# SPMR: CONTINUAL TOOL RETRIEVAL LEARNING VIA MULTI-PATH STRUCTURED PROMPT MEMORY RECOVERY

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Tool retrieval determines which capabilities a large language model agent can actually exercise at any given moment. Generative tool retrieval internalizes tool knowledge into model parameters, turning retrieval from context enumeration into a learned mapping. As deployed tool inventories keep growing, however, a parameterized catalogue turns every tool registration into a model update, and extending a generative tool retriever becomes a continual learning problem. Existing continual learning methods largely attribute forgetting to knowledge loss and respond by freezing or otherwise protecting historical parameters. We argue that preserved parameters do not imply preserved retrieval performance, and decompose retrieval forgetting into three separable mechanisms: **Storage**, whether historical parameters survive; **Access**, whether the router can still reach them; and **Competition**, whether newly added labels displace historical targets even when their logits are unchanged. Motivated by this view, we propose a prompt-based generative tool retriever with a structured prompt memory of box-indexed soft prompts and a multi-path recovery memory mechanism that restores access to historical tools displaced by newly added tools. On ToolBench and ToolRet, our method surpasses most continual learning baselines. These results indicate that forgetting in continual retrieval should be measured along storage, access, and competition separately, rather than attributed to a single memory problem. Code is available at `https://anonymous.4open.science/r/SPMR-3248`.

## 1 INTRODUCTION

Large language model agents increasingly rely on tool calling to interact with the external world—acquiring real-time information, performing precise computation, and operating online services—so that an agent's capabilities are largely determined by the set of tools it can invoke (Schick et al., 2023). Deployed tool inventories, however, already grow far beyond what a context window can hold: ToolBench alone covers more than 16,000 real-world REST APIs (Qin et al., 2024), and one production router already serves 34,396 skills (Murtaza et al., 2026). Tool selection is therefore delegated to a dedicated retrieval module (Patil et al., 2024).

Within this module, the generative-retrieval paradigm trains the model to directly generate the identifier of the target tool for a given query, internal izing tool knowledge into model parameters (Metzler et al., 2021; Tay et al., 2022); ToolGen, for instance, represents each API as a special token in an extended vocabulary (Wang et al., 2025). A parametric catalogue enables end-to-end semantic retrieval, but it also means that registering a new tool is no longer an index write but a model update. This shift turns catalogue maintenance into sequential model adaptation.

In real-world deployments, tool catalogues evolve continuously: new tools are added, old ones deprecated or changed. Therefore, extending a generative tool retriever is a continual-learning problem: the model must learn new tools sequentially while preserving retrieval ability for existing ones. The dominant response in continual learning is to protect what has been learned—constraining updates to parameters important for earlier tasks (Kirkpatrick et al., 2017), revisiting historical data (De Lange et al., 2021), or isolating what has been learned in dedicated components (Wang et al., 2022b;a; Smith et al., 2023). Continual generative retrieval inherits the same instinct: DSI++



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.

refreshes a parametric index as new documents arrive (Mehta et al., 2023), and PromptDSI supports expansion through stateless soft-prompt encoders (Huynh et al., 2025). Across both lines, forgetting is understood from the perspective of knowledge preservation, and success is measured by whether already-acquired knowledge remains intact. For a parametric tool catalogue—entries organized by function, growing on demand as deployment proceeds—continual extension nonetheless remains underexplored, as existing continual generative-retrieval methods take document corpora as their object of study.

We approach forgetting in this scenario from a different perspective, decomposing retrieval failure into three mechanisms: *Storage*, whether the parametric representations of historical tools are preserved; *Access*, whether the routing still allocates sufficient probability mass to historical tools; and *Competition*, whether historical tools can maintain their previous ranks after the label space expands. From this perspective, intact parameter preservation does not necessarily imply preserved retrieval performance: even with lossless storage, historical tools may fail to be returned because of perturbations in access or competition. Therefore, alongside continued improvement in how knowledge is stored, it is also necessary to study mechanisms for recovering perturbed access.

To address this underexplored problem, we propose SPMR (Structured Prompt Memory Recovery), a prompt-based generative tool retriever for continual learning that pairs a structured prompt memory for storing tool knowledge with a multi-path recovery mechanism for restoring access to historical tools.

Our contributions are summarized as follows:

- A new framework, SPMR, that introduces a Storage–Access–Competition analysis of retrieval forgetting, showing that intact parameter preservation alone does not guarantee preserved retrieval performance and that access should be recovered.
- A structured prompt memory that organizes tool descriptions into structured parametric memory via box embeddings, providing an expandable storage form for tool knowledge.
- A multi-path recovery mechanism that aggregates evidence from multiple prompt paths at inference time, restoring access to historical tools damaged by newly added ones and bridging the gap between parameter preservation and retrieval performance.

## 2 Related Work

**Tool retrieval and generative indexing.** Tool retrieval commonly matches a query against externally indexed tool representations, as studied at scale in ToolBench and ToolRet (Qin et al., 2024; Shi et al., 2025). An alternative is to parameterize the index itself: DSI maps queries directly to document identifiers (Tay et al., 2022), while ToolGen represents tools as vocabulary elements and retrieves them through generation (Wang et al., 2025). This distinction becomes consequential under continual updates. External indices can append new entries directly, whereas parameterized indices must acquire new identifiers without corrupting previously learned query–target mappings. We study this latter setting with a globally expanding tool space.

**Continual retrieval and prompt memory.** Continual learning mitigates interference through replay, regularization, or parameter-efficient capacity expansion (Rebuffi et al., 2017; Kirkpatrick et al., 2017; Zenke et al., 2017; Li & Hoiem, 2017; Aljundi et al., 2018; Buzzega et al., 2020; Wang et al., 2022b; Smith et al., 2023). Retrieval adds the difficulty that the prediction space is itself an evolving index. DSI++ combines rehearsal with sharpness-aware optimization for continual indexing (Mehta et al., 2023), whereas PromptDSI stores incremental retrieval knowledge in prompts attached to a frozen backbone (Huynh et al., 2025). PAMT likewise absorbs new documents into a parametric memory head over a frozen encoder (Mekonnen et al., 2026), but leaves the routing among memories untouched. Related prompt-based methods retrieve prompts from input-dependent keys, compose decomposed prompt components, or constrain prompt updates through gradient projection (Wang et al., 2022b; Smith et al., 2023; Qiao et al., 2024). These mechanisms primarily protect knowledge during optimization. In a routed retriever, however, a preserved prompt may cease to be selected, and newly introduced classifier rows can displace historical targets even when their stored scores are unchanged.

