

ActionRAG: Constraint-Aware Action Retrieval for Large Action Models

Inho Lee and Yunsick Sung*

Department of Computer Science and Artificial Intelligence
Dongguk University-Seoul, Seoul 04620, Republic of Korea
inholee@dgu.ac.kr sung@dongguk.edu

Abstract

Large Action Models (LAMs) select and execute actions from external tool spaces. Traditional relevance-based retrieval selects actions that satisfy semantic relevance but violate execution constraints, parameter requirements, or action preconditions. Reactive execution discovers infeasible action orderings only during rollout, which increases replanning cost. Reliable multi-step execution therefore requires feasibility-aware selection and dependency-aware ordering. We propose ActionRAG, a training-free retrieval framework that combines Constraint Induction, Constraint-Aware Retrieval, and Action-of-Thought to select and order feasible actions before execution. Constraint Induction extracts environment-level and task-level constraints from the environment state and the user goal. Constraint-Aware Retrieval ranks actions by semantic relevance and feasibility coverage, filtering infeasible actions before trajectory construction. Action-of-Thought then constructs feasible action trajectories over the filtered action set using inter-action dependencies and preconditions. We evaluate ActionRAG on StableToolBench and MCP-Bench across four backbones under single-tool, multi-step, and cross-category tool-use settings. On StableToolBench, ActionRAG improves the solvable pass rate by 40.3–43.4%p over CoT/ReAct, reaching 60.7%p on cross-category tasks, and reduces token cost by up to $6.2\times$ over DFSDT.

1 Introduction

Large Language Models (LLMs) support Large Action Models (LAMs) for action execution over external tools and environments (Wang et al., 2025a; Zhang et al., 2025; Zhou et al., 2024). LAMs execute API calls, code, and multi-step procedures from natural language goals (Schick et al., 2023).

Traditional LAMs perform action selection through direct generation or relevance-based re-

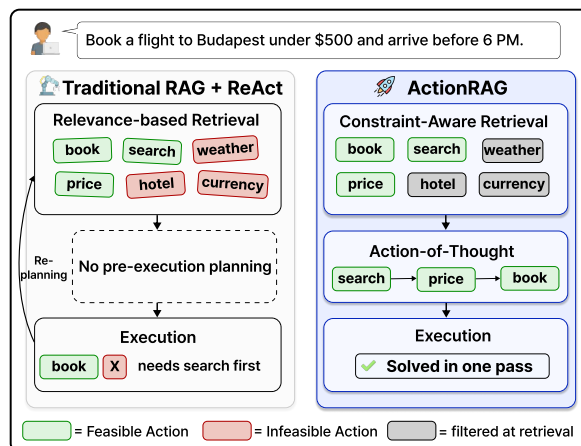


Figure 1: Without feasibility filtering or planning, Traditional RAG keeps off-task actions and calls book before search, triggering re-planning. ActionRAG filters them and orders the rest, solving it in one pass.

trieval adapted from retrieval-augmented generation (RAG). Relevance-based retrieval selects actions that satisfy semantic relevance but violate execution constraints, parameter requirements, or action preconditions (Huang et al., 2022; Qin et al., 2024). Action relevance does not ensure trajectory feasibility in multi-step execution. Reactive execution frameworks such as ReAct (Yao et al., 2023) interleave planning with execution, and therefore discover infeasible action orderings only during rollout, incurring re-planning cost at each failed step. Figure 1 illustrates this failure. Traditional approaches apply feasibility constraints through post-generation validation using solvers, simulators, or execution feedback (Madaan et al., 2023; Chen et al., 2023). These approaches enforce feasibility after action generation and execution planning. Inference therefore executes or attempts infeasible actions, which increases replanning cost and execution instability.

We propose **ActionRAG**, a feasibility-aware retrieval framework for LAMs. ActionRAG induces

* Corresponding author.

environment-level and task-level constraints from the environment state and user goal. *Constraint-Aware Retrieval* ranks actions using semantic relevance and feasibility coverage. The retrieval stage filters infeasible actions before execution. *Action-of-Thought (AoT)* constructs feasible action trajectories over the filtered action set. AoT uses inter-action dependencies and action preconditions for dependency-aware action ordering. ActionRAG performs trajectory construction before execution and reduces exploration of infeasible trajectories during inference.

Experiments on StableToolBench and MCP-Bench evaluate ActionRAG across four backbones under single-tool, multi-step, and cross-category action-execution settings. On StableToolBench, ActionRAG improves the solvable pass rate by 40.3–43.4%p over CoT/ReAct across four backbones, reaching a 60.7%p improvement on cross-category tasks, and reduces token cost by up to $6.2\times$ over DFSDT.

Contributions. Our contributions are summarized as follows:

- We propose **ActionRAG**, a feasibility-aware retrieval framework for action selection in Large Action Models.
- We propose *Constraint-Aware Retrieval*, which augments semantic relevance with a feasibility coverage term for infeasible action filtering, and *Action-of-Thought* for feasible trajectory construction during inference.
- We evaluate ActionRAG on StableToolBench and MCP-Bench across single-tool, multi-step, and cross-category action-execution settings.

2 Related Work

Retrieval for Tool and Action Selection.

Retrieval-augmented approaches have been widely used to support tool selection and external knowledge access in language agents. Early work such as Toolformer (Schick et al., 2023) and subsequent tool-augmented LLM frameworks (Qin et al., 2024; Patil et al., 2024; Kim et al., 2025; Choi et al., 2025) retrieve candidate tools by matching user queries with tool descriptions or API documentation. Similar retrieval paradigms have also been adopted in agentic reasoning frameworks such as ReAct (Yao et al., 2023) and API-based planning

systems (Huang et al., 2022), where semantic similarity is used to identify relevant actions. While these methods improve recall and scalability of tool usage, they primarily focus on relevance matching and do not explicitly incorporate execution feasibility into the retrieval process.

Action Generation and Planning in LAMs.

LAMs extend language models toward executable behavior by integrating structured action spaces, tool invocation, and multi-step decision making. Recent work has demonstrated the ability of LAM-based agents to perform complex workflows across heterogeneous environments (Wang et al., 2024; Chen et al., 2024; Hong et al., 2024; Pan et al., 2025). Action selection is typically performed through direct generation or relevance-based retrieval, with feasibility reasoning handled implicitly within the model’s internal planning process. As a result, LAMs often operate over action spaces that may contain many actions that are not executable under the current state, increasing the burden of downstream validation and error recovery.

Constraint-Aware Planning and Validation.

Constraint reasoning has long been studied in classical planning and has recently been incorporated into language-based agents through symbolic constraints, simulators, or execution feedback. Approaches such as program-of-thought reasoning (Chen et al., 2023), environment simulation and self-refinement (Madaan et al., 2023), and verifier-guided generation improve reliability by identifying infeasible actions or trajectories during execution (Ji et al., 2025). Related work in constrained decoding and planning (Lu et al., 2022; Valmeekam et al., 2023) similarly introduces structured validation mechanisms to enforce consistency and safety. However, in most existing formulations, constraint reasoning operates as a post-generation validation step rather than as a core component of action retrieval and selection.

Scaling Action Spaces in Agentic AI.

Recent progress in agentic AI has substantially expanded the action spaces available to language agents. Standardization efforts such as the Model Context Protocol (MCP) and related interoperability frameworks have lowered the cost of exposing external tools to LLMs, enabling large and heterogeneous tool ecosystems (Singh et al., 2025; Hou et al., 2026; Ray, 2025; Ehtesham et al., 2025; Lumer et al., 2025). As action spaces scale, retrieval-

based selection may return semantically relevant but execution-infeasible actions, while multi-step and cross-tool settings introduce dependencies not captured by relevance alone (Qu et al., 2025; Li, 2025; Plaat et al., 2025). ActionRAG addresses this setting through feasibility-aware retrieval and dependency-aware trajectory construction.

3 ActionRAG Framework

Figure 2 shows the ActionRAG framework. Given an environment state s_t , a user goal g_t , and an action space $\mathcal{A}$, ActionRAG first performs Constraint Induction (CI) to extract structured constraints and retrieval queries. Constraint-Aware Retrieval then ranks actions by semantic relevance and feasibility coverage, filtering infeasible actions before trajectory construction. Action-of-Thought (AoT) constructs a feasible action trajectory over the filtered set using inter-action dependencies and preconditions. The trajectory is finally executed by a Large Action Model (LAM), which grounds parameters from the user goal and intermediate results. ActionRAG is a training-free framework that applies feasibility constraints during retrieval and trajectory construction rather than as post-generation validation, reducing execution of infeasible actions and invalid trajectories during inference.

3.1 Problem Setup

Let $\mathcal{A}$ denote the action space. Each action $a \in \mathcal{A}$ corresponds to a tool invocation with structured parameters, associated with a specification $\tau(a)$ containing functionality, input parameters, and output fields. At inference time, the agent observes an environment state s_t and a user goal g_t . The environment state contains available actions, interaction history, and intermediate execution results. An action trajectory is a sequence $\pi = (a_1, \dots, a_T)$, and executing an action updates the state according to $s_{t+1} = \mathcal{T}(s_t, a_t)$, where $\mathcal{T}$ denotes the environment transition. The objective is to construct a feasible action trajectory that satisfies the user goal under the evolving environment state.

ActionRAG structures inference into three stages: (1) constraint induction, (2) Constraint-Aware Retrieval, and (3) feasible trajectory construction. The resulting trajectory is then executed by the LAM, which is not specific to ActionRAG.

3.2 Constraint Induction

Constraint Induction (CI) decomposes the user goal into sub-goals and extracts retrieval queries and

feasibility constraints from (s_t, g_t) . CI outputs

$$\langle \mathcal{G}_{\text{sub}}, \mathcal{Q}, C_{\text{env}}, C_{\text{task}} \rangle, \quad (1)$$

where $\mathcal{G}_{\text{sub}}$ is a set of sub-goals, $\mathcal{Q}$ is a set of retrieval queries, and $C_{\text{env}}, C_{\text{task}}$ are environment-level and task-level constraints.

Sub-goal decomposition. CI decomposes g_t into a set of atomic sub-goals

$$\mathcal{G}_{\text{sub}} = \{g_1, g_2, \dots, g_m\}, \quad (2)$$

which serve as retrieval units for both action retrieval and trajectory construction.

Retrieval query induction. For each sub-goal, CI generates retrieval queries describing the required action capabilities. The combined query set

$$\mathcal{Q} = \bigcup_{i=1}^m \mathcal{Q}_i \quad (3)$$

maps the user goal to the action space during retrieval.

