

SD-ICL: Stage-Decomposed In-Context Learning for Tool Use in LLM Agents

Kangci Xiong* Xiaohan Ge* Yangyang Qin Fusheng Jin[†]

Beijing Institute of Technology, Beijing, China

1120222342@bit.edu.cn 1120222210@bit.edu.cn

3220251482@bit.edu.cn jfs21cn@bit.edu.cn

*Equal contribution. [†]Corresponding author: jfs21cn@bit.edu.cn

Abstract

Large language models (LLMs) augmented with external tools have demonstrated strong capabilities in API selection, invocation, and reasoning over tool outputs. However, existing training-free tool-use methods usually rely on inference-time context construction to guide the entire tool-use process with a unified input, including tool descriptions, task prompts, demonstrations, trajectories, or planning hints. Tool-use decisions place different emphasis on this contextual evidence: tool/API selection emphasizes task-API correspondence, while call generation additionally uses API schemas, local call transitions, and intermediate observations. We present SD-ICL, a Stage-Decomposed In-Context Learning method that decomposes context construction into tool selection and call generation stages. SD-ICL first retrieves tool selection demonstrations by combining task semantics with tool set relevance and constructs an explicit tool selection state. It then uses this state to retrieve call generation demonstrations through API-set similarity and adjacent call relations, organizing complementary context for the executor. Experiments show that SD-ICL improves task completion and preference quality across the evaluated tool-use benchmarks and model settings.

1 Introduction

LLM agents are increasingly used in tool learning, where models solve user tasks by selecting external tools, invoking APIs, and using tool-returned results. By interacting with external resources such as APIs, databases, and software services, agents can extend their capabilities beyond parametric knowledge alone. Reasoning paradigms such as ReAct (Yao et al., 2023) organize tool use as an iterative loop of reasoning, actions, and observations, allowing the model to adjust subsequent decisions according to intermediate tool results. As tool-use tasks involve larger tool sets, multiple API calls,

and structured tool interfaces, agents must not only decide which tools to use, but also organize invocation order, generate executable arguments, and use observations to continue the task-solving process (Schick et al., 2023; Qin et al., 2024; Patil et al., 2024).

Existing training-free tool-use methods commonly rely on context assembled at inference time from tool descriptions, task prompts, retrieved demonstrations, historical trajectories, planning hints, and other auxiliary information. They require no parameter updates and remain applicable when fine-tuning is unavailable or tool environments change. Most organize this material as a unified input for the full tool-use process, even though the same example may be useful for different reasons at different points in execution. Different decisions place different emphasis on the available evidence. Identifying relevant APIs emphasizes task intent and API functionality; realizing a call trajectory also depends on invocation patterns, API schemas, parameter constraints, and intermediate observations. The distinction is one of emphasis rather than strict separation, and the relevant evidence can overlap across stages.

This mismatch can weaken guidance at either stage. Context selected to identify relevant tools may give little attention to call formatting or argument generation; trajectories with useful call patterns may offer limited help in selecting tools from the current candidate set. A semantically similar task can invoke a different API set, and a task with different wording can still exhibit a useful invocation pattern. Retrieving complete historical trajectories only by task semantic similarity therefore provides the same form of evidence to every stage of tool use.

We propose SD-ICL, or Stage-Decomposed In-Context Learning, which aligns demonstration retrieval with this staged decision structure. Stage 1 jointly considers task semantics and tool set rel-

evance to retrieve tool selection demonstrations. From them, it constructs a tool selection state containing selected tools, selected APIs, a selection rationale, and candidate adjacent call relations. Stage 2 uses the selected APIs and adjacent call relations to retrieve call generation demonstrations by matching historical API sets and adjacent call relations. Together, the two sets of demonstrations provide complementary context to the executor.

Experiments on ToolBench (Qin et al., 2024) and APIBench (Patil et al., 2024) with DeepSeek-V3.2 and Qwen3.5-Flash show that SD-ICL consistently improves tool-use performance over demonstration retrieval based on semantic similarity. SD-ICL achieves average pass rates of 71.8% and 83.9% under DeepSeek-V3.2, and 74.4% and 86.5% under Qwen3.5-Flash, on ToolBench and APIBench, respectively. Component ablations further support the staged context design.

Our contributions are summarized as follows:

1. We propose SD-ICL, a staged context construction method for in-context tool use. It separates tool selection from call generation and connects the two stages through explicit intermediate variables.
2. We design demonstration retrieval and context organization for each stage. Tool selection examples emphasize task semantics and tool set relevance; call generation examples emphasize API-set similarity and adjacent call relations guided by the Stage 1 tool selection state.
3. Experiments on ToolBench and APIBench under DeepSeek-V3.2 and Qwen3.5-Flash, together with an additional evaluation using Qwen3.5-9B, show improvements in task completion and preference quality. Ablations support the staged context design.

2 Related Work

Tool learning and tool-use agents. Tool learning enables LLMs to solve tasks by invoking external APIs, databases, retrieval systems, or software services. ReAct (Yao et al., 2023) interleaves reasoning, actions, and observations, making later actions responsive to intermediate tool results. ToolLLM, API-Bank, and Gorilla distinguish API retrieval or identification, path construction, and API calling across large-scale real APIs, dialogue, and machine learning APIs (Qin et al., 2024; Li et al., 2023; Patil et al., 2024). These distinctions motivate context

that gives appropriate emphasis to API scope and call generation at different stages. GPT4Tools extends tool learning to multimodal settings (Yang et al., 2023).

In-context demonstrations for agents and tool use. ICL enables LLMs to adapt from demonstrations without parameter updates (Brown et al., 2020), and demonstration selection and organization substantially affect model behavior (Rubin et al., 2022; Ye et al., 2023). Learned retrievers and CEIL-style methods select examples for a single prompt, primarily in standard input-output settings. Agent trajectories add actions, observations, and dependencies between calls, so their usefulness may depend on more than similarity between task descriptions. Synapse (Zheng et al., 2024) retrieves complete trajectories as exemplars for computer control, and TRAD (Zhou et al., 2024) retrieves thoughts step by step to reduce irrelevant trajectory context. API-based agents can also learn API functionality directly from in-context demonstrations (Patel et al., 2025). SD-ICL organizes tool-use demonstrations by decision stage: an explicit tool selection state supplies selected APIs and adjacent call relations for retrieving call generation demonstrations.

Training-free tool-use planning and context construction. Training-free methods guide tool use through prompts, feedback, search, or planning. Reflexion, ToolLLM’s depth-first search decision tree (DFSDT), ToolChain*, and Tool-Planner improve execution through feedback, tree-based exploration, A* search, or toolkit-level replanning (Shinn et al., 2023; Qin et al., 2024; Zhuang et al., 2024; Liu et al., 2025). SD-ICL operates before execution and leaves the executor unchanged. It reorganizes successful trajectories into demonstrations for tool selection and call generation, emphasizing API scope in Stage 1 and invocation patterns in Stage 2.

3 SD-ICL: Stage-Decomposed In-Context Learning

SD-ICL is a training-free method that aligns context construction before execution with the staged decisions in tool use. Tool selection places greater emphasis on matching task intent to candidate tools and APIs. Call generation also uses API call formats, argument constraints, observations, and local call transitions. An explicit tool selection state connects the two stages, as shown in Figure 1.

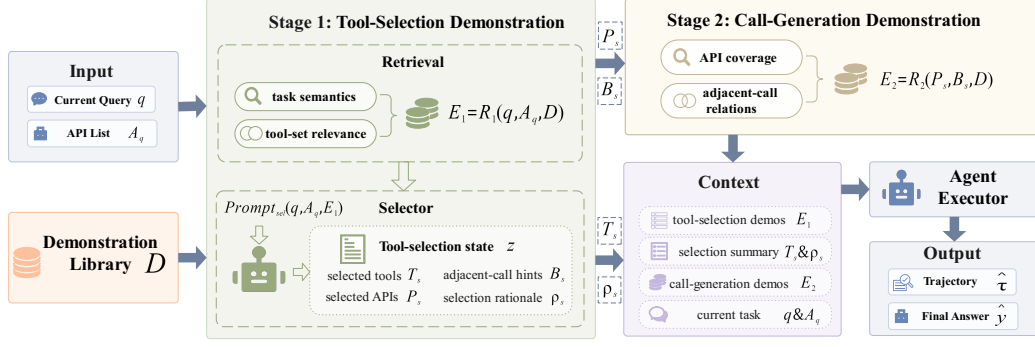


Figure 1: Overview of SD-ICL. SD-ICL constructs the pre-execution context in two stages: it first retrieves tool selection demonstrations and produces a tool selection state, then uses the selected APIs and adjacent call relations to retrieve call generation demonstrations. The staged context is finally provided to an unchanged executor.

