CAFTBENCH: A BENCHMARK FOR CONTINUOUS FINE-TUNING OF TOOL-CALLING AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-based agents increasingly rely on external tools and APIs, yet most deployed systems assume a static tool ecosystem. In practice, APIs evolve: new tools appear, schemas change, and existing parameters are modified. Such changes require agents not only to acquire new tool-use behavior, but also to retain earlier tools, adapt to schema mutations, and compose skills learned at different times. We introduce **CAFTBench**, a static and reproducible benchmark for evaluating continuous fine-tuning of tool-calling agents. CAFTBench consists of **CAFT-30k**, a tool-ordered training corpus of 29,573 tool-calling instances spanning more than 260 tools, and **CAFT-Eval**, a probe suite covering single-tool routing, cross-stage composition, API upgrades, no-tool abstention, and structured-output robustness. We further define five metrics—Historical Routing Failure (HRF), Cross-Stage Composition Rate (CSCR), Negative Transfer and Historical Interference (NTHI), Tool Over-reliance Index (TOI), and Syntax Collapse Rate (SCR)—to disentangle retention, transfer, interference, over-reliance, and output-format degradation. Experiments with Qwen3-8B and LLaMA-3.1-8B-Instruct show that existing continual learning methods exhibit strong backbone-dependent trade-offs: EWC-style regularization improves stability and upgraded-API adaptation on Qwen, while semantic replay improves composition, but the same methods can substantially degrade LLaMA and induce tool over-reliance. These results highlight the need for continual learning methods designed specifically for evolving tool-calling agents.

1 INTRODUCTION

Large Language Models (LLMs) have enabled a new class of tool-using agents that can plan, call external APIs, and integrate returned information into downstream reasoning (Yang et al., 2025; Team et al., 2026; Yang et al., 2026; Jiang et al., 2025; Qin et al., 2023). By delegating computation and data access to external tools, such agents can perform tasks beyond the parametric knowledge of the underlying model. However, most tool-calling systems are trained and evaluated under a static assumption: once the model is deployed, its tool-use knowledge is treated as fixed (Zhou et al., 2026; Fang et al., 2025). This assumption is increasingly unrealistic. Real-world tool ecosystems evolve continuously as new APIs are introduced, old schemas change, mandatory parameters are added, and domain-specific tools emerge (Wang et al., 2026a; Chen et al., 2025a).

A common response to tool evolution is to rely on prompt engineering or in-context learning (ICL) (Dong et al., 2024; de Wynter, 2026; Zhang et al., 2026a). Although these approaches are flexible, they are bounded by context length, increase inference cost, and do not update the model's internal tool-use behavior across time (Dorovatas et al., 2026; Wang et al., 2026b). Continuous fine-tuning offers a complementary path: the model can incorporate newly observed tools and schema changes directly into its weights. Yet continuous fine-tuning also introduces a central continual learning challenge—the model must acquire new tool behavior without forgetting older tools or collapsing its structured-output format.

Tool-calling continual learning differs from standard continual learning in vision or classification tasks (Qu et al., 2025). A lifelong tool-using agent must retain historical routing behavior, adapt when an existing API mutates, abstain from tool use when no tool is needed, and compose tools learned in different stages to solve unseen multi-step tasks (Jiang et al., 2026; Zhang et al., 2026b). API mutations are especially challenging: when a mandatory parameter is added or a function name

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

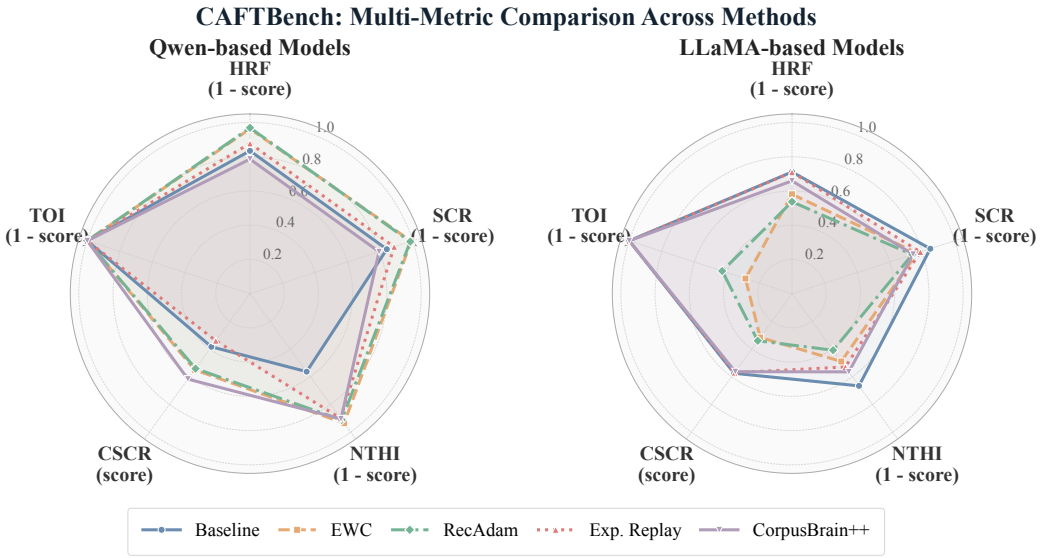


Figure 1: Radar chart comparing five continual learning methods on CAFTBench for Qwen3-8B (left) and LLaMA-3.1-8B-Instruct (right). For error metrics where lower is better (HRF, SCR, NTHI, TOI), we plot $1 - \text{score}$; for CSCR, where higher is better, we plot the raw score. Larger enclosed areas therefore indicate stronger overall performance.

changes, the model must overwrite previously learned behavior rather than merely memorize a new class label (Cheng et al., 2026).

Despite this need, existing benchmarks do not isolate the weight-update dynamics of continuously fine-tuned tool-calling agents. Static tool-calling benchmarks such as BFCL (Patil et al., 2025) provide rigorous format and routing evaluation, but evaluate models at a single temporal snapshot. Lifelong agent benchmarks such as LifelongAgentBench (Zheng et al., 2025) study long-horizon agent behavior in interactive environments, but their results conflate model forgetting with exploration, environment dynamics, and experience collection. The field therefore lacks a static, reproducible benchmark that directly measures how tool-use knowledge is retained, overwritten, and recombined after sequential weight updates.

We introduce **CAFTBench** (Continuous Agent Fine-Tuning Benchmark), a benchmark for evaluating continuous fine-tuning of tool-calling agents. CAFTBench contains **CAFT-30k**, a tool-ordered training corpus with 29,573 tool-calling instances over more than 260 tools, and **CAFT-Eval**, a unified evaluation suite spanning original single-tool probes, cross-stage compositional tasks, upgraded API probes, and no-tool prompts. To disentangle different failure modes, we define five metrics: Historical Routing Failure (HRF), which measures routing failures on historical tools; Cross-Stage Composition Rate (CSCR), which evaluates zero-shot composition of tools learned in different stages; Negative Transfer and Historical Interference (NTHI), which measures adaptation to upgraded APIs; Tool Over-reliance Index (TOI), which captures unnecessary tool calls on plain-text prompts; and Syntax Collapse Rate (SCR), which measures degradation in structured output. Detailed definitions and test logic are provided in Section 4.

We evaluate Qwen3-8B (Yang et al., 2025) and LLaMA-3.1-8B-Instruct (Grattafiori et al., 2024) under representative continual fine-tuning strategies, including Elastic Weight Consolidation (EWC), RecAdam, Experience Replay, CorpusBrain++, and sequential fine-tuning without explicit continual learning. Our results reveal strong method–backbone interactions. On Qwen, EWC-style methods substantially reduce upgraded-API interference and preserve syntax, while CorpusBrain++ yields the strongest cross-stage composition. On LLaMA, however, the same regularization methods degrade routing, composition, and abstention behavior, producing severe tool over-reliance. These findings suggest that continual tool-calling remains far from solved and that benchmark design must separately measure retention, adaptation, composition, abstention, and syntax stability.

