What Does an Agent Need? Bundle-Aware Agent Resource Composition

Anonymous authors
Paper under double-blind review

Abstract

As catalogs of models, tools, and reusable skills expand, determining which resources an agent should use becomes increasingly challenging. The value of a resource depends on the backbone and the other resources it is combined with, and it is known only after execution. Existing methods address parts of this problem. Token-based tool generators tie each tool to a learned token, so they select only from the training catalog. Retrievers and routers score each tool or backbone on its own and miss complementary resources, and bundle reuse and agent recommendation return configurations built for other tasks. How to compose heterogeneous resources for a new task remains underexplored. We formulate Task-Conditioned Agent Resource Composition (AgentComposition): jointly selecting a backbone model and composing a tool–skill bundle that maximize end-to-end execution utility. We propose Retrieval-Grounded Bundle-Aware Agent Generation. It retrieves candidate resources and past configurations of similar tasks, and a fine-tuned generator that reads each resource from its description recombines them by constrained beam search, so new resources need no retraining. A bundle-aware critic ranks the complete configurations. It learns from two proxies for real execution, an LLM teacher and historically successful configurations, and adapts to end-to-end execution feedback when it is available. We evaluate by reference recovery, offline execution with judged and native rewards, and online recommendation with real execution, and we build AgentSelect-Exec with 5,060 executed configurations for this purpose. Our method recovers the complete reference configuration within ten candidates for 67.73% of AgentSelect queries, nearly twice the strongest baseline, and leads all baselines on SkillsBench. The proxy-trained critic already raises the execution score of the top-ranked agent from 7.58 to 8.37 out of 10, and execution feedback lifts it to 8.78, above the strongest baselines. Four target-domain queries suffice to adapt the generator to SkillsBench, and agents recommended online execute better than those of the strongest baselines under both an LLM judge and native verifiers. These results establish resource composition as a learnable and transferable capability, shifting the focus from individual resource relevance to joint task suitability, toward agents that adapt to changing resource ecosystems and evolve through accumulated task experience.

1 Introduction

Large language models (LLMs) have grown from text completion Radford et al. (2019); Brown et al. (2020) into agents that reason, call functions, and act in environments Yao et al. (2023); Schick et al. (2023); Qin et al. (2024a). The Model Context Protocol (MCP) Anthropic (2024) and Agent Skills Anthropic (2025b); Xu & Yan (2026) expose growing catalogs of tools, data, and reusable skills, and agent harnesses organize them around a backbone model Microsoft (2026); OpenAI (2026); Anthropic (2025a). Choosing these resources well makes agents more adaptive Cheng et al. (2025); Hao et al. (2025), keeps the tool context small Liu et al. (2026), and lets agents keep up with changing ecosystems Wu et al. (2026b).

Each new task therefore raises a practical question: which backbone and which resources should an agent use together?

Existing work answers parts of this question. Tool learning and tool retrieval select tools for a fixed backbone Qin et al. (2024b); Hao et al. (2023); Wang et al. (2025); Du et al. (2024); Qu et al. (2024); Yuan et al. (2024), model routing chooses a backbone without tools Ong et al. (2025), and AgentSelect recommends predefined configurations Shi et al. (2026). Composing a configuration for a new task is harder, for four reasons. First, the value of a resource depends on the resources it is combined with. A seven-day, pet-friendly road trip under a fixed budget needs accommodation search, pet-policy checks, route planning, and cost calculation. Accommodation tools alone leave needs unmet, and a past road-trip configuration may miss the pet constraint. Selecting resources one by one misses such bundle effects (Appendix N). Second, the number of configurations grows combinatorially with the catalog. Third, the catalog keeps changing, so methods tied to a fixed set of resources must be retrained. Fourth, a configuration's quality is known only after a costly execution.

We formulate Task-Conditioned Agent Resource Composition (AgentComposition): given a task and its context, select a backbone and compose a tool–skill bundle that maximize end-to-end execution utility. Agent architecture search such as AgentSquare Shang et al. (2025) optimizes how an agent plans, reasons, and remembers. AgentComposition chooses the resources an agent uses. We propose Retrieval-Grounded Bundle-Aware Agent Generation, whose components address these challenges. Retrieval narrows the search to relevant resources and supplies past configurations of similar tasks. A fine-tuned generator composes configurations by constrained beam search, choosing each tool given the backbone and the tools already chosen. It reads each resource from its description, so a new resource is usable as soon as it is indexed. A bundle-level critic estimates execution quality from proxy signals, adapts to execution feedback when available, and reranks the completed configurations.

We evaluate on AgentSelect Shi et al. (2026) and SkillsBench Li et al. (2026) and on three datasets with execution outcomes: BFCL Patil et al. (2025), SWE-Skills-Bench Han et al. (2026), and AgentSelect-Exec, which we build from 5,060 executed configurations. Four findings emerge. (i) Our method recovers complete configurations best, while historical bundles, tool retrievers and model routers, token-based tool generators, bundle generators, and retrieval-augmented LLMs each fall short in ways their designs predict. (ii) Execution feedback makes the critic's rankings agree with execution. The critic then selects better-executing configurations than all proxy-trained variants and the strongest baselines, and it works best as a final reranker. (iii) The learned composition transfers to new domains. Four target queries are enough to adapt the generator to SkillsBench, and Stage 3 adapts the critic when executed candidates are available. (iv) Agents recommended online execute better than those of the strongest baselines, under both an LLM judge and native verifiers, and within our lists, higher-ranked recommendations execute better.

Our contributions are as follows. ❶ We formulate AgentComposition, which composes a backbone and a tool–skill bundle for each task to maximize execution utility, and we identify its bundle-level challenges. ❷ We propose a retrieval-grounded, bundle-aware framework that composes valid configurations from a changing catalog and ranks them with a critic that adapts to execution feedback. ❸ We build AgentSelect-Exec and evaluate by reference recovery, offline execution, transfer to new domains, and online execution[1].

2 Related Work

Agent Construction and Resource Composition. Agent construction decides what an agent can do and how it does it. AgentSquare searches over planning, reasoning, tool-use, and memory modules Shang et al. (2025). ADAS repeatedly writes and tests agent programs Hu et al. (2025b). SkillWeaver discovers and refines reusable skill APIs Zheng et al. (2025), and PSEC combines skill-specific LoRA modules in parameter space Liu et al. (2025). These methods change the agent's design or build new capabilities. We compose existing catalog

[1]Code and data: https://anonymous.4open.science/r/AgentComposition-801C.

resources for a task, choosing a backbone and a tool–skill bundle together. Model routing chooses between a strong and a weak LLM without tools Ong et al. (2025).

Agent Resource Retrieval and Selection. Existing retrieval typically ranks individual resources by their relevance to a query Shi et al. (2025); Moon et al. (2024). PORTS aligns a dense tool retriever with LLM preferences over tool selection Molfetta et al. (2025), and AnyTool collects APIs for a fixed LLM through hierarchical LLM agents Du et al. (2024). Scoring tools one at a time finds useful tools but ignores how they work together, so the top-ranked tools form bundles with many extra tools. Generative methods such as ToolkenGPT, ToolGen, and ToolWeaver let a fixed backbone select tools by generating them Hao et al. (2023); Wang et al. (2025); Fang et al. (2026). ToolWeaver encodes tool co-usage in hierarchical tool codes that are fixed for the whole catalog. These methods give each tool a learned token or code tied to the training catalog, so new tools need new tokens and retraining. Toolken+ reranks ToolkenGPT's top-k tools by their documentation and adds a reject option that skips the current call Yakovlev et al. (2024). It improves individual tool calls during generation. Our method treats resource selection as composition. It selects a backbone and a bundle together and generates configurations from retrieved resources and their descriptions, so a new resource can be used as soon as it is indexed.

Bundle Recommendation and Agent Discovery. Bundle recommendation ranks existing bundles or constructs new ones Sun et al. (2026). BGCN models user–item, user–bundle, and bundle–item interactions through graph convolutions Chang et al. (2020). CrossCBR aligns item- and bundle-level representations Ma et al. (2022). Construction methods include query-conditioned Text2Bundle Hu et al. (2025a), LLM-based adaptive in-context learning Sun et al. (2024), and discrete-diffusion generation of unordered bundles Tu et al. (2026). In agent discovery, AgentSearchBench and AgentSelect retrieve or rank predefined agents or configurations from an existing catalog Wu et al. (2026a); Shi et al. (2026). We treat agent resources as parts of a bundle and generate a new configuration for each task.

3 Method

Problem formulation. AgentComposition maps a task to an agent that executes it end to end. A task Q contains a user request and its context, such as relevant session history. Given backbone models $\mathcal{M}$ and tools and skills $\mathcal{T}$, an agent configuration $A = (M, B)$ pairs a backbone $M \in \mathcal{M}$ with a bundle $B \subseteq \mathcal{T}$ of at most $L_{\max}$ tools. Executing A on Q yields a utility $U(Q, A)$, such as a verifier reward or a judged score. The goal is the configuration with the highest utility,

$$A^\star(Q) = \underset{M \in \mathcal{M},\ B \subseteq \mathcal{T},\ |B| \le L_{\max}}{\arg\max} U\big(Q, (M, B)\big).$$

Three properties make this a bundle composition problem. First, utility is non-additive. The gain of adding a tool t, $U(Q, (M, B \cup \{t\})) - U(Q, (M, B))$, depends on the backbone M and on the tools already in B, which can complement or duplicate t (Appendix N). Retrieval-based selection scores each resource on its own and picks the backbone and the bundle in separate steps. It thus treats this gain as a fixed relevance of t to Q, independent of M and B. AgentComposition selects the backbone and the bundle together, as one configuration. Second, the search space grows combinatorially with the catalog, to $|\mathcal{M}| \sum_{m=0}^{L_{\max}} \binom{|\mathcal{T}|}{m}$ configurations. Third, U is observed only after costly execution. We therefore estimate U from proxy signals, and from execution feedback when it is available. The catalog also changes over time, so new resources must be usable without retraining. The output is a list of up to K distinct configurations ranked by estimated utility, $\mathcal{G}(Q; \mathcal{M}, \mathcal{T}) = (A_Q^{(1)}, \ldots, A_Q^{(K)})$.

Method overview. Our framework has three steps (Figure 1). Retrieval finds task-relevant resources and the past configurations of similar tasks. A fine-tuned generator then recombines these resources into candidate configurations through constrained beam search. Finally, a bundle-aware critic ranks the completed configurations. The critic stands in for execution. It learns from proxy signals and can adapt to execution feedback when it is available. Algorithm 1 in Appendix C gives the procedure.

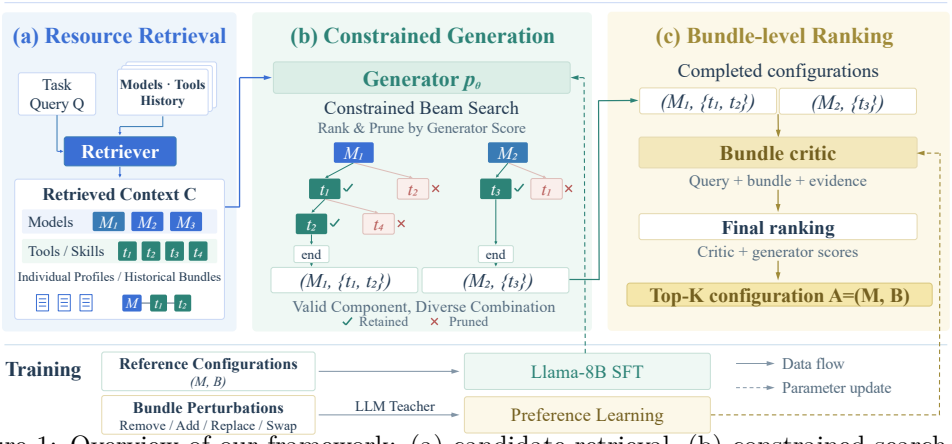


Figure 1: Overview of our framework: (a) candidate retrieval, (b) constrained search with generator-based pruning, and (c) final reranking. The lower panel shows generator and critic training, with solid arrows for data flow and dashed arrows for parameter updates.