3.1 Problem Setup and Overview

Given a task q and its accessible API/tool set $\mathcal{A}_q = \{a_1, \dots, a_n\}$, a tool-use agent must select relevant APIs, generate arguments, interact with tools, and produce a final answer. We represent the fixed base executor as

$$(\hat{\tau}, \hat{y}) = F_\theta(q, \mathcal{A}_q, C),$$

where $\hat{\tau}$ is the generated tool interaction trajectory, $\hat{y}$ is the final answer, and C is the context supplied before execution. The parameters, action space, and tool interaction loop of F_θ remain fixed; SD-ICL changes the construction of C .

SD-ICL builds a demonstration library $\mathcal{D} = \{d_i\}_{i=1}^M$ from historical successful trajectories and extracts tool sets, API sets, and adjacent call relations. Each trajectory is available in a compact tool selection view and a complete call generation view. These representations support construction of the tool selection state, retrieval of call generation demonstrations, and assembly of the staged context.

3.2 Explicit Tool Selection Signal Construction

In tool-use tasks, a single homogeneous context may distribute attention unevenly across tool identification, call formatting, and invocation-path organization. SD-ICL therefore first retrieves demonstrations that emphasize task–tool/API correspondence and generates an explicit tool selection state before execution.

Given the current task q , the candidate API/tool set $\mathcal{A}_q$, and the demonstration library $\mathcal{D}$, Stage 1 retrieves tool selection demonstrations:

$$E_1 = R_1(q, \mathcal{A}_q, \mathcal{D}),$$

where $R_1(\cdot)$ denotes the retrieval function for the tool selection stage, and E_1 denotes the retrieved first-stage demonstrations. Unlike retrieval that only relies on task-text similarity over complete trajectories, Stage 1 jointly considers task semantic similarity and tool set relevance. For a historical demonstration d_i , let its task description be q_i and its actually used tool set be T_i . The candidate tool set for the current task is denoted as:

$$T_{\text{cand}} = \Phi_T(\mathcal{A}_q),$$

where $\Phi_T(\cdot)$ denotes the process of extracting and normalizing tool names from the candidate API/tool set. The fused score for Stage 1 is defined as:

$$\begin{aligned} \text{Score}_1(q, d_i) &= \lambda_1 \bar{s}_{\text{text}}(q, q_i) \\ &\quad + (1 - \lambda_1) \bar{s}_{\text{tool}}(T_{\text{cand}}, T_i), \end{aligned}$$

where $\bar{s}_{\text{text}}$ and $\bar{s}_{\text{tool}}$ denote the task semantic similarity after candidate pool normalization and tool set relevance after candidate pool normalization, respectively, and λ_1 controls their relative contribution after normalization. The normalization procedure is described in Appendix B.4. We denote the number of demonstrations for the tool selection stage as M_1 , and select the top M_1 demonstrations as the first-stage demonstrations:

$$E_1 = \text{Top}_{M_1}(\{(d_i, \text{Score}_1(q, d_i)) \mid d_i \in \mathcal{D}\}).$$

We use SimCSE (Gao et al., 2021) to encode task text and Jaccard similarity to compute tool set relevance. Similarity normalization and candidate retrieval pools are described in the appendix.

After obtaining E_1 , SD-ICL converts the retrieved historical demonstrations into tool selection views. For a historical demonstration d_i , its tool selection view is:

$$V_{\text{sel}}(d_i) = (q_i, T_i, P_i),$$

where q_i is the historical task description, T_i the tool set used, and P_i the APIs actually invoked. This view keeps task–tool/API correspondence and omits full execution details.

The selector G_ϕ then generates the raw tool selection result based on the current task, the candidate API/tool set, and the demonstration views for the tool selection stage:

$$r_{\text{sel}} = G_\phi(\text{Prompt}_{\text{sel}}(q, \mathcal{A}_q, E_1)).$$

The system parses this output to obtain the tool selection state:

$$z_s = (T_s, P_s, B_s, \rho_s).$$

Here, T_s , P_s , B_s , and ρ_s denote the selected tools, selected APIs, candidate adjacent call relations represented as adjacent API bigrams, and selection rationale, respectively. B_s captures local call tendencies and provides a structured query signal for Stage 2 retrieval.

3.3 Demonstration Retrieval Guided by Intermediate Variables

After obtaining the explicit tool selection state, SD-ICL transforms the second-stage retrieval from overall task semantic matching into state-conditioned retrieval. The state narrows the retrieval query to the APIs selected for the task and the local transitions proposed by the selector. Call generation context then places additional emphasis on whether a historical trajectory covers those APIs and whether its local call structure is transferable. Stage 2 uses the selected API set P_s and candidate adjacent call relations B_s to rerank historical successful trajectories:

$$E_2 = R_2(P_s, B_s, \mathcal{D}),$$

where E_2 denotes the demonstrations retrieved for the call generation stage.

Following the demonstration-library representation, each historical demonstration d_i is associated with an invoked API set P_i and a set of adjacent

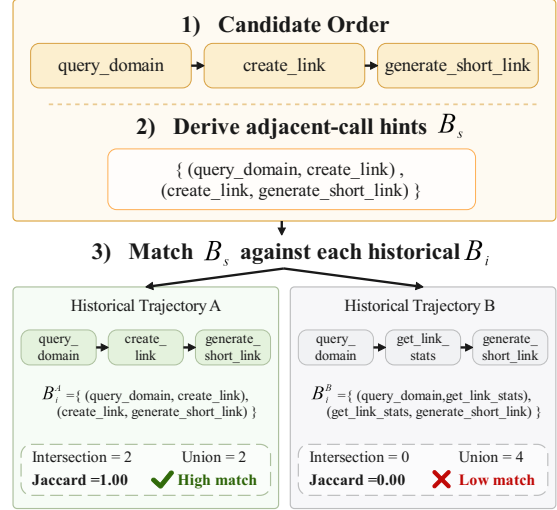


Figure 2: Illustration of adjacent call matching in Stage 2. The candidate order is converted into adjacent call relations B_s , which are then matched against each historical B_i using Jaccard similarity. Trajectories with higher overlap are preferred for call generation demonstration retrieval.

call relations B_i . Stage 2 first computes API-set similarity, measured by Jaccard similarity:

$$s_{\text{api}}(P_s, P_i) = \frac{|P_s \cap P_i|}{|P_s \cup P_i|}.$$

It then computes adjacent call similarity:

$$s_{\text{adj}}(B_s, B_i) = \frac{|B_s \cap B_i|}{|B_s \cup B_i|}.$$

Here, s_{api} measures similarity between the selected and historical API sets, while s_{adj} measures whether the historical adjacent call relations match the candidate adjacent call relations. The metric captures local structure through common adjacent API combinations. Appendix C.8 quantifies adjacent observation-to-argument reuse in successful ToolBench trajectories, and Figure 2 illustrates the matching process. Dependencies spanning non-adjacent calls fall outside this representation, as discussed in the limitations.

Combining API-set similarity and adjacent call relation similarity, the fused Stage 2 score is defined as:

$$\text{Score}_2(z_s, d_i) = \lambda_2 \bar{s}_{\text{api}}(P_s, P_i) + (1 - \lambda_2) \bar{s}_{\text{adj}}(B_s, B_i),$$

where $\bar{s}_{\text{api}}$ and $\bar{s}_{\text{adj}}$ denote normalized API-set similarity and adjacent call similarity, respectively. The

weight λ_2 controls their relative contribution after candidate pool normalization.

The system then selects the top-ranked demonstrations as call generation demonstrations, excluding by default demonstrations already used in Stage 1:

$$E_2 = \text{Top}_{M_2} \left(\{(d_i, \text{Score}_2(z_s, d_i)) \mid d_i \in \mathcal{D} \setminus E_1\} \right).$$

Here, M_2 denotes the number of call generation demonstrations. The two demonstration sets serve different roles: E_1 emphasizes matching tasks to tools and APIs; E_2 emphasizes API call format, argument construction, intermediate observations, and multistep call organization. Section 4 provides the implementation settings.