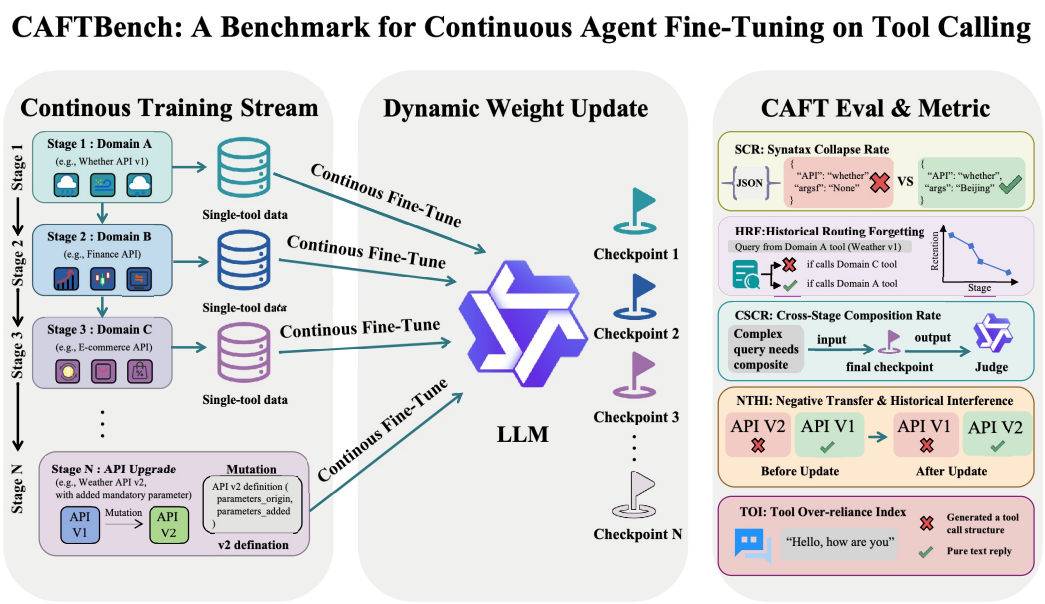


Figure 2: Overview of CAFTBench. Models first learn from a tool-ordered data stream and then adapt to upgraded API schemas. The checkpoints at the end of these two phases are evaluated using static probes. The evaluation suite measures five failure modes: syntax collapse (SCR), historical routing failure (HRF), cross-stage composition (CSCR), upgraded-API interference (NTHI), and tool over-reliance (TOI).

Our work makes four contributions. First, we introduce CAFTBench, a static and reproducible benchmark for continuous fine-tuning of tool-calling agents. Second, we construct CAFT-30k and CAFT-Eval, covering staged tool learning, cross-stage composition, upgraded APIs, and no-tool abstention. Third, we define five disentangled metrics—HRF, CSCR, NTHI, TOI, and SCR—that isolate distinct continual tool-calling failure modes. Finally, we provide empirical evidence that existing continual learning methods exhibit strong backbone-dependent trade-offs, highlighting open challenges in lifelong tool-use adaptation[1].

2 Related Work

CAFTBench lies at the intersection of continual learning for LLMs, tool/function-calling evaluation, and lifelong agent benchmarks.

Continual learning for LLMs. Continual learning has long studied how models acquire new tasks without catastrophic forgetting, but LLM-scale continual learning introduces additional challenges in instruction following, distribution shift, and long-term knowledge retention. Recent benchmarks extend continual learning to LLM settings. SWE-Bench-CL (Joshi et al., 2025), for example, organizes software-engineering tasks into chronological streams and evaluates forgetting and transfer in code generation. TiC-LM (Li et al., 2025) studies time-continual language modeling over web-scale corpora. These efforts are complementary to ours: they evaluate code repair or broad temporal pretraining, whereas CAFTBench focuses on fine-grained tool-calling behavior under sequential fine-tuning.

Tool and function-calling benchmarks. Tool-use evaluation has progressed from static function-call validation to more realistic multi-tool and multi-turn settings. BFCL (Patil et al., 2025) provides rigorous static evaluation of function calling, including AST-based matching and abstention-oriented

[1]The code and dataset is available on `https://anonymous.4open.science/r/CAFTBench-F7D8`

cases. API-Bank (Li et al., 2023), ToolAlpaca (Tang et al., 2023), ToolBench (Qin et al., 2023), ToolEyes (Ye et al., 2024), and MASSIVE-Agents (Kulkarni et al., 2025) broaden evaluation across API planning, multilingual settings, and real-world tool collections. Benchmarks for evolving tools such as ToolEVO and ToolQA-D further examine API or tool-environment changes. However, these benchmarks generally evaluate a fixed model against a fixed tool set, or focus on tool evolution without isolating sequential weight updates. They therefore provide limited insight into how tool-use knowledge changes after continuous fine-tuning.

Lifelong agent evaluation. Lifelong agent benchmarks evaluate whether agents can accumulate skills across long horizons and changing environments. LifelongAgentBench (Zheng et al., 2025) studies skill learning across interactive database, operating-system, and knowledge-graph environments, while related work examines self-evolving agents, skill discovery, and agent memory (Zheng et al., 2026; Atreja, 2025). General interactive agent benchmarks, including AgentBench, WebArena, and OSWorld, similarly evaluate agents in dynamic environments. These settings are important, but interactive environments introduce confounding factors such as exploration strategy, tool execution noise, and experience collection. CAFTBench instead adopts a static probe-based protocol to isolate the model's internal weight-update dynamics.

Positioning of CAFTBench. Table 1 summarizes how CAFTBench differs from representative benchmarks along tool-use, continual-learning, control, and diagnostic-probe dimensions. CAFT-Bench complements prior work by combining three properties that are rarely evaluated together: general tool calling, sequential fine-tuning, and static reproducible measurement. Its evaluation suite separates five failure modes—historical routing failure, cross-stage composition failure, schema-mutation interference, tool over-reliance, and syntax collapse—allowing continual tool-calling behavior to be analyzed without conflating these effects.

Table 1: Positioning of CAFTBench relative to representative benchmarks. ✓, ○, and ✗denote explicit, partial/indirect, and non-primary coverage. Columns use a continual-tool-use interpretation: Seq. FT denotes sequential fine-tuning, API Evo. schema/tool evolution, Weight controlled weight updates, Comp. cross-stage composition, Schema upgraded-schema adaptation, and Drift post-training behavioral drift.

Benchmark	Tool Use		Continual		Control		CFT Probes		
	API	Struct.	Seq. FT	API Evo.	Static	Wt. Upd.	X-Stage	Schema	Drift
CAFTBench (Ours)	✓	✓	✓	✓	✓	✓	✓	✓	✓
BFCL	✓	✓	✗	✗	✓	✗	✗	✗	○
API-Bank	✓	✓	✗	✗	○	✗	✗	✗	○
ToolBench / ToolLLM	✓	✓	✗	✗	○	✗	○	✗	○
ToolAlpaca	✓	✓	✗	✗	○	✗	✗	✗	✗
MASSIVE-Agents	✓	✓	✗	✗	✓	✗	✗	✗	○
LifelongAgentBench	○	○	○	○	✗	✗	○	✗	✗
SWE-Bench-CL	✗	✗	✓	○	○	○	○	○	✗
ToolEVO / ToolQA-D	✓	○	○	✓	○	✗	✗	○	✗
AgentBench / WebArena / OSWorld	○	○	✗	○	✗	✗	○	✗	✗

3 CAFTBENCH: DATASET CONSTRUCTION

The foundation of CAFTBench is a staged training stream paired with carefully controlled evaluation probes. We developed a data collection and generation pipeline that produces the CAFT-30k continuous training corpus and the CAFT-Eval testing suite.

We first collected existing open-source tool-calling datasets and standardized them into a unified structured format. To improve data coverage and diversity, we augmented the corpus with an LLM-based generation pipeline, with prompts provided in the Appendix. This process ensures that each tool has at least 80 distinct training instances. In total, CAFT-30k contains over 260 unique tools and 29,573 training instances. We also construct a single-tool evaluation set containing more than 1,300 instances. The tools span diverse domains, including finance, geography, mathematics, search, and e-commerce. Figure 3 shows the resulting domain distribution.

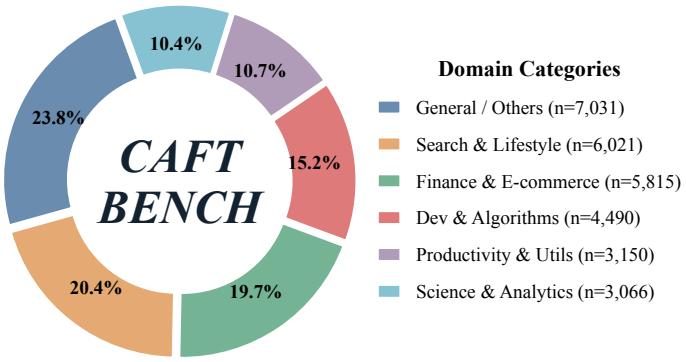


Figure 3: Domain distribution of tools in CAFT-30k. The dataset covers diverse application areas, including search and lifestyle, finance and e-commerce, and development and algorithms, supporting broad evaluation of tool-calling generalization.

Beyond single-tool evaluation, CAFTBench measures whether agents can compose capabilities learned at different stages without additional joint training. We construct a multi-tool compositional probe set using a bottom-up synthesis pipeline inspired by compositional instruction tuning (Chen et al., 2025b). Starting from atomic tool specifications and stage-wise tool partitions, we enumerate feasible cross-stage dependency chains and require downstream calls to consume semantically necessary outputs from upstream calls. We then condition LLM generation on these chains to produce natural user-facing tasks, followed by human verification to remove shortcut-solvable, weakly coupled, or logically inconsistent samples. Each retained instance therefore encodes a non-trivial execution graph that cannot be solved by single-tool recall alone.

