Beyond Static Toolsets: Self-Evolving LLM Tool Agents via Continual Documentation Adaptation

Bin Wu [1] **Edgar Meij** [2] **Emine Yilmaz** [1]

[1] Centre for Artificial Intelligence, University College London
[2] Bloomberg
{bin.wu.23,emine.yilmaz}@ucl.ac.uk
emeij@bloomberg.net

Abstract

Large language models (LLMs) increasingly act as tool-using *agents*, and existing methods for evaluating and optimizing tool usage by LLMs typically assume a static tool environment with fixed APIs and documentation. In practice, toolsets evolve as tools are added, changed, or deprecated, introducing instability for agents that must retain prior competence while adapting to new capabilities. We formalize this challenge as the stability–adaptation dilemma. To address it, we propose ContDa, a continual documentation adaptation framework that provides a generalizable solution to this problem, enabling LLM agents to self-evolve by updating tool documentation. ContDa combines relation-guided exploration, which leverages functionally related existing tools as anchors to probe and identify new tool capabilities, with relation-aware adjustment that organizes overlapping tools and explicitly encodes usage preferences and fallback options among them. We then introduce complementary metrics that disentangle performance from stability and adaptation. Experiments across three evolution patterns on dynamic extensions of StableToolBench and RestBench show that ContDa consistently improves average performance by enhancing the discovery of new capabilities while incurring only limited loss of previously solved tasks, demonstrating documentation adaptation as an effective and lightweight mechanism for robust tool use in evolving environments. Our code is available at https://github.com/Bingo-W/ContDa.

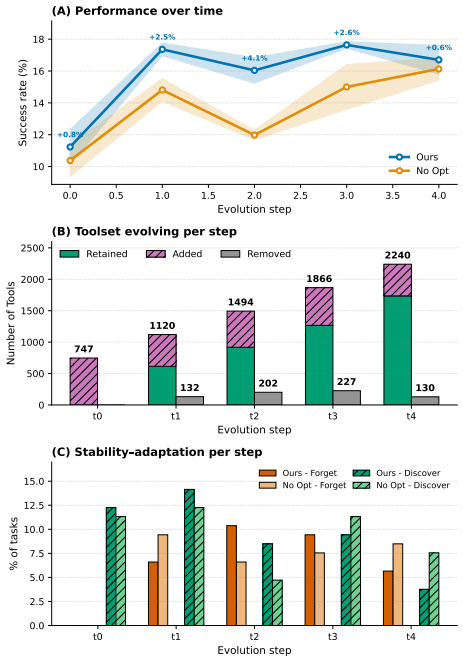

Figure 1: **I2-I** of StableToolBench under mixture evolution over time. (A) Success rate over time: ContDa is consistently superior under evolving environments. (B) Per-step toolset evolution (added/removed/retained) quantifying environment dynamics. (C) Per-step stability–adaptation metrics (*Forget*, *Discover*) show ContDa improves discovery at similar levels of forgetting.

1 Introduction

Large language models (LLMs) are increasingly deployed as tool-using agents (Qu et al., 2025b), capable of interacting with external APIs, software services, and databases to execute complex real-world tasks (Wu et al., 2024; Jimenez et al., 2024; Lei et al., 2025; Hu et al., 2025). Equipping LLMs with executable tools enables capabilities far beyond text-only reasoning, enabling LLMs to have *agency* in the form of observing and interacting with their environment, for instance supporting workflows including numerical computations and accessing private databases. However, practical deployments of LLMs rarely operate over a static toolset. As customer needs and business requirements evolve, enterprise infrastructure is continually updated, leading to tools and interfaces being added or deprecated. Tool-using agents should therefore not only select and call tools, but also adapt to evolving tool ecosystems to remain reliable in real-world settings. This challenge is further ex-

Findings of the Association for Computational Linguistics: ACL 2026, pages 21519–21539

acerbated under emerging standards such as Model Context Protocol (MCP) (Hou et al., 2025), where servers may dynamically change the tool list.

While recent work improves tool use via prompt and documentation optimization (Yuan et al., 2025; Wu et al., 2025), instruction-tuned models (Qin et al., 2024; Patil et al., 2024), and retrieval-based systems for a large toolset (Xu et al., 2024; Shi et al., 2025), most methods assume a static tool universe determined during prompt/documentation optimization or model training. In particular, approaches that refine documentation typically treat tools in isolation (Qu et al., 2025a), presuming stable surrounding tools, and retrieval-based systems generally operate under a closed-world assumption in which the tool corpus is fixed and fully known in advance (Patil et al., 2024). These assumptions limit robustness when tools appear, change, or disappear, a common reality in production environments.

Adapting LLM agents to evolving toolsets presents two core challenges. **Task retention** requires maintaining competence on previously solvable tasks even when key tools are removed, or when newly introduced tools with overlapping functionality create confusion. **Capability discovery** requires identifying and leveraging new tools rather than rigidly relying on legacy ones. These challenges give rise to a stability–adaptation dilemma (Wang et al., 2024b), where the agent should retain prior competence while incorporating new functionality. Retraining a model for every tool change is theoretically possible but computationally expensive and may degrade previously reliable behaviors, further motivating approaches that preserve and extend tool knowledge without model updates.

To address this problem, we introduce ContDa, a CONTinual Documentation Adaptation framework that enables the LLM agent to self-evolve as toolsets change. Rather than updating model weights, ContDa continually adapts the tool documentation that guides the agent. It first performs **relation-guided exploration**, identifying functionally-related tools from descriptions and exploration history, and using these related tools as anchors to generate probing queries that verify stated functionality and uncover potential capabilities for new tools. This targeted exploration prevents the agent from being constrained by incomplete local documentation and supports more reliable capability discovery. After exploring new tools, ContDa performs **relation-aware adjustment** to mitigate confusion among tools with overlapping functionality. It clusters tools with similar functionality and jointly adjusts their documentation to clarify functional distinctions and fallback options. This adjustment resolves ambiguity and reduces conflicts across tool documents. By combining relation-guided exploration with relation-aware adjustment, ContDa discovers new capabilities while preserving prior competence, enabling continual tool adaptation without retraining (see Figure 1).

Our contributions are the following. (1) We formalize tool-use under evolving toolsets and characterize the resulting stability–adaptation challenge of retaining prior competence while incorporating new functionality. (2) We propose ContDa, a continual documentation adaptation framework that enables LLM agents to self-evolve via relation-guided exploration and relation-aware adjustment without retraining. (3) We introduce an evaluation setup for dynamic tool changes, augmenting existing benchmarks with two metrics that quantify stability and adaptation, and show across three evolution patterns that ContDa consistently outperforms baselines.

2 Related Work

Tool-Use LLMs LLMs augmented with external tools have become a core paradigm for agentic systems (Schick et al., 2023; Yang et al., 2023b; Qin et al., 2024; Xi et al., 2025). One line of studies improves tool-use capability by training the underlying model. Early work uses supervised fine-tuning on tool-calling trajectories (Qin et al., 2024; Patil et al., 2024), while later approaches incorporate execution feedback (Gao et al., 2024; Wang et al., 2024a) and develop higher-quality demonstrations through Monte-Carlo search (Chen et al., 2025a), compositional task construction (Chen et al., 2025b), graph-based reasoning (Yin et al., 2025), or simulated agent–user interaction (Prabhakar et al., 2025). Reinforcement-learning-based methods (Guo et al., 2025) further study reward shaping (Qian et al., 2025), execution-grounded rewards (Feng et al., 2025), and curriculum-style optimization (Zhang et al., 2025b; Goldie et al., 2025). While effective, these approaches implicitly assume a *static* and *accurately documented* tool universe. Tracking toolset evolution via repeated model updates is costly and may introduce instability or forgetting. We instead consider the *evolving toolset* setting and focus on maintaining and improving tool competence without modifying model parameters, relying solely on documentation-level adaptation.

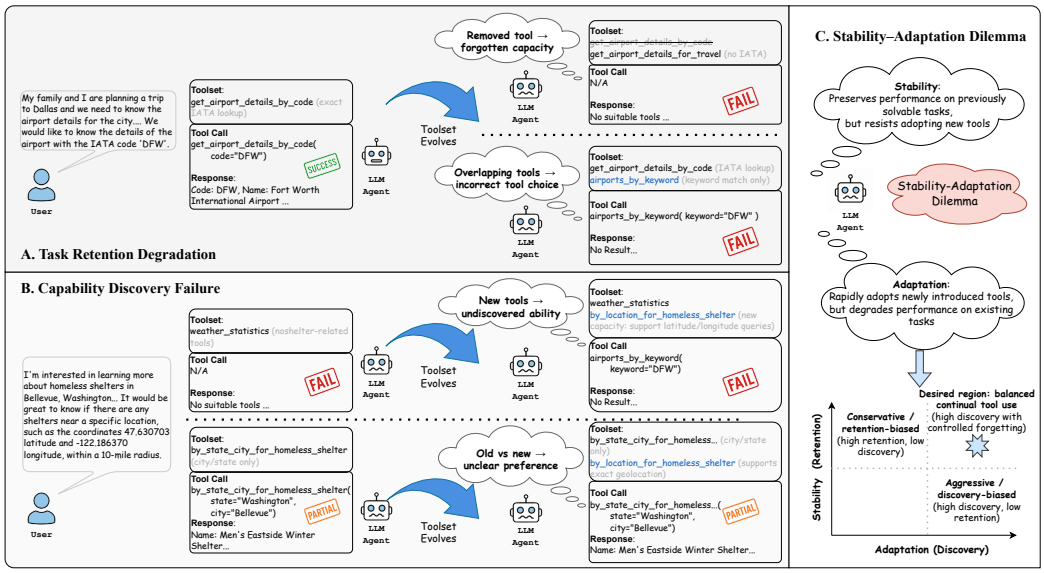

Figure 2: Failure patterns under evolving toolsets. (A) Task retention degradation: Forgetting or confusion when tools overlap or are removed. (B) Capability discovery failure: Newly introduced tools are underutilized. (C) These effects yield the stability–adaptation dilemma, where agents retain prior competence while adopting new capabilities.