Constraint extraction. CI extracts two types of constraints. *Environment-level constraints* C_{env} capture whether an action is feasible under the current state s_t , covering parameter availability, action preconditions, and parameter dependencies across actions. *Task-level constraints* C_{task} capture whether an action contributes to the user goal g_t , covering sub-goal assignment, required outputs, and parameter values grounded in the goal. Together, these define the feasibility function

$$C(a \mid s_t, g_t) = C_{\text{env}}(a \mid s_t) \wedge C_{\text{task}}(a \mid g_t), \quad (4)$$

where $C(a \mid s_t, g_t) = 1$ indicates that action a satisfies both constraint types. $\mathcal{G}_{\text{sub}}$ and $\mathcal{Q}$ guide retrieval, while C_{env} and C_{task} provide feasibility signals for downstream filtering and trajectory construction.

3.3 Constraint-Aware Retrieval

Constraint-Aware Retrieval ranks actions using semantic relevance and feasibility coverage, filtering infeasible actions before trajectory construction and execution.

Sub-goal-level multi-query retrieval. Given the query set $\mathcal{Q}$ generated by CI, each query $q \in \mathcal{Q}_i$ retrieves the top- k actions from the action space:

$$\mathcal{A}_q = \text{RETRIEVE}(q, \mathcal{A}, k). \quad (5)$$

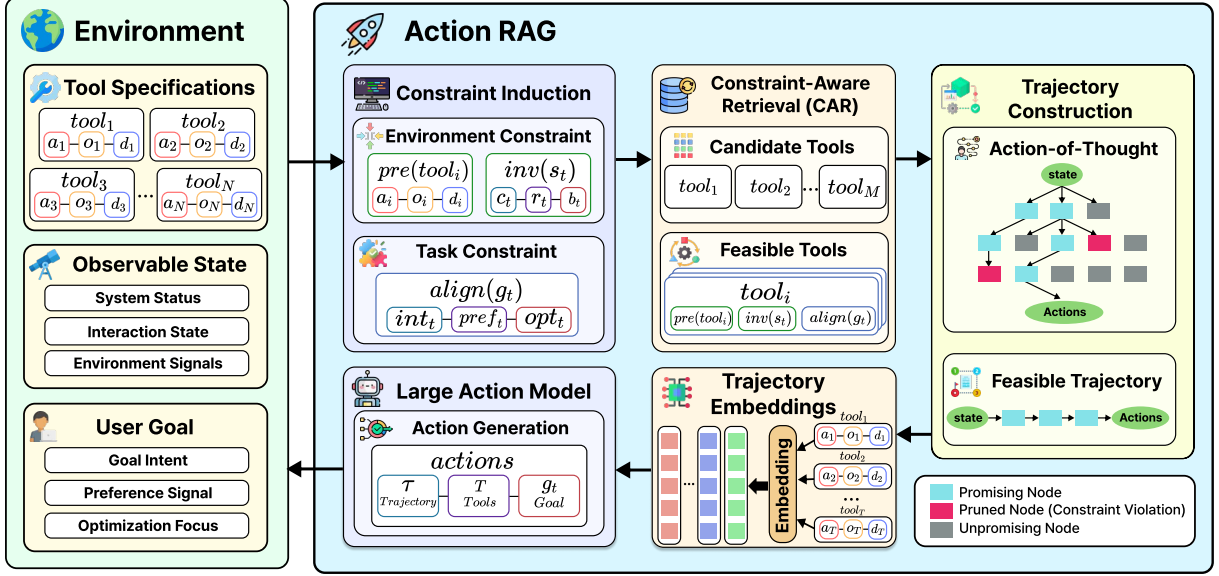


Figure 2: Overview of ActionRAG. The framework induces environment-level and task-level constraints from the environment state and user goal, retrieves candidate actions, filters infeasible actions through Constraint-Aware Retrieval, and constructs feasible action trajectories through Action-of-Thought (AoT) before execution by a Large Action Model.

The retrieved sets are aggregated across all sub-goals to form the candidate pool:

$$\mathcal{A}_{\text{cand}} = \bigcup_{i=1}^m \bigcup_{q \in \mathcal{Q}_i} \mathcal{A}_q. \quad (6)$$

Constraint-aware scoring. For each candidate action $a \in \mathcal{A}_{\text{cand}}$, we compute

$$v(a) = \left[\max_{q \in \mathcal{Q}_a} s_{\text{rel}}(a, q) \right] + \lambda \cdot \phi(a \mid s_t, g_t), \quad (7)$$

where $\mathcal{Q}_a$ denotes the subset of retrieval queries that retrieved action a , $s_{\text{rel}}(a, q)$ denotes the semantic relevance score between a and q , and λ controls the contribution of feasibility coverage. Setting $\lambda = 0$ reduces this to relevance-only retrieval.

The feasibility coverage term $\phi(a \mid s_t, g_t) \in [0, 1]$ measures how well the parameters of a are grounded under the current state. Let $\mathcal{P}_{\text{req}}(a)$ denote the required parameters of a (optional parameters are excluded). We define

$$\phi(a \mid s_t, g_t) = \frac{1}{|\mathcal{P}_{\text{req}}(a)|} \sum_{p \in \mathcal{P}_{\text{req}}(a)} c(p, s_t), \quad (8)$$

where $c(p, s_t) \in [0, 1]$ scores the grounding of each parameter p . A parameter receives full coverage when its value is directly available in s_t or the user goal, partial coverage when it can be supplied by

a grounded state variable or a preceding action output, and zero otherwise. Coverage values and matching thresholds are reported in Appendix B.

Feasible action set construction. The feasible action set is obtained by ranking the candidate pool by $v(a)$:

$$\mathcal{A}_{\text{feasible}} = \text{Top-}K(\mathcal{A}_{\text{cand}}, v), \quad (9)$$

where K is a budget on the number of actions exposed to trajectory construction. $\mathcal{A}_{\text{feasible}}$ defines a reduced action space over which Action-of-Thought (§3.4) constructs the execution trajectory.

Fallback under parse failure. Constraint-Aware Retrieval consumes the structured CI output. When the CI output cannot be parsed, ActionRAG sets $\lambda = 0$ and ranks candidates by semantic relevance alone, and Action-of-Thought receives the candidate pool without C_{env} and C_{task} . The pipeline therefore degrades to relevance-only retrieval rather than terminating.

3.4 Action-of-Thought

Constraint-Aware Retrieval yields a feasible action set $\mathcal{A}_{\text{feasible}}$ but does not specify how the actions should be composed. In multi-step execution, parameter dependencies and ordering constraints

across actions still need to be resolved. *Action-of-Thought (AoT)* constructs an executable trajectory over $\mathcal{A}_{\text{feasible}}$ under the constraints induced by CI.

Dependency-aware trajectory construction. AoT produces a trajectory

$$\pi = (a_1, a_2, \dots, a_T) \quad (10)$$

by selecting and ordering actions from $\mathcal{A}_{\text{feasible}}$ so that each step’s parameter requirements can be supplied by the user goal, the environment state, or the outputs of preceding actions. When a required parameter is unavailable, AoT inserts an intermediate action that produces it. AoT *targets* trajectories that satisfy the per-step feasibility property

$$C(a_t \mid s_t, g_t) = 1, \quad \forall a_t \in \pi, \quad (11)$$

and approximates this objective through structured prompting over $\mathcal{A}_{\text{feasible}}$ together with the CI-derived constraints C_{env} and C_{task} , rather than through formal constraint propagation.

Pre-execution trajectory construction. Reactive approaches interleave planning with execution, exposing the agent to dependency-incompatible action selections at every step. AoT instead commits to the full trajectory before execution, so that infeasible orderings are eliminated at planning time rather than discovered during rollout. This pre-execution structure is the basis for the efficiency gains observed in §4.3 and the failure-mode reductions reported in §4.6.

3.5 Integration with Large Action Models

ActionRAG is independent of the underlying Large Action Model (LAM). Given the trajectory $\pi = (a_1, \dots, a_T)$ produced by AoT, the LAM grounds each action’s parameters from the user goal or the outputs of preceding actions according to the action specification $\tau(a_t)$, and executes the trajectory sequentially. After execution, the LAM synthesizes the final response from the tool outputs in a single call (Appendix A.3). ActionRAG therefore issues three LLM calls per query: Constraint Induction, Action-of-Thought, and response synthesis. Because π is fixed before execution, no additional retrieval or planning is invoked during rollout. By constraining the action space and committing to a feasible trajectory before execution, ActionRAG eliminates infeasible actions and trajectories during inference, rather than discovering them through failed rollouts. Algorithm 1 summarizes the full inference procedure.

Algorithm 1 ActionRAG Inference

Require: User goal g_t , environment state s_t , action space $\mathcal{A}$, retrieval size k , feasible-set budget K

Ensure: Action trajectory π and execution outputs $\{o_t\}$

Stage 1: Constraint Induction (§3.2)

1: $\langle \mathcal{G}_{\text{sub}}, \mathcal{Q}, C_{\text{env}}, C_{\text{task}} \rangle \leftarrow \text{CI}(s_t, g_t)$

Stage 2: Constraint-Aware Retrieval (§3.3)

2: $\mathcal{A}_{\text{cand}} \leftarrow \bigcup_{q \in \mathcal{Q}} \text{RETRIEVE}(q, \mathcal{A}, k)$

3: **for all** $a \in \mathcal{A}_{\text{cand}}$ **do**

4: Compute $v(a)$ ▷ Eq. (7)

5: $\mathcal{A}_{\text{feasible}} \leftarrow \text{Top-}K(\mathcal{A}_{\text{cand}}, v)$

Stage 3: Action-of-Thought (§3.4)

6: $\pi \leftarrow \text{AoT}(g_t, \mathcal{A}_{\text{feasible}}, C_{\text{env}}, C_{\text{task}})$

Execution by LAM (standard rollout, §3.5)

7: **for** $t = 1$ **to** T **do**

8: $o_t \leftarrow \text{EXECUTE}(a_t \mid g_t, o_{1:t-1})$

9: $s_{t+1} \leftarrow \mathcal{T}(s_t, a_t)$

10: **return** $\pi, \{o_t\}_{t=1}^T$

4 Experiments

4.1 Experimental Setup

Benchmarks and models. We evaluate ActionRAG on two complementary benchmarks.¹ StableToolBench (Guo et al., 2024) contains 765 solvable queries across subsets of increasing compositional difficulty (G1–G3), covering single-tool, multi-step, and cross-category tasks. MCP-Bench (Wang et al., 2025b) evaluates tool-use proficiency across schema understanding, task completion, tool usage, and planning. Both benchmarks consist of synthetic queries over public APIs and contain no personally identifying information. We evaluate four open-source LLMs of varying scales: Llama-3.1-8B-Instruct, Llama-3.3-70B-Instruct, Qwen-2.5-7B-Instruct, and Qwen-2.5-72B-Instruct. ActionRAG is training-free and requires no fine-tuning of the underlying LAMs. All inference uses greedy decoding via vLLM (Kwon et al., 2023) on 4×NVIDIA B200 GPUs.