To evaluate tool abstention, we additionally construct no-tool query sets for TOI. The primary no-tool set contains simple everyday utterances sampled from daily-dialogue-style sources (Li et al., 2017), including movie-related and general conversational exchanges, where no external API call is required. To test whether over-reliance persists beyond casual dialogue, we also build a harder no-tool QA set by collecting simple GPT-style question–answer pairs from dataset alpaca-cleaned (Taori et al., 2023). These examples cover basic knowledge, reasoning, and math questions that should be answered directly by the model. We manually filter the QA set to remove prompts with ambiguous tool-use requirements, ensuring that each retained instance has a clear no-tool label.

To evaluate adaptation to evolving APIs, we select 50 high-frequency tools and simulate realistic API upgrades. We mutate tool schemas by adding mandatory parameters and changing function names, such as appending a "v1" or "v2" suffix. Models first fine-tuned on the original tools are then fine-tuned on the upgraded APIs. This subset isolates whether the model can overwrite obsolete tool-use behavior and adapt to schema mutations.

4 CAFTBench: Evaluation Pipeline and Metrics

To ensure reproducibility and eliminate confounding variables associated with dynamic environment interactions, CAFTBench adopts a static evaluation pipeline that does not execute external tools. The pipeline evaluates saved model checkpoints by generating textual tool-call outputs and computing metric-specific scores from the generated trajectories. Overall, the evaluation process consists of three major stages: checkpoint-specific inference, structured output parsing when required, and metric computation.

Method $\mathcal{M}$ first trains on original data D_{orig} in tool order: completed tool groups are not revisited by the incoming stream, while replay methods retain buffered examples. EWC-based methods use five consolidation intervals within this stream (Appendix B). The resulting θ_1 then undergoes API-upgrade

training to produce θ_2. These checkpoints mark phase endpoints, not individual tool transitions. Each is evaluated with temperature zero, and outputs are saved for rescoring.

For metrics that require structured tool-call extraction, CAFTBench uses a lightweight AST-assisted parser to process outputs following the canonical format:

```
Call: tool_name(arg_1=value_1, arg_2=value_2, ...)
```

The parser first verifies whether the generated text matches the expected call pattern, and then deserializes the keyword arguments into a parameter dictionary. This design separates syntax-level failures from routing and argument-level failures. However, not all metrics require AST-level parsing. In particular, the LLM-as-Judge version of CSCR evaluates the raw generated trajectory directly, since multi-tool composition can involve flexible intermediate reasoning and variable execution order.

The evaluation workflow is summarized in Algorithm 1. In the current evaluation protocol, θ_1 is used to evaluate original single-tool behavior, compositional generalization, and tool over-reliance, while θ_2 is used to evaluate adaptation to upgraded APIs.

Algorithm 1 CAFTBench Continual Evaluation Pipeline

Require: Base model θ_0, tool-ordered data D_{orig}, upgraded API data D_{upg}, continual learning method $\mathcal{M}$, evaluation suite $\mathcal{E}$

Phase 1: Sequential Original-Tool Learning
$\theta_1 \leftarrow \text{ToolOrdered_FT}(\theta_0, D_{\text{orig}}, \mathcal{M})$
Phase 2: Continual Fine-Tuning for API Upgrades
$\theta_2 \leftarrow \text{Continual_SFT}(\theta_1, D_{\text{upg}}, \text{method} = \mathcal{M})$
Phase 3: Checkpoint-Specific Static Inference
$\mathcal{E}_{\text{orig}} \leftarrow \{e \in \mathcal{E} \mid e.\text{tag} \in \{\text{route}, \text{comp}, \text{toi}\}\}$
$\mathcal{E}_{\text{upg}} \leftarrow \{e \in \mathcal{E} \mid e.\text{tag} = \text{upg}\}$
for each $e \in \mathcal{E}_{\text{orig}}$ **do**
    $\mathcal{R}[\theta_1][e.\text{id}] \leftarrow \text{Generate}(\theta_1, e.\text{prompt}, \text{temperature} = 0)$
end for
for each $e \in \mathcal{E}_{\text{upg}}$ **do**
    $\mathcal{R}[\theta_2][e.\text{id}] \leftarrow \text{Generate}(\theta_2, e.\text{prompt}, \text{temperature} = 0)$
end for
Phase 4: Metric Computation
Compute HRF and SCR from $\mathcal{R}[\theta_1]$ on original single-tool probes
Compute CSCR from $\mathcal{R}[\theta_1]$ on compositional probes using exact matching or LLM-as-Judge
Compute TOI from $\mathcal{R}[\theta_1]$ on plain-text no-tool probes
Compute NTHI from $\mathcal{R}[\theta_2]$ on upgraded API probes
return Metric report: HRF, SCR, CSCR, TOI from θ_1; NTHI from θ_2

Following inference, CAFTBench computes five metrics that characterize different aspects of continual tool-calling behavior.

Syntax Collapse Rate (SCR) SCR measures whether the model preserves the basic ability to emit a syntactically valid tool-call string. A prediction is valid if it can be parsed into the canonical pattern `Call: tool_name(args)`. SCR is computed on the original single-tool evaluation set:

$$SCR = \frac{1}{|D_{\text{orig}}^{\text{test}}|} \sum_{x \in D_{\text{orig}}^{\text{test}}} \mathbb{I}\left[\text{Parser}(f_{\theta_1}(x)) = \text{Error}\right],$$

where $f_{\theta_1}(x)$ denotes the textual output generated by checkpoint θ_1 for input x. A higher SCR indicates a greater tendency to violate the required output format.

Historical Routing Failure (HRF) HRF measures whether the model selects the correct historical tool on original single-tool instructions. Given an original test instance x, the model is expected to select the correct tool name $t_{\text{true}}(x)$. If the output cannot be parsed, it is counted as a routing failure because no valid tool decision can be recovered. HRF is defined as:

$$HRF = \frac{1}{|D_{\text{orig}}^{\text{test}}|} \sum_{x \in D_{\text{orig}}^{\text{test}}} \mathbb{I}\left[\text{Parser}(f_{\theta_1}(x)) = \text{Error} \ \vee \ \text{Tool}(f_{\theta_1}(x)) \neq t_{\text{true}}(x)\right].$$

HRF measures endpoint routing error after the original tool stream, rather than a before–after forgetting difference.

Cross-Stage Composition Rate (CSCR) CSCR evaluates whether the model can compose tools across different stages or tool groups in response to multi-step instructions. Since compositional tool use can admit flexible reasoning paths and variable execution orders, CAFTBench supports an LLM-as-Judge evaluation mode in addition to exact structural matching. For experiments in this paper, we report the LLM-as-Judge version. In this setting, the raw generated trajectory is passed to an evaluator model, which assigns a score $S \in \{0, 1, \ldots, 5\}$ according to a predefined rubric considering logical consistency, tool selection, and argument correctness. The normalized score is:

$$CSCR = \frac{1}{5 \cdot |D_{\text{comp}}|} \sum_{x \in D_{\text{comp}}} \text{LLM_Judge}\left(x, f_{\theta_1}(x), \mathcal{T}_{\text{req}}(x)\right),$$

where D_{comp} is the compositional evaluation set and $\mathcal{T}_{\text{req}}(x)$ denotes the required tools for instance x. The final value lies in $[0, 1]$, with higher values indicating stronger compositional tool-use ability.

Negative Transfer and Historical Interference (NTHI) NTHI measures failures when adapting to upgraded API specifications after continual fine-tuning. It is computed on upgraded API instances using checkpoint θ_2. A prediction is erroneous if it is unparseable, calls the obsolete or wrong upgraded tool, or omits required arguments under strict schema checking. Formally:

$$NTHI = \frac{1}{|D_{\text{upg}}^{\text{test}}|} \sum_{x \in D_{\text{upg}}^{\text{test}}} \mathbb{I}\Big[\text{Parser}(f_{\theta_2}(x)) = \text{Error} \ \vee \ \text{Tool}(f_{\theta_2}(x)) \neq t_{\text{upg}}(x) \\ \vee \ \delta_{\text{strict}} \cdot \text{MissingRequired}(f_{\theta_2}(x), S_{\text{upg}}(x))\Big].$$

where $t_{\text{upg}}(x)$ is the expected upgraded tool, $S_{\text{upg}}(x)$ is the required-argument schema provided in the current prompt, and δ_{strict} indicates whether strict required-argument checking is enabled. Lower NTHI indicates better adaptation to the upgraded API distribution.