Tool Documentation Optimization Another line of research focuses on enhancing tool use by refining documentation rather than updating the model itself (Pryzant et al., 2023; Yang et al., 2023a; Yuksekgonul et al., 2025). Yuan et al. (2025) standardize heterogeneous API descriptions; probe–refine frameworks (Qu et al., 2025a; Fang et al., 2025b) iteratively improve tool guidance; and Wu et al. (2025) jointly optimize prompts and documentation. However, these methods assume fixed toolsets and treat tools independently, overlooking functional relationships and how they shift as tools are added or removed. Consequently, they often become unstable, forgetting prior behaviors or failing to adopt newly introduced tools. Our method explicitly incorporates evolving tool relations and performs relation-guided exploration and relation-aware adjustment, enabling both retention and discovery.

Self-Evolving Agents Recent efforts investigate how agents can improve themselves over time (Fang et al., 2025a; Gao et al., 2025). Prior work explores self-evolution through model optimization (Gou et al., 2024; Yuan et al., 2024), prompt refinement (Chen et al., 2024; Khattab et al., 2024; Yuksekgonul et al., 2025), long-term memory organization (Zhong et al., 2024; Wang et al., 2025), and adaptive multi-agent workflow design (Zhang et al., 2025a; Zhuge et al., 2024). These directions demonstrate the value of enabling agents to autonomously adjust internal components. Our work complements these efforts by focusing on evolution in the *external tool environment*, offering a lightweight approach. Instead of adjusting model parameters or workflows, we continually refine tool documentation as tools appear or disappear. This expands self-evolving agents to settings where reliable tool use requires dynamic interfaces, not only internal mechanisms. Our approach is also related to memory-based strategies used in self-evolving agents, which store and retrieve past interaction trajectories or task–solution pairs at inference time (Chhikara et al., 2025; Xu et al., 2025). In contrast, ContDa does not maintain an explicit memory module. Instead, it uses accumulated interaction experience to iteratively refine structured tool documentation and relational annotations, without retaining raw trajectories as a retrievable store. This provides a complementary mechanism for continual adaptation by modifying the tool document itself, rather than augmenting inference with stored experiences.

3 Problem Definition: Evolving Toolsets

To evaluate the robustness of LLM-based tool-using agents under dynamic conditions, we consider the *Evolving Toolset* setting, in which the available tools change over time. We formalize such environments as a sequence of toolsets $\{\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_T\}$ with corresponding documentation $\{\mathcal{D}_t\}$. Each tool $t_{i,t} \in \mathcal{T}_t$ is associated with a documentation $d_{i,t} \in \mathcal{D}_t$. At each timestep t, the agent solves a query q using tools from $\mathcal{T}_t$. This setting closely reflects the behavior of the emerging MCP, where client

agents observe only the current tool list at each timestep. While version or modification metadata may be available in some MCP implementations, it is currently optional and not consistently reliable across tools. We therefore make no assumptions about access to such metadata or any change logs.

Stability–Adaptation Dilemma Compared to static environments, evolving toolsets introduce two challenges (Figure 2). (1) **Task retention degradation**: When the toolset changes, the agent may "forget" previously solvable tasks, either because essential tools are removed or because newly introduced tools with overlapping functionality cause confusion. (2) **Capability discovery failure**: Agents often struggle to recognize or utilize newly introduced tools, even when these tools enable previously unsolvable tasks or offer functionality superior to existing ones. Together, these challenges give rise to a stability–adaptation dilemma (Wang et al., 2024b): The agent must preserve established competence while incorporating new capabilities as the toolset evolves. We quantify this dilemma using dedicated metrics introduced in Section 5.

Failure Modes of Retention Degradation Task retention degradation in our setting does not arise from parameter updates, but from changes in retrieval and selection under evolving toolsets. As illustrated in Figure 2(A), this phenomenon can arise from (1) tool removal or (2) overlapping tools. Tool removal directly eliminates previously available capabilities, making certain tasks unsolvable. Overlapping tools typically manifest through failures at different stages of the decision process: the relevant tool may not be retrieved under the updated toolset, or it may be retrieved but not selected due to overlap with similar tools. A small-scale analysis among 84 forgetting cases shows that most failures arise from tool removal or incorrect tool selection under overlap, while retrieval failures occur less frequently (Table 1). This suggests that, although overlapping tools can introduce failures at both retrieval and selection stages, the dominant issue in practice lies in selecting among retrieved candidates rather than retrieving them.

Failure Type	# Case	Percentage
Tool unavailability	54	64.3%
Selection confusion	23	27.3%
Retrieval failure	7	8.4%

Table 1: Breakdown of retention failure modes under evolving toolsets among 84 cases on StableToolBench.

4 Continual Documentation Adaptation

In the evolving toolset setting (§3), the available tools change over time, introducing functional drift as new tools appear and existing ones are removed. Learning-by-interaction approaches improve documentation in static settings by iteratively generating probing queries and refining tool descriptions (Qu et al., 2025a; Fang et al., 2025b), but these do not explicitly address the stability–adaptation dilemma that arises when the tool environment itself evolves. Our goal is to preserve competence while enabling efficient discovery and adoption of new capabilities.

To this end, ContDa self-adapts tool documentation continually through two complementary stages at each timestep t. The core principle underlying both stages is to exploit relations between tools, capturing their functional relevance, i.e., the extent to which tools share similar functionality or can serve overlapping roles in task execution. Such relations provide a natural basis for both stages: tools with similar functionality offer useful signals to guide the exploration of newly introduced tools, while tools with overlapping roles require joint adjustment to clarify their distinctions and fallback behaviors.

Given the current toolset $\mathcal{T}_t$ and prior documentation $\mathcal{D}_{t-1}$, Stage I (*relation-guided exploration*) uses relationships to identify functionally similar tools and refine the documentation of newly introduced tools through targeted probing, while Stage II (*relation-aware adjustment*) uses relationships to organize tools with overlapping roles, clarifying distinctions and establishing fallback behaviors. Repeating these stages produces a documentation sequence $\mathcal{D}_0 \rightarrow \cdots \rightarrow \mathcal{D}_T$ aligned with the evolving toolset.

4.1 Stage I: Relation-Guided Exploration

In an evolving toolset, the agent must continually identify the capabilities of newly introduced tools as the toolset $\mathcal{T}_t$ changes. Prior work (Qu et al., 2025a; Fang et al., 2025b) addresses this by generating and executing natural-language probing queries and using the resulting responses to refine tool documentation. However, when probing is guided solely by a new tool's initial documentation, the resulting queries may be derived from incomplete, outdated, or underspecified descriptions, trapping exploration within existing gaps or errors.

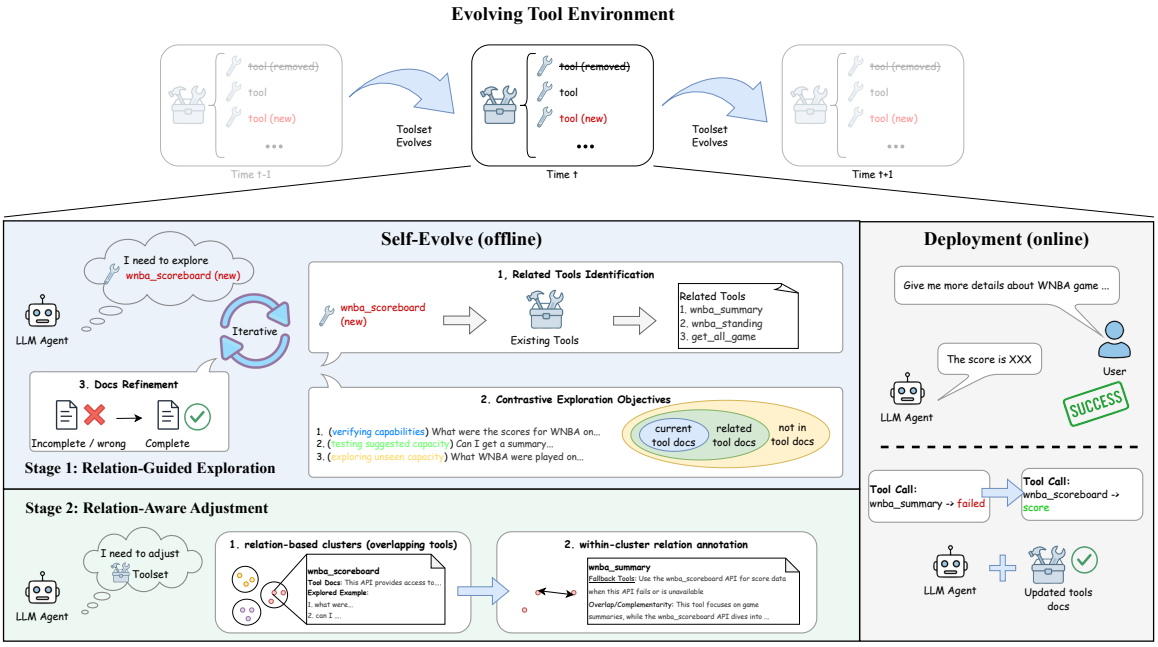

Figure 3: Overview of ContDa. As the toolset evolves (top), the agent adapts documentation rather than model parameters. Stage I performs relation-guided exploration of new tools; Stage II conducts relation-aware adjustment among overlapping tools, yielding continually refined documentation for robust tool use over time.

To address this limitation, Stage I adopts an iterative probe–refine procedure in which the agent generates a small set of natural-language probing queries, executes them using the new tool, and refines the tool documentation based on the observed responses. Specifically, this probing process is conditioned on *related* tools from the existing toolset: when a new tool t_i appears, the agent retrieves related tools from the previous toolset $\mathcal{T}_{t-1}$ as anchors to guide query generation. This relational grounding reduces blind exploration and directs the agent toward capabilities plausible for the tool's role. Across iterations, the agent alternates between related tool identification, contrastive exploration objective construction, query execution, and document refinement, forming a probe–refine loop similar to (Qu et al., 2025a; Fang et al., 2025b) but enhanced by relation-guided query construction.