**Geometric routing and multi-path prediction.** Most semantic retrievers represent queries and candidates as points in an embedding space; region-based representations instead encode sets of compatible points, as in Query2box (Ren et al., 2020). A separate line of work studies conditional routing and prediction fusion: sparse mixture-of-experts layers route each input to a small subset of parameters (Shazeer et al., 2017; Fedus et al., 2022), SMoPE applies the same idea to prompt experts (Le et al., 2026), and LAE and PromptFusion combine complementary adaptation branches (Gao et al., 2023; Chen et al., 2024). IH-PromptDSI connects these ideas differently: description-derived hierarchical boxes serve as persistent prompt keys, and router-selected prompts are evaluated as separate forward paths before their tool scores are aggregated. This formulation makes three failure modes explicit—whether historical retrieval functions are *stored*, whether they remain *accessible*, and whether their targets remain *competitive* as the global tool inventory expands.

# 3 METHOD

## 3.1 PROBLEM FORMULATION AND THE STORAGE–ACCESS–COMPETITION DECOMPOSITION

We study a sequence of tasks $t = 0, \ldots, T$, with $t = 0$ the base task. At task $t$ a set $\Delta\mathcal{Y}_t$ of new tools is registered, expanding the available inventory to

$$\mathcal{Y}_t = \bigcup_{\tau=0}^{t} \Delta\mathcal{Y}_\tau, \qquad \Delta\mathcal{Y}_t \cap \mathcal{Y}_{t-1} = \varnothing \quad (t \geq 1), \tag{1}$$

where each tool $y$ carries a textual description $d_y$ and a unique identifier. The training data of task $t$ are query–tool pairs $\mathcal{D}_t = \{(x_i, y_i)\}_{i=1}^{N_t}$ over newly registered tools only, with $x_i$ a user query and $y_i \in \Delta\mathcal{Y}_t$ its target tool. The model is a function $f_t$ mapping a query to a ranking over the full inventory, and $\rho_t(x, y)$ denotes the rank it assigns to tool $y$. Evaluation covers queries targeting both new and historical tools and does not reveal a target's task. The objective is to learn $f_0, \ldots, f_T$ such that at each step $t$, $f_t$ minimizes the expected ranking loss over all tools seen so far,

$$\min_{f_t} \ \mathbb{E}_{(x,y)\sim\bigcup_{\tau=0}^{t}\mathcal{D}_\tau}\big[\ell(f_t(x), y)\big], \tag{2}$$

while training only on $\mathcal{D}_t$ . This requires both acquiring new tools and avoiding catastrophic forgetting, i.e., ranking performance on historical tools should not degrade significantly after learning $\Delta\mathcal{Y}_t$.

In continual learning, forgetting is usually described in terms of knowledge, where updates for new tasks damage the parameters that encoded earlier ones. For a parametric retriever, however, a returned tool is the outcome of a chain: the parameters must survive, the inference procedure must read the path on which they live, and the resulting score must survive ranking against a label set that has grown since. Protecting parameters alone therefore does not guarantee that a historical tool remains retrievable. We separate this chain into three conditions and say that a historical tool $y$ remains retrievable at task $t$ only when all three hold:

$$\text{retrievable}_t(y) \iff \text{Storage}_t(y) \ \wedge \ \text{Access}_t(y) \ \wedge \ \text{Competition}_t(y). \tag{3}$$

**Storage.** When a tool is registered, its description is encoded into a stored representation, and a classifier row is appended to score it. Let $r_y^{(t)}$ denote this representation after task $t$. Storage holds iff it is retained unchanged from the task $t_y$ at which $y$ was registered, so that a later update never overwrites what was stored:

$$\text{Storage}_t(y) \iff r_y^{(t)} = r_y^{(t_y)}. \tag{4}$$

If this fails, the representation has been overwritten and there is nothing left to recover.

**Access.** Let $\mathcal{C}_p(x)$ denote the set of paths that the inference procedure activates for query $x$ at task $t$. Access holds iff the path $m$ on which $y$'s representation lives is among them:

$$\text{Access}_t(y) \iff m \in \mathcal{C}_p(x) \quad \text{for the query } x \text{ targeting } y, \tag{5}$$

where $m$ is the box whose prompt stores $y$. A preserved representation that is never routed to contributes nothing and behaves exactly as if it had been overwritten.

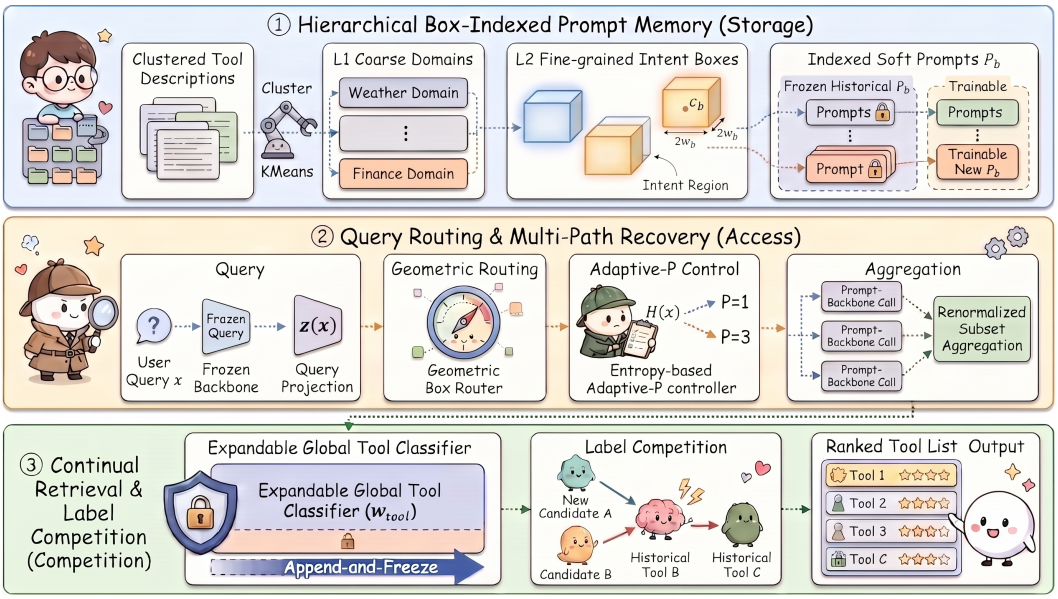


Figure 1: Overview of SPMR. Boxes over tool descriptions index soft prompts; a router selects prompt paths; an expandable classifier ranks all tools seen so far. Inherited entries are frozen during registration, and several paths are aggregated at inference.