Tool Over-reliance Index (TOI) TOI measures whether continual exposure to tool-calling data causes the model to over-invoke tools on plain-text queries that do not require external tool use. Given a no-tool query $x \in D_{\text{plain}}$, the output is considered a false positive if it contains a parseable tool call. TOI is defined as:

$$TOI = \frac{1}{|D_{\text{plain}}|} \sum_{x \in D_{\text{plain}}} \mathbb{I}\left[\text{Is_Tool_Call}(f_{\theta_1}(x)) = \text{True}\right].$$

A higher TOI indicates stronger tool over-reliance and weaker intent discrimination between ordinary conversational queries and tool-use requests.

5 Experiments

We compare continual fine-tuning methods using all five CAFTBench metrics; implementation details are provided in Appendix B.

5.1 Experimental Setup

Backbone models. We evaluate CAFTBench on two representative model families: Qwen3-8B (Yang et al., 2025) and LLaMA-3.1-8B-Instruct (Grattafiori et al., 2024). These two backbones differ in their pretraining distributions, instruction-following behavior, and sensitivity to continual fine-tuning, allowing us to examine whether the observed continual learning dynamics are method-specific or backbone-dependent.

Continual learning methods We compare five training strategies: (1) **Baseline**, which performs supervised fine-tuning on the tool-ordered stream without regularization or replay; (2) **EWC** (Kirkpatrick et al., 2017), which regularizes parameter updates using Fisher-weighted importance estimates;

(3) **RecAdam** (Chen et al., 2020), which combines EWC-style regularization with an objective-shifting schedule; (4) **Experience Replay** (Rolnick et al., 2019), which interleaves new samples with historical buffered samples; and (5) **CorpusBrain++** (Guo et al., 2024), a semantic-aware replay strategy that retrieves relevant historical samples based on semantic similarity.

Evaluation protocol All models are evaluated with the static CAFTBench pipeline described in Section 4. We use deterministic decoding with temperature set to 0.0, top-p set to 1.0, and a maximum generation length of 512 tokens. Inference is performed using vLLM batch generation with tensor parallel size 2 and maximum model length 4096. For CSCR, we use the LLM-as-Judge setting, where Qwen3-30B-A3B-Instruct assigns a score from 0 to 5 according to the rubric described in the Appendix; the judge is served with tensor parallel size 8 and maximum model length 8192. TOI is primarily reported on the pure no-tool test set, and we additionally evaluate a harder no-tool QA set to test whether tool over-reliance persists under more structured plain-text queries.

5.2 Main Results

Table 2 reports the main CAFTBench results across all model-method combinations. For HRF, SCR, NTHI, and TOI, lower values are better, while for CSCR, higher values are better.

Table 2: Main CAFTBench results across continual learning methods and backbone models. HRF, SCR, NTHI, and TOI are error rates, where lower is better; CSCR is a normalized LLM-as-Judge score, where higher is better. Best results within each backbone are shown in bold.

Model	Method	HRF ↓	SCR ↓	NTHI ↓	CSCR ↑	TOI ↓
Qwen3-8B	EWC	0.0368	**0.0092**	**0.0644**	0.5498	**0.0000**
	RecAdam	**0.0307**	0.0153	0.0881	0.5410	**0.0000**
	Exp. Replay	0.1218	0.1142	0.0996	0.3398	**0.0000**
	CorpusBrain++	0.2138	0.2069	0.0958	**0.6172**	**0.0000**
	Baseline(SFT)	0.1655	0.1594	0.4368	0.3838	**0.0000**
LLaMA-3.1-8B-Instruct	Baseline(SFT)	**0.2889**	**0.1502**	**0.3341**	**0.5750**	**0.0000**
	Exp. Replay	0.2904	0.2115	0.4705	0.5706	**0.0000**
	CorpusBrain++	0.3410	0.2559	0.4352	0.5634	**0.0000**
	RecAdam	0.4628	0.2613	0.5916	0.3402	0.5710
	EWC	0.4192	0.2322	0.5103	0.3162	0.7140

Analysis Table 2 shows strong method–backbone interactions. On Qwen3-8B, continual learning methods substantially improve upgraded-API adaptation: NTHI drops from 0.4368 under sequential fine-tuning to below 0.10 for all continual variants. EWC achieves the best syntax and schema stability, with the lowest SCR (0.0092) and NTHI (0.0644), while RecAdam obtains the lowest HRF (0.0307). CorpusBrain++ achieves the best CSCR (0.6172), but with higher HRF and SCR, indicating that better cross-stage composition does not necessarily imply stricter single-tool routing or formatting.

To explain these differences, we inspect saved model outputs and decompose errors into format failures, wrong tool selections, missing upgraded arguments, and false tool calls on no-tool prompts. This diagnostic decomposition, shown in Figure 4, is used only to interpret the five CAFTBench metrics. For Qwen3-8B, most parseable outputs select the correct tool, so performance differences mainly come from whether the model preserves the canonical call format. This explains why EWC improves Qwen: it stabilizes structured outputs, reducing both SCR and NTHI.

LLaMA-3.1-8B-Instruct exhibits a different failure mode. EWC and RecAdam worsen HRF, SCR, NTHI, CSCR, and TOI relative to the baseline. Output inspection shows frequent malformed or repetitive calls, such as repeated `Call:` prefixes, along with wrong tool selections even when parsing succeeds. Their high TOI (0.7140 and 0.5710) further shows that regularization shifts LLaMA toward over-triggering tool calls on no-tool prompts. Replay-based methods avoid this over-reliance and preserve CSCR close to the baseline, but still increase SCR and NTHI, suggesting that replay alone does not enforce stable formatting or upgraded-schema compliance. On the harder no-tool QA set,

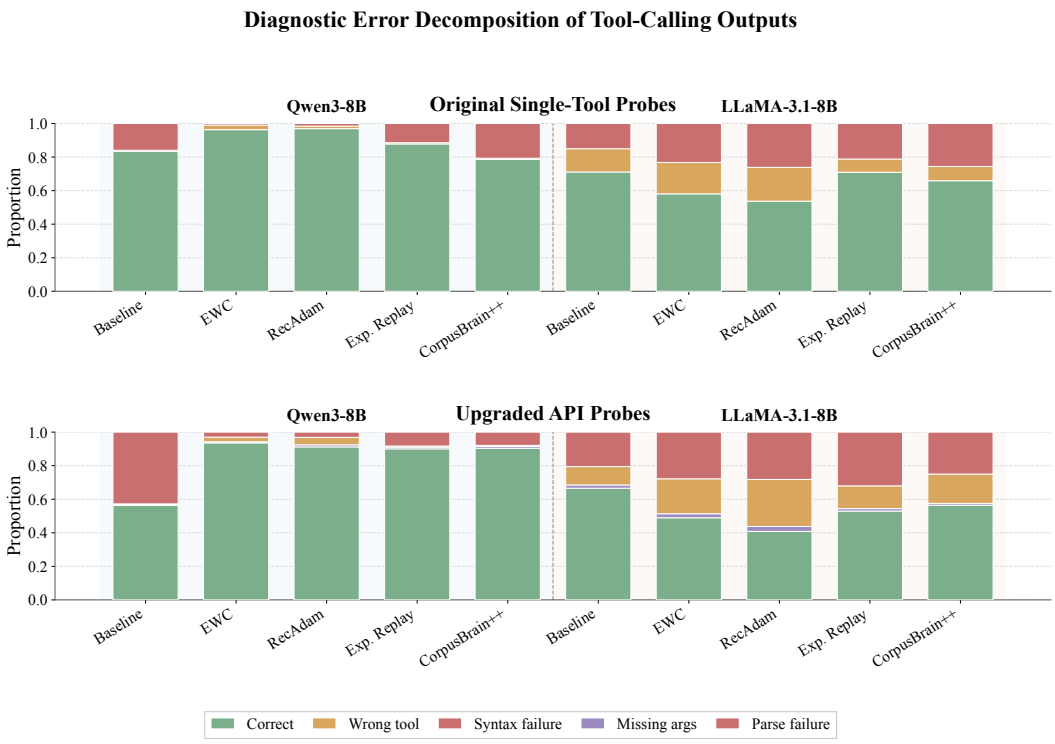


Figure 4: Diagnostic error decomposition of model outputs on original single-tool and upgraded API probes. The decomposition explains HRF, SCR, and NTHI by separating correct calls, syntax or parse failures, wrong-tool calls, and missing required arguments. These categories are used only for analysis and are not introduced as additional benchmark metrics.

LLaMA with EWC and RecAdam still shows high TOI (0.6140 and 0.3540), confirming that this over-reliance is not specific to the conversational no-tool set.