Related Tools Identification For each new tool $t_i \in \mathcal{T}_t \backslash \mathcal{T}_{t-1}$, the agent retrieves a raw set of related tools $\mathcal{N}_i$ at every iteration. Although tools with moderate similarity offer useful functional cues for forming initial hypotheses about t_i, tools that are overly similar provide little beyond redundant signals and tend to confine exploration to the same assumption space as the tool's own documentation. To balance these effects, the final related tools $\mathcal{R}_i$ are selected by sampling from those whose similarity exceeds a threshold to control exploration cost:

$$\mathcal{N}_i = \{t_j \in \mathcal{T}_t \backslash \{t_i\} \mid s(t_i, t_j) \geq \tau\}, \quad (1)$$

$$\mathcal{R}_i = \textsc{Sample}_k(\mathcal{N}_i) \quad (2)$$

where $s(\cdot, \cdot)$ is a relevance score, τ is a minimum similarity threshold, and k controls the number of related tools sampled. This strategy ensures that $\mathcal{R}_i$ includes tools that are informatively related rather than trivially similar, forming a principled basis for relation-guided exploration.

Contrastive Exploration Objectives Given the related toolset $\mathcal{R}_i$ for tool t_i, the agent performs contrastive exploration by generating probing queries that target different regions of the tool's capability space. At each iteration, the agent generates one natural-language probing query for each of three complementary objectives, constructed using both the tool's own documentation $d_{i,t}$ and the documentation of related tools $\{d_{j,t} \mid t_j \in \mathcal{R}_i\}$. These queries are generated iteratively across multiple rounds, allowing the agent to progressively refine its understanding based on observed responses. Specifically, the three objectives correspond to: (1) verify explicitly stated capabilities in $d_{i,t}$, (2) test peer-suggested capabilities that may indicate replacements or extensions in $\{d_{j,t} | t_j \in \mathcal{R}_i\}$, and (3) explore capabilities beyond both sources. This yields a richer and more targeted characterization of the new tool. The resulting traces $\mathcal{T}_i$ are used

to refine the tool documents $d_{i,t}$ and obtain the documentation set $\mathcal{D}'_t$ for newly introduced tools. Detailed documentation refinement and stopping conditions appear in Appendix A.

4.2 Stage II: Relation-Aware Adjustment

Even with refined descriptions of new tools, ambiguity remains when multiple tools share overlapping capabilities or partially substitutable roles. This ambiguity arises at the level of groups of related tools and cannot be resolved by refining tools in isolation, often leading the agent to confuse these tools during selection. Stage II addresses this challenge by jointly adjusting documentation across related tools. The agent forms relation-based clusters that group tools with similar functional roles identified in Stage I. For each cluster, the agent appends concise relational notes that summarize fallback options and areas of overlap or complementarity. This adjustment reduces confusion among tools with closely related behavior.

Tool Clustering The tools are grouped into relation-based clusters $\mathcal{C}_t = \{C_1, C_2, \ldots\}$, based on their refined documents $\mathcal{D}'_t$ and exploration trace $\mathscr{T}$ from Stage I. Each cluster represents a functional neighborhood informed by exploration up to time t, and operating at the cluster level allows the agent to consider tools with overlapping capabilities or substitutable roles jointly.

Relation Specification For each tool t_i, the agent identifies relationships to other tools t_j within the same cluster, i.e., $t_i, t_j \in C_i$. The specified relation involves *fallback usage* to indicate alternative tools when one fails or is unavailable, and the overlap/-complementarity to highlight subtle differences in operating scope or strengths among similar tools. These relations are finally appended into the tool documentation $d_{i,t}$, leading to an updated documentation set $\mathcal{D}^\star_t$ to alleviate the confusion between similar tools. Appendix A provides the detailed clustering procedure and relation extraction rules.

4.3 Continual Procedure

At each timestep t, the agent begins with $\mathcal{T}_t$ and the prior refined documentation $\mathcal{D}^\star_{t-1}$, refines and produces updated documentation $\mathcal{D}^\star_t$, which also serves as the basis for the next timestep. This continual loop allows the agent to track toolset changes, preserve verified competencies, and incorporate new functionality over time. In practice, both stages run offline (e.g., on tool updates or periodic maintenance), and online execution only using D_t, adding no inference-time latency.

Conceptually, Stage I enhances the *adaptation* of the stability–adaptation dilemma by uncovering new tool capabilities, while Stage II supports *stability* by preserving prior competence through role stabilization. Together, they enable ContDa to remain effective under continual toolset evolution.

5 Experimental Setup

In what follows, we answer three research questions: **RQ1:** How well does ContDa maintain performance under different tool-evolution patterns? **RQ2:** How does ContDa balance stability and adaptation? **RQ3:** How do two stages of ContDa jointly improve adaptability and stability?

Benchmark and Simulation We evaluate ContDa under dynamic tool environments constructed from StableToolBench (Qin et al., 2024; Guo et al., 2024), which contains over 16K real-world APIs. Following prior work, we use the I1-I (single-tool), I2-I (intra-category multi-tool), and I3-I (inter-collection multi-tool) subsets (330 tasks) as the evaluation base. We additionally include RestBench (Song et al., 2023), comprising 117 tasks, to assess generalization across benchmarks. To study robustness under tool evolution, we generate a sequence of toolsets where available tools change over time. We consider three evolution patterns: (1) **Expansion-only**: only new tools are incrementally introduced. (2) **Reduction-only**: only existing tools are removed. (3) **Mixture**: both addition and removal occur at each step. At each timestep, the tool-use agent receives only the updated tool list (as in MCP-based deployments) and must solve tasks using the tools currently available. Importantly, following StableToolBench (Guo et al., 2024), tool documentation may already be incomplete or misaligned with actual tool behavior, and we do not introduce additional simulations of documentation updates. We use five time steps and run five separate simulations for each pattern.

Evaluation We follow ToolBench and use success rate as the evaluation protocol (Qin et al., 2024). All results are averaged over five runs. To quantify performance under evolving toolsets, we report one overall metric and two complementary stability–adaptation metrics based on an evaluation function $E(q, r_t)$ applied to task–response pairs: (1) **Average Performance** measures overall task

performance across timesteps and summarizes the stability–adaptation trade-off:

$$\text{AvgPerf} = \frac{1}{T+1} \sum_{t=0}^{T} \mathbb{E}_{q \in \mathcal{Q}}[E(q, r_t)]. \quad (3)$$

(2) ***Forget*** measures adaptation between consecutive timesteps, defined as the proportion of tasks newly solved at t:

$$\textit{Forget}_t = \frac{|\{q | E(q, r_{t-1}) > E(q, r_t)\}|}{|\mathcal{Q}|}. \quad (4)$$

(3) ***Discover***: Adaptability with respect to changes between consecutive timesteps, measuring the proportion of tasks newly solved at t:

$$\textit{Discover}_t = \frac{|\{q | E(q, r_{t-1}) < E(q, r_t)\}|}{|\mathcal{Q}|}. \quad (5)$$

Higher Discover reflects improved adaptation to newly introduced tools, while lower Forget indicates better retention of previously solved tasks. For reporting, we average *Forget* and *Discover* across timesteps, and additionally analyze their per-timestep dynamics when relevant.

Baselines and Models We use `gpt-4o-mini` and `qwen3-30B` as tool-use agents, spanning closed- and open-source models, with `gpt-4.1-mini` as the evaluator using prompts from the original benchmark. Tool retrieval and optimization employ `BAAI/bge-large-en-v1.5`. We compare ContDa against a naive baseline (**No Optimization**) and representative documentation-optimization methods (**EASYTOOL** (Yuan et al., 2025), **DRAFT** (Qu et al., 2025a)), which are among the strongest existing approaches for documentation-based tool-use optimization without modifying model parameters. We also include two ablations of ContDa that remove relation-aware adjustment or relation-guided exploration (**ContDa w/o Ad.**, **ContDa w/o Disc.**). Full implementation details are provided in Appendix B.

6 Results and Discussion

This section evaluates ContDa under evolving tool environments, covering overall performance (RQ1), stability–adaptation dynamics (RQ2), and ablations of discovery and adjustment (RQ3). Additional analyses are in Appendix C.

6.1 Overall Performance (RQ1)

We first compare ContDa with the baselines under the expansion-only, reduction-only, and mixture evolution scenarios. Across all settings, ContDa achieves the highest Average Performance and the strongest *Discover* across all three evolving patterns, with improvements that are statistically significant over the strongest baseline in most cases, see Table 2. This demonstrates consistent advantages under dynamic environments and successful tool adaptation. As shown in Table 2, these gains come with a moderate increase in *Forget* relative to conservative baselines, reflecting the inherent stability–adaptation trade-off. Nevertheless, ContDa maintains a higher overall success rate across all simulation runs, indicating the best stability–adaptation balance. Similar trends are observed on RestBench (Table 3), indicating that the advantages of ContDa generalize across benchmarks.

Scenario-level trends are consistent across both base models. Under expansion-only evolution, ContDa yields the largest performance gains, driven by substantially higher *Discover* as newly introduced tools become available. Under reduction-only evolution, where no new tools are added, ContDa still achieves higher Average Performance than all baselines, indicating that relation-aware adjustment helps mitigate degradation caused by tool removals. Under mixture evolution, the most challenging setting, ContDa consistently outperforms all baselines in both Average Performance and *Discover*, demonstrating robustness to realistic tool churn despite a moderate increase in *Forget*.

While the above results focus on controlled evolution patterns, it is also important to assess robustness under more realistic and less structured conditions. To this end, we conduct an additional one-step evaluation on the original ToolBench APIs, which are sourced from real RapidAPI tools and naturally contain practical inconsistencies such as parameter mismatches, unavailable endpoints, and return-format variations (Guo et al., 2024). Although no benchmark currently models controlled fine-grained schema drift explicitly, these naturally occurring irregularities provide a realistic test bed. As shown in Table 4, ContDa achieves the highest success rate among all methods, indicating its effectiveness under interface variations in real-world settings.