**Competition.** Competition holds iff $y$ still ranks within the top $k$:

$$\text{Competition}_t(y) \iff \rho_t(x, y) \leq k \quad \text{for the query } x \text{ targeting } y. \tag{6}$$

SPMR is built for the conjunction of all three rather than for storage alone. It addresses storage by keeping inherited entries frozen, access by reading several memory paths, and competition by ranking over a growing label set. The mechanisms are developed in the next three sections.

### 3.2 Overview

SPMR keeps the backbone frozen and adds three components (Figure 1). A structured prompt memory stores soft prompts keyed by hierarchical boxes over tool descriptions and grows only by appending. At inference, a query is projected into the description space, where a geometric router scores it against these boxes and adaptively selects the top-$p$ scored prompt , $p$ being a per-query number of paths. Each selected path reads its box's prompt, concatenates it with the query's hidden state, and passes the result through the frozen backbone and a shared classifier to produce a tool logit vector. Then the $p$ vectors are weight-averaged into a single ranking over every tool seen so far. The classifier appends one row per registered tool while inherited rows stay frozen.

### 3.3 Structured Prompt Memory with Box-Indexed Soft Prompts

Descriptions of distinct tools often differ in only a few words, and those words carry the operation, so a flat clustering over description similarity can merge tools that serve different intents while keeping them apart would require one region per tool. Tool functionality instead admits a coarse *domain* (weather, say) and a finer *intent* within it (querying today's weather), and we encode both levels rather than forcing one clustering resolution to serve both.

Specifically, we encode each tool description using a frozen sentence encoder $e_y \in \mathbb{R}^{384}$, partition the resulting embeddings into coarse L1 domain groups using $k$-means, and apply HDBSCAN within each domain to identify dense L2 intent clusters (McInnes et al., 2017). Each resulting region $b \in \mathcal{B}^{(t)}$ is an axis-aligned box $B_b = [\, c_b - w_b,\ c_b + w_b \,]$ over the description space, with center $c_b \in \mathbb{R}^{384}$, half-width $w_b \in \mathbb{R}^{384}_{>0}$, and a floor $\delta > 0$ on the half-width, $\tilde{w}_b = \max(w_b, \delta)$, that keeps boxes from degenerating. Each box indexes a trainable soft prompt $P_b \in \mathbb{R}^{m \times d_{\text{model}}}$ shared by the tools in its region, $d_{\text{model}}$ being the hidden dimension of the frozen backbone, so tools that share a box stay distinct labels and we never need a region per tool.

New tools are then assigned to a compatible existing box or grouped into a new one, with historical centers, widths, and indices left untouched so no global reconstruction is required. At task $t$ we append prompts for $b \in \mathcal{B}^{(t)} \setminus \mathcal{B}^{(t-1)}$ and one classifier row per new tool id, freezing everything inherited along with all box centers and widths, so only the appended entries and the query projection of Section 3.4 remain trainable. The box keys themselves are trained to place a query inside the box of its target tool through a cross-entropy over boxes and a containment penalty,

$$\mathcal{L}_{\text{box}} = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{\exp(r_{b_i}(z_i))}{\sum_{j\in\mathcal{B}^{(t)}} \exp(r_j(z_i))}, \qquad \mathcal{L}_{\text{inside}} = \frac{1}{Nd}\sum_{i=1}^{N} \left\| [\,|z_i - c_{b_i}| - \tilde{w}_{b_i}]_+ \right\|_2^2, \quad (7)$$

where $b_i$ is the box of the target tool $y_i$, $z_i = z(x_i)$, $[\cdot]_+$ is applied elementwise, and $d = 384$ is the embedding dimension.

### 3.4 Query Routing and Classification

The query passes through the first transformer layer of the frozen Llama-3 backbone (Grattafiori et al., 2024), and the hidden states of valid tokens are mean-pooled into $h(x) \in \mathbb{R}^{4096}$. A two-layer projection maps this into the description space so a query can be scored against the box geometry,

$$z(x) = W_2\,\mathrm{GELU}(W_1 h(x)) \in \mathbb{R}^{384}, \quad (8)$$

where $W_1 \in \mathbb{R}^{1024\times 4096}$ and $W_2 \in \mathbb{R}^{384\times 1024}$ are the only trained parameters of this branch. Each query is scored against a box through two distances, one measuring how far it falls outside the box and the other its width-normalized distance to the center,

$$d_b^{\text{out}}(z) = \left\| \left[\,|z - c_b| - \tilde{w}_b\,\right]_+ \right\|_2, \qquad d_b^{\text{ctr}}(z) = \left\| \frac{z - c_b}{\tilde{w}_b + \epsilon} \right\|_2, \quad (9)$$

where $\tilde{w}_b$ is the truncated half-width of Section 3.3 and $\epsilon > 0$ a small constant; the center distance matters because overlapping boxes can all contain the query, leaving $d^{\text{out}}$ zero for every candidate. The two distances become the routing logit and the probability of reading region $b$,

$$r_b(z) = -\frac{d_b^{\text{out}}(z) + \alpha\, d_b^{\text{ctr}}(z)}{\tau}, \qquad s_b^{\text{sem}}(x) = \frac{\exp(r_b(z(x)))}{\sum_{j\in\mathcal{B}^{(t)}} \exp(r_j(z(x)))}, \quad (10)$$

where $\alpha$ weights the center term against the outside term and $\tau$ is a temperature; $b^\star = \arg\max_b s_b^{\text{sem}}(x)$ gives top-1 routing, and Section 3.5 reads several boxes instead.