These results should be interpreted in the context of the tool-ordered stream. Earlier tool groups have already left the incoming stream when later groups are learned; replay methods explicitly reintroduce buffered examples, whereas EWC-based methods carry historical information through parameter regularization. The five EWC consolidation intervals specify when this information is updated, rather than defining the number of tools encountered. Accordingly, θ_1 summarizes performance after sequential original-tool learning, while θ_2 measures subsequent API-upgrade adaptation. This distinction separates the training chronology from the two reported evaluation endpoints, although endpoint scores alone do not reveal when forgetting occurs within the stream.

6 Conclusion

We present CAFTBench, a benchmark for continuous fine-tuning of tool-calling agents. It pairs CAFT-30k with CAFT-Eval, measuring historical routing, stability, API adaptation, composition, and abstention. Its five metrics separate failure modes in interactive benchmarks.

Experiments on Qwen3-8B and LLaMA-3.1-8B show no reliable solution for continual learning. EWC-style regularization boosts syntax and API adaptation on Qwen; semantic replay improves composition. Yet these methods degrade LLaMA, raising routing errors and tool over-reliance. Thus, continual tool-calling is sensitive to backbone; future methods must jointly optimize retention, plasticity, composition, abstention, and format stability. We hope CAFTBench offers a controlled foundation for adapting agents to evolving tool ecosystems.

AI USE STATEMENT

In accordance with the ICLR 2027 AI policy, we disclose that LLMs were used as part of the research workflow, primarily for benchmark construction and evaluation. We also used LLMs for language polishing and code debugging. All LLM outputs were subject to careful human oversight and validation. We take full responsibility for the accuracy and integrity of all content in this paper, including any sections enhanced with LLM assistance.

ETHICS STATEMENT

This work uses only publicly available datasets, which contain no personally identifiable or sensitive information. We did not identify any ethical concerns.

REPRODUCIBILITY STATEMENT

Code and data supporting reproducibility are available in the anonymous repository linked in the paper. Dataset construction and evaluation protocols are described in Sections 3 and 4, with further implementation details provided in the appendix.

REFERENCES

Dhruv Atreja. Alas: Autonomous learning agent for self-updating language models, 2025. URL https://arxiv.org/abs/2508.15805.

Guoxin Chen, Zhong Zhang, Xin Cong, Fangda Guo, Yesai Wu, Yankai Lin, Wenzheng Feng, and Yasheng Wang. Learning evolving tools for large language models, 2025a. URL https://arxiv.org/abs/2410.06617.

Mingyang Chen, Haoze Sun, Tianpeng Li, Fan Yang, Hao Liang, Keer Lu, Bin Cui, Wentao Zhang, Zenan Zhou, and Weipeng Chen. Facilitating multi-turn function calling for llms via compositional instruction tuning, 2025b. URL https://arxiv.org/abs/2410.12952.

Sanyuan Chen, Yutai Hou, Yiming Cui, Wanxiang Che, Ting Liu, and Xiangzhan Yu. Recall and learn: Fine-tuning deep pretrained language models with less forgetting, 2020. URL https://arxiv.org/abs/2004.12651.

Jiali Cheng, Rui Pan, and Hadi Amiri. Investigating tool-memory conflicts in tool-augmented llms, 2026. URL https://arxiv.org/abs/2601.09760.

Adrian de Wynter. Is in-context learning learning?, 2026. URL https://arxiv.org/abs/2509.10414.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. A survey on in-context learning, 2024. URL https://arxiv.org/abs/2301.00234.

Vaggelis Dorovatas, Malte Schwerin, Andrew D. Bagdanov, Lucas Caccia, Antonio Carta, Laurent Charlin, Barbara Hammer, Tyler L. Hayes, Timm Hess, Christopher Kanan, Dhireesha Kudithipudi, Xialei Liu, Vincenzo Lomonaco, Jorge Mendez-Mendez, Darshan Patil, Ameya Prabhu, Elisa Ricci, Tinne Tuytelaars, Gido M. van de Ven, Liyuan Wang, Joost van de Weijer, Jonghyun Choi, Martin Mundt, and Rahaf Aljundi. Modular memory is the key to continual learning agents, 2026. URL https://arxiv.org/abs/2603.01761.

Jinyuan Fang, Yanwen Peng, Xi Zhang, Yingxu Wang, Xinhao Yi, Guibin Zhang, Yi Xu, Bin Wu, Siwei Liu, Zihao Li, Zhaochun Ren, Nikos Aletras, Xi Wang, Han Zhou, and Zaiqiao Meng. A comprehensive survey of self-evolving ai agents: A new paradigm bridging foundation models and lifelong agentic systems, 2025. URL https://arxiv.org/abs/2508.07407.

Aaron Grattafiori et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Jiafeng Guo, Changjiang Zhou, Ruqing Zhang, Jiangui Chen, Maarten de Rijke, Yixing Fan, and Xueqi Cheng. Corpusbrain++: A continual generative pre-training framework for knowledge-intensive language tasks, 2024. URL https://arxiv.org/abs/2402.16767.

Dongfu Jiang, Yi Lu, Zhuofeng Li, Zhiheng Lyu, Ping Nie, Haozhe Wang, Alex Su, Hui Chen, Kai Zou, Chao Du, Tianyu Pang, and Wenhu Chen. Verltool: Towards holistic agentic reinforcement learning with tool use, 2025. URL https://arxiv.org/abs/2509.01055.

Guanyu Jiang, Zhaochen Su, Xiaoye Qu, and Yi R. Fung. Xskill: Continual learning from experience and skills in multimodal agents, 2026. URL https://arxiv.org/abs/2603.12056.

Thomas Joshi, Shayan Chowdhury, and Fatih Uysal. Swe-bench-cl: Continual learning for coding agents, 2025. URL https://arxiv.org/abs/2507.00014.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks, 2017. URL https://arxiv.org/abs/1612.00796.

Mayank Kulkarni, Vittorio Mazzia, Judith Gaspers, Chris Hench, and Jack FitzGerald. MASSIVE-agents: A benchmark for multilingual function-calling in 52 languages. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 20193–20215, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.1099. URL https://aclanthology.org/2025.findings-emnlp.1099/.

Jeffrey Li, Mohammadreza Armandpour, Iman Mirzadeh, Sachin Mehta, Vaishaal Shankar, Raviteja Vemulapalli, Samy Bengio, Oncel Tuzel, Mehrdad Farajtabar, Hadi Pouransari, and Fartash Faghri. Tic-lm: A web-scale benchmark for time-continual llm pretraining, 2025. URL https://arxiv.org/abs/2504.02107.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms, 2023. URL https://arxiv.org/abs/2304.08244.

Yanran Li, Hui Su, Xiaoyu Shen, Wenjie Li, Ziqiang Cao, and Shuzi Niu. DailyDialog: A manually labelled multi-turn dialogue dataset. In Greg Kondrak and Taro Watanabe (eds.), *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 986–995, Taipei, Taiwan, November 2017. Asian Federation of Natural Language Processing. URL https://aclanthology.org/I17-1099/.

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

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. URL https://arxiv.org/abs/2307.16789.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-rong Wen. Tool learning with large language models: a survey. *Frontiers of Computer Science*, 19(8), January 2025. ISSN 2095-2236. doi: 10.1007/s11704-024-40678-2. URL http://dx.doi.org/10.1007/s11704-024-40678-2.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P. Lillicrap, and Greg Wayne. Experience replay for continual learning, 2019. URL https://arxiv.org/abs/1811.11682.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases, 2023. URL https://arxiv.org/abs/2306.05301.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Kimi Team et al. Kimi k2.5: Visual agentic intelligence, 2026. URL https://arxiv.org/abs/2602.02276.

Ruipeng Wang, Yuxin Chen, Yukai Wang, Chang Wu, Junfeng Fang, Xiaodong Cai, Qi Gu, Hui Su, An Zhang, Xiang Wang, Xunliang Cai, and Tat-Seng Chua. Agentnoisebench: Benchmarking robustness of tool-using llm agents under noisy condition, 2026a. URL https://arxiv.org/abs/2602.11348.

Yinjie Wang, Xuyang Chen, Xiaolong Jin, Mengdi Wang, and Ling Yang. Openclaw-rl: Train any agent simply by talking, 2026b. URL https://arxiv.org/abs/2603.10165.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Jian Yang, Wei Zhang, Shawn Guo, Zhengmao Ye, Lin Jing, Shark Liu, Yizhi Li, Jiajun Wu, Cening Liu, X. Ma, Yuyang Song, Siwei Wu, Yuwen Li, L. Liao, T. Zheng, Ziling Huang, Zelong Huang, Che Liu, Yan Xing, Renyuan Li, Qingsong Cai, Hanxu Yan, Siyue Wang, Shikai Li, Jason Klein Liu, An Huang, Yongsheng Kang, Jinxing Zhang, Chuan Hao, Haowen Wang, Weicheng Gu, Ran Tao, Mingjie Tang, Peihao Wu, Jianzhou Wang, Xianglong Liu, Weifeng Lv, and Bryan Dai. Iquest-coder-v1 technical report, 2026. URL https://arxiv.org/abs/2603.16733.