6.2 Stability–Adaptation Dynamics (RQ2)

To examine performance evolution over time, we analyze per-timestep success rates under the mixture setting (Figure 4). All methods show gradual improvement as the toolset expands, where EASYTOOL and DRAFT largely track the no-

	gpt-4o-mini			*qwen3-30B-A3B-Instruct*		
	Ave. Perf.	*Forget* ↓	*Discover* ↑	Ave. Perf.	*Forget* ↓	*Discover*↑
Expansion-only Evolution						
Original	$16.05_{\pm 0.43}$	$3.53_{\pm 0.26}$	$7.95_{\pm 0.20}$	$16.91_{\pm 0.38}$	$4.86_{\pm 0.07}$	$9.07_{\pm 0.12}$
EASYTOOL	$\underline{16.17}_{\pm 0.53}$	$3.79_{\pm 0.15}$	$8.00_{\pm 0.20}$	$17.39_{\pm 0.52}$	$4.73_{\pm 0.22}$	$9.39_{\pm 0.21}$
DRAFT	$16.07_{\pm 0.49}$	$3.85_{\pm 0.30}$	$8.20_{\pm 0.30}$	$17.18_{\pm 0.41}$	$4.94_{\pm 0.20}$	$9.25_{\pm 0.21}$
CONTDA	$\mathbf{17.58}^{**}_{\pm 0.48}$	$4.64_{\pm 0.21}$	$9.15_{\pm 0.08}$	$\mathbf{20.54}^{**}_{\pm 0.42}$	$5.82_{\pm 0.19}$	$10.48_{\pm 0.15}$
Reduction-only Evolution						
Original	$16.05_{\pm 0.43}$	$5.60_{\pm 0.23}$	$7.92_{\pm 0.27}$	$16.91_{\pm 0.38}$	$6.29_{\pm 0.14}$	$9.08_{\pm 0.15}$
EASYTOOL	$\underline{16.17}_{\pm 0.53}$	$5.56_{\pm 0.23}$	$7.95_{\pm 0.18}$	$17.39_{\pm 0.52}$	$6.58_{\pm 0.17}$	$9.30_{\pm 0.27}$
DRAFT	$16.07_{\pm 0.49}$	$5.64_{\pm 0.20}$	$8.19_{\pm 0.29}$	$17.18_{\pm 0.41}$	$6.25_{\pm 0.30}$	$9.20_{\pm 0.25}$
CONTDA	$\mathbf{17.70}_{\pm 0.65}$	$6.53_{\pm 0.21}$	$9.26_{\pm 0.42}$	$\mathbf{19.09}^{*}_{\pm 0.59}$	$6.85_{\pm 0.25}$	$10.48_{\pm 0.32}$
Mixture Evolution						
Original	$15.78_{\pm 0.38}$	$4.97_{\pm 0.31}$	$9.33_{\pm 0.19}$	$17.29_{\pm 0.30}$	$6.07_{\pm 0.17}$	$10.90_{\pm 0.21}$
EASYTOOL	$15.66_{\pm 0.41}$	$4.79_{\pm 0.13}$	$9.28_{\pm 0.09}$	$16.62_{\pm 0.24}$	$6.08_{\pm 0.20}$	$10.51_{\pm 0.19}$
DRAFT	$\underline{15.99}_{\pm 0.32}$	$4.96_{\pm 0.14}$	$9.38_{\pm 0.36}$	$16.51_{\pm 0.36}$	$5.66_{\pm 0.20}$	$10.33_{\pm 0.14}$
CONTDA	$\mathbf{17.46}^{*}_{\pm 0.62}$	$5.55_{\pm 0.22}$	$10.13_{\pm 0.36}$	$\mathbf{18.72}^{**}_{\pm 0.47}$	$6.67_{\pm 0.19}$	$11.82_{\pm 0.25}$

Table 2: Overall performance under evolving toolsets on StableToolBench. We report mean performance and standard error over five runs per evolution pattern (**best** and second-best). Statistical significance is reported when CONTDA outperforms the strongest baseline ($^{*}: p < 0.05$, $^{**}: p < 0.01$, paired t-test).

	Ave. Perf.	*Forget* ↓	*Discover* ↑
Expansion-only Evolution			
Original	$8.85_{\pm 0.79}$	$3.32_{\pm 0.26}$	$4.68_{\pm 0.34}$
EASYTOOL	$8.91_{\pm 0.67}$	$3.32_{\pm 0.28}$	$4.95_{\pm 0.29}$
DRAFT	$\underline{9.41}_{\pm 0.31}$	$3.52_{\pm 0.47}$	$5.62_{\pm 0.35}$
CONTDA	$\mathbf{10.48}_{\pm 0.85}$	$4.17_{\pm 0.36}$	$5.78_{\pm 0.46}$
Reduction-only Evolution			
Original	$8.85_{\pm 0.79}$	$2.53_{\pm 0.21}$	$4.71_{\pm 0.40}$
EASYTOOL	$8.91_{\pm 0.67}$	$2.77_{\pm 0.17}$	$4.88_{\pm 0.30}$
DRAFT	$\underline{9.41}_{\pm 0.31}$	$3.83_{\pm 0.30}$	$5.36_{\pm 0.35}$
CONTDA	$\mathbf{11.65}^{*}_{\pm 0.47}$	$4.68_{\pm 0.25}$	$7.18_{\pm 0.44}$
Mixture Evolution			
Original	$8.67_{\pm 0.45}$	$3.69_{\pm 0.13}$	$5.36_{\pm 0.18}$
EASYTOOL	$8.09_{\pm 0.62}$	$3.35_{\pm 0.17}$	$4.84_{\pm 0.25}$
DRAFT	$\underline{9.49}_{\pm 0.72}$	$3.59_{\pm 0.42}$	$5.58_{\pm 0.36}$
CONTDA	$\mathbf{11.20}^{*}_{\pm 0.94}$	$5.19_{\pm 0.43}$	$7.38_{\pm 0.55}$

Table 3: Performance of *gpt-4o-mini* on RestBench.

Method	Success Rate (%)
Original	17.87
EasyTool	18.72
DRAFT	18.27
CONTDA	**19.75**

Table 4: Performance under realistic interface irregularities on StableToolBench APIs (one-step evaluation).

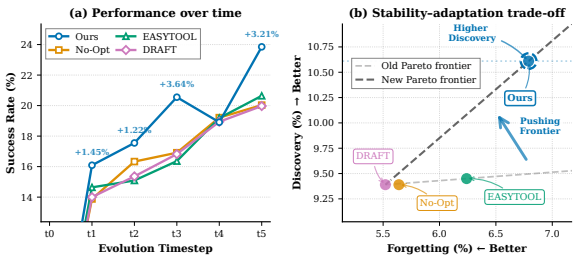

Figure 4: Stability–Adaptation Analysis on StableToolBench. (a) Success rate under mixture evolution; (b) Forgetting-Discovery trade-off under mixture evolution.

optimization baseline, with marginal gains when the toolset becomes nearly complete. This indicates limited adaptivity of non-relation optimization under changing tool environments, consistent with their low *Discover* in Table 2. In contrast, CONTDA exhibits a steeper performance increase over time, driven by more effective discovery of newly introduced tools via relation-aware replacement and identification of functionally superior alternatives.

The Forgetting-Discovery frontier in Figure 4 reveals that CONTDA yields the best trade-off, extending the Pareto frontier (Jin and Sendhoff, 2008) toward greater adaptability. It attains the largest *Discover* gains while maintaining reasonable *Forget* levels, indicating that any instability introduced by exploration is outweighed by its long-term benefits.

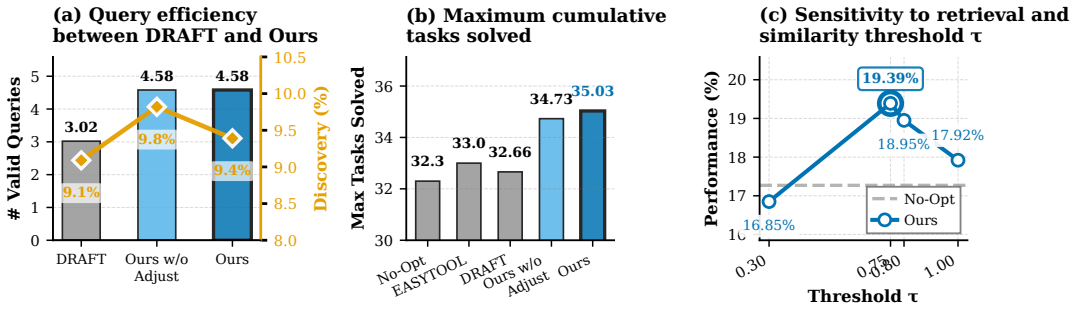

Figure 5: Analysis of relation-guided exploration (Stage I) on StableToolBench: (a) query efficiency, (b) maximum capacity, (c) and sensitivity to retrieval threshold τ.

	Ave. Perf.	*Forget* ↓	*Discover* ↑
	Expansion-Only Evolution		
Original	17.37	4.06	8.16
ContDa w/o Ad.	↑ 19.06	5.17	9.82
ContDa w/o Disc.	↓ 15.95	4.61	8.70
ContDa	↑ 19.40	4.85	9.39
	Mixture Evolution		
Original	17.27	6.06	10.00
ContDa w/o Ad.	↑ 18.53	6.00	10.48
ContDa w/o Disc.	↓ 16.65	6.12	10.06
ContDa	↑ 19.39	6.36	11.21

Table 5: Ablation Study on StableToolBench.

6.3 Discovery & Adjustment Analysis (RQ3)

We isolate the contribution of relation-guided exploration by comparing ContDa with probing-based baselines. As shown in Table 5, Stage I alone (ContDa w/o Ad.) achieves higher *Discover* and better overall performance than non-optimization methods, demonstrating that relational cues are effective for uncovering newly introduced or previously overlooked tools. However, Stage I by itself underperforms the full pipeline, where it attains higher *Discover*, but also incurs higher *Forget*, confirming that exploration without adjustment leads to *Forget* issue. Figure 5 further shows that relation-guided exploration generates a more diverse set of informative probes, yielding a larger cumulative number of tasks solved across timesteps.