Baselines and evaluation. We compare against four representative baselines: CoT/ReAct (Yao et al., 2023), DFSDT (Qin et al., 2024), Smurfs (Chen et al., 2025), and ReWOO (Xu et al., 2023). All methods use the same tool inventory, tool specifications, backbone, retrieval corpus, and context budget, and retrieval-based methods share the same BM25 and BGE-M3 backend and retrieval budget; the ranking and planning strategies are the

¹Licenses of all artifacts used in this work: StableToolBench, MCP-Bench, vLLM (Apache 2.0); BAAI/bge-m3 (MIT); Llama-3.1-8B-Instruct and Llama-3.3-70B-Instruct (Llama Community License); Qwen-2.5-7B-Instruct and Qwen3-235B-A22B-Instruct-2507 (Apache 2.0); Qwen-2.5-72B-Instruct (Qwen License Agreement). All artifacts are used consistently with their intended research use.

only difference. ReWOO is excluded on MCP-Bench because its single-prompt planner exceeds the context window under the full MCP-Bench tool inventory. For StableToolBench, we report Solvable Pass Rate (SoPR) evaluated using GPT-4o as an automated judge and averaged over three runs. MCP-Bench reports two metric families. Valid Tool Name, Schema Compliance, and Execution Success are derived from execution traces against live MCP servers. Task Fulfillment, Grounding, Tool Appropriateness, Parameter Accuracy, Dependency Awareness, and Parameter Efficiency are scored by o4-mini against the MCP-Bench rubric and averaged over five runs. Appendix H reports judge–human agreement for both benchmarks.

Retrieval setup. We use hybrid retrieval combining dense embeddings (BAAI/bge-m3) and BM25 ($k_1 = 1.5$, $b = 0.75$). Scores are combined via weighted sum with $\alpha = 0.6$, and we retrieve the top- $k = 15$ tools per query. s_{rel} is min–max normalized to $[0, 1]$ within the candidate pool and $\phi \in [0, 1]$ by construction, so the two terms of Eq. (7) share a common scale. We fix λ , K , and the coverage thresholds on StableToolBench and transfer them to MCP-Bench unchanged, without benchmark-specific tuning. Appendix C gives the full retrieval setup and metric definitions, and Appendix E reports hyperparameter sensitivity.

4.2 Main Results

Table 1 reports the main results on MCP-Bench and StableToolBench. ActionRAG consistently achieves the best performance across all model scales and benchmark settings.

Consistent improvements across models. On StableToolBench, ActionRAG substantially improves SoPR over all baselines. Llama-3.1-8B-Instruct improves from 14.1% to 55.8%, and Qwen-2.5-7B-Instruct from 21.6% to 65.0%. For larger models, Qwen-2.5-72B-Instruct achieves 72.1% overall SoPR, suggesting that Constraint-Aware Retrieval improves execution reliability consistently across model scales.

Strong gains on compositional tasks. The largest gains are observed on G3_Instruction, which requires cross-category tool coordination. Qwen-2.5-72B-Instruct improves from 21.6% to 80.5%, while Llama-3.3-70B-Instruct improves from 18.2% to 73.8%. ActionRAG achieves up to a 60.7 percentage point improve-

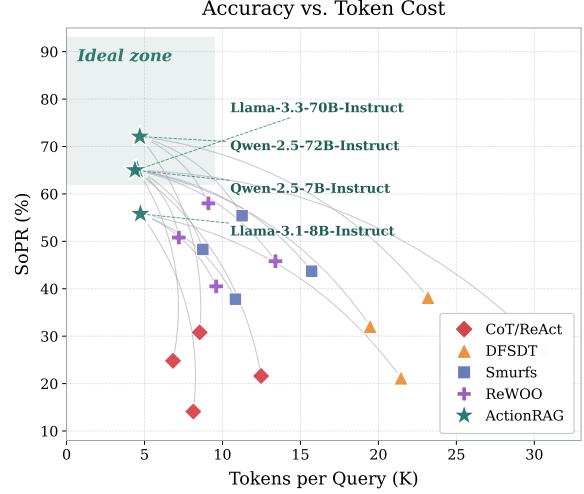


Figure 3: SoPR vs. average token cost per query across four models. The shaded region indicates the desirable regime (high accuracy, low cost). Arrows show the transition from CoT/ReAct to ActionRAG.

ment on G3_Instruction (Qwen-2.5-7B-Instruct: 8.2% to 68.9%), highlighting the importance of dependency-aware trajectory construction.

Improved planning and tool use. ActionRAG improves Tool Appropriateness, Parameter Accuracy, and Dependency Awareness across all four backbones on MCP-Bench. Smurfs reaches higher Task Fulfillment and Parameter Efficiency on Qwen-2.5-7B-Instruct. ActionRAG improves the three execution-derived dimensions across all four backbones, so the gains appear in the execution traces and not only in rubric scores. Appendix G repeats the StableToolBench protocol on a stronger backbone.

4.3 Efficiency

Figure 3 compares accuracy and token cost across methods and models. ActionRAG operates within a narrow token range (4.9–5.3K per query), while Smurfs requires 8.7–15.7K and DFSDT requires 19.5–30.6K, yielding up to $3.0\times$ and $6.2\times$ lower token cost, respectively. Despite the lower cost, ActionRAG consistently achieves higher SoPR across all model scales, indicating that Constraint-Aware Retrieval reduces unnecessary tool exploration.

Stage-level token decomposition. The reported token cost includes all LLM calls in the ActionRAG pipeline: Constraint Induction (CI), Action-of-Thought (AoT), and final response synthesis. Table 2 reports the per-stage breakdown on Qwen-2.5-7B-Instruct (averaged over 765 queries). AoT

(a) MCP-Bench

Model / Method	Overall	Valid Tool Name	Schema Compliance <i>Execution-derived</i>	Execution Success	Task Fulfillment	Grounding <i>Task Completion</i>	Tool Approp.	Param. Accuracy <i>Tool Usage</i>	Dependency Awareness	Param. Efficiency <i>Planning</i>
Llama-3.1-8B-Instruct	0.428 $\pm$.060	0.893 $\pm$.030	0.550 $\pm$.070	0.600 $\pm$.060	0.450 $\pm$.060	0.420 $\pm$.060	0.500 $\pm$.070	0.400 $\pm$.060	0.400 $\pm$.060	0.300 $\pm$.060
+DFSdT	0.465 $\pm$.050	0.901 $\pm$.030	0.580 $\pm$.060	0.620 $\pm$.060	0.480 $\pm$.060	0.460 $\pm$.060	0.530 $\pm$.060	0.420 $\pm$.060	0.420 $\pm$.060	0.320 $\pm$.060
+Smurfs	0.583 $\pm$.061	0.939 $\pm$.032	0.485 $\pm$.076	0.560 $\pm$.066	0.622 $\pm$.058	0.533 $\pm$.058	0.600 $\pm$.067	0.500 $\pm$.059	0.511 $\pm$.066	0.500 $\pm$.068
+ActionRAG	0.735 $\pm$.045	0.964 $\pm$.020	0.793 $\pm$.050	0.693 $\pm$.055	0.671 $\pm$.052	0.694 $\pm$.058	0.824 $\pm$.048	0.612 $\pm$.056	0.741 $\pm$.050	0.624 $\pm$.054
Qwen-2.5-7B-Instruct	0.461 $\pm$.055	0.908 $\pm$.028	0.572 $\pm$.065	0.615 $\pm$.058	0.478 $\pm$.058	0.445 $\pm$.058	0.524 $\pm$.065	0.420 $\pm$.058	0.421 $\pm$.058	0.322 $\pm$.058
+DFSdT	0.496 $\pm$.050	0.917 $\pm$.028	0.602 $\pm$.060	0.638 $\pm$.058	0.506 $\pm$.056	0.482 $\pm$.058	0.554 $\pm$.058	0.448 $\pm$.058	0.447 $\pm$.058	0.345 $\pm$.058
+Smurfs	0.684 $\pm$.050	0.921 $\pm$.030	0.610 $\pm$.065	0.636 $\pm$.060	0.744 $\pm$.045	0.600 $\pm$.055	0.778 $\pm$.048	0.644 $\pm$.058	0.622 $\pm$.060	0.600 $\pm$.062
+ActionRAG	0.748 $\pm$.045	0.922 $\pm$.030	0.921 $\pm$.035	0.685 $\pm$.055	0.700 $\pm$.050	0.662 $\pm$.055	0.863 $\pm$.045	0.775 $\pm$.050	0.650 $\pm$.055	0.550 $\pm$.060
Llama-3.3-70B-Instruct	0.558 $\pm$.050	0.961 $\pm$.022	0.694 $\pm$.058	0.726 $\pm$.052	0.587 $\pm$.052	0.555 $\pm$.055	0.628 $\pm$.055	0.520 $\pm$.052	0.508 $\pm$.052	0.385 $\pm$.052
+DFSdT	0.594 $\pm$.045	0.966 $\pm$.020	0.722 $\pm$.055	0.748 $\pm$.050	0.615 $\pm$.050	0.586 $\pm$.052	0.658 $\pm$.052	0.546 $\pm$.050	0.534 $\pm$.050	0.408 $\pm$.050
+Smurfs	0.736 $\pm$.045	0.980 $\pm$.020	0.624 $\pm$.060	0.750 $\pm$.050	0.733 $\pm$.050	0.722 $\pm$.062	0.911 $\pm$.040	0.622 $\pm$.060	0.656 $\pm$.055	0.622 $\pm$.058
+ActionRAG	0.835 $\pm$.040	1.000 $\pm$.000	0.815 $\pm$.045	0.824 $\pm$.040	0.789 $\pm$.050	1.000 $\pm$.000	0.967 $\pm$.030	0.822 $\pm$.045	0.711 $\pm$.055	0.633 $\pm$.050
Qwen-2.5-72B-Instruct	0.571 $\pm$.048	0.958 $\pm$.022	0.706 $\pm$.056	0.738 $\pm$.050	0.598 $\pm$.050	0.568 $\pm$.053	0.640 $\pm$.053	0.531 $\pm$.050	0.520 $\pm$.050	0.398 $\pm$.050
+DFSdT	0.608 $\pm$.045	0.964 $\pm$.020	0.732 $\pm$.052	0.760 $\pm$.048	0.628 $\pm$.048	0.600 $\pm$.050	0.670 $\pm$.050	0.558 $\pm$.048	0.547 $\pm$.048	0.422 $\pm$.048
+Smurfs	0.748 $\pm$.040	0.992 $\pm$.020	0.638 $\pm$.060	0.762 $\pm$.050	0.748 $\pm$.050	0.736 $\pm$.060	0.918 $\pm$.040	0.638 $\pm$.060	0.671 $\pm$.050	0.638 $\pm$.060
+ActionRAG	0.845 $\pm$.040	1.000 $\pm$.000	0.831 $\pm$.050	0.842 $\pm$.040	0.812 $\pm$.050	1.000 $\pm$.000	0.975 $\pm$.030	0.842 $\pm$.050	0.726 $\pm$.050	0.652 $\pm$.050