Junjie Ye, Guanyu Li, Songyang Gao, Caishuang Huang, Yilong Wu, Sixian Li, Xiaoran Fan, Shihan Dou, Tao Ji, Qi Zhang, Tao Gui, and Xuanjing Huang. Tooleyes: Fine-grained evaluation for tool learning capabilities of large language models in real-world scenarios, 2024. URL https://arxiv.org/abs/2401.00741.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models, 2026a. URL https://arxiv.org/abs/2510.04618.

Ziao Zhang, Kou Shi, Shiting Huang, Avery Nie, Yu Zeng, Yiming Zhao, Zhen Fang, Qishen Su, Haibo Qiu, Wei Yang, Qingnan Ren, Shun Zou, Wenxuan Huang, Lin Chen, Zehui Chen, and Feng Zhao. Skillflow:benchmarking lifelong skill discovery and evolution for autonomous agents, 2026b. URL https://arxiv.org/abs/2604.17308.

Junhao Zheng, Xidi Cai, Qiuke Li, Duzhen Zhang, ZhongZhi Li, Yingying Zhang, Le Song, and Qianli Ma. Lifelongagentbench: Evaluating llm agents as lifelong learners, 2025. URL https://arxiv.org/abs/2505.11942.

Junhao Zheng, Chengming Shi, Xidi Cai, Qiuke Li, Duzhen Zhang, Chenxing Li, Dong Yu, and Qianli Ma. Lifelong learning of large language model based agents: A roadmap, 2026. URL https://arxiv.org/abs/2501.07278.

Huichi Zhou, Siyuan Guo, Anjie Liu, Zhongwei Yu, Ziqin Gong, Bowen Zhao, Zhixun Chen, Menglong Zhang, Yihang Chen, Jinsong Li, Runyu Yang, Qiangbin Liu, Xinlei Yu, Jianmin Zhou, Na Wang, Chunyang Sun, and Jun Wang. Memento-skills: Let agents design agents, 2026. URL https://arxiv.org/abs/2603.18743.

A PROMPTS TEMPLATE

A.1 PROMPTS FOR LLM-AS-JUDGE

CSCR Judge Prompt

```
Task: Evaluate tool-composition quality for one sample.
Use required tools as reference shots for expected decomposition and intent coverage.
Score must be an integer from 0 to 5.

Scoring rubric:
5: Fully correct multi-tool plan, all critical steps present and coherent.
4: Mostly correct, minor omissions/order issues.
3: Partial completion, core intent partly satisfied.
2: Weak completion, major missing steps.
1: Barely related or almost unusable.
0: Invalid/off-topic/no meaningful tool composition.

Return JSON only: {"score": <0-5 integer>, "reason": "short reason"}.

Sample id: <sample_id>
User question: <composite_question>
Required tools (reference shots): [tool_a, tool_b, ...]
Model output:
<predicted tool-calling text>
```

Figure 5: Prompt template used by our LLM-as-Judge for CSCR evaluation. Given the user query, required tools (as reference shots), and the model-generated tool-calling output, the judge assigns an integer score from 0 to 5 according to a predefined rubric on composition quality, and returns a structured JSON response containing both the score and a brief rationale.

A.2 PROMPTS FOR ZERO-SHOT MULTI-TOOL GENERALIZATION

Multi-tool Synthesis Prompt

```
[Sequential Composition Prompt]
You are constructing compositional tasks with Sequential Composition strategy.
Given Task A and Task B, create a subsequent task that depends on the output of Task A,
and write one compositional question that requires both tasks.
Requirements:
1. The compositional question must be realistic and self-contained.
2. Do not mention tool names or function names.
3. The second step must depend on the first step output.
4. Keep deterministic details; avoid ambiguity.
Output JSON only:
{"given_task":"...","subsequent_task":"...","composition_task":"..."}
Input payload:
{"task_a": "<task_a_query>", "task_b_reference": "<task_b_query>"}

[Parallel-then-Sequential Composition Prompt]
You are constructing compositional tasks with Parallel-then-Sequential strategy.
Create one parallel task independent of Task A output, then create a subsequent task
that needs both outputs, and produce one compositional question requiring all steps.
Requirements:
1. Parallel task can be solved at the same time as Task A.
2. Subsequent task must use outputs of Task A and parallel task together.
3. Do not mention tool names or function names.
4. Keep deterministic details; avoid ambiguity.
Output JSON only:
{"given_task":"...","parallel_task":"...","subsequent_task":"...","composition_task":"..."}
Input payload:
{"task_a": "<task_a_query>", "task_parallel_reference": "<task_b_query>",
 "task_subsequent_reference": "<task_c_query>"}
```

Figure 6: Prompt templates for bottom-up multi-tool compositional data synthesis. Given pairs of atomic single-tool tasks sampled from different training stages, an LLM is instructed to compose them into naturalistic multi-step questions via two strategies: Sequential Composition, where a subsequent task is constructed to depend on the output of a preceding task; and Parallel-then-Sequential Composition, where an independent parallel task is first introduced alongside the given task, followed by a subsequent task that requires both outputs. Both templates enforce that the synthesized questions are self-contained and free of explicit tool or function names to prevent shortcut matching.

B Implementation Details of Continual Learning Baselines

In this section, we provide the technical implementation details for the continuous fine-tuning baselines evaluated in CAFTBench, specifically focusing on Elastic Weight Consolidation (EWC) and Experience Replay. Detailed hyperparameter configurations for all training stages are summarized in Table 3.

Sequential tool stream. The original training data are grouped by target tool and presented in tool order. Let $D_{\pi(i)}$ denote the training examples for the i-th tool in the ordering π. The incoming stream is

$$\mathcal{S}_{\text{orig}} = D_{\pi(1)} \rightarrow D_{\pi(2)} \rightarrow \cdots \rightarrow D_{\pi(K)},$$

where K is the number of tools. Training advances through these tool-specific groups rather than globally mixing examples from all tools. Once training on a group is complete, the incoming stream moves to the next tool and does not return to the completed group. Replay methods additionally access historical examples through their memory buffers; this is distinct from the ordering of the incoming data. For the EWC-based methods, the ordered stream is partitioned into five consecutive consolidation intervals for Fisher estimation and parameter-anchor updates (Appendix B). These intervals organize regularization within the tool-learning stream and are distinct from the later API-upgrade phase.

B.1 Basic Elastic Weight Consolidation (EWC)

The basic EWC baseline is designed to mitigate catastrophic forgetting by regularizing the weight updates based on their importance to historical tasks (Kirkpatrick et al., 2017). In our setup, the original tool-ordered stream is partitioned into five consecutive consolidation intervals (chunks $t = 1, \ldots, 5$). Tool order is preserved within this stream, while the interval boundaries specify when Fisher estimates and parameter anchors are updated. In the EWC formulations below, "stage" refers to a consolidation interval, not to the original versus upgraded API phases used for endpoint evaluation.

Mathematical Formulation For each training stage t ($t > 1$), EWC introduces a quadratic penalty term to the standard supervised fine-tuning (SFT) objective. The total loss function is defined as:

$$\mathcal{L}_{\text{total}}(\theta_t) = \mathcal{L}_{\text{SFT}}(\theta_t, \mathcal{D}_t) + \frac{\lambda_{\text{ewc}}}{2} \sum_i \tilde{F}_{t-1,i}(\theta_{t,i} - \theta^*_{t-1,i})^2 \tag{1}$$

where $\mathcal{L}_{\text{SFT}}$ is the cross-entropy loss on the current dataset $\mathcal{D}_t$, $\theta^*_{t-1,i}$ denotes the consolidated optimal parameters from the previous stage, and $\tilde{F}_{t-1,i}$ represents the importance of the i-th parameter. The coefficient λ_{ewc} scales the importance of the EWC penalty.

Fisher Information Matrix and Sequential Fusion The importance of each parameter is estimated using the **diagonal approximation** of the empirical Fisher Information Matrix (FIM). Upon completing Stage t, we compute the local Fisher matrix F_t by performing a dedicated inference pass over the dataset $\mathcal{D}_t$ with the model set to **evaluation mode** to ensure deterministic gradient calculation. For each parameter θ_i, the corresponding diagonal element of F_t is computed by averaging the squared gradients of the log-likelihood:

$$F_{t,i} = \frac{1}{|\mathcal{D}_t|} \sum_{(x,y)\in\mathcal{D}_t} \left(\frac{\partial \mathcal{L}_{\text{SFT}}(x, y; \theta)}{\partial \theta_i} \right)^2 \tag{2}$$

where $\mathcal{L}_{\text{SFT}}$ is the cross-entropy loss. To accumulate knowledge across multiple stages without the linear growth of storage requirements, we employ a **rolling arithmetic fusion** strategy. The consolidated Fisher matrix $\tilde{F}_t$ for the subsequent stage is updated as:

$$\tilde{F}_t = 0.5\tilde{F}_{t-1} + 0.5F_t \tag{3}$$

where $\tilde{F}_1 = F_1$. This ensures that the importance weights consistently reflect a balanced contribution from all previously encountered tool-calling patterns.