We examine the sensitivity of relation-guided exploration to the similarity threshold τ. As shown in Figure 5(c), ContDa consistently outperforms the no-optimization baseline across a broad range of τ, indicating robustness to reasonable hyperparameter choices. Performance peaks at an intermediate threshold, where retrieved tools are sufficiently related to provide informative anchors while avoiding redundancy. When τ is too low or too high (i.e., top-k tools), performance drops, supporting the design of balanced retrieval.

When we analyze the effect of relation-aware adjustment (see Table 5), removing relation-guided discovery (ContDa w/o Disc.) causes a clear drop in Average Performance under both expansion-only and mixture evolution, despite comparable or lower *Forget*, indicating that the degradation mainly stems from reduced *Discover*. Without Stage I to identify and characterize new tools, Stage II constructs relations from incomplete or inaccurate understanding, leading to ineffective adjustment. Consequently, adjustment alone cannot compensate for missing discovery, whereas the full pipeline achieves the best performance by discovering new capabilities and organizing them via relation-aware adjustment.

7 Conclusion

We present ContDa, a continual documentation adaptation framework for LLM-based tool use under evolving toolsets. By integrating relation-guided exploration with relation-aware adjustment, ContDa enables agents to adapt to new tools while preserving prior competence without modifying parameters. Experiments across three evolution patterns on StableToolBench and RestBench show improved overall performance and discovery with limited forgetting, highlighting documentation as an effective substrate for continual tool-use adaptation.

Limitations

While ContDa enhances continual tool-use adaptation without modifying model parameters, it cannot fully eliminate forgetting when tool functionality changes substantially, as all adjustments operate through documentation rather than weight updates. This parameter-free design improves stability but may underreact to drastic behavioral shifts, suggesting potential benefits from integrating lightweight model memory in future work. In addition, our

evaluation relies on controlled simulations of tool evolution derived from StableToolBench and RestBench, which enable systematic comparison across expansion, reduction, and mixture patterns but may not capture all irregularities of real-world MCP-style deployments. We also fix the number of evolution steps and simulation runs (e.g., five) for tractability; exploring sensitivity to longer horizons and different evolution lengths is left to future work. Studying ContDa in production-scale dynamic tool environments, therefore, represents an important next step.

Acknowledgments

Bin Wu is supported by the Bloomberg Data Science Ph.D. Fellowship. We thank Sambhav Kothari for discussions and for providing feedback on our project.

References

Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. 2003. An introduction to mcmc for machine learning. *Machine learning*, 50(1):5–43.

Paul S Bradley, Kristin P Bennett, and Ayhan Demiriz. 2000. Constrained k-means clustering. *Microsoft Research, Redmond*, 20(0):0.

Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje Karlsson, Jie Fu, and Yemin Shi. 2024. Autoagents: a framework for automatic agent generation. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pages 22–30.

Guoxin Chen, Zhong Zhang, Xin Cong, Fangda Guo, Yesai Wu, Yankai Lin, Wenzheng Feng, and Yasheng Wang. 2025a. Learning evolving tools for large language models. In *The Thirteenth International Conference on Learning Representations*.

Mingyang Chen, Tianpeng Li, Fan Yang, Hao Liang, Bin CUI, Wentao Zhang, Zenan Zhou, and 1 others. 2025b. Facilitating multi-turn function calling for llms via compositional instruction tuning. In *The Thirteenth International Conference on Learning Representations*.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building production-ready AI agents with scalable long-term memory. In *ECAI 2025 - 28th European Conference on Artificial Intelligence, 25-30 October 2025, Bologna, Italy - Including 14th Conference on Prestigious Applications of Intelligent Systems (PAIS 2025)*, Frontiers in Artificial Intelligence and Applications, pages 2993–3000. IOS Press.

Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, and 1 others. 2025a. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems. *arXiv preprint arXiv:2508.07407*.

Wei Fang, Yang Zhang, Kaizhi Qian, James R. Glass, and Yada Zhu. 2025b. PLAY2PROMPT: Zero-shot tool instruction optimization for LLM agents via tool play. In *Findings of the Association for Computational Linguistics: ACL 2025*.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*.

Huan-ang Gao, Jiayi Geng, Wenyue Hua, Mengkang Hu, Xinzhe Juan, Hongzhang Liu, Shilong Liu, Jiahao Qiu, Xuan Qi, Yiran Wu, and 1 others. 2025. A survey of self-evolving agents: On path to artificial super intelligence. *arXiv preprint arXiv:2507.21046*.

Shen Gao, Zhengliang Shi, Minghang Zhu, Bowen Fang, Xin Xin, Pengjie Ren, Zhumin Chen, Jun Ma, and Zhaochun Ren. 2024. Confucius: Iterative tool learning from introspection feedback by easy-to-difficult curriculum. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 18030–18038.

Anna Goldie, Azalia Mirhoseini, Hao Zhou, Irene Cai, and Christopher D Manning. 2025. Synthetic data generation & multi-step rl for reasoning & tool use. *arXiv preprint arXiv:2504.04736*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, and 1 others. 2024. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

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 *ACL (Findings)*.

Xinyi Hou, Yanjie Zhao, Shenao Wang, and Haoyu Wang. 2025. Model context protocol (mcp): Landscape, security threats, and future research directions. *arXiv preprint arXiv:2503.23278*.

Mengkang Hu, Yuhang Zhou, Wendong Fan, Yuzhou Nie, Bowei Xia, Tao Sun, Ziyu Ye, Zhaoxuan Jin, Yingru Li, Qiguang Chen, and 1 others. 2025. Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation. *arXiv preprint arXiv:2505.23885*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Yaochu Jin and Bernhard Sendhoff. 2008. Pareto-based multiobjective machine learning: An overview and case studies. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(3):397–415.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, Heather Miller, and 1 others. 2024. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin SU, ZHAOQING SUO, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, and 1 others. 2025. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. In *The Thirteenth International Conference on Learning Representations*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Manoj Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Quoc Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan Wang, Silvio Savarese, and Caiming Xiong. 2025. APIGen-MT: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7957–7968.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations*.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2025a. From exploration to mastery: Enabling llms to master tools via self-driven interactions. In *The Thirteenth International Conference on Learning Representations*.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. 2025b. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343.

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. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. 2025. Retrieval models aren't tool-savvy: Benchmarking tool retrieval for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 24497–24524, Vienna, Austria. Association for Computational Linguistics.

Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, and 1 others. 2023. Restgpt: Connecting large language models with real-world restful apis. *arXiv preprint arXiv:2306.06624*.

Boshi Wang, Hao Fang, Jason Eisner, Benjamin Van Durme, and Yu Su. 2024a. Llms in the imaginarium: Tool learning through simulated trial and error. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10583–10604.

Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. 2024b. A comprehensive survey of continual learning: Theory, method and application. *IEEE transactions on pattern analysis and machine intelligence*, 46(8):5362–5383.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2025. Agent workflow memory. In *Forty-second International Conference on Machine Learning*.

Bin Wu, Edgar Meij, and Emine Yilmaz. 2025. A joint optimization framework for enhancing efficiency of tool utilization in llm agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22361–22373.

Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis Ioannidis, Karthik Subbian, Jure Leskovec, and James Y Zou. 2024. Avatar: Optimizing llm agents for tool usage via contrastive reasoning. *Advances in Neural Information Processing Systems*, 37:25981–26010.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, and 1 others. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101.

Qiancheng Xu, Yongqi Li, Heming Xia, and Wenjie Li. 2024. Enhancing tool retrieval with iterative feedback from large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 9609–9619.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-mem: Agentic memory for LLM agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023a. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.

Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. 2023b. Gpt4tools: Teaching large language model to use tools via self-instruction. *Advances in Neural Information Processing Systems*, 36:71995–72007.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Fan Yin, Zifeng Wang, I-Hung Hsu, Jun Yan, Ke Jiang, Yanfei Chen, Jindong Gu, Long Le, Kai-Wei Chang, Chen-Yu Lee, Hamid Palangi, and Tomas Pfister. 2025. Magnet: Multi-turn tool-use data synthesis and distillation via graph translation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32600–32616, Vienna, Austria. Association for Computational Linguistics.

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

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason E Weston. 2024. Self-rewarding language models. In *Forty-first International Conference on Machine Learning*.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Pan Lu, Zhi Huang, Carlos Guestrin, and James Zou. 2025. Optimizing generative ai by backpropagating language model feedback. *Nature*, 639(8055):609–616.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xiong-Hui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2025a. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations*.

Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. 2025b. Nemotron-research-tool-n1: Tool-using language models with reinforced reasoning. *arXiv preprint arXiv:2505.00024*.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*.

A Additional Method Details

Appendix A details the implementation of ConTDa, expanding on the mechanisms introduced in Section §4. We describe related-tool identification, documentation refinement, and their integration into a continual adaptation procedure for reproducibility.

A.1 Related Tools Identification

For each new tool $t_i \in \mathcal{T}_t \setminus \mathcal{T}_{t-1}$, the agent identifies a raw pool of related tools based on embedding similarity. Let $s(t_i, t_j)$ denote a relevance score computed from the embeddings of tool descriptions or names. We define:

$$\mathcal{N}_i = \{t_j \in \mathcal{T}_t \setminus \{t_i\} : s(t_i, t_j) \geq \tau\}, \quad (6)$$

$$\mathcal{R}_i = \textsc{Sample}_k(\mathcal{N}_i), \quad (7)$$

where τ controls minimal relevance and k limits exploration breadth. Sampling promotes diversity and prevents redundancy by avoiding the exclusive selection of highly similar tools. In practice, we use the same retrieval model employed by the agent (e.g., `BAAI/bge-large-en-v1.5`) to obtain embeddings, and compute relevance scores using cosine similarity over these embeddings. For simplicity, we adopt random sampling and leave the exploration of alternative sampling strategies to future work.