Once a box is selected, its prompt $P_b$ is prepended to the query token embeddings and the sequence passes through the frozen backbone; let $h_b(x) \in \mathbb{R}^{4096}$ be the hidden state at the last valid position. A global classifier then scores every tool in the inventory,

$$\ell_b(x) = W_{\text{tool}}\, h_b(x) \in \mathbb{R}^{|\mathcal{Y}_t|}, \qquad \mathcal{L}_{\text{tool}} = -\frac{1}{N}\sum_{i=1}^{N} \log \mathrm{softmax}\left(\ell_{b_i}(x_i)\right)_{y_i}, \quad (11)$$

where $W_{\text{tool}} \in \mathbb{R}^{|\mathcal{Y}_t|\times 4096}$ holds one row per global tool id and grows as tools are registered. The full per-task objective combines this classification loss with the two geometric terms of Section 3.3,

$$\mathcal{L} = \mathcal{L}_{\text{tool}} + \lambda_{\text{box}}\mathcal{L}_{\text{box}} + \lambda_{\text{inside}}\mathcal{L}_{\text{inside}}. \quad (12)$$

### 3.5 Multi-Path Prompt Recovery

Under top-1 routing, a single mis-ranked box becomes a point of failure: reading only one memory path makes retrieval depend on a routing decision that can be perturbed as the memory grows. SPMR instead forms a candidate set $\mathcal{C}_p(x)$ containing the $p$ highest-scoring boxes and aggregates the corresponding prediction paths, with weights renormalized over this set,

$$\tilde{s}_b(x) = \frac{s_b^{\text{sem}}(x)}{\sum_{j\in\mathcal{C}_p(x)} s_j^{\text{sem}}(x)}, \qquad \ell_{\text{ens}}(x) = \sum_{b\in\mathcal{C}_p(x)} \tilde{s}_b(x)\,\ell_b(x). \quad (13)$$

The number of selected paths $p$ therefore controls the inference cost and serves as a per-query computational budget.

A fixed budget spends the same computation on every query, so we select $p$ per query with a policy over a small ordered set $\mathcal{P} \subset \mathbb{Z}_+$ of candidate path counts, ranging from a frugal to a generous budget. The policy observes only information available before any path is executed: the projected query, the router probabilities with their cumulative sums and decay, and geometric features from the box distances of Section 3.4. It never sees the target tool, the retrieval result, or the task identity. For two adjacent candidates $a < b$ in $\mathcal{P}$, let $\mathrm{hit}_p(x) \in \{0, 1\}$ indicate whether the target appears in the top $k$ of the final classifier ranking when the ensemble uses $p$ paths, and define $y_{a\to b}(x) = \mathrm{hit}_b(x) - \mathrm{hit}_a(x)$, where $+1$ marks a *rescue*, $0$ an unchanged outcome, and $-1$ a *regression*. One tree-ensemble classifier (Breiman, 1996) per adjacent pair is trained on training queries whose outcome under every candidate budget was actually executed, and returns class probabilities from which we take the signed marginal utility

$$\widehat{\Delta}_{a\to b}(x) = P(y_{a\to b} = +1 \mid x) - P(y_{a\to b} = -1 \mid x). \tag{14}$$

With a cost $\lambda \geq 0$ per additional path, the utility of each candidate is its predicted gain minus its cost, accumulated across the chain of adjacent steps: $U_p = 0$ for the smallest candidate, and $U_b = U_a + \widehat{\Delta}_{a\to b} - (b - a)\lambda$ for each adjacent pair $a < b$, so that enlarging the budget is chosen only when the predicted rescue outweighs its price. The policy then takes

$$\pi(x) = \arg\max_{p\in\mathcal{P}} U_p(x). \tag{15}$$

The single global price $\lambda$ is fitted on a calibration split by minimizing the mean selected budget, so one $\lambda$ yields different budgets for different queries.

# 4 Experiments

## 4.1 Experimental Setup

**Datasets** We evaluate SPMR on two tool retrieval benchmarks: ToolBench (Qin et al., 2024) and ToolRet (Shi et al., 2025) . To simulate the continual learning setting, we split each tool inventory into a base task followed by tasks that register new tools only: ToolBench starts from 11,112 tools and adds 640, 640, and 643 tools, while ToolRet starts from 14,175 tools and adds 2,382, 2,373, 2,361, and 2,348 tools. Following Section 3.1, training at task $t$ uses only the queries of task $t$, whose targets are the tools registered at $t$. At each task, evaluation covers the queries of all tasks observed so far. The target's task is never revealed to the model, and the same protocol is applied to all baselines.

**Baselines** We are not aware of prior tool-retrieval work that jointly considers sequential tool introduction and task-agnostic ranking over all tools observed so far; our comparison therefore includes retrieval methods and continual-learning methods adapted to this setting. **Retrieval baselines. DSI++** (Mehta et al., 2023) and **PromptDSI** (Huynh et al., 2025) are the closest continual-retrieval comparators: the DSI++ adaptation replays historical query–tool examples, whereas PromptDSI uses rehearsal-free prompt memory. **Continual-learning baselines. L2P** (Wang et al., 2022b) and **CODA-Prompt** (Smith et al., 2023) provide input-dependent prompt selection and prompt-composition baselines, respectively. **PGP** (Qiao et al., 2024) constrains prompt updates through gradient projection. We additionally include **InsCL** (Wang et al., 2024) as a replay-based comparator and **EWC-DR** (Liu & Chang, 2026) as a regularization-based comparator that corrects EWC's Fisher-based importance estimation.

**Evaluation Metrics** We adopt Recall@$k$ and NDCG@$k$ as retrieval metrics (Manning et al., 2008; Järvelin & Kekäläinen, 2002). The main tables report $k = 5$, and results for the remaining cutoffs are given in Section B. To evaluate continual learning, after training on task $t$ we report the average performance over the tasks seen so far ($A_t$), its average over the whole sequence (AIP), forgetting ($F_t$), and backward transfer (BWT) (Lopez-Paz & Ranzato, 2017; Chaudhry et al., 2018; Rebuffi et al., 2017). Let $P_{t,i}$ denote either Recall@$k$ or NDCG@$k$ on the queries of task $i$ after training through task $t$, with $i \leq t$ and tasks indexed as in Section 3.1. Each quantity below is computed separately for the two metrics.

$$A_t = \frac{1}{t+1}\sum_{i=0}^{t} P_{t,i}, \qquad \mathrm{AIP} = \frac{1}{T+1}\sum_{t=0}^{T} A_t, \tag{16}$$

Table 1: Main results on Recall@5. AIP, FT, and BWT are defined in Section 4.1; lower FT and higher BWT are better. Params is the persistent state beyond the frozen backbone, in MiB, that must be loaded for inference; lower is better. **Best** and second best are marked per column for AIP, FT, and BWT, and "–" denotes a result that is not yet available.