3.1 Retrieval Context

Retrieval provides the generator and critic with $\mathcal{C}(Q) = (\mathcal{M}_r(Q), \mathcal{T}_r(Q), \mathcal{E}_r(Q))$: retrieved LLMs, retrieved tools, and evidence, which includes resource profiles, normalized scores, provenance, and complete historical bundles when available. Semantic retrieval is the default, and a collaborative-signal retriever is added when query–agent interaction histories exist (Appendix B).

3.2 Structured Generator Fine-Tuning

We initialize p_θ from Meta-Llama-3-8B, apply LoRA to attention and feed-forward layers, and train the input embeddings and output projection, keeping other pretrained parameters frozen. This adapts the generator to resource descriptions and the structured output format. Each training instance has a query $Q^{(n)}$, context $\mathcal{C}(Q^{(n)})$, and reference $A^{(n)\star} = (M^{(n)\star}, B^{(n)\star})$. The retrieved context includes the reference resources. Constrained decoding limits outputs to the components in this context, so every output uses valid component names. We serialize $Y(A) = \langle M, \tau_{\text{sep}}, \text{Ser}(B), \tau_{\text{end}} \rangle$. For $B = \{t_1, \ldots, t_m\} \neq \emptyset$, $\text{Ser}(B) = \langle t_1, \ldots, t_m \rangle$ follows the supervised order. For an allowed empty bundle, it is $\langle \tau_\emptyset \rangle$. The markers τ_{sep}, $\tau_\emptyset$, and τ_{end} separate the backbone from the tools, mark an empty bundle, and end the configuration. The decoding order affects what each token is conditioned on, but the bundle itself is an unordered set. Resource names use the existing subword tokens of the vocabulary.

Let $x^{(n)} = \text{Prompt}(I, Q^{(n)}, \mathcal{C}(Q^{(n)}))$, where I is a fixed instruction requesting only a configuration. The target $y^{(n)\star}$ is the tokenization of $Y(A^{(n)\star})$ followed by the model's end-of-sequence token. We minimize

$$\mathcal{L}_{\text{SFT}}(\theta) = -\sum_{n=1}^{N} \sum_{i=1}^{|y^{(n)\star}|} \log p_\theta \left(y_i^{(n)\star} \mid x^{(n)}, y_{<i}^{(n)\star} \right). \tag{1}$$

3.3 Bundle-Level Preference Modeling

The critic $\widehat{U}_\psi(Q, A)$ estimates the execution utility $U(Q, A)$ of a complete configuration. It has an encoder Enc_{ψ_e}, initialized from AgentEmbedding Chen et al. (2026) (an EasyRec model tuned for agent-resource retrieval), and a scalar head g_{ψ_s}. The encoder reads Q, A, and the retrieved evidence $\Gamma(A, \mathcal{E}_r(Q))$ for A's resources. We train the critic in three stages: on LLM-teacher judgments, on counterfactual perturbations judged against reference configurations, and on execution feedback when it is available.

Stage 1: Teacher-supervised preferences. For each training query, we perturb its reference configuration by removing, adding, or replacing tools, or by changing the backbone. An

LLM teacher scores each configuration with $r_i \in [0,1]$. We sample same-query pairs (i,j) with $r_i - r_j \geq 0.05$ and cap the number of pairs that change the backbone, the tools, or both. With $\widehat{U}_i = \widehat{U}_\psi(Q_i, A_i)$, we minimize on minibatch $\mathcal{I}_1$

$$\mathcal{L}^{(1)}_{\text{critic}} = -\sum_{(i,j)\in\mathcal{I}_1} w_{ij} \log \sigma(\widehat{U}_i - \widehat{U}_j), \tag{2}$$

where σ is the logistic sigmoid and the weights $w_{ij} \propto \sqrt{\max(r_i - r_j, 0.05)}$ sum to one. We add 0.1 times the mean squared error between $\sigma(\widehat{U})$ and the teacher scores.

Stage 2: Reference-based refinement. With the backbone fixed, we build preference pairs from tool perturbations that move a configuration toward a reference configuration: removing an extra tool, restoring a missing tool, or replacing a tool at a fixed bundle size. A pair is kept when the perturbation raises precision or recall against that reference without lowering the other. For backbone changes, we keep the teacher comparisons. Over a minibatch $\mathcal{I}_2$, we minimize the margin loss

$$\mathcal{L}^{(2)}_{\text{critic}} = \frac{1}{|\mathcal{I}_2|} \sum_{(i,j)\in\mathcal{I}_2} \log\Big(1 + \exp\Big[m - (\widehat{U}_i - \widehat{U}_j)\Big]\Big), \qquad m = 0.1. \tag{3}$$

Stage 3: Execution-feedback adaptation. When execution outcomes are available (Section 5.1), we fine-tune the Stage 1+2 critic with $\mathcal{L}^{(3)}_{\text{critic}} = \|\sigma(\widehat{U}) - y\|_1 + \mathcal{L}^{(2)}_{\text{critic}}$, where $y \in [0,1]$ is the execution score and Eq. (3) uses same-query pairs with $y_i > y_j$.

3.4 Bundle-Aware Constrained Generation

Search composes a configuration step by step over the retrieved resources. It selects a backbone first and then adds one tool at a time, so each tool is chosen given the backbone and the tools already in the bundle. Two quantities control the search. The proposal budget P is the maximum number of proposals kept at each expansion, and the pruning width $W(\eta)$ is the number of paths kept from η candidates (Eq. (6)).

Constrained decoding. Each hypothesis h holds a backbone $M_h \in \mathcal{M}_r(Q)$, a prefix $\rho_h = \langle M_h, \tau_{\text{sep}}, t_1, \ldots, t_\ell \rangle$, and a tool set B_h. A legal action adds an unused tool from $\mathcal{T}_r(Q) \setminus B_h$ or terminates the bundle with τ_{end}, or with $\tau_\emptyset, \tau_{\text{end}}$ for an empty bundle. A path also ends when $|B_h| = L_{\max}$. Prefix tries over the identifiers restrict decoding to legal actions, so every output is a valid configuration of retrieved resources.

Action and path scoring. Given $x = \text{Prompt}(I, Q, \mathcal{C}(Q))$, an action a with tokens $\pi(a) = \langle v^a_1, \ldots, v^a_{|\pi(a)|} \rangle$ is scored by its length-normalized log-likelihood

$$\bar{\ell}_\theta(a \mid x, \rho) = \frac{1}{|\pi(a)|} \sum_{j=1}^{|\pi(a)|} \log p_\theta\big(v^a_j \mid x, \rho, v^a_{<j}\big), \tag{4}$$

where ρ is empty for a backbone and ρ_h for later actions. The empty-bundle action $\tau_\emptyset$ is legal only at the first tool step, where its proposal score is $\bar{\ell}_\theta(\tau_\emptyset \mid x, \rho_h) - \lambda_\emptyset$ with $\lambda_\emptyset \geq 0$. A path h is scored by the mean log-likelihood over the tokens Ω_h of its selected actions,

$$\bar{L}_\theta(h) = \frac{1}{|\Omega_h|} \sum_{i\in\Omega_h} \log p_\theta(y_i \mid x, y_{<i}). \tag{5}$$

Expansion and pruning. A pruning step that receives η candidates keeps the top $W(\eta)$, with

$$W(\eta) = \min\{\eta,\ b_{\max},\ \max(b_{\min}, \lceil r_{\text{keep}}\, \eta \rceil)\}, \tag{6}$$

where $r_{\text{keep}} \in (0,1]$ is the retain ratio and $1 \leq b_{\min} \leq b_{\max}$ bound the width. Search starts by scoring every backbone in $\mathcal{M}_r(Q)$ with Eq. (4) and an empty prefix. The top P become root proposals, and pruning keeps $W(\eta_0)$ of these η_0 roots as the initial beam. Retrieval thus supplies the candidate backbones, and the generator orders them. At each depth, every

path proposes its top P legal actions by Eq. (4), and τ_{end} is added for a nonempty bundle if it is not among them. Completed paths enter a pool. Unfinished paths with the same backbone and tool set are merged into the one with the best score, and pruning keeps the top $W(\eta)$ of the η merged paths by Eq. (5). Search ends when no unfinished path remains.

Final configuration reranking. Completed paths are deduplicated by (M, B) into $\mathcal{F}(Q)$, each with its best path score $\bar{L}_\theta(Q, A)$. The critic ranks them by $S_{\text{final}}(Q, A) = \widehat{U}_\psi(Q, A)$, and we return the top K configurations as $\mathcal{H}_K(Q)$. Without a critic, $S_{\text{final}}(Q, A) = \bar{L}_\theta(Q, A) - \lambda_{\text{len}}|B|$, where $\lambda_{\text{len}} \geq 0$ favors compact bundles.

4 Experiments

4.1 Datasets

AgentSelect Shi et al. (2026) pairs queries with agent configurations over 231 LLMs and 12,099 tools. SkillsBench Li et al. (2026) runs agents with and without skills on tasks checked by hidden tests. From its successful runs, we build queries over 19 LLMs, 155 skills, and 20 tools and split them by task. We use it as a new resource domain (Section 5.2). Three further datasets record the execution outcome of each configuration. AgentSelect-Exec executes the ten top-ranked configurations of each AgentSelect query and scores each run with an LLM judge. We use it for Stage 3 training and the execution analysis (Section 5.1). BFCL Patil et al. (2025) and SWE-Skills-Bench Han et al. (2026) provide native rewards for transfer to new domains (Section 5.2). Table 1 summarizes the splits, and Appendix D.1 details their construction.

Table 1: Dataset statistics.

	Queries			Pairs		
Dataset	Train	Val.	Test	Train	Val.	Test
AgentSelect	5,022	627	629	12,180	1,507	1,514
AgentSelect-Exec	200	106	200	2,000	1,060	2,000
SkillsBench	19	–	78	109	–	506
BFCL	161	35	44	2,898	630	792
SWE-Skills-Bench	690	60	230	5,520	480	1,840

4.2 Implementation Details

The generator is Meta-Llama-3-8B fine-tuned with LoRA on AgentSelect. The critic starts from AgentEmbedding Chen et al. (2026), is trained by Stages 1 and 2 on AgentSelect, and is adapted by Stage 3 on the AgentSelect-Exec training split. Search uses adaptive beam bounds $(b_{\min}, b_{\max}) = (1, 2)$, retain ratio $r_{\text{keep}} = 0.4$, proposal budget $P = 5$, $L_{\max} = 6$, and $K = 10$. The critic reranks completed configurations without guiding search. Appendix D.2 gives all training and search settings, and Appendix K reports the parameter grid.