A.2 Contrastive Exploration Objectives

Given $\mathcal{R}_i$, the agent formulates probing queries that examine different regions of the tool's potential capability space. Specifically, the queries fall into three categories:

- **Self-verification (Known / Overlap)**: validate functionality that is explicitly stated in $d_{i,t}$ or strongly aligned with capabilities of related tools.
- **Peer-inspired (Unknown-from-Peers / Comparative Frontier)**: test functionality suggested in the documentation of tools in $\mathcal{R}_i$ but not confirmed in t_i, which may indicate extensions or replacements of existing capabilities.
- **Open-ended exploration (Unknown-Novel / Emergent Frontier)**: probe for entirely new functionality not implied by either the documentation of t_i or that of related tools.

These three tracks allow the agent to contrast retained behavior with peer-suggested and novel behaviors, enabling a more complete characterization of the tool.

A.3 Redundancy Filtering

To avoid repetitive and redundant exploration, we filter out the generated queries that are too similar to the already explored ones. Specifically, we compute the semantic similarity between the generated queries and each of the already-explored queries, and the query will be filtered out if the semantic similarity to any already-explored queries surpasses the threshold.

A.4 Documentation Refinement

After executing the probing queries, the agent refines the documentation of the new tool by incorporating evidence obtained from the collected query-response pairs and by contrasting this evidence with the documentation of related tools. Following the previous studies in textual optimization (Qu et al., 2025a; Yuksekgonul et al., 2025; Wu et al., 2025), our refinement procedure consists of two steps.

Step 1: Suggestion Generation Let $\mathscr{T}_i = \{(q_\ell, r_\ell)\}$ denote the explored query-response pairs for tool t_i, and let $\{d_{j,t} \mid t_j \in \mathcal{R}_i\}$ denote the documentation of related tools. The agent synthesizes these sources to produce a set of revision suggestions S_i. This process compares the explored query-response pairs in $\mathcal{T}_i$ with (1) the capabilities described in $d_{i,t}$, (2) the functionality implied by related tools, and (3) any emergent behaviors uncovered through open-ended probes. Formally,

$$S_i = \textsc{Suggest}\big(d_{i,t},\ \mathscr{T}_i,\ \{d_{j,t} : t_j \in \mathcal{R}_i\}\big), \quad (8)$$

where $\textsc{Suggest}(\cdot)$ identifies inconsistencies between documentation and actual behavior, missing or unsupported peer-aligned capabilities, and novel capacities revealed during exploration. The suggestions emphasize improving the accuracy, completeness, clarity, and conciseness of the documentation.

Step 2: Document Updating Given the suggestion set S_i, the agent generates a refined document $d'_{i,t}$ by revising the existing description and integrating supporting information from related tools:

$$d'_{i,t} = \textsc{Refine}\big(d_{i,t},\ S_i,\ \{d_{j,t} : t_j \in \mathcal{R}_i\}\big). \quad (9)$$

The operator $\textsc{Revise}(\cdot)$ rewrites the documentation to reflect verified behavior, incorporate overlaps with peers, clarify unsupported peer features, and position any emergent capabilities identified during exploration. The refinement further proposes high-level directions for subsequent probing, focusing on

unresolved peer-suggested behavior and ambiguous novel signals. This document updating step is applied iteratively within Stage I until the termination condition is reached.

A.5 Termination Condition

Inspired by the existing work (Qu et al., 2025a), we also employ a tool-adaptive termination mechanism to manage the relation-guided exploration. The probing-refinement process is performed iteratively until meeting either semantically converged or beyond budget.

Refinement Converged Following the existing work, we also detect the semantically converged via measuring the degree of change Δ between iterations. Specifically, both lexical and semantical metrics are employed for calculating the degree of change between iteration k and $k-1$:

$$\Delta = \frac{cos(e_k^t, e_{k-1}^t) + BLEU(d_k, d_{k-1})}{2}, \quad (10)$$

where $cos(\cdot,\cdot)$ is the cosine distance between semantic embedding and BLEU is the word-match score for computing n-gram overlap (Papineni et al., 2002). If the change Δ exceeds a predefined termination threshold, the relation-guided exploration is stopped.

A.6 Relation-Based Clustering

To prepare for Stage II, tools are grouped into clusters using embeddings derived from:

- refined documentation $d'_{i,t}$, and
- exploration trace summaries $\mathscr{T}_i$.

Regarding the input context length of LLMs, we constrain the number of tools in each tool. To implement it, we employ the constraint K-means clustering algorithm (Bradley et al., 2000), whose goal is:

$$\min_{\{C_k\}} \sum_{k=1}^{K} \sum_{t_i \in C_k} \| t_i - \mu_k \|^2 \quad s.t.\ |C_k| \leq \lambda, \quad (11)$$

where K is number of clusters, μ_k is the centroid of the k-th cluster, and λ is the maximum number of tools in each cluster. We apply the clustering with cosine distance and a threshold λ to obtain clusters $\mathcal{C}_t = \{C_1, C_2, \ldots\}$.

A.7 Relation Extraction and Specification

Within each cluster C_ℓ, the agent identifies two types of relations:

Fallback Usage Tools t_i and t_j form a fallback pair if their functional overlap exceeds γ, computed via textual embedding similarity or overlap in validated capabilities.

Overlap / Complementarity Tools that serve similar purposes but differ in scope, precision, or input/output patterns are annotated with contrastive notes specifying their strengths, limitations, and usage boundaries.

These relations are embedded into documentation templates that specify preference rules (e.g., "prefer t_j for longer inputs", "use t_i when t_j fails").

A.8 Algorithmic Summary

Algorithm 1 summarizes the overall procedure.

A.9 Practical Cost and Deployment Considerations

ContDa is designed as a documentation maintenance process rather than an online inference-time component. Relation-guided exploration and relation-aware adjustment can be run asynchronously, for example when new tools are added or removed, or as a scheduled periodic job (e.g., nightly). As a result, the additional LLM calls introduced by ContDa are amortized over all subsequent user queries until the next tool change. The overall computational cost therefore scales with the frequency of toolset evolution rather than with query volume, making the approach suitable for real-world deployments where tool ecosystems evolve incrementally while serving a large number of user requests.

B Additional Experimental Setup Details

Appendix B provides additional details of the experimental setup used in our evaluation. We describe the benchmarks, evolving toolset simulation, evaluation protocol, model and prompt configurations, and baselines, which together ensure reproducibility and clarity beyond the main paper.

B.1 Benchmark and Task Construction

We build on StableToolBench (Guo et al., 2024), which includes over 16,000 real-world APIs collected from RapidAPI along with their natural-language documentation. From this benchmark, we select three subsets used in Qin et al. (2024):

- **I1-I**: single-tool instructions;
- **I2-I**: intra-category multi-tool instructions;

Algorithm 1: CONTDA: Continual Documentation Adaptation Under Evolving Toolsets

Input: Toolsets $\{\mathcal{T}_t\}_{t=0}^{T}$; initial documentation $\mathcal{D}_0^\star$; score function $s(\cdot,\cdot)$; similarity threshold τ; sample size k.

Output: Final documentation $\mathcal{D}_T^\star$.

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

  `// Initialize documentation for retained tools`

  foreach $t_i \in (\mathcal{T}_t \cap \mathcal{T}_{t-1})$ **do**

    $d'_{i,t} \leftarrow d^\star_{i,t-1}$;

  $\textsc{New}_t \leftarrow \mathcal{T}_t \setminus \mathcal{T}_{t-1}$;

  `// Stage I: Relation-Guided Exploration`

  foreach $t_i \in \textit{New}_t$ **do**

    $m \leftarrow 0$; $d^{(0)}_{i,t} \leftarrow d_{i,t}$; $\mathscr{T}_i \leftarrow \varnothing$;

    repeat

      $m \leftarrow m + 1$;

      `// Identify related tools dynamically`

      $\mathcal{N}_i^{(m)} \leftarrow \{t_j \in \mathcal{T}_t \setminus \textsc{New}_t : s(d^{(m-1)}_{i,t}, d'_{j,t}) \geq \tau\}$;

      $\mathcal{R}_i^{(m)} \leftarrow \textsc{Sample}_k(\mathcal{N}_i^{(m)})$;

      `// Generate probing queries (verify, suggested, explore)`

      $\mathsf{Q}_i^{(m)} \leftarrow \textsc{GenerateProbes}(d^{(m-1)}_{i,t}, \mathcal{R}_i^{(m)})$;

      `// Remove redundant queries`

      $\mathsf{Q}_i^{(m)} \leftarrow \textsc{FilterQueries}(\mathsf{Q}_i^{(m)}, \mathscr{T}_i)$;

      `// Execute queries`

      $\mathscr{T}_i^{(m)} \leftarrow \textsc{Execute}(\mathsf{Q}_i^{(m)})$;

      $\mathscr{T}_i \leftarrow \mathscr{T}_i \cup \mathscr{T}_i^{(m)}$;

      `// Refine documentation using execution results`

      $d^{(m)}_{i,t} \leftarrow \textsc{Refine}(d^{(m-1)}_{i,t}, \mathscr{T}_i^{(m)}, \mathcal{R}_i^{(m)})$;

    until *TerminationCondition*($d^{(m)}_{i,t}$, $d^{(m-1)}_{i,t}$)

    $d'_{i,t} \leftarrow d^{(m)}_{i,t}$;

  `// Stage II: Relation-Aware Documentation Adjustment`

  $\mathcal{C}_t \leftarrow \textsc{BuildClusters}(\mathcal{T}_t, \{d'_{i,t}\}, \{\mathscr{T}_i\})$;

  foreach *cluster* $C \in \mathcal{C}_t$ **do**

    $\mathsf{Rel}_C \leftarrow \textsc{AnnotateRelations}(C)$;

    foreach $t_i \in C$ **do**

      $d^\star_{i,t} \leftarrow \textsc{Refine}(d'_{i,t}, \mathsf{Rel}_C)$;

  $\mathcal{D}_t^\star \leftarrow \{d^\star_{i,t} : t_i \in \mathcal{T}_t\}$;

- **I3-I**: inter-collection multi-tool instructions.