3.4 Staged Context Assembly and Execution

After the two stages, SD-ICL assembles the tool selection demonstrations E_1 , a tool selection summary S_z derived from z_s , and the call generation demonstrations E_2 .

SD-ICL first constructs a tool selection summary from the selected tools and the selection rationale:

$$S_z = \text{Summary}(T_s, \rho_s).$$

Only T_s and ρ_s enter this summary; P_s and B_s enter the final context indirectly through the Stage 2 demonstrations retrieved under their guidance.

The system then assembles the final context:

$$C_{\text{SD-ICL}} = [\text{Format}_{\text{sel}}(E_1); \text{Format}_{\text{sum}}(S_z); \text{Format}_{\text{gen}}(E_2); \text{Format}_{\text{task}}(q, \mathcal{A}_q)].$$

$\text{Format}_{\text{sel}}$ inserts the tool-selection view, while $\text{Format}_{\text{gen}}$ inserts full successful trajectories for call formatting, argument generation, and execution-path organization.

Finally, the staged context is passed to the executor:

$$(\hat{\tau}, \hat{y}) = F_{\theta}(q, \mathcal{A}_q, C_{\text{SD-ICL}}).$$

The final tool actions are generated by the executor from the staged context, task state, and tool observations.

4 Experiments

We evaluate whether SD-ICL improves tool-use performance over demonstration retrieval based on

semantic similarity and how its key components contribute. Main results use DeepSeek-V3.2 and Qwen3.5-Flash, with an additional evaluation using Qwen3.5-9B. Within each model setting, controlled context-construction methods share the same executor, candidate API/tool set, maximum execution steps, demonstration budget, and evaluation scripts.

4.1 Experimental Setup

Datasets. We evaluate on ToolBench and APIBench. ToolBench is a large-scale benchmark for real API tool use, containing 16,464 APIs, 49 categories, and 3,451 tools, covering single-tool and multi-tool scenarios (Qin et al., 2024). For ToolBench, we use the 765 solvable queries provided by StableToolBench across the six G1-Inst., G1-Tool, G1-Cat., G2-Inst., G2-Cat., and G3-Inst. subsets, together with its local virtual API service (Guo et al., 2024). Pass., Win., and judge configurations follow the evaluation protocol defined in Appendix B. APIBench contains HuggingFace, TensorFlow Hub, and TorchHub subsets (Patil et al., 2024). Most APIBench tasks involve shorter invocation chains and focus on API selection and argument generation; we use it as a complementary benchmark for shorter API-call scenarios.

Baselines. Within each base-model block, Zero-shot, Text-ICL, and SD-ICL are controlled context-construction methods. Zero-shot uses no demonstrations. Text-ICL retrieves four complete successful trajectories by task semantic similarity and inserts them as homogeneous demonstrations. Tool-Planner uses the corresponding base model but introduces an execution-time planning mechanism, so we treat it as a planning-based reference (Liu et al., 2025). ToolLLaMA and Gorilla are external reference systems, differing in base model, training regime, retrieval component, or execution mechanism (Qin et al., 2024; Patil et al., 2024). TRAD is additionally evaluated on ToolBench under DeepSeek-V3.2. We additionally evaluate Qwen3.5-9B.

Evaluation metrics. Pass. is the primary task-completion metric following the benchmark evaluation protocol. Win. is a complementary StableToolBench-style preference metric, with Zero-shot reported as 50.0%. We report per-subset results and sample-size-weighted aggregate averages.

Implementation details. All training-free methods share the same executor and evaluation pipeline within each base-model block. The executor uses a single-chain tool-use process with at most 64 Thought–Action–Observation steps. Text-ICL uses 4 complete trajectory demonstrations. SD-ICL uses $M_1 = 2$ tool selection demonstrations and $M_2 = 2$ call generation demonstrations. For main results, the executor and Stage 1 selector use the same base model; the selector is called once before execution to produce the tool selection state. We fix $\lambda_1 = 0.4$ for Stage 1 retrieval and $\lambda_2 = 0.5$ for Stage 2 retrieval across all experiments. Ablations and diagnostics use DeepSeek-V3.2 unless otherwise stated. Appendices A and B provide prompt templates, hyperparameters, and evaluation details. To separate the extra selector call from stage-guided retrieval, we include SD-ICL w/o SGR, which keeps the selector call and tool selection summary but removes Stage 2 guided retrieval.

4.2 Main Results

Table 1 reports ToolBench results. SD-ICL consistently improves over Text-ICL under both main model settings. Under DeepSeek-V3.2, Avg. Pass. rises from 57.0% to 71.8%, a gain of 14.8 percentage points; under Qwen3.5-Flash, it rises from 62.2% to 74.4%, a gain of 12.2 points. The largest gains occur on G2-Cat. and G3-Inst., with a clear increase on G1-Cat. as well. These subsets involve demanding tool or API scope identification and multistep call organization. Within the DeepSeek-V3.2 block, ToolLLaMA leads on G2-Inst. and Tool-Planner on G2-Cat. Tool-Planner also leads several Qwen3.5-Flash subsets, and SD-ICL has the highest aggregate result among the controlled context construction methods.

The Qwen3.5-Flash block shows a similar pattern in the second main model setting. On ToolBench, SD-ICL improves Avg. Win. over Text-ICL by 11.1 points. The gain is especially visible on G3-Inst., where Text-ICL drops to 54.1% under Qwen3.5-Flash but SD-ICL still reaches 82.0% Pass. The results are consistent across these two main settings; component effects are analyzed with DeepSeek-V3.2 ablations in Section 4.3.

Comparison with TRAD. Table 3(a) reports the aggregate ToolBench comparison under DeepSeek-V3.2; subset results appear in Table 13. Text-ICL obtains 57.0% Avg. Pass. and 58.6% Avg. Win., TRAD obtains 65.0% and 64.8%, and SD-ICL ob-

tains 71.8% and 70.6%, respectively. TRAD retrieves thoughts step by step during execution. SD-ICL constructs tool selection and call generation demonstrations before execution.

Table 2 reports APIBench results. SD-ICL improves over Text-ICL under both main model settings: Avg. Pass. increases from 78.9% to 83.9% under DeepSeek-V3.2 and from 82.8% to 86.5% under Qwen3.5-Flash. Gorilla has the highest APIBench scores, particularly on TorchHub, where its API-oriented adaptation matches strong platform conventions. SD-ICL records the highest aggregate scores among the controlled context construction methods. Under Qwen3.5-Flash, Avg. Win. increases from 70.0% to 79.0%. TorchHub shows the largest Pass. increase over Text-ICL, from 57.0% to 72.6%; Tool-Planner and Gorilla retain higher scores on this subset. Section 4.3 reports the component analysis.

Additional Qwen3.5-9B evaluation. Table 3(b) reports aggregate Pass. results with Qwen3.5-9B; full results appear in Table 14. On ToolBench, SD-ICL improves over Text-ICL from 49.0% to 59.0% (+10.0 percentage points). On APIBench, it improves Avg. Pass. from 74.8% to 79.8% (+5.0 percentage points), with positive improvements on all three platforms.

4.3 Ablation Study

Mechanism-level ablations. We conduct component ablations with DeepSeek-V3.2 in a common base model setting. Three variants isolate changes to the staged design: w/o SGR keeps the selector call and tool selection summary but removes Stage 2 guided retrieval; w/o ETS removes the explicit tool selection state; w/o TCR keeps the selector state and staged views but removes guidance from selected APIs and adjacent call relations during Stage 2 retrieval. Appendix B.6, Table 9, summarizes the variants.