4.3 Baselines

We compare five baseline families (Appendix H). (i) Retrieval-based composition pairs a retrieved LLM with independently retrieved tools (Component-Ret) or with a historical bundle (Bundle-Ret). (ii) Tool retrieval and model routing covers PORTS Molfetta et al. (2025) (a tool retriever aligned with LLM preferences), AnyTool Du et al. (2024) (API retrieval by hierarchical LLM agents), and RouteLLM Ong et al. (2025) (routing queries between LLMs). (iii) Token-based tool generation covers ToolGen Wang et al. (2025) and ToolWeaver Fang et al. (2026), which generate learned tool tokens or codes. (iv) Bundle generation adapts sequential Text2Bundle Hu et al. (2025a) and discrete-diffusion DDBC-Adapt Tu et al. (2026). (v) Retrieval-augmented LLMs generate configurations from the same retrieved evidence as ours, using RAG-Llama, GPT-5.6-Sol/Terra/Luna, DeepSeek-V4-Pro, Qwen3.8-27B, and Kimi-K2.6. Families (ii) and (iii) select either tools or a backbone, so we complete each configuration with the other module of Component-Ret (Appendix H.4).

4.4 Evaluation Metrics

Each method returns 10 configurations, and metrics are averaged over queries on a 0–100 scale. Our primary metric, Rank-Discounted Component Recall (RDCR@10), discounts

component recall by rank, rewarding partial recovery near the top. RDCP@10 similarly discounts component precision and penalizes components absent from the matched reference. ToolR@1 is the share of required tools covered at rank 1, and CompR@1 also counts the reference LLM. Tool-Hit@10 asks whether any candidate covers all required tools. CR-Hit@1, CR-Hit@10, and CR-MRR@10 additionally require the reference LLM, with MRR rewarding earlier first hits. Appendix I defines all metrics, and Table 2 reports them.

Table 2: Agent recommendation performance on AgentSelect (%, higher is better). The top three scores in each column are shaded.

	Tool Recovery		Agent Recovery				Ranked Quality	
Method	ToolR @1	Tool-Hit @10	CompR @1	CR-Hit @1	CR-Hit @10	CR-MRR @10	RDCR @10	RDCP @10
Component-Ret	27.15	12.40	22.61	1.91	6.04	2.92	20.72	11.78
Bundle-Ret	90.06	97.62	66.74	10.33	12.40	11.19	33.81	32.55
PORTS	72.23	55.95	54.20	9.52	32.74	15.73	52.30	26.01
AnyTool	11.54	5.41	13.33	0.00	2.60	1.06	10.96	12.14
RouteLLM	27.16	14.88	24.74	5.54	9.10	6.50	21.96	10.83
ToolGen	37.95	28.40	31.91	4.16	15.60	7.10	30.28	31.05
ToolWeaver	21.68	14.29	19.14	0.60	5.36	1.84	18.77	19.95
Text2Bundle	40.28	31.32	31.99	2.07	3.82	2.44	31.28	17.66
DDBC-Adapt	70.61	60.25	54.50	8.90	32.43	15.61	49.92	44.73
RAG-Llama	52.68	49.92	40.58	3.97	6.04	4.48	15.79	16.23
RAG-GPT-5.6-Sol	78.98	69.79	57.86	4.93	30.37	13.17	54.76	52.69
RAG-GPT-5.6-Terra	81.92	71.86	60.23	7.00	31.80	14.78	56.37	54.24
RAG-GPT-5.6-Luna	74.23	62.48	55.57	6.36	26.07	12.53	51.68	50.63
RAG-DeepSeek-V4-Pro	73.04	58.66	54.42	6.36	27.03	12.29	52.36	43.61
RAG-Qwen3.8-27B	82.81	73.29	61.71	7.79	31.00	14.46	57.94	40.48
RAG-Kimi-K2.6	87.63	81.08	65.03	9.38	36.73	17.10	62.83	37.29
Ours (generator only)	89.99	83.15	66.94	12.24	67.73	29.02	65.88	65.28
Ours (with critic)	90.19	87.92	66.05	9.70	58.51	23.35	65.48	62.71

4.5 Overall Recommendation Performance

This section tests whether composing a configuration as a whole recovers complete configurations better than reusing, retrieving, or generating resources one at a time. Table 2 shows that it does. The two variants of ours rank first and second in RDCR@10 and RDCP@10, and each baseline family falls short in ways its design predicts.

Historical bundles hold the right resources, but not the right configuration. Bundle-Ret reaches the highest Tool-Hit@10 (97.62%), so the bundles of similar past queries contain most required tools. Reused as they are, they rarely match the whole configuration of a new task (12.40% CR-Hit@10), so we recompose the retrieved resources.

Selecting resources one by one does not compose them. Retrievers and routers score each component on its own. PORTS finds relevant tools but adds many that the task does not need (26.01% RDCP@10), and token-based generators such as ToolGen and ToolWeaver, which tie each tool to a learned token, recover even less. RAG-LLMs read the same evidence as ours and select relevant resources, with high ToolR@1 and CompR@1, but they rarely recover the backbone and all tools together (at most 36.73% CR-Hit@10).

Bundle generators fix the backbone or the vocabulary. Text2Bundle builds a bundle for a fixed CF-selected LLM and rarely recovers the whole configuration (3.82% CR-Hit@10). DDBC-Adapt generates the backbone and the tools together from the components of its training vocabulary and reaches 32.43%, about half of ours.

Joint composition recovers complete configurations. Our generator chooses each tool given the backbone and the tools already chosen. It recovers the complete configuration within ten candidates on 67.73% of queries, nearly twice the best baseline. Constrained decoding keeps every output valid, whereas 3% of free outputs name unknown components (Appendix L).

Reference recovery does not measure execution. Critic reranking slightly lowers recovery, e.g., RDCP@10 from 65.28% to 62.71%. The critic estimates execution quality, and a

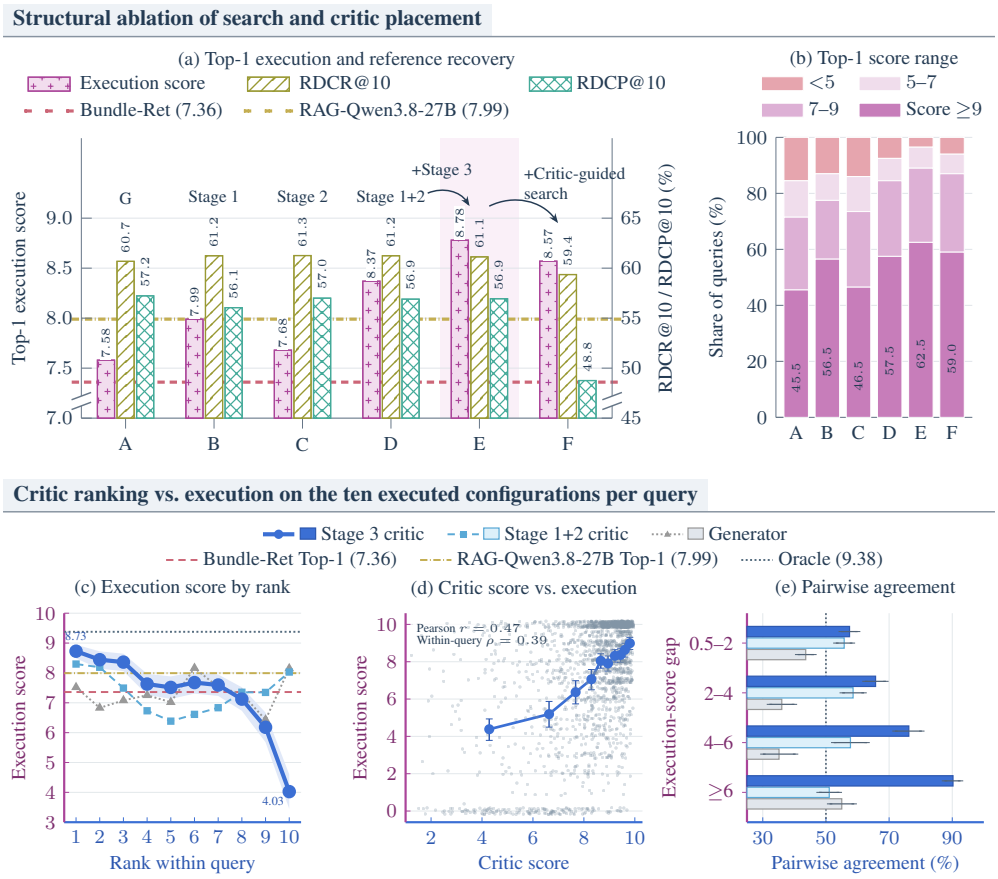


Figure 2: Execution analysis on the 200 AgentSelect-Exec test queries. Top-1 scores in (a)–(b) come from re-executing each top-ranked configuration, and (c)–(e) use the recorded executions of AgentSelect-Exec. In (a), purple bars give execution scores, and hatched bars give reference recovery. Bands and error bars are 95% query-bootstrap intervals.

configuration that departs from the reference can still execute well. Section 5.1 therefore evaluates rankings by execution.

5 Further Analysis

5.1 From Reference Configurations to Execution Signals

Our critic is meant to estimate execution quality, which reference recovery does not capture. On AgentSelect-Exec, we ask which supervision lets the critic predict execution quality, where in the pipeline it should act, and whether its full ranking agrees with execution. Execution feedback (Stage 3) lets the critic pick the best-executing configurations and order the rest as execution does, and the critic works best as a final reranker.

Figure 2a–b compares six rankers on the 200 test queries. A ranks candidates by generator scores (G). B, C, and D rerank them with critics trained on Stage 1, Stage 2, and both stages. E adds Stage 3, and F also applies the Stage 3 critic during beam search. A–E rerank the same ten executed configurations per query (Appendix F), and F ranks the pool of its own search.

Each signal brings the critic closer to execution. Both proxy signals improve the top choice over generator scores, and together they help more (D, 8.37 versus 7.58 for A). Execution feedback helps most. E reaches 8.78 and cuts the share of queries scoring below 5 from 15.5% to 3.5%. Reference recovery stays flat across A–E (60.7%–61.3% RDCR@10), so it misses

these gains. Even the proxy-trained critic D selects better configurations than Bundle-Ret (7.36) and RAG-Qwen3.8-27B (7.99).

The critic belongs at the end of search. Guiding beam search with the Stage 3 critic (F) lowers the top choice to 8.57. The critic is trained on complete configurations, so it misjudges partial ones and prunes good branches early. We therefore apply it only after search.

The adapted critic orders all candidates as execution does. Beyond the top choice (Figure 2c–e), execution quality falls steadily with the Stage 3 critic's rank, from 8.73 at rank 1 to 4.03 at rank 10, so configurations that fail in execution sink to the bottom. The Stage 1+2 critic and the generator spread such failures across their lists. Within queries, the Stage 3 critic orders 68.0% of configuration pairs correctly, against 55.8% for the Stage 1+2 critic and 43.1% for the generator. On pairs whose execution scores differ by at least 6 points, it is correct 90.2% of the time (Figure 2e).

5.2 Generalization to New Domains

Our generator reads resources from their descriptions, so its learned composition should carry over to new domains. We test this on SkillsBench, BFCL, and SWE-Skills-Bench and ask which component to adapt. The learned composition transfers. A few target queries are enough to adapt the generator, and executed candidates are enough to adapt the critic.

On SkillsBench, we fine-tune the AgentSelect-trained generator on the 19 training queries and rank candidates by its scores. A few-shot variant trains on 4 of them and validates on the other 15. We compare it with fine-tuning the base Meta-Llama-3-8B (Figure 3). BFCL and SWE-Skills-Bench provide executed configurations with native rewards. There we rank candidates with the generator (Generator Rank) or the critic (Critic Rank), each with and without fine-tuning on target training tasks. Stage 3 fine-tunes the critic. RAG-Qwen3.8-27B ranks the same candidates from task and resource descriptions.