These subsets together comprise 330 tasks, which form a fixed evaluation pool shared across all evolution scenarios and timesteps.

In addition to StableToolBench, we also evaluate on RestBench (Song et al., 2023), a benchmark designed for RESTful API-based tool use (a subset of StableToolBench toolset). We filter out the task-oriented instruction (e.g., delete or add songs), and finally obtain 117 tasks.

We focus on the end-to-end pipeline, involving tool retrieval and tool use. Regarding the large available toolset, we first perform the retrieval to identify the top-k relevant tools, and then, given tasks and the retrieved tools, LLMs generate the final response based on the ReAct framework (Yao et al., 2023).

B.2 Evolving Toolset Simulation

To comprehensively capture diverse real-world patterns of tool evolution, we design three simulation scenarios:

- **Expansion-only evolution:** New tools are incrementally introduced over time, while no previously available tools are removed. This setting evaluates the agent's ability to discover and leverage newly introduced functionalities.
- **Reduction-only evolution:** At each time step, a subset of existing tools is removed without introducing new ones. This scenario assesses the agent's ability to retain competence and re-adapt when familiar tools disappear.
- **Mixture evolution:** Both new tools are introduced and old tools are removed at each step. This represents the most general and challenging case, requiring the agent to simultaneously retain prior capabilities and explore new ones.

Markov-based Simulation Process. We simulate the above evolving toolsets, using a Markov-based random sampling process (Andrieu et al., 2003), defined over a finite tool pool $\mathcal{T}$. At the initial step ($t = 0$), an initial subset $\mathcal{S}_0 \subseteq \mathcal{T}$ is randomly sampled. For each subsequent step ($t > 0$), the subset evolves via partial replacement as follows:

1. A random subset $\mathcal{R}_t \subseteq \mathcal{S}_{t-1}$ is selected for removal.
2. A replacement subset $\mathcal{A}_t \subseteq (\mathcal{T} \setminus \mathcal{S}_{t-1})$ of the same cardinality as $\mathcal{R}_t$ is randomly sampled.
3. The updated subset of available tools is then given by:

$$\mathcal{S}_t = (\mathcal{S}_{t-1} \setminus \mathcal{R}_t) \cup \mathcal{A}_t. \tag{12}$$

This process generates a sequence of tool subsets $\{\mathcal{S}_t\}_{t=0}^{T}$, each depending only on its immediate predecessor, thereby forming a first-order Markov chain. By adjusting the probabilities of removal and addition, we can flexibly instantiate the three evolution scenarios described above.

Connection to MCP. Although our simulation explicitly adds and removes tools between steps, the observed signal closely mirrors MCP-based deployments. Under Model Context Protocol (MCP), servers notify clients of changes via the `tools/list_changed` event, which exposes only the updated list of available tools, without any explicit diff information. Agents must infer which tools are new, removed, or retained by comparing consecutive lists. Our Markov-based evolution thus provides a controlled abstraction of the coarse-grained update signal that real agents would observe in MCP environments.

B.3 Evaluation Protocol

We adopt the LLM-as-a-Judge success-rate protocol from ToolBench. For each timestep t, toolset T_t, and query $q \in Q$, the base model (either `gpt-4o-mini-2024-07-18` or `qwen3-30B-A3B-Instruct-2507`) produces a response $r_t(q)$ via tool calls. An evaluation model then decides whether the response is successful.

Let $E(q, r_t) \in \{0, 1\}$ denote the binary judgment returned by the evaluator for (q, r_t). The success rate at timestep t is

$$\text{SuccessRate}_t = \mathbb{E}_{q \in Q}[E(q, r_t)]. \tag{13}$$

We then compute the temporal metrics introduced in Section 3:

$$\textit{Forget}_t = \frac{|\{q : E(q, r_{t-1}) > E(q, r_t)\}|}{|Q|}, \tag{14}$$

$$\textit{Discover}_t = \frac{|\{q : E(q, r_{t-1}) < E(q, r_t)\}|}{|Q|}, \tag{15}$$

$$\text{AvgPerf} = \frac{1}{T+1} \sum_{t=0}^{T} \text{SuccessRate}_t. \tag{16}$$

Each evolution scenario (expansion-only, reduction-only, mixture) is simulated with five random seeds for the Markov process, and we report mean values (and standard deviations where relevant) across these runs.

B.4 Models and Prompt Setup

Base models. We consider two base LLMs as the tool-using agents:

- `gpt-4o-mini-2024-07-18`;
- `qwen3-30B-A3B-Instruct-2507`.

For each base model, we run the full evolving-toolset evaluation pipeline with and without ContDa, yielding two sets of results (one per base).

Evaluator model. For both bases, we use `gpt-4.1-mini-2025-04-14` as the evaluator in the LLM-as-a-Judge protocol. Using a separate, stronger evaluator model avoids contamination between the model being optimized and the model providing evaluation feedback.

System Prompt for Relation-Guided Exploration

You are an assistant that helps optimize API/tool documentation. Your current role is **Query Proposal (Iterative Mode)**.

Goal Generate natural, realistic user queries to explore the tool's functionality.

Instructions

1. Generate exactly one query for each of the three tracks:
   - **Known (Overlap):** validate functionality already described in the current doc or strongly aligned with peers.
   - **Unknown-from-Peers (Comparative Frontier):** probe functionality described in peers but not yet confirmed in this tool.
   - **Unknown-Novel (Emergent Frontier):** probe for functionality not suggested by either current docs or peers (entirely new possibilities).
2. Each query must require calling the tool **once**.
3. Make queries **natural** — avoid exposing API names, parameter names, or technical jargon.
4. Provide **specific values** for all required parameters.
5. Extract the parameters in the exact format the tool expects.
6. Avoid queries too similar to those already in **ExploredQueries**.
7. Use **Suggestions** (if any) and **SimilarToolsDocs** to guide query choice.
8. For each query, include a short **Rationale**.

Response Format Return a JSON object with exactly these fields:

- "Stage": must be "QueryProposal"
- "KnownQuery": object with "User Query", "Parameters", "Rationale"
- "UnknownFromPeersQuery": object with "User Query", "Parameters", "Rationale"
- "UnknownNovelQuery": object with "User Query", "Parameters", "Rationale"

Retriever model. For all bases, including testing and optimization in our ContDa, we employ `BAAI/bge-large-en-v1.5` as the retrieval model, and top-k is set as five.

Decoding settings. For task execution and evaluation, we set the decoding parameters as: (1) temperature $= 2 \times 10^{-4}$ (2) top-$p = 0.99$ for reproducibility, and disable the reasoning ability of the Qwen3 model for fair comparison.

These settings are applied consistently across timesteps and across both base models. During the documentation evolution stage (i.e., when ContDa uses an LLM to rewrite or refine tool descriptions), we do not impose strict sampling constraints, allowing more diverse generations that can reveal a broader set of potential behaviors.

Prompts. We reuse the official prompts from ToolBench for task execution and automatic evaluation. The prompts used for ContDa's relation-guided exploration and relation-aware adjustment (e.g., for generating probing queries, summarizing exploration traces, and rewriting documentation

System Prompt for Relation-Aware Adjustment

You are refining documentation for a set of tools that belong to the same functional cluster. Your goal is to append a concise relational supplement to each tool's existing description.

========== INSTRUCTIONS ==========
STEP 1. Understand the Cluster Briefly identify what these tools share in purpose or functionality. (This is for your own reasoning; do not include this summary in the output.)
STEP 2. Append a Relational Supplement: **append** a short section labeled 'Relational Notes' (or similar) that summarizes:

- **Status:** Functional / Broken for some inputs (cache usable)
- **Fallback Tools:** When and why to use other tools instead (e.g., "Use X when this tool fails or has no cached result.")
- **Overlap/Complementarity:** Any differences or preferences between tools in the cluster.

Output Format:
Return only valid JSON:
{ "tool_a": "### Relational Notes: ...",
"tool_b": "### Relational Notes: ..." }

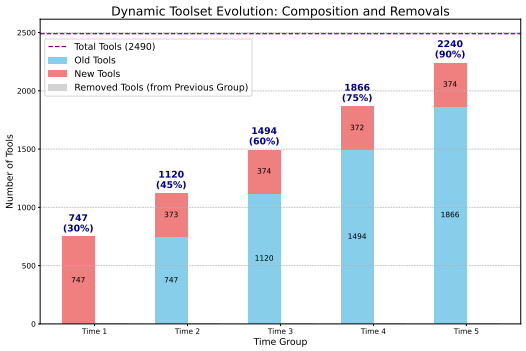

Figure 6: Statistical Information of one simulated evolving toolset with Expansion-only evolution.

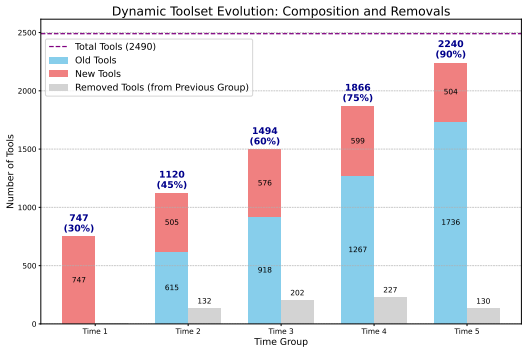

Figure 8: Statistical Information of one simulated evolving toolset with Mixture evolution.

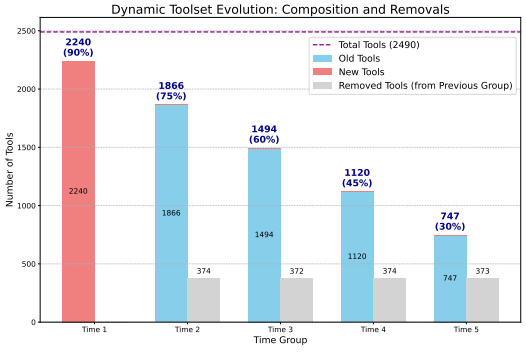

Figure 7: Statistical Information of one simulated evolving toolset with Reduction-only evolution.

with relational annotations) are provided verbatim in Appendix B.7.