(b) StableToolBench

Model / Method	SoPR (%)	G1_Instruction	G1_Category	G1_Tool	G2_Instruction	G2_Category	G3_Instruction
Llama-3.1-8B-Instruct	14.1 $\pm$ 1.3	14.7 $\pm$ 2.8	16.3 $\pm$ 3.0	17.1 $\pm$ 3.0	11.3 $\pm$ 3.1	18.5 $\pm$ 3.5	6.6 $\pm$ 3.2
+DFSdT	21.2 $\pm$ 1.5	21.5 $\pm$ 3.2	24.2 $\pm$ 3.5	24.7 $\pm$ 3.4	17.9 $\pm$ 3.7	25.8 $\pm$ 3.9	13.1 $\pm$ 4.3
+Smurfs	37.8 $\pm$ 1.7	39.1 $\pm$ 3.8	38.5 $\pm$ 3.9	37.9 $\pm$ 3.9	34.8 $\pm$ 4.6	37.5 $\pm$ 4.3	39.0 $\pm$ 6.2
+ReWOO	40.5 $\pm$ 1.8	41.8 $\pm$ 3.9	42.3 $\pm$ 3.9	41.0 $\pm$ 3.9	38.9 $\pm$ 4.7	40.1 $\pm$ 4.4	38.2 $\pm$ 6.2
+ActionRAG	55.8 $\pm$ 1.8	56.4 $\pm$ 3.9	52.3 $\pm$ 4.0	58.2 $\pm$ 3.8	57.6 $\pm$ 4.8	50.0 $\pm$ 4.7	65.6 $\pm$ 6.1
Qwen-2.5-7B-Instruct	21.6 $\pm$ 1.5	20.2 $\pm$ 3.1	24.8 $\pm$ 3.5	24.1 $\pm$ 3.4	15.1 $\pm$ 3.5	28.2 $\pm$ 4.0	8.2 $\pm$ 3.5
+DFSdT	27.3 $\pm$ 1.6	28.2 $\pm$ 3.5	32.7 $\pm$ 3.8	31.6 $\pm$ 3.7	22.6 $\pm$ 4.1	33.9 $\pm$ 4.2	14.8 $\pm$ 4.5
+Smurfs	43.7 $\pm$ 1.8	45.2 $\pm$ 3.9	43.8 $\pm$ 4.0	44.8 $\pm$ 4.0	40.5 $\pm$ 4.8	42.4 $\pm$ 4.4	45.5 $\pm$ 6.4
+ReWOO	45.8 $\pm$ 1.8	47.5 $\pm$ 3.9	47.2 $\pm$ 4.0	46.9 $\pm$ 4.0	43.8 $\pm$ 4.8	45.0 $\pm$ 4.4	40.3 $\pm$ 6.3
+ActionRAG	65.0 $\pm$ 1.8	63.2 $\pm$ 3.8	63.4 $\pm$ 3.9	65.8 $\pm$ 3.9	72.6 $\pm$ 4.4	59.7 $\pm$ 4.5	68.9 $\pm$ 6.1
Llama-3.3-70B-Instruct	24.8 $\pm$ 1.6	23.5 $\pm$ 3.3	27.2 $\pm$ 3.6	28.6 $\pm$ 3.6	22.7 $\pm$ 4.1	28.5 $\pm$ 4.0	18.2 $\pm$ 4.9
+DFSdT	32.1 $\pm$ 1.7	31.0 $\pm$ 3.6	35.3 $\pm$ 3.9	36.4 $\pm$ 3.8	29.8 $\pm$ 4.4	36.4 $\pm$ 4.3	23.7 $\pm$ 5.4
+Smurfs	48.3 $\pm$ 1.8	47.1 $\pm$ 3.9	49.5 $\pm$ 4.0	51.2 $\pm$ 4.0	45.5 $\pm$ 4.8	50.0 $\pm$ 4.5	46.4 $\pm$ 6.4
+ReWOO	50.8 $\pm$ 1.8	49.7 $\pm$ 3.9	52.6 $\pm$ 4.0	54.0 $\pm$ 4.0	48.5 $\pm$ 4.8	52.5 $\pm$ 4.5	45.5 $\pm$ 6.4
+ActionRAG	65.1 $\pm$ 1.7	65.0 $\pm$ 3.8	66.0 $\pm$ 3.8	65.8 $\pm$ 3.9	68.9 $\pm$ 4.4	55.7 $\pm$ 4.5	73.8 $\pm$ 5.6
Qwen-2.5-72B-Instruct	30.8 $\pm$ 1.7	29.6 $\pm$ 3.6	34.5 $\pm$ 3.8	35.1 $\pm$ 3.8	28.4 $\pm$ 4.4	35.6 $\pm$ 4.3	21.6 $\pm$ 5.3
+DFSdT	38.2 $\pm$ 1.8	37.5 $\pm$ 3.8	41.6 $\pm$ 3.9	42.4 $\pm$ 3.9	35.0 $\pm$ 4.6	42.0 $\pm$ 4.4	28.2 $\pm$ 5.8
+Smurfs	55.4 $\pm$ 1.8	56.0 $\pm$ 3.9	55.1 $\pm$ 4.0	56.8 $\pm$ 3.9	52.8 $\pm$ 4.8	54.4 $\pm$ 4.5	57.2 $\pm$ 6.3
+ReWOO	58.0 $\pm$ 1.8	58.3 $\pm$ 3.9	59.6 $\pm$ 3.9	60.2 $\pm$ 3.9	56.4 $\pm$ 4.8	58.2 $\pm$ 4.4	53.6 $\pm$ 6.4
+ActionRAG	72.1 $\pm$ 1.6	70.2 $\pm$ 3.6	71.0 $\pm$ 3.6	72.6 $\pm$ 3.6	74.5 $\pm$ 4.2	63.1 $\pm$ 4.3	80.5 $\pm$ 5.0

(a) Valid Tool Name, Schema Compliance, and Execution Success are derived from execution traces; the remaining six dimensions are judged by o4-mini (mean over 5 runs). ReWOO is excluded on MCP-Bench (§4.1). (b) G3_Instruction requires cross-category tool coordination (most challenging). SoPR = avg. Solvable Pass Rate; mean $\pm$ std over 3 runs judged by GPT-4o. **Bold** = best per column within each model block. First row per block = CoT/ReAct baseline.

Table 1: Main results on MCP-Bench (a) and StableToolBench (b). ActionRAG consistently achieves the best performance across all model scales, with the largest gains on cross-category compositional tasks requiring multi-tool coordination.

contributes the largest share of both tokens (45.8%) and latency (11.4s), followed by CI (33.3%, 6.8s) and synthesis (20.9%, 4.2s). Tool execution adds 5.5s without invoking the LLM. Input tokens dominate output tokens by $4.9\times$ (4,080 vs. 838 per query), so prompt-time tool specifications and constraints are the main contributors to token cost.

Wall-clock latency. ActionRAG reduces mean end-to-end latency by 70.4% over CoT/ReAct and by 53.7% over ReWOO (Table 3). ActionRAG issues three LLM calls per query regardless of task difficulty, while CoT/ReAct issues 3 to 11 calls depending on how many steps fail during rollout. The fixed call count also bounds the tail: ActionRAG

reaches a p95 latency of 38.3s, against 174.6s for CoT/ReAct and 189.5s for ReWOO. ReWOO issues two calls but plans over the full tool inventory in one prompt, which produces its long tail. Each ablation removes one LLM call and lowers mean latency to 19.4s and 17.8s, so the third call costs 8.5s and 10.1s respectively.

4.4 Constraint-Aware Retrieval Quality

Table 4 evaluates retrieval and constraint induction quality independently of end-to-end execution. Sub-table (a) reports cross-domain retrieval performance over candidate pools containing gold tools and random distractors, and trajectory tool recall, which measures whether the tools appearing in suc-

Stage	Input	Output	Total	%	Time
CI	1,435	203	1,637	33.3	6.8s
AoT	1,929	324	2,252	45.8	11.4s
Tool execution	—	—	—	—	5.5s
Synthesis	717	312	1,029	20.9	4.2s
Pipeline	4,080	838	4,918	100.0	27.9s

Table 2: Per-stage token and latency breakdown of ActionRAG on Qwen-2.5-7B-Instruct (mean over 765 queries on StableToolBench). Tool execution contributes latency but no LLM tokens. Columns are per-query means and may not sum exactly because each entry is rounded independently.

Method	Calls	Mean	Median	p95
CoT/ReAct	6.4	94.1s	79.4s	174.6s
DFSDT	14.2	212.7s	188.5s	402.3s
Smurfs	9.8	141.3s	129.8s	268.4s
ReWOO	2	60.3s	19.5s	189.5s
AoT only (w/o CI)	2	19.4s	18.6s	27.1s
CI only (w/o AoT)	2	17.8s	17.1s	25.3s
ActionRAG (full)	3	27.9s	26.9s	38.3s

Table 3: End-to-end latency on StableToolBench (Qwen-2.5-7B-Instruct, 765 queries). Calls denotes LLM calls per query; CoT/ReAct, DFSDT, and Smurfs vary with the number of failed steps, while ReWOO and ActionRAG are fixed. Each ActionRAG ablation removes one LLM call.

Successful execution trajectories are recovered within the top-10 candidates. Constraint-Aware Retrieval improves tool recall over BM25 and dense retrieval baselines at both R@5 and R@10. ActionRAG also achieves the highest trajectory tool recall among end-to-end methods, so its gains stem from retrieval as well as generation. Sub-table (b) evaluates Constraint Induction quality against a vanilla prompting baseline. CEQ-Pr/Rec/F1 measure whether induced constraints recover the relevant tool set, while PGR evaluates whether extracted parameters are grounded in the user query. Larger models generally achieve stronger constraint induction quality, and all models substantially outperform vanilla prompting. CEQ-F1 ranges from 0.487 to 0.712 across backbones while the end-to-end gain over CoT/ReAct stays within 40.3–43.4%p, so ActionRAG remains effective under variation in constraint induction quality.

