Mialon, Clémentine Fourrier, Thomas Wolf, Yann LeCun, and Thomas Scialom. Gaia: a benchmark for general ai assistants. In *International Conference on Learning Representations*, volume 2024, pp. 9025–9049, 2024.

Guozhao Mo, Wenliang Zhong, Jiawei Chen, Xuanang Chen, Yaojie Lu, Hongyu Lin, Ben He, Xianpei Han, and Le Sun. Livemcpbench: Can agents navigate an ocean of mcp tools? *arXiv preprint arXiv:2508.01780*, 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, 2022.

Cheng Qian, Emre Xiong, Zijie Liu, Xingyao Chen, and Heng Ji. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*, 2025.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations*, 2024.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 3982–3992, 2019.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Shuaijie She, Yu Bao, Yu Lu, Lu Xu, Tao Li, Wenhao Zhu, Shujian Huang, Shanbo Cheng, Lu Lu, and Yuxuan Wang. Dupo: Enabling reliable llm self-verification via dual preference optimization. *arXiv preprint arXiv:2508.14460*, 2025.

WenHao Wang, Peizhi Niu, Zhao Xu, Zhaoyu Chen, Jian Du, Yaxin Du, Xianghe Pang, Keduan Huang, Yanfeng Wang, Qiang Yan, and Siheng Chen. MCP-flow: Facilitating LLM agents to master real-world, diverse and scaling MCP tools. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5087–5116, 2026. doi: 10.18653/v1/2026.acl-long.231. URL https://aclanthology.org/2026.acl-long.231/.

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 (ICLR)*, 2023.

Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in neural information processing systems*, 33: 5824–5836, 2020.

Congmin Zheng, Jiachen Zhu, Jianghao Lin, Xinyi Dai, Weiwen Liu, Yong Yu, Weinan Zhang, and Mengyue Yang. Cold: Counterfactually-guided length debiasing for process reward models in mathematical reasoning. *arXiv preprint arXiv:2507.15698*, 2025.

A Implementation Details

Backbone and RL algorithm. All MCPDisCo variants and the untuned baseline start from the same publicly released checkpoint, `Qwen3-4B`. The RL fine-tuning objective is optimized with a GRPO (Shao et al., 2024)-style policy-gradient update in which the KL to a frozen reference policy is added to the loss as a soft constraint. The advantage estimator is a group-relative baseline: for each prompt we roll out $G = 8$ trajectories in parallel and take the group-mean reward as the value baseline, which sidesteps the need for a separate critic head.

Optimization. We use AdamW with $\beta_1 = 0.9, \beta_2 = 0.95$, weight decay 0.01, a peak learning rate of 1×10^{-6}, and a cosine schedule with 5% linear warmup. The global batch size is 32 prompts (i.e., $32 \times G = 256$ trajectories per update) and we train for 512 policy-update steps, which corresponds to roughly one and a half epochs over the $8{,}000$-sample training set. The KL coefficient to the reference policy is $\beta_{\mathrm{KL}} = 1 \times 10^{-3}$. All variants use the same optimizer and schedule; only the reward attribution differs.

Reward hyperparameters. The reasoning reward weight is $\lambda = 0.5$ (Eq. 4 of the main paper), balancing D_{rel} and S_{coh}. Sentence embeddings for S_{coh} are computed with the `all-MiniLM-L6-v2` encoder (Reimers & Gurevych, 2019); task keywords for D_{rel} are extracted with KeyBERT (top-10 n-grams, $n \in \{1, 2\}$). For the tool syntax reward, the format base is $R_{\mathrm{format}} = 1$, and the additive bonuses $s_{\mathrm{base}}, s_{\mathrm{req}}, s_{\mathrm{srv}}, s_{\mathrm{tool}}$ are each set to 1; severe formatting errors receive a -2.0 penalty and missing required keys receive a -3.0 penalty (main paper, §3.3).

Rollout and decoding. Rollouts use sampling with temperature 0.9 and top-$p = 0.95$; the per-trajectory token budget is 8192 tokens, of which up to 4096 may be consumed by chain-of-thought. Each trajectory has a hard cap of 20 tool-invocation steps to bound MCP-server exposure per query.

Tool servers are pinned to the same 70-server pool used at evaluation to ensure a consistent action distribution.

Infrastructure and runtime. Training is performed on a single node with $8\times$ NVIDIA A100-80GB GPUs. Rollouts are sharded across the GPUs and dispatch to MCP servers over a local network. A full 512-step run takes approximately 36 hours end-to-end, with roughly 70% of wall-clock spent on MCP tool invocations rather than gradient updates.

Evaluation protocol. All numbers in Tables 1 and 6 are computed at greedy decoding (temperature 0) on the official LiveMCPBench task splits and the MCP-Atlas, GAIA OOD split, with a 20-step tool-invocation cap and a 8192-token generation budget matching the training setup. The Success Rate is defined by the benchmark's official judge; Avg. CoT Tokens and Avg. Chain are computed only over completed trajectories.