For the primary Pass. metric, aligned per-example stratified paired bootstrap gives differences of +14.8 points for full SD-ICL versus Text-ICL (95% CI [+10.3, +19.2]) and +8.0 points versus w/o SGR (95% CI [+3.4, +12.5]). The corresponding differences are +6.3 points versus w/o ETS (95% CI [+1.5, +10.8]) and +2.3 points versus w/o TCR (95% CI [−0.7, +5.2]); Table 4 reports the point estimates, and Table 11 reports the complete paired analysis. The intervals exclude zero for Text-ICL, w/o SGR, and w/o ETS, while the

Model	Method	G1-Inst.		G1-Tool		G1-Cat.		G2-Inst.		G2-Cat.		G3-Inst.		Avg.	
		Pass.	Win.	Pass.	Win.	Pass.	Win.	Pass.	Win.	Pass.	Win.	Pass.	Win.	Pass.	Win.
ToolLLaMA-2-7B	ToolLLaMA	63.8	62.6	63.9	59.5	60.1	54.9	81.1	68.9	68.6	61.3	65.6	73.8	66.4	62.0
DeepSeek-V3.2	Zero-shot	45.4	50.0	53.8	50.0	58.2	50.0	39.6	50.0	41.1	50.0	37.7	50.0	47.6	50.0
	Text-ICL	54.6	56.4	55.0	54.4	62.9	59.5	59.2	60.4	52.6	59.7	59.2	67.2	57.0	58.6
	Tool-Planner	60.1	64.4	71.5	65.2	66.7	62.8	76.4	73.6	79.0	65.3	67.2	68.9	69.7	66.0
	SD-ICL	63.8	66.3	67.7	67.7	78.4	73.2	72.6	71.7	73.4	70.2	82.0	82.0	71.8	70.6
Qwen3.5-Flash	Zero-shot	55.2	50.0	57.0	50.0	62.1	50.0	41.5	50.0	46.8	50.0	36.1	50.0	52.2	50.0
	Text-ICL	64.4	58.9	58.2	57.6	69.9	62.1	63.2	60.4	58.1	60.5	54.1	65.6	62.2	60.3
	Tool-Planner	66.3	66.3	75.3	67.1	69.3	64.7	77.4	75.5	80.6	67.7	70.5	73.8	72.9	68.2
	SD-ICL	69.9	67.5	69.0	69.0	79.7	75.8	73.6	72.6	77.4	71.0	82.0	75.4	74.4	71.4

Table 1: Results on ToolBench. Within each base-model block, Zero-shot, Text-ICL, and SD-ICL are controlled context-construction methods; Tool-Planner is a planning-based reference. ToolLLaMA-2-7B is an external reference system. Pass. denotes task completion rate, and Win. denotes the complementary StableToolBench-style preference metric.

Model	Method	HuggingFace		TensorFlow		TorchHub		Avg.	
		Pass.	Win.	Pass.	Win.	Pass.	Win.	Pass.	Win.
Gorilla	Gorilla	89.2	89.0	90.3	90.4	93.0	94.1	90.0	90.1
DeepSeek-V3.2	Zero-shot	68.1	50.0	62.2	50.0	53.2	50.0	64.3	50.0
	Text-ICL	85.8	68.9	75.9	66.4	55.9	52.2	78.9	66.2
	Tool-Planner	71.4	75.4	75.6	72.4	86.0	77.4	74.5	74.5
	SD-ICL	88.1	76.8	84.2	76.5	61.8	73.7	83.9	76.4
Qwen3.5-Flash	Zero-shot	73.8	50.0	62.8	50.0	68.8	50.0	69.0	50.0
	Text-ICL	89.1	70.9	81.4	70.8	57.0	62.9	82.8	70.0
	Tool-Planner	81.4	77.3	80.7	75.0	91.4	81.2	82.2	76.8
	SD-ICL	89.5	78.6	86.3	80.1	72.6	76.9	86.5	79.0

Table 2: Results on APIBench. Within each base-model block, Zero-shot, Text-ICL, and SD-ICL are controlled context-construction methods; Tool-Planner is a planning-based reference, and Gorilla is an external reference system. Pass. denotes task completion rate, and Win. denotes the complementary StableToolBench-style preference metric.

(a) DeepSeek-V3.2 ToolBench		
Method	Avg. Pass.	Avg. Win.
Text-ICL	57.0	58.6
TRAD	65.0	64.8
SD-ICL	71.8	70.6
(b) Qwen3.5-9B		
Method	ToolBench Pass.	APIBench Pass.
Text-ICL	49.0	74.8
SD-ICL	59.0	79.8

Table 3: Additional results with TRAD and Qwen3.5-9B.

Method	ToolBench		APIBench	
	Avg. Pass.	Avg. Win.	Avg. Pass.	Avg. Win.
Text-ICL	57.0	58.6	78.9	66.2
SD-ICL w/o SGR	63.8	62.5	81.0	69.5
SD-ICL w/o ETS	65.5	63.0	77.5	68.9
SD-ICL w/o TCR	69.5	65.6	82.2	70.9
SD-ICL	71.8	70.6	83.9	76.4

Table 4: Ablation results on ToolBench and APIBench. w/o SGR, i.e., Text-ICL + Tool Selection Summary, removes stage-guided retrieval while keeping the extra selector call and tool selection summary; w/o ETS removes explicit tool selection state; w/o TCR removes retrieval conditioned on the tool selection state.

interval for w/o TCR includes zero.

Direct context-segment removal. We also remove individual segments from the final executor context while preserving their upstream computation. For w/o E1, the selector still uses E1 to construct the Stage 1 state, but E1 is omitted from

the executor context. For w/o Sz, P_s and B_s still condition Stage 2 retrieval, but the tool selection summary is omitted. For w/o E2, Stage 2 retrieval proceeds normally and its call generation demonstrations are omitted from the executor context.

Variant	Avg. Pass.	Avg. Win.
Full SD-ICL	71.8	70.6
w/o E1	66.8	66.0
w/o Sz	62.9	62.6
w/o E2	67.5	65.2

Table 5: Direct context-segment removal results on ToolBench under DeepSeek-V3.2.

Table 5 reports the aggregate results, and Table 12 provides the subset-level results. Removing E1, Sz, or E2 decreases both Avg. Pass. and Avg. Win. in this ToolBench configuration. For the primary Pass. metric, full SD-ICL exceeds w/o E1 by 5.0 points (paired-bootstrap 95% CI [+0.6, +9.5]), w/o Sz by 8.9 points (95% CI [+4.3, +13.7]), and w/o E2 by 4.3 points (95% CI [+0.2, +8.4]). All three intervals exclude zero; Win. remains a complementary point-estimate metric.

5 Analysis

Case study. The representative short-link case in Table 17 requires domain query $\rightarrow$ link creation $\rightarrow$ short-link generation. Zero-shot omits link creation, Text-ICL recovers that step but misses the second short-link service call, and SD-ICL covers all subgoals. In this example, semantic trajectory retrieval alone fails to supply both the required API scope and a usable local call organization. Stage 1 emphasizes the set of tools and APIs needed by the task, and Stage 2 emphasizes the call patterns used to realize that scope.

Selector quality and downstream performance. Across 765 examples, the selector attains macro precision, recall, and F1 of 94.8%, 96.0%, and 95.2% against the benchmark reference path API sets. Complete reference path coverage, defined as including every API in the corresponding reference set, is 85.0%. Pass. is 72.8% with complete coverage and 66.1% with incomplete coverage, a descriptive gap of +6.7 points. These reference sets provide a benchmark comparison target, and coverage is associated with downstream completion rather than specifying a unique tool use path.

Local dependency evidence. Among traceable observation-to-argument reuse cases in the ToolBench demonstration trajectories, 69.2% obtain the reused value from the immediately preceding tool step. In the same trajectories, 89.5% contain at least two calls and 35.2% contain observation reuse in a

subsequent argument. This prevalence makes adjacent call relations useful for local retrieval structure.

Boundary across subsets. Gains are more visible on ToolBench subsets with longer or multitool trajectories. APIBench contains shorter calls and stronger platform conventions, and specialized systems retain an advantage on TorchHub. Staged context construction and platform-specific adaptation thus address different parts of tool use performance.

Cost and budget. SD-ICL adds one selector call before execution compared with Text-ICL. Holding that call and the tool selection summary fixed, full SD-ICL improves ToolBench Pass. over w/o SGR by 8.0 points (paired bootstrap 95% CI [+3.4, +12.5]). The comparison isolates the contribution of Stage 2 guided retrieval beyond the selector summary.

6 Conclusion

We present SD-ICL, a training-free method that constructs context in tool selection and call generation stages. Stage 1 retrieves demonstrations that emphasize task semantics and tool set relevance, then produces an explicit tool selection state. Stage 2 uses the selected APIs and candidate adjacent call relations as soft signals for retrieving call generation demonstrations. The final context combines these two demonstration sets with a summary of the tool selection state, leaving the executor unchanged.