4.5 Robustness to Task Complexity

The gap between ActionRAG and the baselines widens with task difficulty (Figure 4). Baselines lose 23–61% of their Easy-task SoPR on

(a) Retrieval Quality			
Retriever / Method	R@5↑	R@10↑	G3 R@10↑
<i>Retriever, cross-domain candidate pool</i>			
Random	0.071	0.142	0.156
BM25	0.578	0.688	0.699
Dense (BGE-M3)	0.901	0.972	0.945
Constraint-Aware Retrieval	0.918	0.985	0.950
<i>End-to-end method, successful trajectories</i>			
CoT/ReAct	—	0.819	0.748
DFSDT	—	0.861	0.781
Smurfs	—	0.887	0.812
ReWOO	—	0.901	0.824
ActionRAG	—	0.925	0.843

(b) Constraint Induction Quality				
Model	CEQ-Pr↑	CEQ-Rec↑	CEQ-F1↑	PGR↑
Vanilla prompting	0.182	0.385	0.247	0.358
Llama-3.1-8B-Instruct	0.532	0.450	0.487	0.637
Qwen-2.5-7B-Instruct	0.550	0.481	0.512	0.639
Llama-3.3-70B-Instruct	0.689	0.682	0.685	0.782
Qwen-2.5-72B-Instruct	0.714	0.710	0.712	0.798

Table 4: Retrieval and constraint induction quality on StableToolBench. Retriever rows measure R@5 and R@10 over cross-domain candidate pools containing gold tools and random distractors; method rows measure recall of the tools used in successful execution trajectories. Metric definitions are given in Appendix C.2.

Hard tasks, while ActionRAG loses less than 10%. Constraint-Aware Retrieval reduces cascading errors in long-horizon multi-tool execution.

4.6 Failure Mode Analysis

We analyze structural failure modes in CoT/ReAct’s failed queries (Qwen-2.5-7B-Instruct). Three modes—stuck loops (18.6%), under-exploration (28.4%), and over-exploration (13.2%)—account for 60.2% of failures. ActionRAG reduces each mode by 77–99% and lowers the total structural failure rate to 5.0% through structured tool filtering and pre-planned trajectories (Table 14, Appendix I.2).

Where ActionRAG’s own failures originate. Retrieval is rarely the bottleneck for ActionRAG. Across 268 failed queries on Qwen-2.5-7B-Instruct, every gold tool appears in the candidate pool before Constraint-Aware Retrieval, which removes a gold tool in 2 cases. Attributing each failure to its earliest failing stage (Appendix I.2), synthesis accounts for the largest share at 43.3%: every gold tool executes and the final answer is still incorrect. Planning accounts for 27.2% overall and 63% on G3_Instruction, where trajectory length is longest.

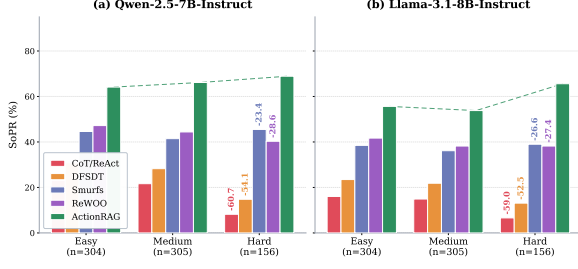


Figure 4: ActionRAG loses the least SoPR as task difficulty increases. Tasks are stratified as Easy (≤ 2 steps and ≤ 4 tools) and Hard (≥ 3 steps and ≥ 5 tools). Numbers above the Hard bars report the relative SoPR reduction from Easy to Hard, $(\text{SoPR}_{\text{Easy}} - \text{SoPR}_{\text{Hard}}) / \text{SoPR}_{\text{Easy}}$, for CoT/ReAct, DFSDT, Smurfs, and ReWOO in that order; ActionRAG’s reduction appears above its own bar. Baselines lose 23–61% of their Easy-task SoPR on Hard tasks, while ActionRAG stays within 10%.

Method	Llama-3.1-8B-Instruct	Qwen-2.5-7B-Instruct	Llama-3.1-70B-Instruct	Qwen-2.5-72B-Instruct
CoT/ReAct	14.1 $\pm$ 1.3	21.6 $\pm$ 1.5	24.8 $\pm$ 1.6	30.8 $\pm$ 1.7
DFSDT	21.2 $\pm$ 1.5	27.3 $\pm$ 1.6	32.1 $\pm$ 1.7	38.2 $\pm$ 1.8
Smurfs	37.8 $\pm$ 1.7	43.7 $\pm$ 1.8	48.3 $\pm$ 1.8	55.4 $\pm$ 1.8
ReWOO	40.5 $\pm$ 1.8	45.8 $\pm$ 1.8	50.8 $\pm$ 1.8	58.0 $\pm$ 1.8
AoT only (w/o CI)	47.5 $\pm$ 1.8	58.4 $\pm$ 1.8	58.3 $\pm$ 1.7	65.4 $\pm$ 1.7
CI only (w/o AoT)	50.2 $\pm$ 1.8	60.7 $\pm$ 1.8	60.8 $\pm$ 1.7	67.8 $\pm$ 1.7
ActionRAG (full)	55.8$\pm$1.8	65.0$\pm$1.8	65.1$\pm$1.7	72.1$\pm$1.6

Table 5: Ablation study on StableToolBench (SoPR %). CI = Constraint Induction; AoT = Action-of-Thought.

4.7 Ablation Study

Table 5 presents ablation results where Constraint Induction (CI) and Action-of-Thought (AoT) are removed independently. CI-only consistently outperforms AoT-only across all model scales, indicating that action-space reduction is the primary contributor to ActionRAG’s gains. The full pipeline further improves performance over CI-only, suggesting that AoT provides additional robustness by organizing filtered tools into feasible execution trajectories. Appendix D separates the internal components of CI and reports the contribution of each.

4.8 Effect of Feasibility Coverage

Feasibility coverage contributes unevenly across subsets (Table 6). Moving from $\lambda = 0$ to the default $\lambda = 0.05$ raises SoPR by 14.8%p on G3_Instruction and 5.5%p on G1_Instruction, and lowers SoPR by 7.8%p on G1_Category and 4.0%p on G2_Category. ϕ contributes where trajectories chain parameters across actions. G3_Instruction requires cross-category coordination, where a candidate whose required parameters cannot be

λ	SoPR	G1-I	G1-C	G1-T	G2-I	G2-C	G3-I
0.00	64.3 $\pm$ 1.7	57.7 $\pm$ 3.9	71.2 $\pm$ 3.6	65.8 $\pm$ 4.0	68.9 $\pm$ 4.6	63.7 $\pm$ 4.4	54.1 $\pm$ 6.4
0.05	65.0$\pm$1.8	63.2$\pm$3.8	63.4$\pm$3.9	65.8$\pm$3.9	72.6$\pm$4.4	59.7$\pm$4.5	68.9$\pm$6.1
0.10	63.9 $\pm$ 1.9	61.3 $\pm$ 3.7	64.1 $\pm$ 4.1	65.2 $\pm$ 4.0	71.7 $\pm$ 4.5	58.9 $\pm$ 4.3	63.9 $\pm$ 6.3
0.20	62.8 $\pm$ 1.8	59.8 $\pm$ 4.0	64.8 $\pm$ 3.8	64.4 $\pm$ 4.1	70.8 $\pm$ 4.7	57.3 $\pm$ 4.6	59.0 $\pm$ 6.5
Δ	+0.7	+5.5	-7.8	0.0	+3.7	-4.0	+14.8

Table 6: Feasibility weight λ over all StableToolBench subsets (Qwen-2.5-7B-Instruct, SoPR %, mean $\pm$ std over 3 runs). CI outputs and candidate pools are held fixed; only λ varies. $\lambda = 0$ disables feasibility coverage. The $\lambda = 0.05$ row is the paper default and matches Table 1(b). Δ is the change from $\lambda = 0$ to the default.

grounded is unusable regardless of its description match. G1_Category and G2_Category select a single tool from a semantically homogeneous pool, where relevance already separates the gold tool and the coverage term only reorders near-ties. The benchmark-level gain is therefore +0.7%p while the subset-level gain reaches +14.8%p, and a larger weight trades the compositional gain for the single-tool regression (62.8% at $\lambda = 0.2$). The same pattern holds on Llama-3.1-8B-Instruct (Appendix F).

5 Conclusion

We proposed **ActionRAG**, a training-free framework that treats tool-use planning as a feasibility-first problem: Constraint-Aware Retrieval filters infeasible actions before execution and Action-of-Thought constructs feasible trajectories over what remains. Experiments on StableToolBench and MCP-Bench across four backbones show consistent gains at every scale, with the largest improvements on compositional multi-tool tasks and lower token cost and latency than competing planners. ActionRAG composes LLM-prompted constraint induction, hybrid retrieval, and pre-execution trajectory construction; the feasibility coverage term is its new element, and that term contributes most where trajectories chain parameters across actions. Structuring the action space before generation offers a training-free path to reducing infeasible execution as tool ecosystems scale.

Limitations

We evaluate ActionRAG on StableToolBench and MCP-Bench, both controlled text-based environments; performance under dynamic APIs, changing tool availability, or non-textual action spaces (e.g., GUI or robotics) remains to be validated. MCP-Bench executes tool calls against live MCP servers, and Valid Tool Name, Schema Compliance, and

Execution Success are derived from those execution traces, which provides partial closed-loop evidence. Tools with side effects, authentication requirements, and rate limits fall outside both benchmarks, and evaluation under irreversible actions remains future work. Our constraint induction evaluation uses benchmark-provided relevant API sets as proxy ground truth, and because multiple feasible tool trajectories may exist for a single query, these metrics may not capture all valid constraint structures. Finally, ActionRAG follows a predefined planning pipeline without execution-time feedback or adaptive replanning, and extending the framework to closed-loop planning remains future work.

Ethical Considerations

ActionRAG is evaluated using existing public tool-use benchmarks (StableToolBench and MCP-Bench) and pre-trained open-source language models. The proposed framework does not involve collection of new human data and does not require interaction with real-world external systems during evaluation. ActionRAG is designed to improve action selection efficiency by filtering infeasible actions before trajectory construction, which may reduce unnecessary tool exploration and inference overhead. However, this work does not evaluate deployment-time safety or security properties. A system that selects executable actions more reliably also increases the consequence of an incorrect selection, so permission boundaries and risk checks on irreversible actions are required for real-world deployment involving external tool execution.

Acknowledgments

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) under the Artificial Intelligence Convergence Innovation Human Resources Development (IITP-2026-RS-2023-00254592) grant funded by the Korea government (MSIT). This research was supported by the ANCHOR program through the Seoul ANCHOR Center, funded by the Ministry of Education (MOE) and the Seoul Metropolitan Government, Republic of Korea (2026-ANCHOR-01-007-04). This work was supported by the Dongguk University Research Fund of 2026.