Training Logic For the Basic EWC baseline, the model follows the tool-ordered stream across the five consolidation intervals. The optimization process focuses on minimizing the combined loss, where the importance of previous knowledge is governed by the regularization strength λ_{ewc}. All training is conducted in `bf16` precision to ensure numerical stability. Specific values for λ_{ewc} and other optimization parameters are provided in Table 3.

B.2 EWC WITH OBJECTIVE SHIFTING (RECADAM)

This baseline, hereafter referred to as **EWC (RecAdam)**, incorporates the **Objective Shifting** mechanism from the RecAdam optimizer (Chen et al., 2020) into the EWC framework. While the original RecAdam shifts the optimization target from a pre-training simulation (L2 regularization) to the downstream task, our implementation adapts this logic to transition from historical knowledge preservation (EWC penalty) to current tool-calling adaptation. Coupled with an **Online Fisher Decay** strategy, this approach ensures a fluid transition between stages and prevents the importance weights from becoming biased toward the initial stages in a long-term continuous learning stream.

Mathematical Formulation Unlike the static penalty in the original EWC, this variant dynamically adjusts the balance between knowledge recall and learning of new tasks within each stage. For a training stage t, let $\tau = s/S \in [0, 1]$ represent the normalized training progress (current step s over total steps S). The total loss function is defined as:

$$\mathcal{L}_{\text{total}}(\theta_t) = \lambda(\tau)\mathcal{L}_{\text{SFT}}(\theta_t, \mathcal{D}_t) + (1 - \lambda(\tau))\frac{\lambda_{\text{ewc}}}{2}\sum_i \tilde{F}_{t-1,i}(\theta_{t,i} - \theta^*_{t-1,i})^2 \tag{4}$$

The annealing coefficient $\lambda(\tau)$ follows a sigmoid schedule, which is strictly aligned with the RecAdam formulation:

$$\lambda(\tau) = \frac{1}{1 + \exp(-k \cdot (\tau - \tau_0))} \tag{5}$$

where k is the slope and τ_0 is the midpoint of the transition. This mechanism enforces a "Recall-then-Learn" strategy: during the early phase ($\tau < \tau_0$), the model focuses on recalling historical API knowledge by emphasizing the EWC penalty; in the later phase ($\tau > \tau_0$), it gradually shifts its optimization focus to fully adapt to the current stage's tool-calling distribution.

Online Fisher Fusion with Decay To maintain a single consolidated Fisher matrix across all five stages while ensuring that recent knowledge is not overshadowed by the distant past, we adopt an online Fisher decay strategy. Upon completing Stage t, the accumulated Fisher matrix $\tilde{F}_t$ is updated using an exponential moving average:

$$\tilde{F}_t = \gamma\tilde{F}_{t-1} + (1 - \gamma)F_t \tag{6}$$

where $\gamma \in [0, 1]$ is the decay factor and F_t is the empirical Fisher matrix calculated on the current dataset $\mathcal{D}_t$ using the same diagonal approximation and evaluation mode as described in Section B.1. This rolling update ensures that the parameter importance weights remain numerically stable and reflect a prioritized memory of recent tool-calling schemas.

Training Logic For the **EWC (RecAdam)** baseline, the model follows the same tool-ordered stream and five consolidation intervals. Following (Chen et al., 2020), the optimization process transitions from historical knowledge recall to new task adaptation via a sigmoid-based shifting mechanism. All training is conducted in `bf16` precision to ensure numerical stability. Specific values for the sigmoid slope k, midpoint τ_0, decay factor γ, and other optimization parameters are provided in Table 3.

B.3 BASIC EXPERIENCE REPLAY

As a representative of rehearsal-based methods, a **Basic Experience Replay** baseline is implemented to mitigate catastrophic forgetting by periodically re-introducing historical samples during the training of new tasks (Rolnick et al., 2019). The focus lies on maintaining a balanced distribution of new and old knowledge at the gradient update level through a structured data interleaving strategy.

Reservoir-based Memory Management A fixed-size memory buffer $\mathcal{M}$ is maintained with a capacity of $N = 1500$ samples. To ensure that $\mathcal{M}$ provides an i.i.d. (independent and identically distributed) representation of the entire historical data stream, **Reservoir Sampling** is utilized. For each new sample x_i encountered at step i, it is added to the buffer if $|\mathcal{M}| < N$; otherwise, it replaces a random existing sample in $\mathcal{M}$ with a probability of N/i. The choice of $N = 1500$ is informed by the ablation studies in Rolnick et al. (2019), which demonstrate that once the buffer reaches a sufficient threshold to represent the task distribution, further increasing the buffer size yields diminishing returns in stability.

Interleaved Optimization Strategy To prevent the model from oscillating between disjoint task distributions, a sample-level interleaving strategy is implemented. Rather than appending replay data as a separate block, a coupled training stream is constructed:

$$\mathcal{D}_{\text{train}} = \{x_{\text{new}}^{(1)}, x_{\text{old}}^{(1)}, x_{\text{new}}^{(2)}, x_{\text{old}}^{(2)}, \ldots, x_{\text{new}}^{(n)}, x_{\text{old}}^{(n)}\} \tag{7}$$

where x_{new} is drawn from the current tool group in the incoming stream and x_{old} is sampled uniformly from $\mathcal{M}$. Random shuffling is deliberately disabled to preserve this 1:1 interleaving ratio. This design ensures that every forward pass and subsequent gradient update consists of an exactly balanced set of new tool-calling instances and historical ones, strictly following the 50/50 experience ratio recommended in the original study to optimize for both stability and plasticity.

Training Logic For the Basic Experience Replay baseline, the optimization process interleaves current tool-calling data with historical instances. Maintaining a 1:1 ratio between incoming and buffered samples within each gradient update ensures a balanced representation of new and old knowledge. All training is conducted in `bf16` precision to ensure numerical stability. Specific values for the buffer capacity N and other optimization parameters are provided in Table 3.

B.4 Semantic-Aware Experience Replay (CorpusBrain++)

To optimize the rehearsal process, a **Semantic-Aware Experience Replay** mechanism is implemented. Moving beyond uniform random sampling, this approach leverages semantic clustering (Guo et al., 2024) to selectively retrieve historical instances that share significant functional context with the current instructions. This targeted replay strategy inherently minimizes task interference and facilitates precise knowledge consolidation.

Semantic-Aware Memory Retrieval Both incoming and buffered samples are projected into a dense semantic space via the locally deployed `all-MiniLM-L6-v2` sentence encoder. To prioritize functional semantics, the textual representation explicitly concatenates the tool description and the user query, effectively adapting the document-level retrieval of CorpusBrain++ (Guo et al., 2024) to our specific tool-calling context. For each new sample, its cosine similarity is computed against all instances currently residing in the reservoir buffer $\mathcal{M}$. To explicitly reinforce tool-specific schema retention, a constant similarity bonus δ is added to the scores of buffered samples that invoke the identical tool. Instead of retrieving the absolute maximum, the paired historical sample is uniformly drawn from the top-k candidates, effectively striking a balance between high semantic relevance and local sampling diversity to mitigate overfitting. Concurrently, the buffer $\mathcal{M}$ is updated via Reservoir Sampling to maintain an unbiased global distribution of the historical data stream.

Semantic-Coupled Optimization Strategy Following the exact interleaving formulation of the basic Experience Replay baseline, the retrieved historical sample is paired 1:1 with the incoming sample. Processing highly correlated schemas within the same gradient update acts as a localized constraint, forcing the model to explicitly distinguish and consolidate the boundaries between structurally similar tools while learning new instructions.

Training Logic For the **Semantic-Aware Experience Replay (CorpusBrain++)** baseline, the optimization process semantically couples incoming queries with relevant historical instances. Processing correlated schemas within the same gradient update explicitly forces the model to distinguish boundaries between similar tools. All training is conducted in `bf16` precision to ensure numerical stability. Specific values for the top-k candidates, similarity bonus δ, and other optimization parameters are provided in Table 3.

Table 3: Hyperparameter settings for all continual learning methods evaluated in this study.