| | ToolBench | | | | ToolRet | | | |
|---|---|---|---|---|---|---|---|---|
| Method | AIP ↑ | FT ↓ | BWT ↑ | Params ↓ | AIP ↑ | FT ↓ | BWT ↑ | Params ↓ |
| L2P | 57.21±3.30 | 48.61±1.46 | $-48.61 \pm 1.46$ | 212.32 | 70.48±0.98 | 12.08±1.39 | $-12.08 \pm 1.39$ | 378.04 |
| InsCL | 74.86±0.66 | 17.97±0.93 | $-13.51 \pm 0.65$ | 255.72 | 73.01±0.27 | 35.79±0.87 | $-35.41 \pm 0.90$ | 421.45 |
| PGP | 44.98±0.58 | 45.61±0.63 | $-45.61 \pm 0.63$ | 44.78 | 64.91±2.51 | 9.56±2.89 | $-9.56 \pm 2.89$ | 60.35 |
| DSI++ | 39.86±0.09 | 47.00±0.10 | $-47.00 \pm 0.10$ | 19.14 | 73.12±0.03 | 27.53±0.26 | $-27.53 \pm 0.26$ | 34.72 |
| CODA | **80.20**±1.67 | **1.10**±0.52 | $\mathbf{-1.09 \pm 0.53}$ | 211.27 | **88.63**±0.17 | **4.42**±0.22 | $\mathbf{-4.42 \pm 0.22}$ | 378.90 |
| EWC-DR | 71.81±0.24 | 11.30±0.91 | $-11.30 \pm 0.91$ | 36.65 | 60.50±0.30 | 29.44±0.39 | $-29.44 \pm 0.39$ | 52.23 |
| PromptDSI | 45.62±2.47 | 23.27±0.70 | $-23.27 \pm 0.70$ | 165.16 | 49.57±2.09 | 21.94±5.07 | $-21.94 \pm 5.07$ | 253.79 |
| SPMR (ours) | 58.15±2.38 | 2.21±0.54 | $-2.21 \pm 0.54$ | 196.99 | 69.38±1.95 | 9.17±1.07 | $-9.17 \pm 1.07$ | 307.88 |

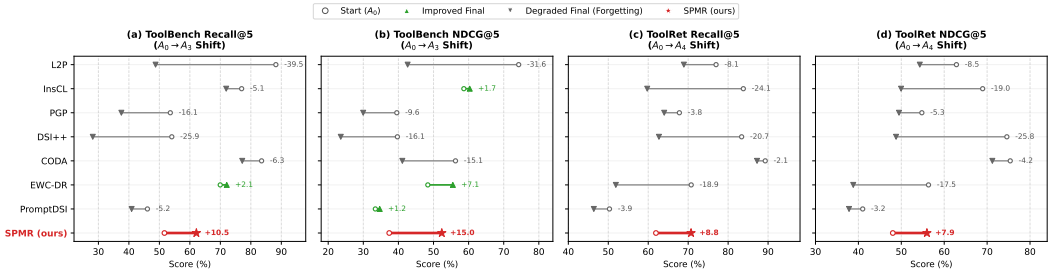


Figure 2: Base-to-final shift of the per-task average $A_t$ on (a) ToolBench Recall@5, (b) ToolBench NDCG@5, (c) ToolRet Recall@5, and (d) ToolRet NDCG@5. Each row is one method: the circle marks the base evaluation $A_0$, the triangle the final evaluation $A_T$ (upward: improved; downward: degraded), and the line the shift between them; SPMR (ours) is emphasized.

$$F_t = \frac{1}{t}\sum_{i=0}^{t-1}\Big(\max_{i' \in \{0,\dots,t-1\}} P_{i',i} - P_{t,i}\Big), \qquad \mathrm{BWT} = \frac{1}{T}\sum_{t=0}^{T-1}\big(P_{T,i} - P_{i,i}\big). \tag{17}$$

**Implementation Details** We use Llama-3-8B (Grattafiori et al., 2024) as a frozen backbone, with query embeddings produced by a sentence encoder (Reimers & Gurevych, 2019). We train with AdamW (Loshchilov & Hutter, 2019) on eight RTX 4090 GPUs. Full training configuration, data splits, and hyperparameters are given in Appendix A.

### 4.2 Comparison Results

Table 1 reports Recall@5, and Figure 2 shows the base-to-final shift behind it. Appendix B provides the full supporting evidence, including per-task Recall@5 and NDCG@5 on both ToolBench and ToolRet, the $A_t$ trajectories with AIP/FT/BWT breakdowns. On absolute accuracy CODA leads every column, so it is the ceiling rather than the target: SPMR sits in the middle of the pack, ahead of the rehearsal-free and retrieval-only methods yet below CODA, InsCL, and EWC-DR. The picture reverses on the axes that measure continual learning, where SPMR is second only to CODA and ahead of every other baseline. Its forgetting is about twice CODA's but a fifth of the next-best baseline's, so SPMR protects what has been learned far better than the methods it trails on accuracy, and backward transfer tells the same story. This stability shows in Figure 2: SPMR is the only method that ends above its base score on all four panels, whereas every other method, including CODA, ends below on at least one. The stability does not come at a memory cost: SPMR stores less persistent state than the strong baselines it beats on stability (CODA, L2P, InsCL), while the lightweight baselines that store less (DSI++, EWC-DR, PGP) pay for their small footprint in forgetting. The trade-off is therefore explicit: SPMR spends a modest amount of memory to buy the stability that the lightweight baselines give up, and spends less memory than the heavyweight baselines that come close to its stability.

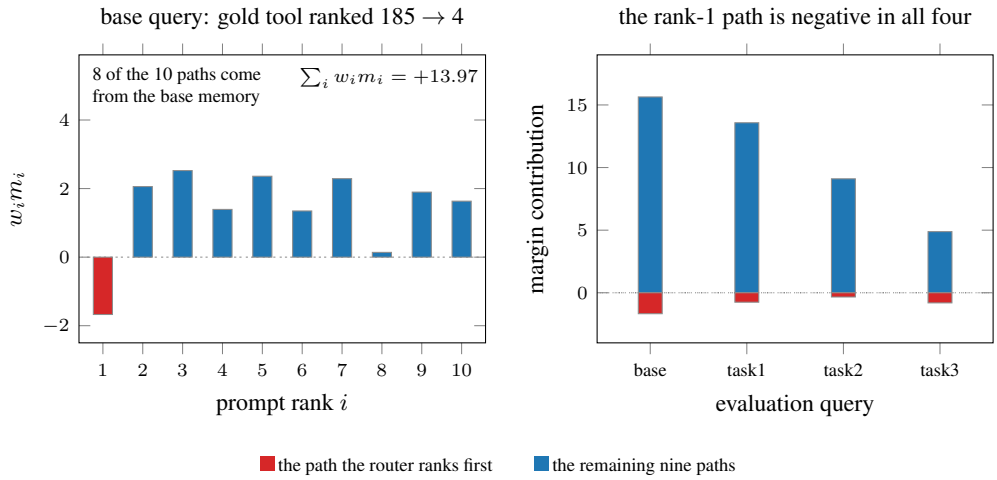


Figure 3: Multi-path recovery on ToolBench. Each trace is one query from a 300-query evaluation sample per stage, taken at the final (task3) checkpoint with $p$=10; $w_i$ is the normalized router weight and $m_i$ the fixed-rival margin of path $i$.