References

- Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F. Karlsson, Jie Fu, and Yemin Shi. 2024. [AutoAgents: A Framework for Automatic Agent Generation](#). In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pages 22–30.
- Junzhi Chen, Juhao Liang, and Benyou Wang. 2025. [Smurfs: Multi-agent system using context-efficient DFSDT for tool planning](#). 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)*, pages 3281–3298, Albuquerque, New Mexico. Association for Computational Linguistics.
- Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *Transactions on Machine Learning Research*.
- Yein Choi, Sungwoo Kim, Yipene Cedric Francois Bassole, and Yunsick Sung. 2025. Enhanced Retrieval-Augmented Generation Using Low-Rank Adaptation. *Applied Sciences*, 15(8):4425.
- Abul Ehtesham, Aditi Singh, Gaurav Kumar Gupta, and Saket Kumar. 2025. A Survey of Agent Interoperability Protocols: Model Context Protocol (MCP), Agent Communication Protocol (ACP), Agent-to-Agent Protocol (A2A), and Agent Network Protocol (ANP). *arXiv preprint arXiv:2505.02279*.
- Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. [StableToolBench: Towards stable large-scale benchmarking on tool learning of large language models](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11143–11156, Bangkok, Thailand. Association for Computational Linguistics.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *International Conference on Learning Representations*.
- Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2026. [Model Context Protocol \(MCP\): Landscape, Security Threats, and Future Research Directions](#). *ACM Transactions on Software Engineering and Methodology*. Just Accepted; online 16 February 2026.
- Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. In *Proceedings of the 39th International Conference on Machine Learning*, volume

- 162 of *Proceedings of Machine Learning Research*, pages 9118–9147. PMLR.
- Jiabao Ji, Yongchao Chen, Yang Zhang, Ramana Rao Kompella, Chuchu Fan, Gaowen Liu, and Shiyu Chang. 2025. Collision- and Reachability-Aware Multi-Robot Control with Grounded LLM Planners. *arXiv preprint arXiv:2505.20573*.
- Donghyeon Kim, Minki Park, and Yunsick Sung. 2025. Enhanced Semantic Retrieval with Structured Prompt and Dimensionality Reduction for Big Data. *Mathematics*, 13(15):2469.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.
- Xinzhe Li. 2025. [A Review of Prominent Paradigms for LLM-Based Agents: Tool Use, Planning \(Including RAG\), and Feedback Learning](#). In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 9760–9779, Abu Dhabi, UAE. Association for Computational Linguistics.
- Ximing Lu, Sean Welleck, Peter West, Liwei Jiang, Jungo Kasai, Daniel Khoshabi, Ronan Le Bras, Lianhui Qin, Youngjae Yu, Rowan Zellers, Noah A. Smith, and Yejin Choi. 2022. [NeuroLogic A*esque Decoding: Constrained Text Generation with Lookahead Heuristics](#). In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 780–799, Seattle, United States. Association for Computational Linguistics.
- Elias Lumer, Faheem Nizar, Anmol Gulati, Pradeep Honaganahalli Basavaraju, and Vamse Kumar Subbiah. 2025. Tool-to-Agent Retrieval: Bridging Tools and Agents for Scalable LLM Multi-Agent Systems. *arXiv preprint arXiv:2511.01854*.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-Refine: Iterative Refinement with Self-Feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594.
- Zhenyu Pan, Haozheng Luo, Manling Li, and Han Liu. 2025. Chain-of-Action: Faithful and Multimodal Question Answering through Large Language Models. In *International Conference on Learning Representations*.
- Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. [Gorilla: Large Language Model Connected with Massive APIs](#). In *Advances in Neural Information Processing Systems*, volume 37, pages 126544–126565.
- Aske Plaat, Max J. van Duijn, Niki van Stein, Mike Preuss, Peter van der Putten, and Kees Joost Batenburg. 2025. [Agentic Large Language Models, a Survey](#). *Journal of Artificial Intelligence Research*, 84. Article 29.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-World APIs. In *International Conference on Learning Representations*.
- Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2025. Tool Learning with Large Language Models: A Survey. *Frontiers of Computer Science*, 19(8):198343.
- Partha Pratim Ray. 2025. [A Survey on Model Context Protocol: Architecture, State-of-the-Art, Challenges and Future Directions](#). *TechRxiv*. Posted 18 April 2025.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems*, volume 36, pages 68539–68551.
- Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talei Khoei. 2025. [A Survey of the Model Context Protocol \(MCP\): Standardizing Context to Enhance Large Language Models \(LLMs\)](#). *Preprints.org*. Posted 2 April 2025.
- Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. 2023. On the Planning Abilities of Large Language Models: A Critical Investigation. In *Advances in Neural Information Processing Systems*, volume 36, pages 75993–76005.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024. Voyager: An Open-Ended Embodied Agent with Large Language Models. *Transactions on Machine Learning Research*.
- Lu Wang, Fangkai Yang, Chaoyun Zhang, Juntao Lu, Jiaxu Qian, Shilin He, Pu Zhao, Bo Qiao, He Huang, Si Qin, Qisheng Su, Jiayi Ye, Yudi Zhang, Jianguang Lou, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Qi Zhang. 2025a. Large Action Models: From Inception to Implementation. *Transactions on Machine Learning Research*.
- Zhenting Wang, Qi Chang, Hemani Patel, Shashank Biju, Cheng-En Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Shah, Yujia Bao, and Eugene Siow. 2025b. MCP-Bench: Benchmarking Tool-Using LLM Agents with Complex Real-World Tasks via MCP Servers. *arXiv preprint arXiv:2508.20453*.

Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. 2023. ReWOO: Decoupling Reasoning from Observations for Efficient Augmented Language Models. *arXiv preprint arXiv:2305.18323*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations*.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Zhiwei Liu, Yihao Feng, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. 2025. xLAM: A family of large action models to empower AI agent systems. 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)*, pages 11583–11597, Albuquerque, New Mexico. Association for Computational Linguistics.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *International Conference on Learning Representations*.

A ActionRAG Prompt Templates

ActionRAG uses a three-stage pipeline: (1) Constraint Induction (CI), (2) Action-of-Thought (AoT), and (3) response synthesis by the Large Action Model. Each stage issues one LLM call, matching the per-stage token breakdown in Table 2. We provide the prompt templates used in our final experiments below.

A.1 Stage 1: Constraint Induction

Constraint Induction (CI) produces structured environment ($\mathcal{C}_{env}$) and task ($\mathcal{C}_{task}$) constraints that guide retrieval and downstream planning.

Stage 1 — Constraint Induction

You are a constraint analysis engine for
 $\hookrightarrow$ tool-calling tasks.

User Query
 {query}

Available Tools
 {tools_json}

YOUR TASK

Analyze the query to produce structured
 $\hookrightarrow$ constraints
 that will guide a downstream planner.

Step 1: Sub-goal Decomposition
 Break the user query into independent sub-
 $\hookrightarrow$ goals.

Step 2: Environment Constraints ($\mathcal{C}_{env}$)
 - chaining_requirements: Which tool outputs
 $\hookrightarrow$ feed
 into which tool inputs?
 - api_notes: Important observations (auth,
 $\hookrightarrow$ format,
 rate limits)

Step 3: Task Constraints ($\mathcal{C}_{task}$)
 - required_info: What the answer must
 $\hookrightarrow$ contain
 - extractable_params: Parameters directly
 $\hookrightarrow$ extractable
 from the query
 - sub_goal_dependencies: Which sub-goals
 $\hookrightarrow$ depend on
 others?

Output Format (JSON only)
 {
 "sub_goals": [...],
 "C_env": {
 "chaining_requirements": [
 {"source_tool":..., "provides":...,
 "needed_by":..., "for_param":...}
],
 "api_notes": [...]
 },
 "C_task": {
 "required_info": [...],
 "extractable_params": {...},
 "sub_goal_dependencies": [...]
 }
 }

A.2 Stage 2: Action-of-Thought

AoT receives the CI output together with tools ranked by $v(a) = [\max_{q \in \mathcal{Q}_a} s_{rel}(a, q)] + \lambda \phi(a | s_t, g_t)$ and generates executable action trajectories.

Stage 2 — Action-of-Thought

You are a constraint-aware planning engine
 $\hookrightarrow$ that
 produces executable action plans.

You receive:

1. A user goal
2. Available tools ranked by
 $v(a) = s_{rel} + \lambda \phi(a)$
3. Constraint Analysis:

```

sub-goals, C_env, C_task

## Planning Process
1. Map each sub-goal to appropriate tools
2. Respect chaining requirements in C_env
3. Fill parameters from
   C_task.extractable_params
4. Use outputs from previous tools when
   ↪ needed
5. Verify all sub-goals are covered

## Output format
EXECUTION PLAN:
[
  {"name": "tool_function_name",
   "arguments": {"param1": "value1"}},
  {"name": "another_tool",
   "arguments": {"id": " "
   ↪ FROM_PREVIOUS_RESULT"}}
]

```

A.3 Stage 3: Response Synthesis

The Large Action Model executes the trajectory produced by AoT and synthesizes the final response from the tool outputs. This stage invokes no retrieval or planning component.

Stage 3 — Response Synthesis

```

You are an AI assistant that synthesizes
  ↪ API call
results into a helpful final answer.

## User Query
{query}

## Tool Call Results
{tool_results}

## Your Task
Based on the API results above, provide a
  ↪ concise and
helpful answer to the user's query.

Rules:
- Use information from all successful API
  ↪ calls
- If some calls failed, use the successful
  ↪ results
- Include relevant details from the results
- If no useful result was obtained, state
  ↪ this clearly

Provide your answer:

```

B Constraint-Aware Retrieval

Constraint-Aware Retrieval scores each candidate as

$$v(a) = \left[\max_{q \in \mathcal{Q}_a} s_{\text{rel}}(a, q) \right] + \lambda \cdot \phi(a \mid s_t, g_t), \quad (12)$$

where $\mathcal{Q}_a$ denotes the subset of retrieval queries that retrieved action a , and

$$\phi(a \mid s_t, g_t) = \frac{1}{|\mathcal{P}_{\text{req}}(a)|} \sum_{p \in \mathcal{P}_{\text{req}}(a)} c(p, s_t). \quad (13)$$

Here, s_t denotes the currently grounded state variables, and $\mathcal{P}_{\text{req}}(a)$ denotes the required parameters of action a (optional parameters are excluded). Per-parameter coverage $c(p, s_t) \in [0, 1]$ is defined as

$$c(p, s_t) = \begin{cases} 1 & p \in s_t, \\ \gamma_s & p \approx_s s_t, \\ \gamma_c & p \approx_c o(a'), \\ 0 & \text{otherwise,} \end{cases} \quad (14)$$

where $\approx_s$ denotes a semantic match $\max_{e \in s_t} \cos(\psi(p), \psi(e)) > \theta_s$ to a grounded state variable, and $\approx_c$ denotes a chain match $\max_{a' \neq a} \cos(\psi(p), \psi(o(a')))) > \theta_c$ to a preceding action output. ψ is the frozen BGE-M3 sentence encoder, and $o(a')$ is the output description of action a' . Default hyperparameters are $\lambda = 0.05$, $\gamma_s = 0.7$, $\gamma_c = 0.4$, $\theta_s = \theta_c = 0.5$.