B.5 Baselines

We compare ContDa against the following methods:

- **No Optimization**: the base model uses the original tool documentation without any modification.
- **EASYTOOL** (Yuan et al., 2025): standardizes and condenses heterogeneous API documentation into a concise, unified schema to improve usability for LLM agents.
- **DRAFT** (Qu et al., 2025a): adopts a trial-and-error procedure that iteratively gathers experience, learns from successful and failed

trajectories, and rewrites tool documentation accordingly.

To analyze the contribution of different components of ContDa, we also include:

- **ContDa w/o Adjust**: removes the relation-aware adjustment stage and only applies relation-guided exploration independently per tool.
- **ContDa w/o Discovery**: removes relation-guided exploration of similar tools, focusing instead on stabilizing documentation for existing tools (under reduction-only scenarios, this variant coincides with full ContDa).

B.6 Toolset Evolution Statistics

Figure 6, Figure 7, and Figure 8 report example statistics of the simulated evolving toolsets, respectively, including the number of retained, added, and removed tools at each timestep for mixture, expansion-only, and reduction-only evolution, respectively. These plots illustrate the degree of tool churn and the coverage of the global tool pool under each scenario.

B.7 Prompt Details

In this subsection, we provide the full prompt templates used in our experiments, including the prompts used by ContDa for relation-guided exploration and relation-aware adjustment. As for (i) task execution prompts for both base models, (ii) the evaluation prompt for the LLM-as-a-Judge, and (iii) the prompt used in baselines, we follow the default prompt from their original repository.

C Additional Analysis Details

Appendix C provides additional analysis of the experimental results beyond the main paper. We examine scenario-wise performance under different tool evolution patterns and present qualitative case studies that illustrate how relation-guided exploration and relation-aware adjustment improve robustness and reliability in practice.

C.1 Scenario-wise Performance Analysis

Table 2 and Table 3 reveal clear scenario-dependent behaviors that highlight different aspects of the stability–adaptation trade-off.

Expansion-only evolution primarily stresses capability discovery. As new tools are continually introduced without removals, improvements in Average Performance closely track increases in *Discover*. Across both benchmarks, ContDa achieves the largest gains in Avg. Perf., driven by substantially higher Discovery than all baselines, while incurring only a moderate increase in *Forget*. In contrast, EASYTOOL and DRAFT show limited performance gains despite slight improvements in Discovery, suggesting that local or isolated documentation refinement is insufficient to systematically surface new tool capabilities as the toolset grows.

Reduction-only evolution isolates the challenge of retention under deletions. Here, no new capabilities are available to be discovered, and performance depends on how well agents adapt when previously relied-upon tools disappear. While all methods experience higher Forget than in expansion-only settings, ContDa consistently maintains higher Average Performance than baselines on both StableToolBench and RestBench. This indicates that relation-aware adjustment helps re-anchor tool usage to remaining alternatives, whereas baselines tend to overfit to outdated documentation and suffer sharper degradation when tools are removed.

Mixture evolution combines both pressures and is the most challenging setting. In this case, ContDa shows the strongest overall advantage, achieving the highest Avg. Perf. and Discovery across models and benchmarks, despite a moderate increase in Forget. Compared to expansion-only evolution, the relative gains in Discovery are smaller, reflecting the competing effect of tool removals; compared to reduction-only evolution, the gains are substantially larger, highlighting the benefit of actively discovering and integrating new tools. Baselines, by contrast, either fail to capitalize on newly added tools (low Discovery) or become brittle under simultaneous deletions, leading to inferior overall performance.

Taken together, these results show that different evolution patterns emphasize different failure modes—discovery in expansion-only, retention in reduction-only, and their interaction in mixture evolution. ContDa consistently performs well across all three by explicitly balancing discovery and stability, rather than optimizing for a single scenario in isolation.

C.2 Case Study: Quality Analysis

To better understand the effect of our proposed ContDa, we perform two case studies for quality

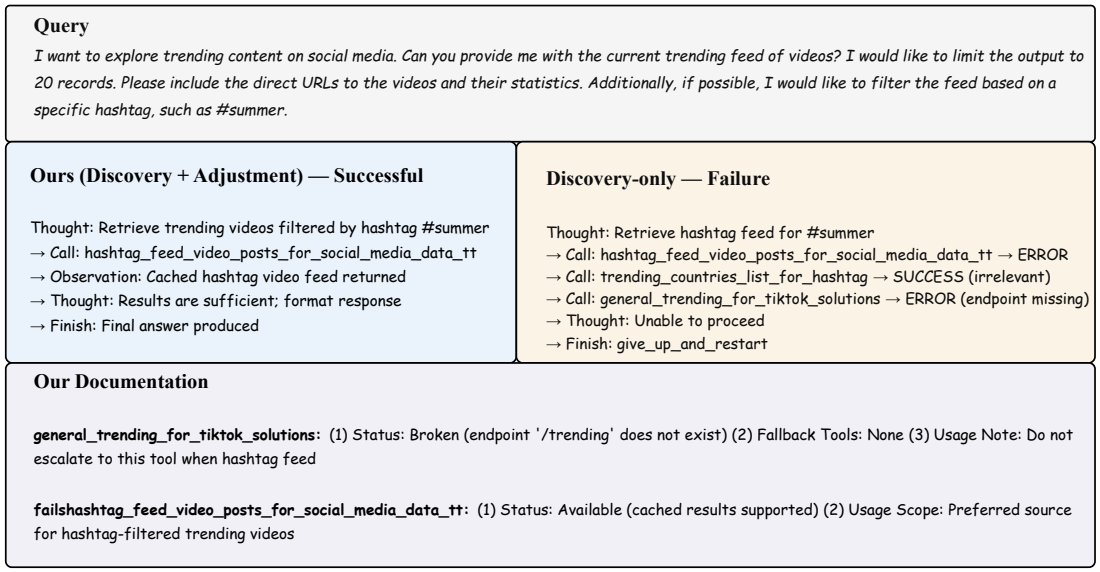

Figure 9: Case study on trending video retrieval with hashtag filtering under unreliable tool endpoints.

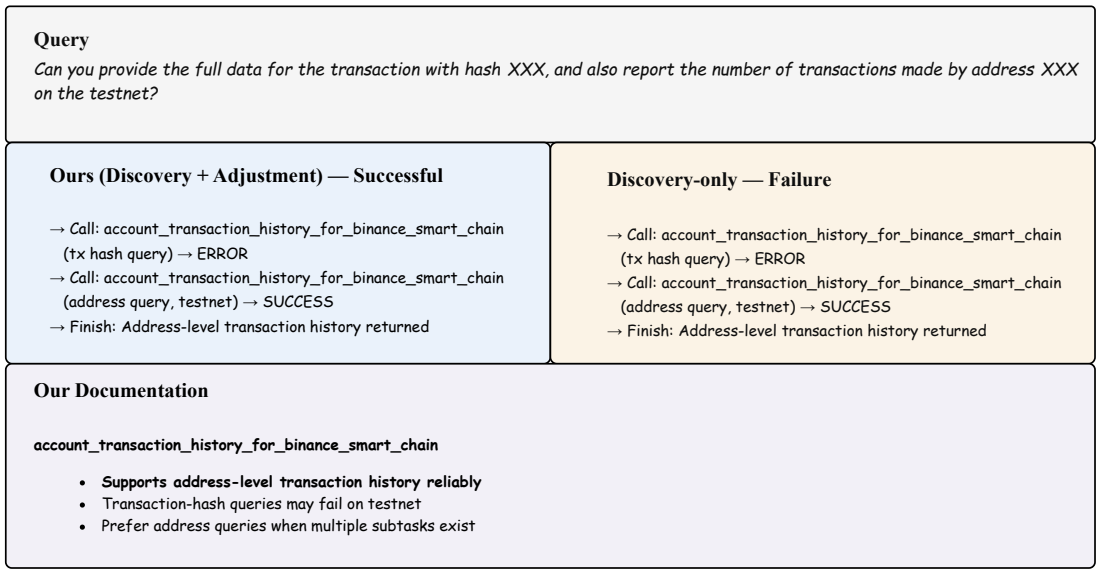

Figure 10: Case study on blockchain transaction querying with partial tool failures.

analysis.

C.2.1 Overlapping Tools and Relation-Aware Adjustment

Figure 9 illustrates how relation-aware documentation adaptation stabilizes tool usage in a video retrieval task with hashtag-based content filtering under unreliable tool environments. Although both methods initially select the same hashtag feed API, the **ContDa w/o Adjust** baseline treats the initial failure as a transient event and escalates to loosely related and non-operational tools, ultimately abandoning the task. In contrast, our method encodes reliability-related relations—such as broken status and the absence of valid fallbacks—directly in the tool documentation, which discourages failure escalation and guides the agent to commit to the usable hashtag feed. Importantly, the improvement does not arise from discovering new tools or adding task-specific knowledge, but from encoding negative and relational information that regularizes tool selection after failures. This case concretely supports our

claim that documentation adaptation is essential for robust tool use beyond discovery in dynamic and unreliable tool ecosystems.

C.2.2 Discovering Hidden Capabilities via Relation-Guided Exploration

Figure 10 illustrates how documentation adaptation can affect tool retrieval and execution order, enabling graceful degradation under partial failures. In this task, the original system retrieves the same blockchain history tool but does not prioritize between transaction-level and address-level queries. As a result, a failure in the transaction-hash lookup leads to premature termination, despite the address-level subtask being feasible. In contrast, our method encodes usage-level guidance in the adapted documentation, which biases retrieval toward the more reliable address-level query. Consequently, the agent successfully completes part of the task even when the transaction lookup fails. This example highlights that documentation adaptation not only stabilizes tool usage, but can also reshape retrieval priorities to maximize task completion under unreliable tool behavior.