### 4.3 Ablations Study

**Multi-path recovery & Adaptive K-selection strategy.** We ablate two design choices: (1) reading multiple prompt paths per query versus using a single routing path, and (2) using an adaptive path budget versus a fixed one. For the first, going from $p$=1 to $p$=10 raises AIP from 47.06 to 60.38, cuts forgetting from 4.61 to 2.84, and improves BWT from $-4.39$ to $-1.84$ (Table 2). For the second, sweeping the fixed budget over $p \in \{1, \ldots, 10\}$ gives an average Recall@5 of 64.09 and a best value of 65.92, while the adaptive policy reaches 66.04 using only 6.01 paths per query on average (Table 8). In other words, the adaptive policy essentially matches the best fixed setting and edges slightly past it, while using 40% fewer paths than $p$=10. Together, these results show that multiple paths reduce forgetting, and that choosing the path count per query gets the same or better retrieval at lower cost.

Table 2: Continual-learning metrics of the adaptive path budget against fixed budgets on ToolBench (Recall@5). **Best** per column.

| Budget | AIP | FT | BWT |
|---|---|---|---|
| Fixed $p$=1(w/o multi-path) | 47.06 | 4.61 | $-4.39$ |
| Fixed $p$=3 | 58.18 | 2.98 | $-2.52$ |
| Fixed $p$=5 | 59.83 | 3.09 | $-2.36$ |
| Fixed $p$=8 | 60.31 | 2.87 | $-1.98$ |
| Fixed $p$=10 | **60.38** | **2.84** | $\mathbf{-1.84}$ |
| Adaptive $p$ | 60.24 | 3.00 | $-2.31$ |

Table 3: Box keys against prototype keys ($w_b = \delta$) on ToolBench (Recall@5). **Best** per column.

| Key | AIP | FT | BWT |
|---|---|---|---|
| Box (ours) | **61.17** | **3.07** | $\mathbf{-3.07}$ |
| Prototype | 47.37 | 6.70 | $-6.70$ |

**w/o box keys.** A box key matches a query by whether it falls inside the region, whereas a prototype key matches by nearest center alone. This distinction matters for forgetting: when a query drifts toward a neighboring tool, a nearest-center match switches it to that tool at once, while a containment match keeps it with its original box as long as it stays inside, so the historical tool remains retrievable. Box keys reach an AIP of 61.17 against 47.37 for prototypes (Table 3), so the same memory and budget recover far more once the key can contain a query. Forgetting drops from 6.70 to 3.07 and backward transfer improves from $-6.70$ to $-3.07$, which is the containment behavior the analysis above predicts: a query that drifts toward a neighboring tool stays with its original box, so earlier tools remain retrievable instead of being displaced.

### 4.4 Analysis

To see why multiple paths help, we take the task3 checkpoint on ToolBench with $p$=10, so that the ten selected prompts of each query are available, and inspect one query that top-1 routing ($p$=1) gets wrong but the ensemble rescues. Figure 3 plots, for each of the ten paths, its weighted margin contribution $w_i m_i$ against its prompt rank $i$ (left), and repeats the decomposition for one such query

per evaluation set (right), with the rank-1 path in red and the rest in blue. The rank-1 path—the one a single-path model would trust—is the only negative contributor in every trace, so its vote actively hurts and is why $p$=1 fails; the remaining paths agree on the gold tool and their positive margins outvote it, which is how the rescue happens. The vote works because those paths carry evidence that is correlated but not redundant, the regime in which rank fusion is effective (Cormack et al., 2009): a few decisive paths sit on top of a broad agreement rather than an average of identical opinions.

## 5 LIMITATIONS

Two limitations remain. First, SPMR addresses tool retrieval only: it does not close the loop of calling a retrieved tool, adjusting from its feedback, and calling again, nor does it implement multi-tool, multi-step tool-chain execution. Second, SPMR achieves strong continual stability with less persistent memory than CODA, but remains less accurate. This gap may reflect CODA's greater prompt-learning capacity through end-to-end, input-conditioned composition of decomposed prompt components. Future work will explore augmenting our structured memory with decomposed and residual prompt refinement while preserving frozen historical entries and multi-path recovery. Third, we only explore tool deletion and update as a preliminary post-hoc study (Appendix C), whose end-to-end performance remains bounded by router accuracy.

## 6 CONCLUSION

We study continual tool retrieval, where new tools must be registered from new-task data while queries are ranked against a task-agnostic, globally expanding inventory. SPMR organizes tool descriptions into an append-only memory of box-indexed prompts and uses adaptive multi-path inference to recover tools that may be missed under top-1 routing. Across two benchmarks, SPMR achieves a favorable trade-off between continual stability and persistent memory, while path-budget experiments and case analysis demonstrate that additional prompt paths can rescue top-1 routing failures. More broadly, our formulation and empirical analysis indicate that parameter preservation alone does not ensure retrievability: historical knowledge may remain stored while its prompt path becomes inaccessible, and an expanding label space may further alter its rank. This motivates a broader principle for continual retrieval—training-time preservation should be complemented by inference-time access recovery. Future work will strengthen prompt composition and competition-aware learning, improve tool addition, update, and deletion, and extend structured prompt memory from single-step retrieval to multi-tool execution with feedback.

### AI USE STATEMENT

In this work, we used generative AI tools to aid and polish writing, improve readability, and retrieve, identify, and summarize related literature. We did not use them to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate mathematical claims or write proofs, propose or refine hypotheses, design our methodology or experiments, or interpret results; these required-disclosure subtasks were performed without AI assistance or are not applicable to this work. All AI-assisted work, including suggested references and edited text, was verified and reviewed by the authors, who take full responsibility for the final content of this work.

### REPRODUCIBILITY STATEMENT

Reproducibility details are provided in Appendix A (training configuration, data splits, and hyperparameters) and Appendix B (additional results). The complete data processing steps for the continual tool retrieval benchmarks are described in Section 4.1.