Across ToolBench and APIBench, SD-ICL consistently improves over Text-ICL in the evaluated model settings. The controlled TRAD comparison further shows gains over step-wise retrieval during execution, while the Qwen3.5-9B results extend the evaluation to an additional model setting. Mechanism ablations and direct context removal experiments further test the roles of staged retrieval, the tool selection state, and the final context segments. Their paired analyses support the overall staged design, with the w/o TCR comparison remaining inconclusive. Organizing demonstrations around tool selection and call generation improves tool use in these settings without changing the executor or its interaction loop.

Limitations

The evaluation covers ToolBench and APIBench. Performance on newer executable and agentic function calling benchmarks remains open. The component analyses use DeepSeek-V3.2 on ToolBench, and the Qwen3.5-9B experiment evaluates the full method in an additional model setting.

SD-ICL uses the same base model for the selector and executor, adding one selector call before execution relative to Text-ICL. A smaller selector may offer a different accuracy–cost trade-off, which is outside the present evaluation.

The generated tool selection state directly shapes subsequent retrieval, so missing APIs or inaccurate adjacent call relations can propagate into Stage 2. The bigram representation captures local transitions but omits long-range data flow. Improving selector reliability and representing richer trajectory structure are natural extensions of the method.

Ethical Considerations

This paper studies context construction for tool use in LLM agents. The experiments are based on public tool-use benchmarks and do not involve collecting, annotating, or inferring new personal private data. The method also does not introduce user profiling, sensitive attribute recognition, or personal long-term memory mechanisms.

The core function of SD-ICL is to influence tool selection and call generation through pre-execution context construction. Therefore, if the tool selection state is generated incorrectly, the model may be more likely to refer to inappropriate tools or APIs during subsequent execution, increasing the risk of erroneous tool invocation. In real deployments, if the underlying executor is connected to external services with actual side effects, such as payment, account management, data modification, automated operations, or high-privilege internal systems, incorrect tool selection and argument generation may introduce safety, privacy, or accountability risks. Practical applications should therefore incorporate permission control, invocation allowlists, confirmation for high-risk operations, logging and auditing, and human oversight to prevent the model from directly executing irreversible or high-risk operations.

In addition, staged context construction may improve the automation capability of tool-use agents, and should not be used to bypass platform restrictions, conduct API abuse at scale, automate attacks, or access data without authorization. The proposed

method should be used in controlled, authorized, and auditable tool-use environments. In particular, when real user data or external system-state changes are involved, tool permissions, data-access scopes, and execution consequences should be clearly constrained.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China under Grant No. 62272045.

References

- 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 M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. [Language models are few-shot learners](#). In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901.
- Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. [SimCSE: Simple contrastive learning of sentence embeddings](#). In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910. Association for Computational Linguistics.
- 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. Association for Computational Linguistics.
- Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. [API-Bank: A comprehensive benchmark for tool-augmented LLMs](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116. Association for Computational Linguistics.
- Yanming Liu, Xinyue Peng, Jiannan Cao, Shi Bo, Yuwei Zhang, Xuhong Zhang, Sheng Cheng, Xun Wang, Jianwei Yin, and Tianyu Du. 2025. [Tool-Planner: Task planning with clusters across multiple tools](#). In *The Thirteenth International Conference on Learning Representations*.
- Bhrij Patel, Ashish Jagmohan, and Aditya Vempaty. 2025. [Learning API functionality from In-Context demonstrations for tool-based agents](#). In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 18318–18336, Suzhou, China. Association for Computational Linguistics.

- 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.
- 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 *The Twelfth International Conference on Learning Representations*.
- Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2022. [Learning to retrieve prompts for in-context learning](#). In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2655–2671. Association for Computational Linguistics.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, 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.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. [Reflexion: Language agents with verbal reinforcement learning](#). In *Advances in Neural Information Processing Systems*, volume 36.
- Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023. [GPT4Tools: Teaching large language model to use tools via self-instruction](#). In *Advances in Neural Information Processing Systems*, volume 36.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. [ReAct: Synergizing reasoning and acting in language models](#). In *The Eleventh International Conference on Learning Representations*.
- Jiacheng Ye, Zhiyong Wu, Jiangtao Feng, Tao Yu, and Lingpeng Kong. 2023. [Compositional exemplars for in-context learning](#). In *Proceedings of the 40th International Conference on Machine Learning*, pages 39818–39833. PMLR.
- Longtao Zheng, Rundong Wang, Xinrun Wang, and Bo An. 2024. [Synapse: Trajectory-as-exemplar prompting with memory for computer control](#). In *The Twelfth International Conference on Learning Representations*.
- Ruiwen Zhou, Yingxuan Yang, Muning Wen, Ying Wen, Wenhao Wang, Chunling Xi, Guoqiang Xu, Yong Yu, and Weinan Zhang. 2024. [TRAD: Enhancing LLM agents with step-wise thought retrieval and aligned decision](#). In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–13.
- Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor Bursztyn, Ryan A. Rossi, Somdeb Sarkhel, and Chao Zhang. 2024. [ToolChain*: Efficient action space navigation in large language models with A* search](#). In *The Twelfth International Conference on Learning Representations*.

Symbol	Description
$q, \mathcal{A}_q$	Current task and its accessible API/tool set
$\mathcal{D}$	Historical demonstration library
E_1, E_2	Retrieved tool-selection and call-generation demonstrations
z_s	Structured tool selection state
T_s, P_s	Selected tool set and selected API set
B_s, ρ_s	Candidate adjacent call relations and selection rationale
S_z	Tool selection summary constructed from T_s and ρ_s
C	Pre-execution context supplied to the executor
M_1, M_2	Numbers of demonstrations retrieved for Stages 1 and 2
λ_1, λ_2	Fusion weights for the Stage 1 and Stage 2 retrieval scores
$\hat{\tau}, \hat{y}$	Generated tool-interaction trajectory and final answer

Table 6: Core notation used in the main text.

A Prompt Templates and Output Schemas

This appendix provides the main templates related to prompt construction and structured outputs in SD-ICL. The main text only outlines the prompt construction procedure; this appendix further describes the specific forms of the tool selection stage, staged context assembly, and output fields. All templates are used for pre-execution context construction and do not change the tool-use loop of the base ReAct-style executor.

A.1 Tool Selection Prompt

The tool selection prompt is used for explicit tool selection state construction in Stage 1. Its inputs include the current task, the candidate API/tool set accessible to the current task, and the tool selection demonstrations retrieved in Stage 1. The selector is required to output structured fields, including selected tools, selected APIs, candidate adjacent call relations, and the selection rationale.

```
You are an assistant for tool selection
→ in a tool-use agent.
Given a user task, a list of available
→ APIs, and several successful tool
→ selection examples,
identify the tools and APIs that are
→ likely to be useful for solving the
→ current task.
Your output must be a valid JSON object
→ with the following fields:
{
  "selected_tools": [...],
  "selected_apis": [...],
  "candidate_order": [...],
  "reason": "...
}
```

Field definitions:

- **selected_tools**: a list of tool names
→ selected from the available API/tool
→ list.
- **selected_apis**: a list of API/action
→ names selected from the available API
→ list.
- **candidate_order**: a possible high-level
→ order of API calls. This order is
→ only used to derive candidate
→ adjacent API relations for
→ demonstration retrieval. It is not a
→ fixed execution plan.
- **reason**: a concise explanation of why
→ these tools and APIs are selected.

Requirements:

1. Only select tools and APIs that appear
→ in the current available API/tool
→ list.
2. Do not invent unavailable APIs.
3. Prefer APIs that directly support the
→ user task.
4. The **candidate_order** should describe
→ plausible local call transitions, but
→ it must not be treated as a mandatory
→ execution path.
5. Keep the reason concise and
→ task-specific.