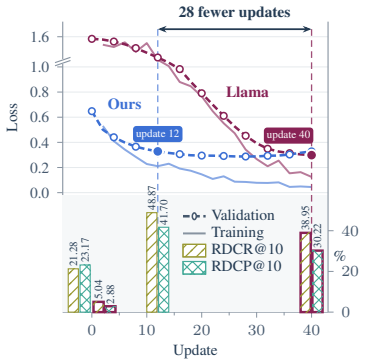


Figure 3: Few-shot adaptation on SkillsBench: training and validation loss (lines) and test RDCR@10 and RDCP@10 (bars). Llama is shown in orange.

Composition transfers to a new resource domain. On SkillsBench, ours leads all baselines on every metric (Table 3) and recovers the complete configuration within ten candidates on 47.44% of the queries, against at most 33.33% for the baselines. RAG-LLMs transfer unevenly. Kimi-K2.6, strong on AgentSelect, falls from 36.73% to 3.85% CR-Hit@10. Constrained decoding matters more in the new domain. Without target training, half of the free outputs (51.3%) are invalid.

Table 3: Agent recommendation performance on SkillsBench (%, higher is better). The top three scores in each column are shaded.

	Tool Recovery		Agent Recovery				Ranked Quality	
Method	ToolR @1	Tool-Hit @10	CompR @1	CR-Hit @1	CR-Hit @10	CR-MRR @10	RDCR @10	RDCP @10
RAG-GPT-5.6-Sol	71.72	41.03	56.22	0.00	24.36	4.24	57.31	54.47
RAG-GPT-5.6-Terra	69.71	32.05	52.62	1.28	16.67	4.91	53.86	44.62
RAG-GPT-5.6-Luna	35.67	3.85	31.97	0.00	2.56	0.40	32.74	47.98
RAG-DeepSeek-V4-Pro	37.05	1.28	33.07	0.00	1.28	0.43	33.72	34.73
RAG-Qwen3.8-27B	71.02	41.03	56.17	1.28	33.33	9.73	57.47	50.25
RAG-Kimi-K2.6	41.76	3.85	40.14	2.56	3.85	3.21	38.64	34.98
Ours	73.61	53.85	61.01	15.38	47.44	24.91	60.31	60.87

A few queries suffice to adapt the generator. Trained on AgentSelect, our generator starts far ahead of the original Llama (21.28 versus 5.04 RDCR@10) and needs 12 updates instead of 40 to reach a similar validation loss (Figure 3). At these checkpoints it also recovers more of the references (48.87 versus 38.95 RDCR@10). The generator thus carries the task format and the composition it learned on AgentSelect into the new domain.

With executed candidates, adapting the critic suffices. Without target training, the Stage 1+2 critic already beats RAG-Qwen3.8-27B on BFCL (90.91 versus 88.64 Top-1) but trails it slightly on SWE-Skills-Bench (Table 4). Stage 3 on target executions lifts the critic to 93.18 on BFCL and 77.13 on SWE-Skills-Bench, above RAG-Qwen3.8-27B. This matches or beats fine-tuning the generator.

Table 4: Offline ranking on BFCL and SWE-Skills-Bench (%, higher is better). Top-1 is the native reward of the top-ranked configuration. The best score in each column is shaded.

	BFCL			SWE-Skills-Bench		
Method	Top-1	RDCR	RDCP	Top-1	RDCR	RDCP
RAG-Qwen3.8-27B	88.64	92.75	92.75	76.85	71.70	71.00
Ours, without target training						
Generator Rank	72.73	90.04	90.04	75.01	71.43	70.71
Critic Rank	90.91	93.56	93.56	73.79	70.21	70.34
Ours, with target fine-tuning						
Generator Rank	90.91	95.68	95.68	77.24	70.68	70.48
Critic Rank	93.18	97.02	97.02	77.13	70.10	70.73

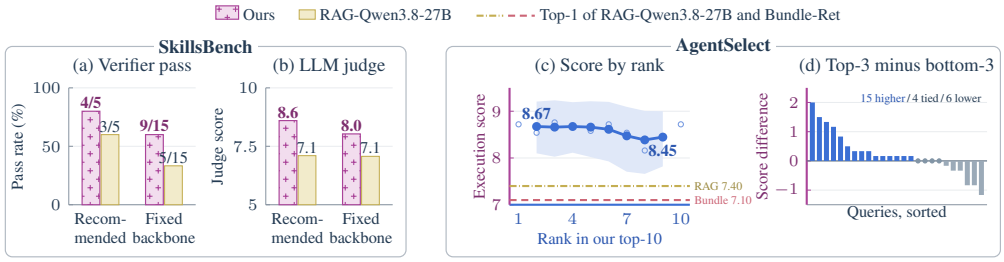


Figure 4: Real execution. SkillsBench: (a) verifier pass rate and (b) judge score of the top-ranked agents. AgentSelect: (c) score by rank in our top-10 lists, with baseline top-1 scores as lines, and (d) per query, the mean score of ranks 1–3 minus that of ranks 8–10.

5.3 Online Recommendation and Execution

Beyond reranking recorded executions, we ask whether the agents our pipeline recommends online also execute better. We run the full pipeline on held-out queries and score each executed recommendation with an LLM judge and, on SkillsBench, with native verifiers (Appendix M). On SkillsBench, our recommendations execute better than those of RAG-Qwen3.8-27B under both the native verifier and the LLM judge (Figure 4a–b). They also pass more verifiers when every method runs on the same fixed backbone, so the gain comes from the selected skills. On AgentSelect, every rank of our top-10 lists executes better than the top-1 agents of both baselines (Figure 4c). Within the lists, higher ranks execute better. The score falls from 8.67 at ranks 1–3 to 8.45 at ranks 8–10, and ranks 1–3 outscore ranks 8–10 on 15 of 25 queries and trail on 6 (Figure 4c–d). Appendix M.4 also ranks official SkillsBench runs offline.

6 Conclusion and Future Work

We formulate AgentComposition, which composes a backbone and a tool–skill bundle for each task to maximize execution utility, and propose a retrieval-grounded, bundle-aware framework for it. Across reference recovery, offline execution, new domains, and online execution, composing a configuration as a whole outperforms selecting resources one by one, and a critic adapted to execution feedback ranks configurations as execution does. Because the generator reads resources from their descriptions, the learned composition transfers to new domains with a few target queries. Our method is limited by retrieval recall, and its critic relies on proxy signals when no execution feedback is available. Future work will use the adapted critic as a reinforcement-learning reward for the generator.

References

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

Anthropic. Effective harnesses for long-running agents. https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents, November 2025a.

Anthropic. Equipping agents for the real world with agent skills, 2025b. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), Advances in Neural Information Processing Systems, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Jianxin Chang, Chen Gao, Xiangnan He, Depeng Jin, and Yong Li. Bundle recommendation with graph convolutional networks. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '20, pp. 1673–1676, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380164. doi: 10.1145/3397271.3401198. URL https://doi.org/10.1145/3397271.3401198.

Tingwei Chen, Yunxiao Shi, Zhengdong Chu, Qingsong Wen, and Min Xu. Adapting embedding models for agent capability retrieval. arXiv preprint arXiv:2607.17347, 2026. Accepted for oral presentation at the AgentSearch Workshop, SIGIR 2026.

Zihao Cheng, Hongru Wang, Zeming Liu, Yuhang Guo, Yuanfang Guo, Yunhong Wang, and Haifeng Wang. ToolSpectrum: Towards personalized tool utilization for large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), Findings of the Association for Computational Linguistics: ACL 2025, pp. 20679–20699, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1063. URL https://aclanthology.org/2025.findings-acl.1063/.

Yu Du, Fangyun Wei, and Hongyang Zhang. AnyTool: Self-reflective, hierarchical agents for large-scale API calls. In Proceedings of the 41st International Conference on Machine Learning, volume 235 of Proceedings of Machine Learning Research, pp. 11812–11829. PMLR, 2024. URL https://proceedings.mlr.press/v235/du24h.html.

Bowen Fang, Wen Ye, Yunyue Su, Jinghao Zhang, Qiang Liu, Yesheng Liu, Xin Sun, Shu Wu, Jiabing Yang, Baole Wei, and Liang Wang. Toolweaver: Weaving collaborative semantics for scalable tool use in large language models. In The Fourteenth International Conference on Learning Representations, 2026. URL https://arxiv.org/abs/2601.21947.

Zhicheng Guo, Sijie Cheng, Yuchen Niu, Hao Wang, Sicheng Zhou, Wenbing Huang, and Yang Liu. StableToolBench-MirrorAPI: Modeling tool environments as mirrors of 7,000+ real-world APIs. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), Findings of the Association for Computational Linguistics: ACL 2025, pp. 5247–5270, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.273. URL https://aclanthology.org/2025.findings-acl.273/.

Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. SWE-Skills-Bench: Do agent skills actually help in real-world software engineering? arXiv preprint arXiv:2603.15401, 2026.

Shibo Hao, Tianyang Liu, Zhen Wang, and Zhiting Hu. ToolkenGPT: Augmenting frozen language models with massive tools via tool embeddings. In Thirty-seventh Conference on Neural Information Processing Systems, 2023. URL https://openreview.net/forum?id=BHXsb69bSx.

Yupu Hao, Pengfei Cao, Zhuoran Jin, Huanxuan Liao, Yubo Chen, Kang Liu, and Jun Zhao. Evaluating personalized tool-augmented LLMs from the perspectives of personalization and proactivity. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 21897–21935, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1064. URL https://aclanthology.org/2025.acl-long.1064/.

Juntong Hu, Shixuan Zhu, Chuan Cui, Qi Shen, Yu Ji, and Zhihua Wei. Text2bundle: Towards personalized query-based bundle generation. ACM Trans. Recomm. Syst., 4(2), November 2025a. doi: 10.1145/3706125. URL https://doi.org/10.1145/3706125.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In The Thirteenth International Conference on Learning Representations, 2025b. URL https://arxiv.org/abs/2408.08435.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han chung Lee, and Dawn Song. Skillsbench: Benchmarking how well agent skills work across diverse tasks, 2026. URL https://arxiv.org/abs/2602.12670.

Marianne Menglin Liu, Daniel Garcia, Fjona Parllaku, Vikas Upadhyay, Syed Fahad Allam Shah, and Dan Roth. ToolScope: Enhancing LLM agent tool use through tool merging and context-aware filtering. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 34095–34119, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1573. URL https://aclanthology.org/2026.acl-long.1573/.

Tenglong Liu, Jianxiong Li, Yinan Zheng, Haoyi Niu, Yixing Lan, Xin Xu, and Xianyuan Zhan. Skill expansion and composition in parameter space. In The Thirteenth International Conference on Learning Representations, 2025. URL https://openreview.net/forum?id=GLWf2fq0bX.

Yunshan Ma, Yingzhi He, An Zhang, Xiang Wang, and Tat-Seng Chua. Crosscbr: Cross-view contrastive learning for bundle recommendation. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD '22, pp. 1233–1241, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393850. doi: 10.1145/3534678.3539229. URL https://doi.org/10.1145/3534678.3539229.

Microsoft. Agent harness. https://learn.microsoft.com/agent-framework/agents/harness, 2026. Microsoft Agent Framework.

Lorenzo Molfetta, Giacomo Frisoni, Nicolò Monaldini, and Gianluca Moro. PORTS: Preference-optimized retrievers for tool selection with large language models. In Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing,

Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.507. URL https://aclanthology.org/2025.emnlp-main.507/.