C Implementation Details

C.1 Hybrid Retrieval

For each query q , the retrieval score of action a is

$$s_{\text{rel}}(a, q) = \alpha \cdot s_{\text{dense}}(a, q) + (1 - \alpha) \cdot s_{\text{BM25}}(a, q), \quad (15)$$

where s_{dense} is the cosine similarity between BGE-M3 embeddings of the query and the tool description, and s_{BM25} is the standard BM25 score with $k_1=1.5, b=0.75$. Both scores are min-max normalized within the candidate pool before combination. We use $\alpha=0.6$ and retrieve the top $k=15$ tools per sub-goal query.

C.2 Evaluation Metrics

For each query q , the gold tool set $\mathcal{T}_q^*$ is obtained from the StableToolBench relevant_apis field (e.g., `[("vimeo", "searchvideos"), ...]`), converted to function-name form `{api_name}_for_{tool_name}`. Throughout the

metrics below, two tool names match under *fuzzy matching* if they are equal or one is a substring of the other, tolerating routine naming variations (e.g., search vs. search_videos). All metrics are computed per query and averaged over queries.

CEQ-Pr / CEQ-Rec / CEQ-F1. The Constraint Extraction Quality (CEQ) metrics evaluate whether CI’s induced constraints recover the relevant tool set. We take $\hat{\mathcal{T}}_q$ as the union of all `source_tool` and `needed_by` entries in $C_{\text{env.chaining_requirements}}$, and report the standard precision, recall, and F1 of $\hat{\mathcal{T}}_q$ against $\mathcal{T}_q^*$ under fuzzy matching.

PGR (Parameter Grounding Rate). Let $\hat{\mathcal{P}}_q$ denote the (key, value) pairs in $C_{\text{task.extractable_params}}$ with non-null values. A pair is *grounded* if its value appears in q as a substring, or contains at least one whitespace-split token of length greater than two that appears in q . PGR is the fraction of grounded pairs in $\hat{\mathcal{P}}_q$.

Trajectory Tool Recall (R@10). Let $\mathcal{T}_q^{\text{top10}}$ denote the top-10 tools returned by the retriever for query q . R@10 is the average of $|\mathcal{T}_q^* \cap \mathcal{T}_q^{\text{top10}}|/|\mathcal{T}_q^*|$ over queries. The G3 R@10 figure restricts the average to G3_Instruction queries.

D Constraint Induction Components

The ablation in Table 5 evaluates the overall contributions of Constraint Induction (CI) and Action-of-Thought (AoT). To further isolate the contribution of individual components within CI, we start from the full ActionRAG pipeline and remove each CI component independently while keeping AoT and the remaining pipeline fixed.

Configuration	SoPR (%)	Δ
Full ActionRAG	65.0 $\pm$ 1.8	—
— Task-side signal ($\mathcal{Q}$, C_{task})	56.0 $\pm$ 2.0	−9.0
— Environment constraint (C_{env})	62.1 $\pm$ 1.9	−2.9
— Sub-goal decomposition	65.0 $\pm$ 1.8	0.0

Table 7: Component-wise ablation of Constraint Induction within the full ActionRAG pipeline on StableToolBench (Qwen-2.5-7B-Instruct, all six subsets, mean $\pm$ std over 3 runs). Each row removes one CI component while keeping AoT and the remaining pipeline fixed. The Full ActionRAG row matches Table 1(b).

Among the CI components, the induced task-side signal contributes the largest performance gain (Table 7). Removing $\mathcal{Q}$ and C_{task} lowers SoPR by 9.0%p, while removing C_{env} lowers it by 2.9%p.

Removing explicit sub-goal decomposition leaves SoPR unchanged in this setting. These results indicate that the contribution of CI primarily comes from the induced task- and environment-level feasibility signals rather than from decomposition alone.

StableToolBench and MCP-Bench provide no gold C_{task} , C_{env} , parameter-flow edges, or unique feasible trajectory, and multiple valid constraint structures can solve the same query. We therefore use component-wise removal within the full pipeline as an end-to-end measure of the contribution of the induced constraint signals, rather than relying on an oracle-versus-extracted comparison.

E Hyperparameter Sensitivity

We analyze sensitivity to the five hyperparameters introduced in Appendix B (λ , K , θ_s , θ_c , and (γ_s, γ_c)) on Qwen-2.5-7B-Instruct and Llama-3.1-8B-Instruct. Each axis is varied independently with the remaining hyperparameters fixed to their defaults.

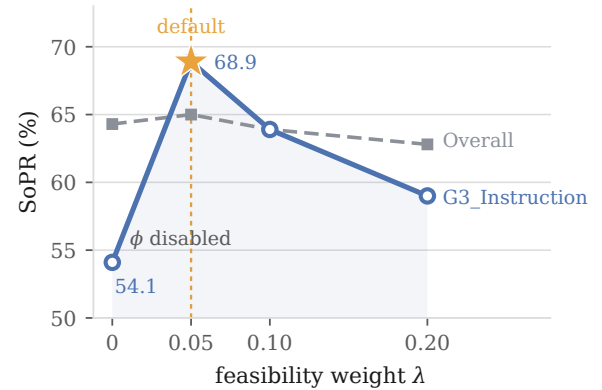


Figure 5: Feasibility weight λ on G3_Instruction and at benchmark level (Qwen-2.5-7B-Instruct, same run as Table 6). The star marks the default $\lambda=0.05$. G3_Instruction ranges over 14.8%p across the sweep while the benchmark-level curve ranges over 2.2%p.

Feasibility weight λ . Figure 5 contrasts the λ axis on G3_Instruction with the benchmark-level curve; Section 4.8 reports every subset (Table 6). Setting $\lambda=0$ lowers SoPR from 68.9% to 54.1% on G3_Instruction and from 65.0% to 64.3% at benchmark level. The gap between the two curves is the subset composition effect analyzed in Section 4.8: ϕ moves compositional trajectories and leaves single-tool retrieval close to unchanged.

Other hyperparameters. Table 8 reports the same axes on the G3_Instruction subset, where

Hyperparameter	Value	SoPR (%)
θ_s (semantic threshold)	0.3	65.6
	0.4	67.2
	0.5 (default)	68.9
	0.6	57.4
	0.7	46.0
θ_c (chain threshold)	0.3	67.2
	0.4	68.9
	0.5 (default)	68.9
	0.6	65.6
	0.7	60.7
K (retrieval budget)	3	65.6
	5	67.2
	10	68.9
	15 (default)	68.9
	20	67.2
(γ_s, γ_c) (partial coverage)	(0.5, 0.2)	62.3
	(0.6, 0.3)	67.2
	(0.7, 0.4) (default)	68.9
	(0.8, 0.5)	67.2
	(1.0, 0.7)	60.7

Table 8: Sensitivity of the remaining four hyperparameters on Qwen-2.5-7B-Instruct, G3_Instruction. Defaults are highlighted.

the effects are largest. θ_s shows the largest variation (-22.9% p at $\theta_s=0.7$): higher thresholds reject correct parameter-state alignments and reduce the coverage signal. θ_c varies by at most -8.2% p over the same range, K by $\pm 3.3\%$ p across $[3, 20]$, and (γ_s, γ_c) by at most 1.7% p within ± 0.1 of the default. SoPR stays above 65% across $K \in [3, 20]$, $\theta_s \in [0.3, 0.5]$, $\theta_c \in [0.3, 0.6]$, and (γ_s, γ_c) within ± 0.1 .

Full-benchmark sensitivity. Table 9 extends the remaining four axes to the full benchmark on both small backbones. The ranking of each axis is preserved across the two backbones, and the default configuration is the best setting on both.

F Feasibility Weight on Llama-3.1-8B-Instruct

Table 10 repeats the λ sweep of Section 4.8 on Llama-3.1-8B-Instruct under the same protocol: CI outputs and candidate pools are held fixed and only λ varies.

The subset-level pattern matches Qwen-2.5-7B-Instruct. Feasibility coverage contributes most on G3_Instruction ($+13.1\%$ p) and on G1_Instruction ($+4.9\%$ p), and is mildly negative on the two category subsets (-5.6% p and -3.6% p). The benchmark-level gain is $+3.4\%$ p.

G Evaluation with a Stronger Backbone

The four backbones of Table 1 are 7B to 72B dense instruction-tuned models. Table 11 evaluates Ac-

Hyperparameter	Value	Qwen-2.5-7B-Instruct	Llama-3.1-8B-Instruct
θ_s	0.3	62.1	53.2
	0.4	63.8	54.7
	0.5 (default)	65.0	55.8
	0.6	59.4	50.9
	0.7	52.7	44.8
θ_c	0.3	63.6	54.4
	0.4	64.8	55.6
	0.5 (default)	65.0	55.8
	0.6	63.2	54.1
	0.7	60.1	51.3
K	3	62.0	53.1
	5	63.5	54.3
	10	64.7	55.4
	15 (default)	65.0	55.8
	20	64.2	55.0
(γ_s, γ_c)	(0.5, 0.2)	60.8	52.0
	(0.6, 0.3)	63.6	54.4
	(0.7, 0.4) (default)	65.0	55.8
	(0.8, 0.5)	64.1	54.9
	(1.0, 0.7)	60.3	51.5

Table 9: Sensitivity over the full StableToolBench benchmark (overall SoPR %, 765 queries, two backbones). Defaults are highlighted and match Table 1(b).