Current task:

```
{query}
```

Available APIs/tools:

```
{api_list}
```

Tool selection demonstrations:

```
{stage1_examples}
```

Now output the JSON object for the

```
→ current task.
```

A.2 Tool Selection Output Schema

The Stage 1 selector outputs a structured tool selection state:

$$z_s = (T_s, P_s, B_s, \rho_s),$$

where T_s denotes selected tools, P_s denotes selected APIs, B_s denotes candidate adjacent call relations, and ρ_s denotes the selection rationale. In implementation, the selector returns the following JSON object:

```
{
  "selected_tools": [
    "tool_name_1",
    "tool_name_2"
  ],
  "selected_apis": [
    "api_name_1",
    "api_name_2"
  ],
  "candidate_order": [
    "api_name_1",
    "api_name_2",
    "api_name_3"
  ],
  "reason": "A concise rationale for
→ selecting these tools and APIs."
}
```

The selected tool list corresponds to T_s , the selected API list corresponds to P_s , and the rationale corresponds to ρ_s . The candidate adjacent call relation set B_s is derived from `candidate_order`:

$$B_s = \{(u_k, u_{k+1}) \mid 1 \leq k < |\text{candidate_order}|\}.$$

If B_s is empty, Stage 2 uses only API-set similarity for retrieval. If P_s is empty, Stage 2 falls back to the Stage 1 retrieval score.

A.3 Staged Context Assembly Template

The final context of SD-ICL consists of four parts: Stage 1 tool selection demonstrations, the explicit tool selection summary, Stage 2 call generation demonstrations, and the current task. The overall template is as follows:

```
You are a tool-use agent. You can solve
→ the user task by reasoning step by
→ step and calling available APIs.
[Stage 1: Tool Selection Demonstrations]
The following examples show how similar
→ tasks are matched with relevant tools
→ and APIs.
They are provided to help identify the
→ tool/API scope of the current task.
{formatted_stage1_examples}
[Tool Selection Summary for the Current
→ Task]
The following tools are likely to be
→ useful:
{selected_tools}
Selection rationale:
{selection_reason}
This summary is only a soft guide. You
→ may still use any API from the
→ available API list if it is necessary
→ for solving the task.
[Stage 2: Call Generation Demonstrations]
The following examples show successful
→ tool-use trajectories.
They are provided to help with API call
→ formatting, argument generation,
→ intermediate observation use, and
→ multi-step call organization.
{formatted_stage2_examples}
[Current Task]
User query:
{query}
Available APIs/tools:
{api_list}
Now solve the current task using the
→ available APIs. Follow the original
→ ReAct-style tool-use process.
```

A.4 Placeholder Descriptions

Placeholder	Description
{query}	The current natural-language task to be solved.
{api_list}	The available API/tool list for the current task, including tool names, API names, functionality descriptions, parameter schemas, and return descriptions.
{stage1_examples}	Tool selection demonstrations retrieved in Stage 1, mainly showing historical tasks, used tools, and actually called APIs.
{formatted_stage1_examples}	Tool selection view examples inserted into the final staged context.
{selected_tools}	The tool-level selection result generated by Stage 1 and retained in the final context.
{selection_reason}	The selection rationale generated by Stage 1 and used as soft guidance.
{formatted_stage2_examples}	Full tool-use trajectory examples retrieved in Stage 2, mainly showing Thought, Action, Observation, and final answer.

Table 7: Placeholder descriptions for the staged context template.

A.5 Stage-Specific Example Views

Demonstrations for the tool selection stage use a simplified view:

$$V_{\text{sel}}(d_i) = (q_i, T_i, P_i),$$

where q_i denotes the historical task, T_i denotes the tool set used in the successful trajectory, and P_i denotes the API set invoked in the successful trajectory. This view does not display the complete execution trajectory, and mainly serves task-tool/API matching.

Demonstrations for the call generation stage use the trajectory view:

$$V_{\text{gen}}(d_i) = (q_i, \{(\text{Thought}_{i,t}, \text{Action}_{i,t}, \text{Observation}_{i,t})\}_{t=1}^{K_i}, y_i).$$

This view preserves the complete tool-use process and provides references for API call formatting, argument generation, intermediate observation use, and multi-step call organization.

B Evaluation Protocol and Implementation Details

B.1 Benchmarks

We evaluate SD-ICL on ToolBench and APIBench. ToolBench is used as the main benchmark for complex tool use evaluation, covering single-tool tasks, intra-category multi-tool tasks, and intra-collection

multi-tool tasks. Following the ToolBench evaluation setting, we report results on G1-Inst., G1-Tool, G1-Cat., G2-Inst., G2-Cat., and G3-Inst.

APIBench is used as a complementary benchmark for shorter API-call scenarios. We evaluate on its three platform subsets: HuggingFace, TensorFlow Hub, and TorchHub. For both benchmarks, aggregate results are computed as sample-size-weighted averages over the corresponding test subsets.

B.2 Base Executor and Tool Environment

Within each base-model block, training-free methods use the same base executor, candidate tool set, maximum execution steps, and evaluation scripts. The base executor follows the ReAct-style single-chain inference process in StableToolBench (Guo et al., 2024). During execution, the agent invokes tools step by step in the form of Thought–Action–Observation, and advances subsequent decisions according to tool-returned results.

The maximum number of execution steps for each sample is set to 64. If the model outputs the final answer within 64 steps, the task terminates. If the step limit is reached before completion, execution stops and the existing trajectory is saved.

The tool environment uses the local virtual tool service provided by StableToolBench. This service first queries cached historical tool responses. If there is no cache hit, the model simulates a format-compatible tool return and writes it into the cache. This mechanism reduces the impact of real API instability, remote service unavailability, and changing return values on experimental reproducibility.

TRAD evaluation. We implement the step-wise retrieval procedure of TRAD in the ToolBench evaluation pipeline. The comparison uses the same 765 ToolBench test examples as SD-ICL, a DeepSeek-V3.2 executor, the same tool environment and maximum execution steps, and the same evaluation scripts.

B.3 Demonstration Library Construction

The demonstration library of SD-ICL is built from historical tool-use trajectories in the training split. Test samples are not included in the demonstration library and do not participate in retrieval or context construction. Each retained demonstration contains the task description, the original successful tool-use trajectory, the actually used tool set, the API set, and adjacent call relations.

For a historical successful trajectory d_i , let its effective API invocation sequence be:

$$u_i = (u_{i,1}, u_{i,2}, \dots, u_{i,K_i}),$$

where K_i denotes the number of effective tool invocation steps. The corresponding API set and adjacent call relation set are:

$$P_i = \{u_{i,k} \mid 1 \leq k \leq K_i\},$$

$$B_i = \{(u_{i,k}, u_{i,k+1}) \mid 1 \leq k < K_i\}.$$

The tool set T_i is obtained by mapping API/actions to their corresponding tool names. Each final demonstration is represented as:

$$d_i = (q_i, C_i, T_i, P_i, B_i),$$

where q_i denotes the task description and C_i denotes the original interaction trajectory. The demonstration library supports both Stage 1 tool selection demonstration retrieval and Stage 2 call generation demonstration retrieval.

B.4 Retrieval and Hyperparameters

SD-ICL uses fixed hyperparameters in all experiments and does not tune them separately for different test subsets.

Because the retrieval signals have different raw value ranges and empirical distributions, SD-ICL normalizes all similarity scores before score fusion. For each test instance, normalization is performed locally within the corresponding candidate retrieval pool, rather than globally over the entire demonstration library. Let $\mathcal{C}$ denote the candidate pool for a retrieval stage. For a similarity function $s_m(\cdot)$, its normalized value for demonstration $d_i \in \mathcal{C}$ is defined as:

$$\bar{s}_m(d_i) = \frac{s_m(d_i) - m_{\min}}{m_{\max} - m_{\min} + \epsilon},$$

$$m_{\min} = \min_{d_j \in \mathcal{C}} s_m(d_j), \quad m_{\max} = \max_{d_j \in \mathcal{C}} s_m(d_j).$$

where ϵ is a small constant used to avoid division by zero. We use the same min–max normalization within the candidate pool for both Stage 1 and Stage 2. In our implementation, the candidate retrieval pool size is set to 80.