Category / Method	Hyperparameter	Value
Shared Settings	Training Epochs	3
	LR Scheduler	Cosine
	Warmup Ratio	0.1
	Precision	`bf16`
Basic EWC (Kirkpatrick et al., 2017)	Learning Rate	5×10^{-5}
	Effective Batch Size	128
	Regularization Strength λ_{ewc}	400.0
EWC (RecAdam) (Chen et al., 2020)	Learning Rate	5×10^{-5}
	Effective Batch Size	128
	Regularization Strength λ_{ewc}	400.0
	Sigmoid Slope k	20.0
	Midpoint τ_0	0.5
	Fisher Decay γ	0.9
Basic Experience Replay (Rolnick et al., 2019)	Learning Rate	2×10^{-5}
	Effective Batch Size	512
	Buffer Capacity N	1500
Semantic-Aware ER (CorpusBrain++) (Guo et al., 2024)	Learning Rate	2×10^{-5}
	Effective Batch Size	512
	Buffer Capacity N	1500
	Top-k Candidates	5
	Similarity Bonus δ	0.3

C Details of the Data Preprocessing and Automated Alignment Pipeline

To construct the **CAFT-30k** training corpus and the **single-tool routing** evaluation subset of **CAFT-Eval**, we design and implement an automated processing pipeline. This pipeline encompasses not only foundational data cleaning but also LLM-driven semantic alignment and long-tail data augmentation mechanisms. This section details the core engineering implementations of our pipeline.

C.1 Multi-Source Data Standardization and Schema Extraction

The raw data is sourced from 10 open-source datasets. Due to the highly heterogeneous schema definitions and dialogue formats across these sources, we first employ hard parsing and heuristic rules to unify them into a standardized five-turn closed-loop trajectory. Each trajectory strictly comprises `system_prompt`, `user_instruction`, `tool_call_text`, `tool_result`, and `final_response`.

C.2 Schema-Based Quality Assurance and Legitimate Empty Call Recovery

During the cleaning phase, we dynamically construct tool specification dictionaries by parsing the `<functions>` XML tags within the system prompts.

Out-of-Schema Parameter Detection We utilize Abstract Syntax Tree (AST) parsing to analyze the call strings generated by the assistant, cross-referencing the parameter keys in real-time. Any instance containing undefined parameters or hallucinated tool names is strictly discarded.

Legitimate Empty Call Recovery For empty parameter calls formatted as `Call: func()`, the pipeline traces back to the tool's definition. If the `properties` field in the schema is genuinely empty, the call is classified as a legitimate invocation and retained, rather than being blindly discarded as a "missing parameter" error. This mechanism significantly enhances the dataset's coverage and the model's accuracy when handling zero-parameter tools (e.g., `get_current_time`).

C.3 Semantic Clustering and LLM-Driven Tool Reforging

To address the phenomena of polymorphic tools and parameter name redundancy (e.g., synonymous parameter names across different datasets), we adopt a three-stage alignment strategy:

Lexical Normalization We use the `inflect` library for singular/plural conversions and standardize camelCase and kebab-case naming conventions into a unified `snake_case` format.

Semantic Clustering We introduce the `all-MiniLM-L6-v2` model to map tool names into a high-dimensional vector space, utilizing community detection algorithms to identify tool clusters with a semantic similarity threshold exceeding 0.85.

LLM Reforging We prompt GPT-4o to act as an API architect to align the parameters of the identified tool clusters, merge synonymous parameters (e.g., `q` and `query`), and generate broader descriptions for the unified tools and their schemas.

C.4 Long-Tail Augmentation and Dynamic Distractor Injection

To mitigate the severe long-tail distribution problem inherent in real-world tool libraries, we implement a parallel augmentation engine:

Long-Tail Augmentation For rare tools with fewer than 110 samples, we leverage Few-Shot prompting to guide the LLM in synthesizing highly diverse, 5-turn closed-loop interaction trajectories.

Dynamic Negative Sampling When assembling the fine-tuning data, the pipeline randomly selects 3 non-target tools from the global tool registry to inject into the system prompt as distractors. This forces the model to learn true semantic routing logic rather than relying on positional biases or rote memorization.

Strict Format Constraints We deprecate verbose JSON structures for tool invocation outputs, adopting a compact `Call: func(k="v")` signature format. This approach significantly reduces parsing failure rates and inference latency.

C.5 Deduplication and Stratified Sampling

Finally, the pipeline performs strict deduplication by computing a feature fingerprint for the entire message sequence. During the dataset splitting phase, we employ a tool-name-based stratified sampling strategy. Specifically, we extract exactly 5 samples per unique tool to constitute the **single-tool routing** probe within **CAFT-Eval**, ensuring comprehensive coverage across all tool categories. The remaining robust samples naturally form the **CAFT-30k** training corpus, facilitating a rigorous evaluation of the model's foundational routing capabilities without data leakage.

D Additional Experiments and Dataset Analysis

This section reports additional experiments on hyperparameter sensitivity, backbone diversity, and the domain distribution of the evaluation probes.

D.1 Hyperparameter Sensitivity

We report additional experiments on **Qwen3-8B** for parameter selection and sensitivity analysis of continual fine-tuning methods under selected regularization and replay-buffer settings. Minor numerical differences may arise from variations in the computational environment.

Table 4: Hyperparameter sensitivity analysis results on **Qwen3-8B**.

Method	**Setting**	**HRF**	**SCR**	**NTHI**	**CSCR**	**TOI**
CorpusBrain++	Buffer = 1500	0.1548	0.1464	0.1241	0.4036	0.000
CorpusBrain++	Buffer = 750	0.1027	0.0958	0.0843	0.3494	0.000
EWC	$\lambda = 200$	0.0291	0.0161	0.0391	0.4140	0.855
EWC	$\lambda = 400$	0.0429	0.0184	0.0291	0.4008	0.901
EWC	$\lambda = 800$	0.0199	0.0069	0.0429	0.3980	0.841
RecAdam	$\lambda = 200,\ k = 10$	0.0268	0.0092	0.0460	0.4106	0.718
RecAdam	$\lambda = 200,\ k = 20$	0.0383	0.0084	0.0406	0.3764	0.939
Replay	Buffer = 1500	0.1625	0.1556	0.0943	0.3476	0.000
Replay	Buffer = 750	0.1663	0.1563	0.0644	0.3382	0.000

D.2 BACKBONE DIVERSITY

We report additional experiments on **Olmo3-7B** and **Ministral-3-14B-Instruct** to examine continual fine-tuning behavior across additional backbone models.

Table 5: Results on **Olmo3-7B** and **Ministral-3-14B-Instruct**.

Model	Method	HRF ↓	SCR ↓	NTHI ↓	CSCR ↑	TOI ↓
Olmo3-7B	Baseline (SFT)	**0.004598**	0.001533	**0.026820**	0.6200	**0.000**
	EWC	0.055172	0.049042	0.046743	0.5330	**0.000**
	RecAdam	0.020690	0.008429	0.054406	0.5310	**0.000**
	Replay	0.006130	0.001533	**0.026820**	0.6284	**0.000**
	CorpusBrain++	**0.004598**	**0.000000**	0.028352	**0.6370**	**0.000**
Ministral-3-14B-Instruct	Baseline (SFT)	0.006130	0.003831	0.029885	0.4742	0.032
	EWC	0.122605	0.050575	0.062835	0.3544	0.273
	RecAdam	0.160920	0.051341	0.031418	0.2192	0.502
	Replay	**0.004598**	**0.000000**	0.030651	0.5788	**0.000**
	CorpusBrain++	0.005364	0.003065	**0.026820**	**0.5824**	0.001

D.3 EVALUATION-PROBE DOMAIN DISTRIBUTION

The evaluation probes are categorized using the same domain taxonomy as the training data. The original and upgraded probes are paired at the tool level and therefore have identical domain distributions.

Table 6: Domain distribution of the original and upgraded evaluation probes.

Domain	Original	Upgraded	Percentage
General / Others	315	315	24.1%
Search & Lifestyle	270	270	20.7%
Finance & E-commerce	255	255	19.5%
Dev & Algorithms	200	200	15.3%
Productivity & Utils	140	140	10.7%
Science & Analytics	125	125	9.6%
Total	**1,305**	**1,305**	**100.0%**

LIMITATIONS AND BROADER IMPACT

CAFTBench is a static benchmark and does not execute external tools, so it isolates model-side continual fine-tuning behavior rather than end-to-end agent reliability in live environments. Although training follows an ordered sequence of tools, the reported evaluation uses the original-learning and API-upgrade endpoints; it does not characterize intermediate forgetting trajectories or historical-tool retention after API upgrades. Our experiments cover two 8B-scale open-source backbones and representative continual learning baselines; broader model families, larger scales, and additional update schedules may exhibit different trade-offs. CAFTBench is intended to support safer and more reproducible evaluation of agents under evolving tool ecosystems. However, stronger tool-calling agents may also amplify risks from incorrect, unnecessary, or unauthorized tool use, so deployment should include permission control, monitoring, and abstention safeguards.