Suhong Moon, Siddharth Jha, Lutfi Eren Erdogan, Sehoon Kim, Woosang Lim, Kurt Keutzer, and Amir Gholami. Efficient and scalable estimation of tool representations in vector space, 2024. URL https://arxiv.org/abs/2409.02141.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In The Thirteenth International Conference on Learning Representations, 2025. URL https://openreview.net/forum?id=8sSqNntaMr.

OpenAI. The next evolution of the agents sdk. https://openai.com/index/the-next-evolution-of-the-agents-sdk/, April 2026.

Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In International Conference on Machine Learning, 2025. URL https://openreview.net/forum?id=2GmDdhBdDk.

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. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In The Twelfth International Conference on Learning Representations, 2024a. URL https://openreview.net/forum?id=dHng2O0Jjr.

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. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In The Twelfth International Conference on Learning Representations, 2024b. URL https://openreview.net/forum?id=dHng2O0Jjr.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Towards completeness-oriented tool retrieval for large language models. In Proceedings of the 33rd ACM International Conference on Information and Knowledge Management, pp. 1930–1940. ACM, 2024. doi: 10.1145/3627673.3679847. URL https://doi.org/10.1145/3627673.3679847.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. URL https://api.semanticscholar.org/CorpusID:160025533.

Xubin Ren and Chao Huang. EasyRec: Simple yet effective language models for recommendation. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pp. 17717–17732, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.894. URL https://aclanthology.org/2025.emnlp-main.894/.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In Thirty-seventh Conference on Neural Information Processing Systems, 2023. URL https://openreview.net/forum?id=Yacmpz84TH.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic LLM agent search in modular design space. In The Thirteenth International Conference on Learning Representations, 2025. URL https://openreview.net/forum?id=mPdmDYIQ7f.

Yunxiao Shi, Wujiang Xu, Tingwei Chen, Haoning Shang, Ling Yang, Yunfeng Wan, Zhuo Cao, Xing Zi, Dimitris N. Metaxas, and Min Xu. Agentselect: Benchmark for narrative query-to-agent recommendation. In Forty-third International Conference on Machine Learning, 2026. URL https://openreview.net/forum?id=4M5Kj2UqaM.

Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. In Findings of the Association for Computational Linguistics: ACL 2025, pp. 24497–24524, Vienna, Austria, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.findings-acl.1258. URL https://aclanthology.org/2025.findings-acl.1258/.

Meng Sun, Lin Li, Ming Li, Xiaohui Tao, Dong Zhang, Qing Xie, Peipei Wang, and Jimmy Xiangji Huang. A survey on bundle recommendation: Methods, applications, and challenges. ACM Comput. Surv., 58(11), May 2026. ISSN 0360-0300. doi: 10.1145/3802820. URL https://doi.org/10.1145/3802820.

Zhu Sun, Kaidong Feng, Jie Yang, Xinghua Qu, Hui Fang, Yew-Soon Ong, and Wenyuan Liu. Adaptive in-context learning with large language models for bundle generation. In Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '24, pp. 966–976, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704314. doi: 10.1145/3626772.3657808. URL https://doi.org/10.1145/3626772.3657808.

Teng Tu, Ai Li, Yunshan Ma, Shuo Xu, Xiaohao Liu, Haokai Ma, Liang Pang, and Tat-Seng Chua. Discrete diffusion for bundle construction. In The Fourteenth International Conference on Learning Representations, 2026. URL https://openreview.net/forum?id=dKyhgfe50H.

Renxi Wang, Xudong Han, Lei Ji, Shu Wang, Timothy Baldwin, and Haonan Li. Toolgen: Unified tool retrieval and calling via generation. In The Thirteenth International Conference on Learning Representations, 2025. URL https://openreview.net/forum?id=XLMAMmowdY.

Bin Wu, Arastun Mammadli, Xiaoyu Zhang, and Emine Yilmaz. Agentsearchbench: A benchmark for ai agent search in the wild, 2026a. URL https://arxiv.org/abs/2604.22436.

Bin Wu, Edgar Meij, and Emine Yilmaz. Beyond static toolsets: Self-evolving LLM tool agents via continual documentation adaptation. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), Findings of the Association for Computational Linguistics: ACL 2026, pp. 21519–21539, San Diego, California, United States, July 2026b. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1082. URL https://aclanthology.org/2026.findings-acl.1082/.

Renjun Xu and Yang Yan. Agent skills for large language models: Architecture, acquisition, security, and the path forward. In First Workshop on Agent Skills, 2026. URL https://openreview.net/forum?id=Er0p92BsmW.

Konstantin Yakovlev, Sergey Nikolenko, and Andrey Bout. Toolken+: Improving LLM tool usage with reranking and a reject option. In Findings of the Association for Computational Linguistics: EMNLP 2024, 2024. URL https://arxiv.org/abs/2410.12004.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In The Eleventh International Conference on Learning Representations, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R. Fung, Hao Peng, and Heng Ji. CRAFT: Customizing LLMs by creating and retrieving from specialized toolsets. In The Twelfth International Conference on Learning Representations, 2024. URL https://openreview.net/forum?id=G0vdDSt9XM.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. SkillWeaver: Web agents can self-improve by discovering and honing skills. arXiv preprint arXiv:2504.07079, 2025. URL https://arxiv.org/abs/2504.07079.

A AI Usage Statement

In this work, we used generative AI tools to assist with translation, dataset cleaning and reformatting. Additionally, we used these tools to improve the readability of the manuscript, summarize and analyze existing literature, and identify relevant references. We reviewed all AI-assisted work, including checking the references suggested by these tools. We take responsibility for the final content of this work, including all text, claims, and artifacts produced with the assistance of generative AI.

B Retrieval Design

The AgentSelect configuration in Appendix D.2 combines CF LLM retrieval, CF historical-bundle retrieval, and semantic individual-tool retrieval. SkillsBench uses the fixed inventory described in Appendix D.2.

CF retrieval. Let d_M and d_G be profiles of backbone $M \in \mathcal{M}$ and historical tool bundle $G \in \mathcal{G} \subseteq 2^{\mathcal{T}}$. Two trained two-tower retrievers use TF–IDF features and query-conditioned supervision from AgentSelect Shi et al. (2026):

$$s^{\mathrm{sp}}_{\mathrm{LLM}}(Q, M) = \left\langle g^{\mathrm{LLM}}_{\phi_Q}(\mathrm{TFIDF}(Q)), g^{\mathrm{LLM}}_{\phi_M}(\mathrm{TFIDF}(d_M)) \right\rangle,$$
$$s^{\mathrm{sp}}_{\mathrm{bun}}(Q, G) = \left\langle g^{\mathrm{bun}}_{\phi_Q}(\mathrm{TFIDF}(Q)), g^{\mathrm{bun}}_{\phi_G}(\mathrm{TFIDF}(d_G)) \right\rangle.$$

Historical bundles remain intact as retrieval evidence; their tools are also merged and deduplicated into the candidate tool inventory.

Semantic retrieval. Dense encoders match query text to component profiles. We use tuned EasyRec models Ren & Huang (2025) trained with AgentSelect supervision; the representations are further studied in AgentEmbedding Chen et al. (2026). For $u = M$ or t, with corresponding component encoder e_u, the score is $\langle e_Q(Q), e_u(d_u) \rangle$.

Context assembly. The selected rankings provide $\mathcal{M}_r(Q)$ and $\mathcal{T}_r(Q)$. Search roots drawn from $\mathcal{M}_r(Q)$ are ordered by the generator score (Section 3.4). Evidence $\mathcal{E}_r(Q)$ keeps component profiles, normalized scores, source branches, and retrieved historical bundles. On AgentSelect, the three retrieval outputs occupy separate context sections, with shortened profiles to fit the input budget. Appendix J measures the coverage of each branch.

C Constrained Generation and Bundle-Critic Reranking

Algorithm 1 gives the search and final ranking of Section 3.4. Generator scores guide expansion and pruning, and the critic ranks the completed configurations. Every expansion, including the root step, keeps at most P proposals scored by Eq. (4), and every pruning step keeps $W(\eta)$ of its η inputs (Eq. (6)).

D Experimental Setup Details

D.1 Datasets

AgentSelect. AgentSelect aggregates over 40 sources: 111,179 queries, 107,721 agents, and 251,103 interactions, with a catalog of 231 LLM-only entries and 12,099 tools. We evaluate on all 629 AgentSelect test queries.

SkillsBench. We sample 1,000 successful SkillsBench trajectories with verifier reward 1.0, which give 97 queries across 79 tasks and 615 query–agent pairs. The inventory contains 19

Algorithm 1 Constrained Generation with Optional Bundle-Critic Reranking

Require: Query Q, context $\mathcal{C}(Q)$, generator p_θ, optional critic $\widehat{U}_\psi$, proposal budget $P \geq 1$, pruning width $W(\cdot)$ (Eq. (6)), tool limit $L_{\max}$, output size $K \geq 1$
Ensure: Up to K distinct ranked configurations $\mathcal{H}_K(Q)$

Score each backbone in $\mathcal{M}_r(Q)$ by Eq. (4), keep the top P as root proposals, and keep the top $W(\eta_0)$ of these η_0 roots, each followed by τ_{sep}, as the initial beam
$\mathcal{F}_{\text{paths}} \leftarrow \emptyset$ (completed pool)
If $L_{\max} = 0$, complete the roots with the empty-bundle representation and move them to $\mathcal{F}_{\text{paths}}$
while the beam is nonempty do
  Enumerate the legal unused-tool and termination actions of each path
  Keep the top P legal actions of each path by Eq. (4), with the no-tool penalty $\lambda_\emptyset$ on $\tau_\emptyset$, and add τ_{end} of a nonempty bundle if it is not among them
  Extend the paths and update their scores by Eq. (5)
  Move terminated paths and paths with $L_{\max}$ tools to $\mathcal{F}_{\text{paths}}$
  Merge unfinished paths with the same (M, B) into the best ordering, and keep the top $W(\eta)$ of the η merged paths by $\bar{L}_\theta$
end while
Deduplicate $\mathcal{F}_{\text{paths}}$ by (M, B) into $\mathcal{F}(Q)$, keeping the best path score $\bar{L}_\theta(Q, A)$ of each configuration
if a critic is available then
  $S_{\text{final}}(Q, A) \leftarrow \widehat{U}_\psi(Q, A)$ for each $A \in \mathcal{F}(Q)$
else
  $S_{\text{final}}(Q, A) \leftarrow \bar{L}_\theta(Q, A) - \lambda_{\text{len}}|B|$
end if
return $\text{Top}_K(\mathcal{F}(Q); S_{\text{final}}(Q, \cdot))$

LLM identifiers, 155 skills, and 20 tool or action components. For SkillsBench, we split by task, training on 16 tasks (19 queries) and evaluating on the remaining 63 tasks (78 queries). Appendix E provides further details on dataset construction.

Execution datasets. Three further datasets record the execution outcome of each configuration. AgentSelect-Exec attaches execution feedback to 506 AgentSelect queries: for each query, we execute the ten configurations our method ranks highest, calling the selected backbone, simulating tool responses with an LLM, and scoring each run with a blinded LLM judge on completion (0–4), correctness (0–4), and constraint adherence (0–2). Building it consumes about 50.4M tokens, and Appendix F details the construction and cost. BFCL Patil et al. (2025) and SWE-Skills-Bench Han et al. (2026) release execution results for fixed configurations, which we use as native rewards, with each task forming one query. From BFCL we take the 240 executable Python tasks, each run by 18 backbones with the task's function menu. The reward is 1 if the calls pass the execution check and 0 otherwise. SWE-Skills-Bench (v2) contains 980 software-engineering tasks over 98 skills, each run under eight configurations, namely four model–harness pairs with the task's skill enabled or disabled. The reward is the fraction of hidden tests passed. We split both benchmarks by task into training, validation, and test tasks, 161/35/44 on BFCL and 690/60/230 on SWE-Skills-Bench, where the ten tasks of a skill stay in one split. Appendix G details both benchmarks, Section 5.1 uses AgentSelect-Exec, and Section 5.2 uses BFCL and SWE-Skills-Bench.