For Stage 1, let $\mathcal{C}_1$ be the candidate pool for the current task. We normalize the task semantic similarity s_{text} and the tool set relevance s_{tool} within $\mathcal{C}_1$, and compute:

Setting	Value
Base executor	StableToolBench ReAct-style single-chain DeepSeek-V3.2 / Qwen3.5-Flash: main results; Qwen3.5-9B: additional evaluation;
Executor model	DeepSeek-V3.2: ablations and diagnostics
Executor max output length	1024
Executor temperature	1.0 (default)
Executor top-p	1.0 (default)
Max execution steps	64
Max observation length	1024
	same as executor for main and Qwen3.5-9B results; DeepSeek-V3.2 for mechanism ablations,
Selector model	context-removal ablations, and selector diagnostics
Selector temperature	0.0
Selector top-p	1.0 (default)
Selector max output length	512
Selector output format	JSON object
Judge model	gpt-4o
Judge temperature	0.2
Judge max output length	1000
Stage 1 examples M_1	2
Stage 2 examples M_2	2
Text-ICL examples	4
Stage 1 fusion weight λ_1	0.4
Stage 2 fusion weight λ_2	0.5
Candidate retrieval pool size	80
Semantic encoder	sup-simcse-roberta-base
SimCSE hidden size	768
SimCSE max length	256

Table 8: Implementation settings and hyperparameters. DeepSeek-V3.2 and Qwen3.5-Flash are used for the main results, and Qwen3.5-9B is used for an additional evaluation. Mechanism ablations, direct context-removal ablations, repeated-run stability checks, sensitivity analyses, and selector diagnostics use DeepSeek-V3.2 unless otherwise stated. Temperature and top-p follow the corresponding chat-completion API defaults unless otherwise specified. The judge uses the OpenAI API model identifier gpt-4o.

$$\text{Score}_1(q, d_i) = \lambda_1 \bar{s}_{\text{text}}(q, q_i) + (1 - \lambda_1) \bar{s}_{\text{tool}}(T_{\text{cand}}, T_i).$$

For Stage 2, let $\mathcal{C}_2$ be the candidate pool after excluding demonstrations already selected in Stage 1. We normalize the API-set similarity s_{api} and the adjacent call similarity s_{adj} within $\mathcal{C}_2$, and compute:

$$\text{Score}_2(z_s, d_i) = \lambda_2 \bar{s}_{\text{api}}(P_s, P_i) + (1 - \lambda_2) \bar{s}_{\text{adj}}(B_s, B_i).$$

If B_s is empty, Stage 2 uses only the normalized API-set similarity:

$$\text{Score}_2(z_s, d_i) = \bar{s}_{\text{api}}(P_s, P_i).$$

If P_s is empty, Stage 2 falls back to the Stage 1 retrieval score based on normalized task semantics and tool set relevance.

B.5 LLM-as-a-Judge Protocol

We use Pass. as the primary task-completion metric and Win. as a complementary preference metric.

Pass. follows the task-completion evaluation protocol of ToolBench / ToolEval and measures whether the candidate tool-use trajectory completes the task. Win. follows the StableToolBench-style preference evaluation (Guo et al., 2024) and compares a candidate method with the common Zero-shot reference output. For all automatic judging, we use gpt-4o with the same prompt template and decoding settings across methods.

Specifically, for each test sample, the judge reads the candidate method’s output and the Zero-shot output for the same task, and gives a relative preference judgment by considering both final answer quality and the tool-use process. Since Zero-shot is used as the common reference, its Win. is reported as 50.0%.

The gpt-4o judge prompt template is as follows:

```
You are an evaluator for tool-use agents.
Given a user task and two candidate
→ solutions, decide which solution
→ better completes the task.
You should consider:
1. whether the final answer satisfies the
→ user request;
2. whether the tool calls are relevant
→ and sufficient;
3. whether required intermediate
→ information is obtained;
4. whether the solution avoids
→ unnecessary or invalid tool calls;
5. whether the reasoning and final answer
→ are consistent with the observations.
User task:
{query}
Candidate A:
{answer_a}
Candidate B:
{answer_b}
Please choose one of the following
→ labels:
- A is better
- B is better
- Tie
Then briefly explain the reason.
```

B.6 Ablation Variant Definitions

Method	Stage Views	Selector State	Stage-2 Guided Retrieval	Tool Selection Summary
Text-ICL	✗	✗	✗	✗
w/o SGR	✗	✓	✗	✓
w/o ETS	✓	✗	✗	✗
w/o TCR	✓	✓	✗	✓
SD-ICL	✓	✓	✓	✓

Table 9: Component differences among Text-ICL, SD-ICL variants, and full SD-ICL. Stage Views denotes different formats for tool selection and call generation demonstrations; Selector State denotes explicit z_s ; Stage-2 Guided Retrieval denotes retrieval conditioned on P_s and B_s .

C Statistical Stability and Uncertainty

C.1 Repeated-Run Stability

To characterize run-to-run inference variance, we repeat SD-ICL three times on ToolBench under DeepSeek-V3.2. Each run uses the same data split, base model, maximum number of execution steps, demonstration budget, and evaluation protocol, while independently performing demonstration retrieval, context construction, agent inference, result conversion, and automatic evaluation. We report this analysis for full SD-ICL as a representative stability check.

Method	Run 1	Run 2	Run 3	Mean $\pm$ Std.
SD-ICL Avg. Pass.	71.76	71.63	71.90	71.76 $\pm$ 0.13

Table 10: Repeated-run stability of SD-ICL on ToolBench. Results are reported as percentages. The standard deviation is computed over three independent runs.

As shown in Table 10, SD-ICL obtains ToolBench average pass rates of 71.76%, 71.63%, and 71.90% across three independent runs. The standard deviation is 0.13 percentage points, and the three runs show low aggregate variability.

C.2 Paired Statistical Analysis

Method comparisons use aligned per-example Pass. outcomes. For ToolBench, stratified paired bootstrap resamples aligned examples within each of the six subsets, after which aggregate differences are computed from the resampled outcomes. We report 95% percentile intervals.

Comparison	Δ Pass.	Paired-bootstrap 95% CI
Full vs. Text-ICL	+14.8	[+10.3, +19.2]
Full vs. w/o SGR	+8.0	[+3.4, +12.5]
Full vs. w/o ETS	+6.3	[+1.5, +10.8]
Full vs. w/o TCR	+2.3	[−0.7, +5.2]
Full vs. w/o E1	+5.0	[+0.6, +9.5]
Full vs. w/o Sz	+8.9	[+4.3, +13.7]
Full vs. w/o E2	+4.3	[+0.2, +8.4]

Table 11: Paired statistical analysis of ToolBench Pass. differences under DeepSeek-V3.2. Differences are reported in percentage points.

The paired intervals for full SD-ICL versus Text-ICL, w/o SGR, and w/o ETS exclude zero, whereas the interval for w/o TCR includes zero. All three direct context-segment removal intervals exclude zero.

C.3 Full Results for Direct Context-Segment Removal

Table 12 reports subset-level Pass. and Win. results for the direct context-segment removal variants. Pass. remains the primary metric, with Win. reported as a complementary metric.

C.4 Full Results with TRAD

Table 13 reports the subset-level comparison among Text-ICL, TRAD, and SD-ICL on the same ToolBench evaluation set.

C.5 Qwen3.5-9B Results

This additional Qwen3.5-9B evaluation is reported in Table 14, with subset-level ToolBench Pass. and platform-level APIBench Pass.

C.6 Selector Quality Analysis

Precision, recall, and F1 are computed per example against the API set associated with the benchmark reference path and then macro-averaged. Complete reference-path coverage means that every API in the corresponding reference set is included in the selector output. Conditional Pass. values characterize the association between coverage and downstream task completion.

The selector obtains high agreement with the reference API sets. Complete reference-path coverage is associated with higher downstream Pass.

C.7 Additional Ablation on Semantic Encoder

Under DeepSeek-V3.2, we also test an SD-ICL w/o Sem variant that removes the Stage 1 semantic encoder. This variant does not use SimCSE semantic representations for retrieval in the tool selection stage, but retains the explicit tool selection state and retrieval guided by intermediate variables. The gap between SD-ICL w/o Sem and full SD-ICL is relatively small: on ToolBench, full SD-ICL exceeds w/o Sem by 1.1 percentage points in average pass rate; on APIBench, it exceeds w/o Sem by 1.5 percentage points. These small gaps characterize limited sensitivity to the choice of semantic encoder in this setting.