## REFERENCES

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Computer Vision – ECCV 2018*, volume 11207 of *Lecture Notes in Computer Science*, pp. 144–161. Springer, 2018. doi: 10.1007/978-3-030-01219-9_9.

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996. doi: 10.1007/BF00058655.

Pietro Buzzega, Matteo Boschini, Angelo Porrello, Davide Abati, and Simone Calderara. Dark experience for general continual learning: A strong, simple baseline. In *Advances in Neural Information Processing Systems*, volume 33, pp. 15920–15930, 2020.

Arslan Chaudhry, Puneet K. Dokania, Thalaiyasingam Ajanthan, and Philip H. S. Torr. Riemannian walk for incremental learning: Understanding forgetting and intransigence. In *Computer Vision – ECCV 2018*, volume 11215 of *Lecture Notes in Computer Science*, pp. 556–572. Springer, 2018. doi: 10.1007/978-3-030-01252-6_33.

Haoran Chen, Zuxuan Wu, Xintong Han, Menglin Jia, and Yu-Gang Jiang. Promptfusion: Decoupling stability and plasticity for continual learning. In *Computer Vision – ECCV 2024*, pp. 196–212, 2024. doi: 10.1007/978-3-031-73021-4_12. URL https://www.ecva.net/papers/eccv_2024/papers_ECCV/html/11826_ECCV_2024_paper.php.

Gordon V. Cormack, Charles L. A. Clarke, and Stefan Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 758–759. ACM, 2009. doi: 10.1145/1571941.1572114.

Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3366–3385, 2021. doi: 10.1109/TPAMI.2021.3057446.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL https://jmlr.org/papers/v23/21-0998.html.

Qiankun Gao, Chen Zhao, Yifan Sun, Teng Xi, Gang Zhang, Bernard Ghanem, and Jian Zhang. A unified continual learning framework with general parameter-efficient tuning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11449–11459. IEEE, 2023. URL https://openaccess.thecvf.com/content/ICCV2023/html/Gao_A_Unified_Continual_Learning_Framework_with_General_Parameter-Efficient_Tuning_ICCV_2023_paper.html.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Tuan-Luc Huynh, Thuy-Trang Vu, Weiqing Wang, Yinwei Wei, Trung Le, Dragan Gasevic, Yuan-Fang Li, and Thanh-Toan Do. Promptdsi: Prompt-based rehearsal-free continual learning for document retrieval. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2025*, volume 16019 of *Lecture Notes in Computer Science*, pp. 383–401. Springer, 2025. doi: 10.1007/978-3-032-06109-6_22. URL https://arxiv.org/abs/2406.12593.

Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

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. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114.

Minh Le, Bao-Ngoc Dao, Huy Nguyen, Quyen Tran, Anh Nguyen, and Nhat Ho. One-prompt strikes back: Sparse mixture of experts for prompt-based continual learning. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/099ba28b20462dac7bf057d21c27a011-Abstract-Conference.html.

Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947, 2017. doi: 10.1109/TPAMI.2017.2773081.

Xuan Liu and Xiaobin Chang. Elastic weight consolidation done right for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3930–3940, 2026. URL https://arxiv.org/abs/2603.18596.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, volume 30, pp. 6467–6476, 2017.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1711.05101.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*, 2(11):205, 2017.

Sanket Vaibhav Mehta, Jai Gupta, Yi Tay, Mostafa Dehghani, Vinh Q. Tran, Jinfeng Rao, Marc Najork, Emma Strubell, and Donald Metzler. Dsi++: Updating transformer memory with new documents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://arxiv.org/abs/2212.09744.

Kidist Amde Mekonnen, Yubao Tang, and Maarten de Rijke. A parametric memory head for continual generative retrieval. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1347–1358, 2026.

Donald Metzler, Yi Tay, Dara Bahri, and Marc Najork. Rethinking search: Making domain experts out of dilettantes. *SIGIR Forum*, 55(1):13:1–13:27, 2021.

Syed Shariyar Murtaza, Yifan Nie, Utkarsh Soni, Eugene Wen, and Arvid Frydenlund. When synthetic data hurts: On catastrophic forgetting in skill retrieval for LLM agents. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing: Industry Track*, 2026. URL https://arxiv.org/abs/2609.10750.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. In *Advances in Neural Information Processing Systems*, volume 37, pp. 126544–126565, 2024. doi: 10.52202/079017-4020.

Jingyang Qiao, Zhizhong Zhang, Xin Tan, Chengwei Chen, Yanyun Qu, Yong Peng, and Yuan Xie. Prompt gradient projection for continual learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/46c2a9a6f2b2be68682013eb1173c801-Abstract-Conference.html.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.16789.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017. doi: 10.1109/CVPR.2017.587.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of EMNLP-IJCNLP*, pp. 3982–3992. Association for Computational Linguistics, 2019. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410/.

Hongyu Ren, Weihua Hu, and Jure Leskovec. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/2002.05969.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.

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

James Seale Smith, Leonid Karlinsky, Vyshnavi Gutta, Paola Cascante-Bonilla, Donghyun Kim, Assaf Arbelle, Rameswar Panda, Rogerio Feris, and Zsolt Kira. Coda-prompt: Continual decomposed attention-based prompting for rehearsal-free continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023. URL https://arxiv.org/abs/2211.13218.

Yi Tay, Vinh Q. Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, Tal Schuster, William W. Cohen, and Donald Metzler. Transformer memory as a differentiable search index. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://arxiv.org/abs/2202.06991.

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

Yifan Wang, Yafei Liu, Chufan Shi, Haoling Li, Chen Chen, Haonan Lu, and Yujiu Yang. Inscl: A data-efficient continual learning paradigm for fine-tuning large language models with instructions. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 663–677. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.naacl-long.37. URL https://aclanthology.org/2024.naacl-long.37/.

Zifeng Wang, Zizhao Zhang, Sayna Ebrahimi, Ruoxi Sun, Han Zhang, Chen-Yu Lee, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Dualprompt: Complementary prompting for rehearsal-free continual learning. In *European Conference on Computer Vision*, pp. 631–648, 2022a. URL https://arxiv.org/abs/2204.04799.

Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. Learning to prompt for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 139–149, 2022b. URL https://arxiv.org/abs/2112.08654.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 3987–3995. PMLR, 2017.