Subset	$\lambda=0$	$\lambda=0.05$	$\lambda=0.10$	$\lambda=0.20$	Δ
G1_Instruction	51.5 $\pm$ 4.1	56.4 $\pm$ 3.9	55.0 $\pm$ 3.8	53.4 $\pm$ 4.0	+4.9
G1_Category	57.9 $\pm$ 3.9	52.3 $\pm$ 4.0	53.1 $\pm$ 4.2	53.8 $\pm$ 3.9	-5.6
G1_Tool	58.2 $\pm$ 4.0	58.2 $\pm$ 3.8	57.6 $\pm$ 4.1	56.8 $\pm$ 4.2	0.0
G2_Instruction	54.2 $\pm$ 5.0	57.6 $\pm$ 4.8	56.9 $\pm$ 4.9	55.9 $\pm$ 5.1	+3.4
G2_Category	53.6 $\pm$ 4.8	50.0 $\pm$ 4.7	49.2 $\pm$ 4.5	47.6 $\pm$ 4.6	-3.6
G3_Instruction	52.5 $\pm$ 6.6	65.6 $\pm$ 6.1	61.7 $\pm$ 6.2	56.2 $\pm$ 6.5	+13.1
Overall	52.4 $\pm$ 1.8	55.8 $\pm$ 1.8	55.1 $\pm$ 1.7	53.7 $\pm$ 1.9	+3.4

Table 10: Feasibility weight λ over all StableToolBench subsets (Llama-3.1-8B-Instruct, SoPR %, mean $\pm$ std over 3 runs). The shaded column is the paper default and matches Table 1(b). Δ is the change from $\lambda = 0$ to the default.

tionRAG with Qwen3-235B-A22B-Instruct-2507 (235B total parameters, 22B active) under the same StableToolBench protocol.

ActionRAG improves SoPR on every subset with the stronger backbone. The gain grows with compositional difficulty: $+7.2\%$ p averaged over G1, $+9.6\%$ p over G2, and $+17.5\%$ p on G3_Instruction. The smallest gain occurs on G1_Instruction ($+4.3\%$ p), the subset where relevance-only selection is already close to sufficient. Token cost stays within the range reported in Section 4.3 at 5.2K tokens per query. Backbone capability alone therefore does not remove the benefit of feasibility-aware retrieval and dependency-aware trajectory construction.

H LLM-Judge and Human Alignment

Both StableToolBench and MCP-Bench use LLM-based judges, following their respective official

Subset	CoT/ReAct	ActionRAG	Δ
G1_Instruction	76.3 $\pm$ 3.4	80.6 $\pm$ 3.1	+4.3
G1_Category	71.9 $\pm$ 3.6	83.4 $\pm$ 2.9	+11.5
G1_Tool	72.4 $\pm$ 3.5	78.3 $\pm$ 3.3	+5.9
G2_Instruction	69.5 $\pm$ 4.6	84.6 $\pm$ 3.5	+15.1
G2_Category	66.7 $\pm$ 4.3	70.7 $\pm$ 4.2	+4.0
G3_Instruction	68.3 $\pm$ 6.0	85.8 $\pm$ 4.4	+17.5
Overall	71.5$\pm$1.7	80.0$\pm$1.4	+8.5

Table 11: StableToolBench SoPR (%) with Qwen3-235B-A22B-Instruct-2507, mean $\pm$ std over 3 runs. ActionRAG uses the same prompts and hyperparameters as in Table 1, without model-specific tuning.

evaluation protocols: StableToolBench uses GPT-4o, while MCP-Bench uses o4-mini. To validate these protocols, we annotated random samples and measured agreement between human and LLM evaluations (Table 12).

Benchmark	n	Cohen’s κ	Acc.	Pearson r
STB Solved/Unsolved	120	0.81	0.91	–
MCP Task Fulfillment	60	–	0.75	0.83
MCP Grounding	60	–	0.78	0.86
MCP Tool Appropriateness	60	–	0.72	0.80
MCP Parameter Accuracy	60	–	0.76	0.84

Table 12: LLM-judge and human agreement on randomly sampled queries. STB achieves Cohen’s κ = 0.81, indicating strong agreement. For MCP-Bench rubric scores, within-1 accuracy ranges from 0.72 to 0.78 with Pearson correlations between 0.80 and 0.86.

I Supplementary Quantitative Analysis

We provide additional analyses for Qwen-2.5-7B-Instruct on StableToolBench (765 queries).

I.1 Per-Group Net Improvement

Group	n	Wins	Regr.	Net
G1_Inst	163	78 (48%)	13 (8%)	+65
G1_Cat	153	65 (42%)	11 (7%)	+54
G1_Tool	158	73 (46%)	12 (8%)	+61
G2_Inst	106	65 (61%)	4 (4%)	+61
G2_Cat	124	52 (42%)	13 (10%)	+39
G3_Inst	61	38 (62%)	1 (2%)	+37

Table 13: Per-group wins and regressions versus CoT/ReAct.

ActionRAG achieves positive net gains across all STB subsets, with the largest relative improvement on G3_Instruction and regressions remaining within 10% across all subsets (Table 13).

I.2 Failure Analysis

ActionRAG substantially reduces structural planning failures, particularly repeated-loop and under-exploration behaviors. This suggests that explicit

Failure Mode	CoT	ActionRAG
Stuck Loop ($\geq 3 \times$ same tool)	18.6%	4.2%
Under-exploration (1 tool used)	28.4%	0.3%
Over-exploration (≥ 5 calls)	13.2%	0.5%
Total structural	60.2%	5.0%

Table 14: Structural failure modes on StableToolBench (Qwen-2.5-7B-Instruct).

Earliest stage	Condition	n	%
Planning (CI/AoT)	Gold tool never invoked	73	27.2
Execution	Gold tool invoked, call failed	77	28.7
Synthesis	All calls succeeded, answer wrong	116	43.3
Pipeline	Runtime failure	2	0.8
Total		268	100.0

Table 15: Earliest failing stage for ActionRAG’s failed queries on StableToolBench (Qwen-2.5-7B-Instruct, all six subsets, 765 queries).

constraint induction stabilizes tool-use trajectories before execution begins.

Most remaining regressions involve incorrect answer generation despite selecting the same tools as CoT/ReAct (60.0% of regression cases), consistent with the stage-level attribution in Table 15. This suggests that the primary remaining bottleneck lies in response grounding and synthesis rather than retrieval or trajectory planning. The second-largest regression source (20.0%) is API execution error, which arises from transient failures in the underlying tool environment and is independent of the planning framework. The remaining regression categories—insufficient tool coverage (9.1%), false abstention (7.3%), and repeated calls (3.6%)—together account for about 20% of regressions, indicating that ActionRAG’s planning behavior itself is rarely the cause of failure on queries that CoT/ReAct happens to solve. These observations motivate future work on tighter integration between feasibility-aware planning and response-level reasoning, for example by propagating constraint signals into the LAM’s answer generation stage.

J Qualitative Case Studies

We present three StableToolBench case studies that illustrate how Constraint Induction (CI) and Action-of-Thought (AoT) interact during inference (Figure 6): (i) **multi-hop parameter chaining** (Case A, G1_Tool), (ii) **two-stage filtering** (Case B, G2_Cat), and (iii) **aggressive action-space pruning** (Case C, G1_Inst). Tools are annotated as **SELECT**, **CHAIN**, or **PRUNE** according to the AoT decision.

Case A: Multi-Step Parameter Chaining			Target Shopping API 5 tools G1_Tool
“Suggest Target ingredients with product details, prices, and availability. Also find nearby stores around ZIP 94611 where I can purchase them.”			
AoT Reasoning		CI retains all 5 (same API domain)	
Tool	$\mathcal{C}_{env}$	$\mathcal{C}_{task}$	Decision
nearby_stores	Satisfied (ZIP in query)	Aligned	SELECT
product_search	Violated — needs store_id	Aligned	CHAIN
product_details	Violated — needs store_id+tcin	Aligned	CHAIN
product_fulfillment	Violated — needs tcin	Aligned	CHAIN
search_autocomplete	—	Not aligned	PRUNE
Trajectory: nearby_stores $\xrightarrow{\text{store_id}}$ product_search $\xrightarrow{\text{tcin}}$ product_details $\xrightarrow{\text{tcin}}$ product_fulfillment store_id from nearby_stores stays bound for every subsequent call.			
Result: Solved (5 $\rightarrow$ 4 tools, 3-hop dependency chain). CoT/ReAct calls product_search first without store_id and fails.			

Case B: CI + AoT Combined Filtering			C++17 Compiler + Password Gen 7 tools, 4 APIs G2_Cat
“Fetch available C++ 17 versions for a new project. Also generate a strong password for documentation.”			
Stage 1 — CI: Excludes hls_for_twitter (cross-domain: Twitter mp4 converter).			7 $\rightarrow$ 6
Stage 2 — AoT (over 6 remaining tools):			
Tool	$\mathcal{C}_{env}$	$\mathcal{C}_{task}$	Decision
c_17_versions	Satisfied	Aligned (C++17 requested)	SELECT
generate_password	Satisfied	Aligned (password requested)	SELECT
pascal_versions	Satisfied	Not aligned (Pascal $\neq$ C++)	PRUNE
get_products	Satisfied	Not aligned (no e-commerce sub-goal)	PRUNE
get_categories	Satisfied	Not aligned (no e-commerce sub-goal)	PRUNE
get_products_in_cat	Satisfied	Not aligned (no e-commerce sub-goal)	PRUNE
Trajectory: c_17_versions, generate_password			6 $\rightarrow$ 2
Result: Solved (7 $\xrightarrow{\text{CI}}$ 6 $\xrightarrow{\text{AoT}}$ 2). CI removes cross-domain noise, while AoT prunes within-domain mismatches.			

Case C: Aggressive Pruning + Parameter Chaining			AliExpress 10 tools G1_Instance
“List products in category 100003819 with brand/price filters. Provide customer feedback with ratings and photos.”			
AoT Reasoning		CI retains all 10 (single API domain)	
Tool	$\mathcal{C}_{env}$	$\mathcal{C}_{task}$	Decision
category_products	Satisfied	Aligned (category ID in query)	SELECT
feedbacks	Violated (needs productid)	Aligned (feedback requested)	CHAIN
shipping	—	Not aligned (not requested)	PRUNE
bestsales	—	Not aligned	PRUNE
historic_prices	—	Not aligned	PRUNE
historic_sales	—	Not aligned	PRUNE
categories	—	Not aligned (user has ID)	PRUNE
currencies	—	Not aligned	PRUNE
countries	—	Not aligned	PRUNE
sales_history	—	Not aligned	PRUNE
Trajectory: category_products $\xrightarrow{\text{productid}}$ feedbacks			10 $\rightarrow$ 2
Result: Solved (80% pruned, with parameter chaining). CoT/ReAct over-explores same-domain tools and fails to form the dependency chain.			

Figure 6: Qualitative case studies of ActionRAG on StableToolBench. Case A demonstrates multi-hop parameter chaining across dependent APIs. Case B illustrates two-stage filtering where CI removes cross-domain distractors and AoT prunes within-domain mismatches. Case C shows aggressive same-domain pruning while preserving the required dependency chain.