D.2 Implementation

Retrieval. On AgentSelect, we concatenate CF LLM, CF Historical Bundle, and Semantic Individual Tool Retrieval outputs as separate context sections, preserving bundle structure and shortening component profiles to fit the input budget. Text2Bundle and subsequent generative methods, including ours, share this evidence. Component-Ret uses LLM and individual-tool outputs, and Bundle-Ret uses LLM and historical-bundle outputs.

Llama Fine-Tuning. We fine-tune Llama-3-8B on AgentSelect for one epoch using LoRA rank 16, scaling factor 32, dropout 0.05, learning rate 10^{-4}, and effective batch size 32. We update LoRA parameters, token embeddings, and the output head, with a maximum

sequence length of 8,192 tokens. Loss covers only configuration output tokens. For SkillsBench, we further fine-tune the AgentSelect model on the 109 training configurations for three epochs with learning rate 5×10^{-5} and effective batch size 8, updating only LoRA parameters while freezing embeddings and the output head. Generator scores rank the SkillsBench configurations in Table 3 (Section 3.4). Section 5.2 uses a few-shot protocol.

Critic Training. We initialize the critic from AgentEmbedding Chen et al. (2026), using 1,000 AgentSelect training queries and 150 held-out validation queries. Stage 1 uses four epochs of teacher-supervised pairwise ranking plus a pointwise loss weighted by 0.1, and we select the checkpoint with the highest validation NDCG@10. Stage 2 refines with preference pairs for redundant additions, missing tools, and replacements, retaining teacher comparisons between LLMs. It uses a 0.1 pairwise margin without pointwise loss, and we select its checkpoint on the validation queries. Both stages update the encoder and scoring head with AdamW, using learning rates $(5 \times 10^{-6}, 10^{-4})$ and $(2 \times 10^{-6}, 2 \times 10^{-5})$ and effective batch sizes of 32 and 24 pairs, respectively. Stage 3 continues from the Stage 1+2 critic on the AgentSelect-Exec training split. It minimizes $\mathcal{L}^{(3)}_{\text{critic}}$ (Section 3.3). Training runs for four epochs with AdamW (weight decay 0.01), learning rates $(2 \times 10^{-6}, 10^{-4})$, and four queries per batch, selecting the checkpoint with the lowest validation loss. For BFCL and SWE-Skills-Bench, target fine-tuning likewise starts from the Stage 1+2 critic, trains for three epochs on each fraction of the training tasks with a task-balanced loss on native rewards, and selects the checkpoint on the validation tasks.

Search Configuration. On AgentSelect, we use validation-selected adaptive beam bounds $(b_{\min}, b_{\max}) = (1, 2)$, together with retain ratio $r_{\text{keep}} = 0.4$ in Eq. (6). We further set the proposal budget $P = 5$, $L_{\max} = 6$, and $K = 10$. Retrieval supplies at most 10 backbones in $\mathcal{M}_r(Q)$. On both AgentSelect and SkillsBench, the no-tool penalty is $\lambda_\emptyset = 10$. For example, when retrieval supplies 10 backbones, the generator proposes 5 roots and pruning keeps $W(5) = 2$ of them. In general, these settings give $W(\eta) = 1$ for $\eta \leq 2$ and $W(\eta) = 2$ otherwise, so at most two paths remain active after each pruning step.

E Construction of the SkillsBench Recommendation View

Trace extraction. We use runs from the public SkillsBench Li et al. (2026) leaderboard that use skills and pass the verifier with reward 1.0, and sample 1,000 of them (Appendix D.1). Each trace gives its backbone in the result record. A skill counts as used when the trace activates it or reads or executes its files. Tools keep their observed names, and harness actions form a separate category. The used skills and tools form the component set.

Queries and reference configurations. We group traces by source task and normalized initial query, merge duplicate model–component-set profiles, and retain up to ten distinct successful configurations per query. All references of a query have the same verifier reward and are unordered. Evaluation matches each candidate against any individual reference (Appendix I). All queries of a task fall in one split. The main recommendation experiment uses the train/test split in Table 1; Section 5.2 uses a separate few-shot protocol.

Inventory. The inventory contains the observed components and backbones. Each skill is described by its public skill document. Methods receive the query text and the descriptions of all catalog components.

F Construction of AgentSelect-Exec

AgentSelect-Exec attaches execution feedback to AgentSelect queries. We build it in four steps and report its token cost.

Candidate configurations. We take 506 AgentSelect queries. The backbones are 18 LLMs: GPT-6-Astra, GPT-6-Luna, GPT-6.1-Sol, GPT-6.1-Terra, DeepSeek-R1, DeepSeek-V3.2, DeepSeek-V4-Pro, DeepSeek-V4.1-Flash, Qwen3-8B, Qwen3-32B, Qwen3.6-35B-A3B, Qwen3.8-27B, Kimi-K2.6, GLM-4.5-Air, GLM-5.3, Seed-OSS-36B-Instruct, LongCat-2.0, and Hy4-preview. For each query, our method (Section 3) ranks configurations over these

backbones with the Stage 1+2 critic (Section 3.4), and we keep the ten highest-ranked configurations, 5,060 in total.

Execution. Each configuration is executed once. The harness calls the selected backbone through its provider API for at most 12 assistant turns and 24 tool calls, with up to 4,000 output tokens per turn. The configuration's tools are exposed with their documented schemas. A GPT-5.6-Luna simulator with a per-configuration state answers every tool call.

Judging. A GPT-5.6-Terra judge with low reasoning effort scores each run on completion (0–4), correctness (0–4), and constraint adherence (0–2). The judge sees the user request, the final answer, the observed evidence, submitted artifacts, and the termination status, and is blind to the recommending method, the backbone identity, and the rank. The mean total score over all runs is 7.38 out of 10.

Splits. We split the queries at random into 200 training, 106 validation, and 200 test queries, each with ten executed configurations (Table 1).

Cost. Table 5 reports the token usage, about 50.4M in total.

Table 5: Token usage for building AgentSelect-Exec.

Role	Model	Input tokens	Output tokens	Total tokens
Backbone	18 LLMs	19,777,574	4,413,132	24,190,706
Tool simulator	GPT-5.6-Luna	12,870,440	1,622,589	14,493,029
Judge	GPT-5.6-Terra	10,206,606	1,496,896	11,703,502
Total		42,854,620	7,532,617	50,387,237

G BFCL and SWE-Skills-Bench

Both benchmarks release execution results for a fixed set of configurations on every task. We treat each task as one query, its executed configurations as the candidate pool, and the released outcome of each run as its native reward. We split each benchmark by task into training, validation, and test tasks (Table 1).

BFCL. BFCL Patil et al. (2025) evaluates function calling: given a user request and function documentation, a model selects the functions to call and fills in their arguments, and each call is checked either against the expected structure or by executing it. Its first release contains 2,000 question–function–answer examples across 40 application domains. We use its 240 executable Python tasks: 100 single-function, 50 multiple-function (choosing among several functions), 50 parallel-call, and 40 parallel multiple-function tasks. We take the released results of 18 models: Claude-3-Haiku, Claude-3-Sonnet, Claude-3-Opus, Claude-3.5-Sonnet, Command-R+, FireFunction-v1 and v2, GPT-3.5-Turbo-0125, GPT-4-0613, GPT-4-1106-Preview, GPT-4-0125-Preview, GPT-4-Turbo-2024-04-09, GPT-4o-2024-05-13, GPT-4o-2024-08-06, GPT-4o-mini, Functionary-Small-v3.1 and v3.2, and Functionary-Medium-v3.1, giving 4,320 runs. Every model receives the task's full function menu, so a task's configurations differ only in the backbone. A run's reward is 1 if its calls pass the execution check and 0 otherwise. The 161/35/44 train/validation/test tasks hold 2,898/630/792 runs.

SWE-Skills-Bench. SWE-Skills-Bench Han et al. (2026) measures how much an agent skill helps in software engineering. Each skill is paired with a real code repository and ten requirement tasks, whose acceptance criteria are converted into hidden executable tests. Each task is run twice under the same repository, environment, and model configuration, once without the skill and once with the designated skill loaded. We use version 2 and its released results, which cover 98 skills and 980 tasks under four model–harness pairs: Codex with GPT-5.4-mini, Codex with Kimi-K2.6, Claude Code with Kimi-K2.6, and Claude Code with Claude Haiku 4.5. The skills span API development (26), developer tools (18), data science and machine learning (17), deployment and operations (14), analytics and monitoring (12), and security and testing (11). Each task thus has eight configurations, and each of the 7,840 runs is rewarded by the fraction of hidden tests passed, in $[0, 1]$. All ten tasks of a skill fall into the same split, giving 69/6/23 skills, 690/60/230 tasks, and 5,520/480/1,840 runs for training, validation, and test. On both benchmarks, RAG-Qwen3.8-27B receives

the task and the full descriptions of all candidate resources and returns a strict ranking of the executed pool.

H Baseline Adaptations and Output Protocols

H.1 Retrieval-Based Composition

Component-Ret. We independently retrieve LLMs via CF and tools via semantic retrieval. The top-10 LLMs are paired with the same six highest-ranked tools, forming ten configurations ranked by LLM CF score.

Bundle-Ret. Separate CF models retrieve one top-ranked LLM and ten historical tool bundles. Each intact bundle is paired with that LLM and ranked by bundle CF score.

H.2 Text2Bundle

We adapt Text2Bundle Hu et al. (2025a) by mapping items to tools and using the query as the request. We omit user-ID embeddings and personalization rewards. Queries and tool descriptions use AgentEmbedding Chen et al. (2026), an EasyRec Ren & Huang (2025) model tuned for agent-resource retrieval. We retain the Transformer state encoder, action-relation modeling, and learned termination action.

Pairwise bundle-completion pretraining precedes Double-DQN training with experience replay. The action pool deduplicates tools from retrieved historical bundles and semantic retrieval. Missing target tools are added to the pool during training, and validation and inference use the retrieved candidates.

The CF-selected LLM stays fixed while beam search extends the single-output policy to Top-K tool sets. Paths are ranked by negative cumulative action regret: each increment is the chosen action's Q-value minus the highest valid-action Q-value. The greedy path scores zero and comes first. Duplicate tool sets keep their best path, and a size cap ends search.

H.3 DDBC-Adapt

We adapt DDBC Tu et al. (2026), which uses masked discrete diffusion and residual quantization. DDBC-Adapt operates over a component vocabulary built from the training split. Frozen MPNet embeddings of LLM and tool descriptions, PCA, and residual k-means replace its representation learner and quantizer. An extra code separates components with identical quantized codes.

A query-conditioned denoiser jointly generates one LLM and a tool set with typed boundaries. Training masks component codes, preserves structural tokens, and randomizes tool order. Decoding retains absorbing masks, enforces code compatibility, and excludes duplicate tools, so it can generate new combinations of the components in its training vocabulary.

Task-trained semantic adapters estimate query–component relevance over the training vocabulary. Relevance probabilities are summed over legal components sharing each code value, and the resulting log masses augment denoiser logits. Distinct valid configurations are ranked by LLM and summed tool relevance, weighted by the predicted probability of their tool count.