C.8 Adjacent Dependency Statistics

We examine the distance between tool observations and subsequent calls that reuse them as arguments. The analysis uses successful training split trajectories from the ToolBench demonstration library and is independent of the base executor. For each argument value that can be traced to a previous

(a) Pass.							
Method	G1-Inst.	G1-Tool	G1-Cat.	G2-Inst.	G2-Cat.	G3-Inst.	Avg.
Full SD-ICL	63.8	67.7	78.4	72.6	73.4	82.0	71.8
w/o E1	59.5	63.3	73.9	67.0	66.9	77.0	66.8
w/o Sz	56.4	59.5	69.9	62.3	62.9	72.1	62.9
w/o E2	60.1	63.9	74.5	67.0	67.7	78.7	67.5

(b) Win.							
Method	G1-Inst.	G1-Tool	G1-Cat.	G2-Inst.	G2-Cat.	G3-Inst.	Avg.
Full SD-ICL	66.3	67.7	73.2	71.7	70.2	82.0	70.6
w/o E1	62.0	63.3	68.6	67.0	64.5	78.7	66.0
w/o Sz	59.5	60.1	65.4	62.3	61.3	73.8	62.6
w/o E2	61.3	62.7	68.0	65.1	64.5	77.0	65.2

Table 12: Full ToolBench results for direct context-segment removal under DeepSeek-V3.2.

(a) Pass.							
Method	G1-Inst.	G1-Tool	G1-Cat.	G2-Inst.	G2-Cat.	G3-Inst.	Avg.
Text-ICL	54.6	55.0	62.9	59.2	52.6	59.2	57.0
TRAD	59.5	62.7	70.6	65.1	63.7	73.8	65.0
SD-ICL	63.8	67.7	78.4	72.6	73.4	82.0	71.8

(b) Win.							
Method	G1-Inst.	G1-Tool	G1-Cat.	G2-Inst.	G2-Cat.	G3-Inst.	Avg.
Text-ICL	56.4	54.4	59.5	60.4	59.7	67.2	58.6
TRAD	62.0	62.0	66.0	65.5	64.5	75.4	64.8
SD-ICL	66.3	67.7	73.2	71.7	70.2	82.0	70.6

Table 13: Full ToolBench results for the comparison with TRAD under DeepSeek-V3.2.

observation, the script records the number of tool steps between the current call and its source. The resulting distribution estimates how often adjacent call relations capture data reuse in multistep trajectories.

Among these trajectories, 89.5% contain at least two tool calls, 35.2% contain an observation reused in a subsequent argument, and 27.6% of adjacent call pairs involve reusable information. For traceable argument values, 69.2% originate in the immediately preceding tool step. Adjacent call relations therefore cover a substantial share of the observed local reuse patterns. Reuse across more distant steps remains outside the bigram representation.

C.9 Ablation Visualization

Figure 3 visualizes the average Pass. and Win. results for Text-ICL, SD-ICL variants, and full SD-ICL on ToolBench and APIBench.

C.10 Sensitivity to Retrieval Weights

We examine sensitivity to the retrieval weights λ_1 and λ_2 under DeepSeek-V3.2. The default values are $\lambda_1 = 0.4$ for Stage 1 and $\lambda_2 = 0.5$ for Stage 2. On ToolBench, one weight is varied at a time with

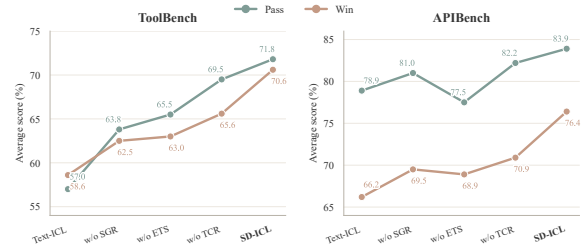


Figure 3: Average Pass. and Win. results for Text-ICL, SD-ICL variants, and full SD-ICL under DeepSeek-V3.2.

the other held fixed. The base executor, demonstration budget, maximum execution steps, and evaluation protocol match the DeepSeek-V3.2 main experiments.

The default configuration achieves an average pass rate of 71.8%. Values of 0.2 and 0.6 for λ_1 yield 71.1% and 69.9%; values of 0.25 and 0.75 for λ_2 yield 69.7% and 68.6%. Every tested setting remains above Text-ICL, with greater variation across the tested λ_2 values than across λ_1 .

(a) ToolBench Pass.							
Method	G1-Inst.	G1-Tool	G1-Cat.	G2-Inst.	G2-Cat.	G3-Inst.	Avg.
Text-ICL	53.4	50.6	57.5	42.5	42.7	36.1	49.0
SD-ICL	59.5	58.9	66.7	55.7	55.6	50.8	59.0

(b) APIBench Pass.				
Method	HuggingFace	TensorFlow	TorchHub	Avg.
Text-ICL	81.9	72.5	48.4	74.8
SD-ICL	85.0	79.1	57.0	79.8

Table 14: Full Pass. results with Qwen3.5-9B on ToolBench and APIBench.

Measure	Value
Evaluated examples	765
Macro Precision	94.8%
Macro Recall	96.0%
Macro F1	95.2%
Complete reference-path coverage	85.0%
Pass. given complete coverage	72.8%
Pass. given incomplete coverage	66.1%
Descriptive gap	+6.7 pp

Table 15: Selector quality and downstream Pass. grouped by reference-path coverage on ToolBench.

Method	TB Pass.	TB Win.	APIB Pass.	APIB Win.
SD-ICL w/o Sem	70.7	68.6	82.4	72.8
SD-ICL	71.8	70.6	83.9	76.4

Table 16: Additional ablation on the Stage 1 semantic encoder under DeepSeek-V3.2. TB denotes ToolBench and APIB denotes APIBench.

D Additional Case and Failure Examples

D.1 Representative Case Study

A representative ToolBench case requires domain query, link creation, and short-link generation. The simplified trajectories below summarize which required step is omitted by each baseline; SD-ICL completes the full sequence.

Method	Simplified trajectory	Result
Zero-shot	Domain query → missing: link creation → short-link generation	Fail
Text-ICL	Domain query → link creation → missing: second short-link service	Fail
SD-ICL	Domain query → link creation → short-link generation	Pass

Table 17: A representative ToolBench case. SD-ICL covers all required subgoals, while Zero-shot and Text-ICL omit different key steps.

D.2 Tool Selection Failure

An incomplete Stage 1 tool selection state can direct Stage 2 toward demonstrations with insufficient

coverage. This occurs when the selector misses necessary APIs or chooses tools that are semantically similar but functionally mismatched.

The selected tools may cover explicit requirements yet miss an implicit subgoal, or the selected API set may be too narrow to retrieve a complete invocation chain. Candidate adjacent call relations can likewise describe a plausible local path without including a required cross-tool transition.

The selector analysis reports the corresponding association between reference path coverage and downstream Pass.

D.3 Demonstration-Coverage Failure

Failures also arise when the demonstration library offers little coverage of a target task. Low-frequency tools, platform-specific APIs, and rare argument formats leave few useful trajectories for either retrieval stage.

TorchHub provides the clearest example. SD-ICL scores below Gorilla and Tool-Planner on this APIBench subset under both main model settings. Strong platform conventions, specific API formats, and sparse trajectory coverage favor the specialized training or explicit planning used by those systems.

D.4 Argument-Generation and Recovery Failure

Even with relevant tools from Stage 1 and similar trajectories from Stage 2, the base executor must generate the final API arguments within the ReAct loop. Errors may arise during argument generation and recovery as well as during tool selection.

Observed failures include missing required arguments, formats inconsistent with API requirements, repeated invalid calls after tool errors, limited use of previous observations, and termination before all subgoals are complete.

These errors occur after context construction and expose limitations in the executor’s argument vali-

dation, error diagnosis, and feedback-based recovery. These failures arise during execution rather than during staged retrieval.

D.5 Long-Range Dependency Failure

Stage 2 uses candidate adjacent call relations to represent local API transitions. The representation cannot encode every longer range data flow dependency.

For example, if an entity ID returned by the tool at step t needs to be used as an argument at step $t + 3$, the bigram structure may not directly express this cross-step dependency:

$$(u_t, u_{t+1}), \quad (u_{t+1}, u_{t+2}).$$

Here, Stage 2 may retrieve demonstrations with a similar local invocation order that provide insufficient guidance for later argument reuse.

Candidate adjacent call relations offer a compact local signal, but complete call dependency graphs remain outside the representation. Longer windows, explicit data flow graphs, or dependency-aware trajectory representations could cover these extended invocation chains.