H.4 Adapted Tool Retrieval and LLM Routing

ToolGen Wang et al. (2025), ToolWeaver Fang et al. (2026), PORTS Molfetta et al. (2025), AnyTool Du et al. (2024), and RouteLLM Ong et al. (2025) each address one part of agent resource use and do not output a complete configuration. ToolGen selects the next tool for a fixed backbone within its agent loop, ToolWeaver generates tools as hierarchical code sequences that capture tool co-usage, PORTS aligns a dense tool retriever with LLM preferences over tool selection, AnyTool collects individual APIs through hierarchical LLM agents, and RouteLLM routes a query between a strong and a weak LLM without tools. We adapt

each method to produce configurations while keeping its core mechanism, and pair its output with the remaining module of Component-Ret.

ToolGen. Following ToolGen, we first train tool-token memorization on 47,800 document-to-token examples, one per tool in our catalog. We then fine-tune the model to generate the complete tool sequence of each training configuration. Beam search produces candidate tool sets, each paired with the CF backbone ranking, and the fusion weight of tool and backbone scores is selected on 616 validation queries.

ToolWeaver. We encode each tool document with a frozen MPNet encoder and quantize it with a residual-quantized autoencoder into two code levels of 1,024 entries each. A graph term in the quantizer pulls together tools that co-occur in training configurations, so co-usage shapes the codes, and every catalog tool keeps a unique two-token code. The language model starts from the ToolGen memorization checkpoint, and each new code token is initialized from the memorized tokens of the tools it covers. With LoRA, we first align queries with the codes of single tools from the training configurations and then fine-tune the model to generate the codes of complete training bundles. Beam search over valid codes produces bundles of one to six tools, which are combined with the CF backbone ranking as in ToolGen, with weights selected on validation queries.

PORTS. We split each training configuration into query–tool positives, and a frozen Meta-Llama-3-8B scores the identifier of the positive tool when conditioned on each candidate tool document. A BGE-base encoder, shared by queries and tool documents, learns to match this preference distribution over the positive and three hard negatives, together with an odds-ratio preference term that ranks the positive above each negative. Hard negatives are the encoder's nearest non-positive tools and are refreshed during training. At test time, the top two retrieved tools form the bundle, and its size and the checkpoint are selected by tool F1 on validation queries. The bundle is paired with the CF backbone ranking.

AnyTool. We keep AnyTool's category–service–API navigation and solvability check, use the collected APIs as the tool set, and pair it with the CF backbone ranking.

RouteLLM. We replace the backbone ranking of Component-Ret with RouteLLM's matrix-factorization (MF) router, trained on backbone positives with TF-IDF query features, and keep the six semantically retrieved tools.

Table 6 lists each method's tool and backbone sources, and Table 2 gives the results.

Table 6: Sources of tools and backbones in the adapted baselines.

Method	Tools from	Backbones from
Component-Ret	semantic retrieval	CF ranking
PORTS	LLM-aligned dense retrieval	CF ranking
AnyTool	hierarchical API retrieval	CF ranking
RouteLLM	semantic retrieval	MF router
ToolGen	generated tool sequence	CF ranking
ToolWeaver	generated hierarchical tool codes	CF ranking
Ours	joint generation	joint generation

I Evaluation Metric Definitions

Each method returns up to ten ranked configurations per query. Let (M_r, B_r) be the backbone and tool set at rank r, and (M^*, B^*) a reference configuration. The backbone and each tool count as one component (on SkillsBench, tools include skills and actions). A query can have several references, and every metric uses the reference that a candidate matches best. The component recall c_r and precision p_r at rank r give the recall-based metrics and

the rank-discounted scores:

$$c_r = \frac{\mathbb{I}[M_r = M^*] + |B_r \cap B^*|}{1 + |B^*|}, \qquad p_r = \frac{\mathbb{I}[M_r = M^*] + |B_r \cap B^*|}{1 + |B_r|},$$

$$\text{ToolR@1} = \frac{|B_1 \cap B^*|}{|B^*|}, \qquad \text{CompR@1} = c_1,$$

$$\text{RDCR@10} = \frac{\sum_{r=1}^{10} w_r\, c_r}{\sum_{r=1}^{10} w_r}, \qquad \text{RDCP@10} = \frac{\sum_{r=1}^{10} w_r\, p_r}{\sum_{r=1}^{10} w_r}, \qquad w_r = \frac{1}{\log_2(r+1)}.$$

Missing ranks count as zero. Extra tools do not lower RDCR but lower RDCP.

The complete-recovery metrics ask whether a reference is fully recovered. Tool-Hit@10 is 1 if any of the top ten configurations contains all tools of a reference, regardless of the backbone. CR-Hit@k also requires the reference backbone and checks the top k configurations ($k = 1, 10$). CR-MRR@10 is the reciprocal rank of the first such hit, and 0 if there is none in the top ten. All of them allow extra tools. Every score is averaged over queries and reported on a 0–100 scale.

J Retrieval Analysis

These studies measure the coverage of each retrieval branch in Appendix B.

Evaluation protocol. Tool retrieval uses 1,000 unique AgentSelect Part II queries sampled at random. Macro recall averages the fraction of reference tools retrieved per query, and the all-covered rate requires their complete coverage. For historical bundles, these measures use the merged tool set. Exact hit requires one retrieved bundle to equal the reference tool set, while complete hit permits additional tools. The cutoff K counts tools or bundles, respectively, and Avg. Tools counts the unique retrieved tools after merging. LLM retrieval uses all 23,073 unique Part I queries, each with multiple ranked reference LLMs. Precision (P), recall (R), MRR, and MAP use binary relevance. Hit requires any relevant LLM, and the all-covered rate (All) requires every reference LLM. For nDCG, reference rank r receives gain $1/\log_2(r+1)$.

Table 7: Individual-tool retrieval (%).

K	Macro Recall	All Covered
5	39.95	21.10
10	48.30	30.20
15	51.93	34.00
20	55.74	38.00
50	65.46	50.60

Tool retrieval. Individual-tool retrieval covers few reference tools: increasing K from 5 to 50 raises macro recall from 39.95% to 65.46% and the all-covered rate from 21.10% to 50.60% (Table 7). Historical bundles cover far more. At $K = 5$, they yield 5.52 unique tools on average and 88.09% macro recall (Table 8). Merging adds only 0.50 percentage points of complete coverage beyond a single bundle's complete hit. Increasing K to 20 raises the merged all-covered rate to 90.00%, with 17.21 tools on average.

LLM retrieval. At $K = 10$, CF LLM retrieval has a 97.95% hit rate and 0.8565 MRR (Table 9). Increasing K to 50 mainly recovers additional relevant models: recall rises from 57.07% to 92.34%, while the hit rate rises to 99.91%.

Table 8: CF tool-bundle retrieval (%, except Avg. Tools).

K	Exact Hit	Complete Hit	Macro Recall	All Covered	Avg. Tools
5	81.20	85.10	88.09	85.60	5.52
10	85.00	87.60	89.49	88.20	9.34
15	86.70	88.50	90.10	89.20	13.19
20	88.20	89.30	90.69	90.00	17.21

Table 9: CF LLM retrieval (P, R, Hit, and All in %).

K	P	R	Hit	All	MRR	MAP	nDCG
5	69.43	35.43	94.75	0.00	0.8521	0.6387	0.6327
10	56.21	57.07	97.95	3.61	0.8565	0.4877	0.6244
15	45.30	68.81	98.99	13.15	0.8574	0.5519	0.6744
20	37.57	76.01	99.45	23.00	0.8576	0.5816	0.7011
50	18.29	92.34	99.91	64.63	0.8578	0.6250	0.7512

K Sensitivity to Beam Bounds and Critic Weight

Validation protocol. With the generator and critic of Table 2, we test adaptive beam bounds $(b_{\min}, b_{\max})$ of $W(\eta)$ in Eq. (6) over $\{(1,2),(2,4),(3,5),(4,7),(6,10)\}$ and critic weights $\alpha_c \in \{0, 0.5, 1.0, 1.5, 2.0\}$ on validation queries, holding other settings fixed. To see how much the critic should count, this study ranks the final candidates by a weighted sum of the critic score (weight α_c) and the generator score (weight 0.15), both standardized within the candidate pool, minus 0.1 per tool. At $\alpha_c = 0$, ranking uses generator scores alone, and a large α_c approaches ranking by the critic alone.

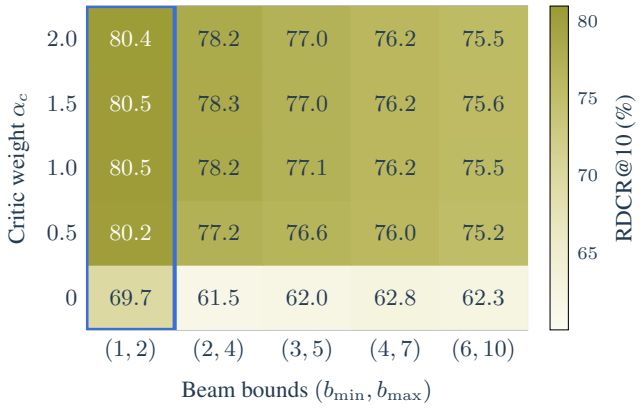


Figure 5: Validation RDCR@10 over beam bounds and critic weights. The outlined column is the bounds (1, 2) that we use.

Beam bounds. Beam bounds matter more than the critic weight. Bounds (1, 2) give the highest RDCR@10 at every tested weight (Figure 5). Without the critic, widening the bounds from (1, 2) to (4, 7) lowers RDCR@10 from 69.7% to 62.8%. With the critic, RDCR@10 falls by about 5 points from (1, 2) to (6, 10), so we use (1, 2).

Critic weight. Critic reranking raises RDCR@10 at every beam setting, e.g., from 69.7% to 80.2% at (1, 2) and from 62.8% to 76.0% at (4, 7). Once the critic is used, its weight matters little. Across positive weights, RDCR@10 varies by at most 1.1 points per beam setting, and at (1, 2) it spans 80.2–80.5%. The largest weight, $\alpha_c = 2.0$, where the critic dominates the ranking, reaches 80.4%, on par with $\alpha_c = 0.5$. Generator scores therefore add little once the critic ranks the candidates, and our final ranking uses the critic alone (Section 3.4).

L Output Validity of Constrained Decoding

Table 10 compares single outputs of free and constrained single-path decoding.

Table 10: Decoding validity (%).

	AgentSelect		SkillsBench	
	Free	Constr.	Free	Constr.
Valid	97.0	100.0	48.7	100.0
Incomplete	0.0	0.0	3.8	0.0
Malformed	0.0	0.0	43.6	0.0
Unknown name	3.0	0.0	3.8	0.0
Same config.	92.0		39.7	

An output is valid if it ends within 256 new tokens, follows the configuration format, and names only catalog components. On SkillsBench, the AgentSelect-trained generator is applied without target training. Same configuration is the share of queries on which both decodings return the same backbone and tool set. Free generation is not always valid. On AgentSelect, 3% of its outputs name components outside the catalog. On SkillsBench, about half of its outputs are invalid, mostly malformed, because the generator has not seen the target format. Constrained decoding keeps every output valid in both settings, and the two decodings agree far less often on SkillsBench than on AgentSelect.

M Online Recommendation and Execution

This section details the study of Section 5.3. Appendices M.1 and M.2 give the setup. Appendices M.3 and M.4 cover SkillsBench: the online execution of the top-ranked configurations, and an offline ranking of the official SkillsBench runs. Appendix M.5 covers AgentSelect, including our full top-10 lists.

M.1 Queries, resources, and candidate generation

The study uses 25 AgentSelect test queries and five SkillsBench test queries. The SkillsBench queries cover travel planning, dialogue parsing, invoice fraud detection, bike rebalancing, and software dependency auditing.

All queries within a dataset share one resource inventory (Table 11).

Table 11: Resources and runtime budgets of the online study.

Setting	AgentSelect	SkillsBench
Queries	25	5
Backbone candidates	14	14
API endpoints / skills	66 / 16	66 / 16
Maximum components	6	10
Assistant turns / tool calls	12 / 24	24 / 48
Output tokens per response	4,000	6,000

All 14 backbone models form the candidate set $\mathcal{M}_r(Q)$, and the generator keeps four root proposals. Each root receives one component expansion before the first pruning step, and each active path considers up to ten legal component proposals. Pruning uses the settings of Section 4.2.

On AgentSelect, the Stage 3 critic ranks the completed configurations. On SkillsBench, the generator score $\bar{L}_\theta$ ranks them. Each query receives ten configurations.

M.2 Execution and scoring

Of the 66 APIs, 11 call public services or a local adapter, and the other 55 are simulated by GPT-5.6-Luna following MirrorAPI Guo et al. (2025). The simulator sees the endpoint description, schema, and arguments, and is blind to the recommendation rank, score, and backbone. Each configuration runs at temperature zero within the budgets of Table 11.

On SkillsBench, the selected skills are mounted in network-isolated containers, and the original hidden tests check the exported artifacts. On AgentSelect, the selected skills are given to the backbone as textual instructions.

GPT-5.6-Terra judges each run from the user request, the outputs, the tool observations, and any verifier results, blind to the rank, scores, and backbone. It assigns 0–4 points for task completion, 0–4 for correctness, and 0–2 for constraint satisfaction.

M.3 SkillsBench with Native Verifiers

Setting. The five tasks are those of Appendix M.1. The baseline is RAG-Qwen3.8-27B. Each method recommends an execution LLM and a set of skills. Ours selects Qwen3.8-27B for all five tasks, and RAG-Qwen3.8-27B selects DeepSeek-V4-Pro. Two tests follow. The first executes each recommended configuration once. The second runs the skills of each method three times with a fixed LLM backbone, GPT-5.6-Luna, which isolates skill selection. The native verifier decides pass or fail, and GPT-5.6-Terra scores each run with the rubric of Appendix M.2. Execution tokens are counted in the first test.

Table 12: Execution on SkillsBench.

	Recommended configuration (one run)						Fixed LLM backbone (three runs)			
	Verifier		Judge (0–10)		Tokens (K)		Passes		Judge (0–10)	
Task	Ours	RAG	Ours	RAG	Ours	RAG	Ours	RAG	Ours	RAG
Travel planning	✓	✓	8.0	7.5	205	417	0/3	0/3	5.80	5.17
Dialogue parsing	✓	✓	10.0	10.0	76	342	3/3	2/3	10.00	9.03
Invoice fraud	✓	✗	10.0	8.0	118	135	3/3	0/3	10.00	8.67
Bike rebalancing	✗	✗	5.0	0.0	688	569	0/3	0/3	4.50	3.17
Dependency audit	✓	✓	10.0	10.0	47	129	3/3	3/3	9.83	9.33
All tasks	4/5	3/5	8.6	7.1	1,134	1,593	9/15	5/15	8.03	7.07

Results. Table 12 gives the per-task results, where RAG denotes RAG-Qwen3.8-27B and bold marks the better value of each pair. With the recommended configurations, ours passes the verifier on four of five tasks and RAG-Qwen3.8-27B on three. Ours also has a higher mean judge score (8.6 versus 7.1) and uses 29% fewer tokens. On invoice fraud detection, only ours passes. On bike rebalancing, both fail, and ours completes part of the task (5.0 versus 0.0). With the fixed LLM backbone, our skills pass 9 of 15 runs, compared with 5, and score higher on all five tasks (8.03 versus 7.07).

M.4 Native execution on official SkillsBench runs

Data. We use the official SkillsBench runs released with the benchmark, executed in OpenHands with three backbones (Opus 4.8, Gemini 3.5 Flash, and MiniMax M3), each with and without the task's skills. We split the tasks into 36 training, 14 validation, and 21 test tasks, and report results on the test tasks. A configuration is a backbone with or without skills; its reward is the mean hidden-test reward over its 1–3 repeated runs (93 configurations, 255 runs). The Stage 1+2 critic, trained on AgentSelect, scores each configuration from the task request, the backbone, and the skills.

Baselines and metrics. The backbone prior ranks configurations by their backbone's mean reward on the training tasks; the skill prior ranks skill-enabled configurations first. Top-1 is the reward of the top-ranked configuration, averaged over tied configurations. Pairwise concordance counts correctly ordered within-task pairs with different rewards, with predictor ties counted as 0.5; same-backbone pairs differ only in whether skills are enabled, and in both-succeed pairs both configurations receive a nonzero reward. nDCG@3 uses linear gains. Intervals are 95% bootstrap intervals over tasks.

Table 13: Official SkillsBench test runs (rewards and concordance in %).

Ranker	Top-1 reward	nDCG@3	Pairwise concordance			Spearman ρ
			All	Same backbone	Both succeed	
Random	41.9 [28.4, 56.2]	–	50.0	50.0	50.0	0.00
Backbone prior	45.2 [26.9, 63.5]	67.3	56.4	50.0	66.2	0.16
Skill prior	49.0 [35.2, 63.2]	81.9	61.6	68.0	56.8	0.29
Ours (Stage 1+2 critic)	54.0 [36.5, 69.8]	82.1	69.8	72.0	73.0	0.36
Oracle	72.1 [57.8, 84.8]	100.0	100.0	100.0	100.0	–

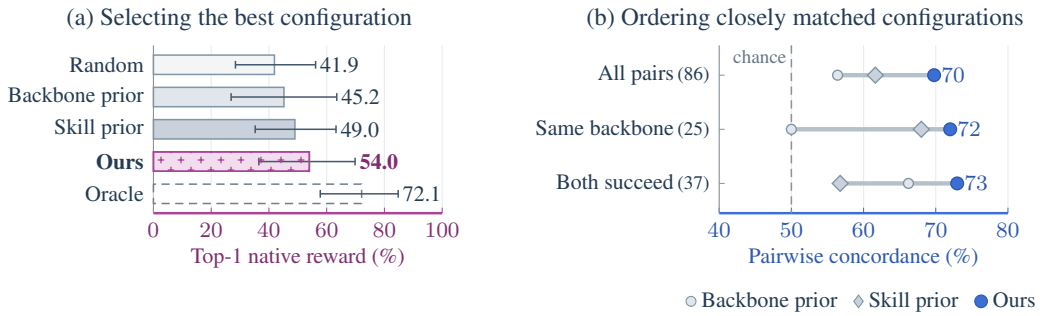


Figure 6: Native execution on official SkillsBench runs. (a) Top-1 native reward with 95% task-bootstrap intervals. (b) Within-task pairwise concordance on pairs with different rewards (pair counts in parentheses).

Results. The critic's top choice reaches a native reward of 54.0%, compared with 41.9% for random selection, 45.2% for the backbone prior, and 49.0% for the skill prior, closing 40% of the gap between random selection and the oracle (72.1%, Figure 6a and Table 13). Within tasks, it orders 69.8% of configuration pairs correctly, above both priors. The advantage holds for the hardest pairs: 72.0% when both configurations share a backbone and differ only in skills, and 73.0% when both succeed with a nonzero reward.

M.5 AgentSelect

Setting. The baselines are RAG-Qwen3.8-27B and Bundle-Ret. For ours, we use the top-ranked configuration of the full method (Stages 1–3). Each top-ranked configuration runs and is judged as in Appendix M.2. We also record turns and wall time per query, total tokens, and backbone API cost.

Table 14: Execution on AgentSelect.

Method	Execution quality (↑)						Execution cost (↓)			
	Total	Compl.	Corr.	Constr.	≥ 8 (%)	Answers	Turns	Time (s)	Tokens (K)	Cost (CNY)
Bundle-Ret	7.10	2.58	2.86	1.66	48	22	3.96	106.2	257.6	0.62
RAG-Qwen3.8-27B	7.40	2.82	3.00	1.58	56	23	3.04	72.1	231.4	0.86
Ours	8.72	3.32	3.52	1.88	72	25	3.04	42.2	180.6	0.50

Results. Table 14 reports the judge's total score and its three parts, the share of queries scoring at least 8, and the number of queries that end with a final answer (Answers). Ours has the highest mean score (8.72, compared with 7.40 and 7.10) and the highest score in all three parts. 72% of its queries score at least 8, compared with 56% and 48%. It uses 3.04 assistant turns per query, as many as RAG-Qwen3.8-27B and fewer than Bundle-Ret (3.96). Ours returns a final answer on all 25 queries, while RAG-Qwen3.8-27B and Bundle-Ret miss two and three. It also takes the least time (42 seconds per query, compared with 72 and 106) and uses 22% to 30% fewer tokens.

Ranking within the top-10 list. For each of the 25 queries, our full method returns a top-10 list, and we execute all ten configurations in the same harness (250 runs). The list is

uniformly strong: the mean score is 8.59, every rank averages at least 8.16, and 73.6% of runs score at least 8 (Figure 7). Quality concentrates at the top of the list. The 3-rank moving average falls from 8.67 (ranks 1–3) to 8.45 (ranks 8–10). From ranks 1–3 to 4–7 to 8–10, the share of runs scoring at least 8 falls from 77.3% to 76.0% to 66.7%, and the share below 6 rises from 5.3% to 6.0% to 9.3%. Per query, ranks 1–3 outscore ranks 8–10 in 15 queries and trail in 6 (4 ties; one-sided sign test, $p = 0.039$).



Figure 7: Executing the top-10 list of ours on 25 AgentSelect queries (250 runs). (a) Execution score by rank: 3-rank moving average with a 95% query-bootstrap band; rings are per-rank means; the dash-dotted line is the mean over all ten. (b) Share of runs per score band in each rank group. (c) Per-query mean of ranks 1–3 minus ranks 8–10, sorted.

N Case Study

We examine a query from the end-to-end evaluation to illustrate how tool value depends on its companion resources. The user requests a domain-similarity check, NS records, and a nameserver IP. We fix DeepSeek-V4-Pro and the similarity tool S, then vary DNS Lookup D and NS Lookup N. Tool contributions depend on the bundle. The raw critic scores for S, $S{+}D$, $S{+}N$, and $S{+}D{+}N$ are -5.56, 0.11, 1.42, and 3.14 (Figure 8). Adding D to S yields a 5.67 gain, but only 1.72 when N is already present, indicating diminishing returns from overlapping lookup capabilities. An additive extrapolation would predict 7.09 for the full bundle rather than the observed 3.14, showing a clear non-additive interaction. Critic ranking agrees with execution quality. The end-to-end judge scores of S, $S{+}D$, $S{+}N$, and $S{+}D{+}N$ are 6.5, 9, 10, and 9.5. The critic ranks the two bundles with N first and S alone last, as execution does. The three-tool agent does not call N, and $S{+}N$ achieves the best execution with fewer tools. This case illustrates both context-dependent tool value and the agreement between the critic's ranking and observed execution quality.

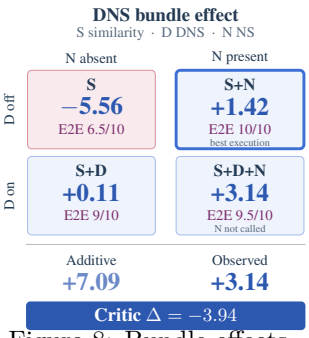


Figure 8: Bundle effects.