## A Implementation Details and Hyperparameters

Queries are truncated to 256 tokens, mean-pooled at the first transformer layer, and mapped by a two-layer projection (4096–1024–384, GELU, dropout 0.15) into the 384-dimensional space of a frozen `all-MiniLM-L6-v2` sentence encoder (Reimers & Gurevych, 2019). Each L2 region stores a soft prompt of length 10 initialized as $\mathcal{N}(0, 0.02^2)$. Routing uses a width floor of 0.05, a center weight of 0.05, and temperature $0.2$. We train with AdamW (Loshchilov & Hutter, 2019) (weight decay 0.01, gradient clipping 1.0 in the base task and 0.5 afterwards) on eight RTX 4090 GPUs. The base task runs for 20 epochs and each later task for up to 30, with learning rates $10^{-4}$

and $10^{-3}$ and $10^{-5}$ for the query projection; batch size is 16 on ToolBench and 4/8 on ToolRet. Early stopping patience is 5, and results are averaged over seeds 42–46 with standard deviations reported.

## B ADDITIONAL COMPARISON RESULTS

Tables 4–7 give the full per-task results behind Tables 1 and **??**. For ToolBench, $A_t$ with $t = 0, \ldots, 3$ is the average over the tasks seen so far after training through task $t$, so $A_3$ is the final average; for ToolRet, the same holds with $t = 0, \ldots, 4$. Table 8 gives the full path-budget sweep behind Table 2.

Table 4: Per-task results on ToolBench (Recall@5). **Best** and second best are marked per column.

| Method | $A_0$ | $A_1$ | $A_2$ | $A_3$ | AIP | FT | BWT |
|---|---|---|---|---|---|---|---|
| L2P | **88.21**±0.12 | 45.31±4.47 | 46.55±4.44 | 48.75±4.37 | 57.21±3.30 | 48.61±1.46 | $-48.61 \pm 1.46$ |
| InsCL | 76.98±0.69 | 74.34±0.66 | 76.22±0.43 | 71.92±1.17 | 74.86±0.66 | 17.97±0.93 | $-13.51 \pm 0.65$ |
| PGP | 53.59±0.11 | 47.11±1.13 | 41.67±0.91 | 37.53±0.54 | 44.98±0.58 | 45.61±0.63 | $-45.61 \pm 0.63$ |
| DSI++ | 54.07±0.09 | 42.72±0.13 | 34.48±0.42 | 28.15±0.12 | 39.86±0.09 | 47.00±0.10 | $-47.00 \pm 0.10$ |
| CODA | 83.52±0.33 | **81.29**±2.03 | **78.80**±2.38 | **77.18**±2.59 | **80.20**±1.67 | **1.10**±0.52 | $\mathbf{-1.09 \pm 0.53}$ |
| EWC-DR | 69.96±0.37 | 72.90±0.24 | 72.28±0.21 | 72.10±0.69 | 71.81±0.24 | 11.30±0.91 | $-11.30 \pm 0.91$ |
| PromptDSI | 46.08±3.49 | 50.51±2.78 | 44.99±1.97 | 40.90±1.91 | 45.62±2.47 | 23.27±0.70 | $-23.27 \pm 0.70$ |
| SPMR (ours) | 51.66±2.51 | 58.59±1.84 | 60.21±2.93 | 62.13±2.71 | 58.15±2.38 | 2.21±0.54 | $-2.21 \pm 0.54$ |

Table 5: Per-task results on ToolBench (NDCG@5). Conventions follow Table 4.

| Method | $A_0$ | $A_1$ | $A_2$ | $A_3$ | AIP | FT | BWT |
|---|---|---|---|---|---|---|---|
| L2P | **74.26**±0.08 | 42.00±4.44 | 41.65±4.19 | 42.63±3.82 | 50.14±3.10 | 46.93±1.85 | $-46.93 \pm 1.85$ |
| InsCL | 58.64±0.63 | 51.29±0.70 | **62.24**±0.23 | **60.32**±0.70 | **58.12**±0.54 | 13.95±0.47 | $-3.10 \pm 0.45$ |
| PGP | 39.54±0.06 | 40.82±0.81 | 34.14±0.84 | 29.97±0.51 | 36.12±0.49 | 41.72±0.30 | $-41.72 \pm 0.30$ |
| DSI++ | 39.71±0.06 | 35.27±0.10 | 28.44±0.25 | 23.58±0.05 | 31.75±0.05 | 39.27±0.12 | $-39.27 \pm 0.12$ |
| CODA | 65.48±0.26 | **57.55**±2.44 | 53.51±2.57 | 50.63±2.79 | 56.79±1.89 | **1.36**±0.60 | $\mathbf{-1.36 \pm 0.62}$ |
| EWC-DR | 48.52±0.14 | 56.55±1.44 | 56.26±0.82 | 56.19±1.04 | 54.38±0.74 | 13.21±1.24 | $-13.21 \pm 1.24$ |
| PromptDSI | 33.43±2.36 | 43.26±2.00 | 38.61±1.59 | 34.68±1.51 | 37.49±1.81 | 24.81±0.61 | $-24.81 \pm 0.61$ |
| SPMR (ours) | 37.39±1.85 | 48.75±1.40 | 50.86±2.59 | 52.36±2.31 | 47.34±1.90 | **5.51**±0.65 | $-5.51 \pm 0.65$ |

Table 6: Per-task results on ToolRet (Recall@5), with $t = 0, \ldots, 4$.

| Method | $A_0$ | $A_1$ | $A_2$ | $A_3$ | $A_4$ | AIP | FT | BWT |
|---|---|---|---|---|---|---|---|---|
| L2P | 76.98±0.23 | 67.57±1.32 | 69.47±1.50 | 69.45±1.81 | 68.92±1.07 | 70.48±0.98 | 12.08±1.39 | $-12.08 \pm 1.39$ |
| InsCL | 83.81±0.54 | 80.20±0.51 | 73.00±0.54 | 68.31±0.16 | 59.75±0.78 | 73.01±0.27 | 35.79±0.87 | $-35.41 \pm 0.90$ |
| PGP | 67.80±0.20 | 63.25±4.96 | 64.82±2.55 | 64.70±2.22 | 64.00±2.65 | 64.91±2.51 | 9.56±2.89 | $-9.56 \pm 2.89$ |
| DSI++ | 83.38±0.07 | 79.55±0.05 | 72.75±0.05 | 67.24±0.18 | 62.68±0.23 | 73.12±0.03 | 27.53±0.26 | $-27.53 \pm 0.26$ |
| CODA | **89.29**±0.16 | **89.63**±0.17 | **88.88**±0.16 | **88.11**±0.17 | **87.24**±0.22 | **88.63**±0.17 | **4.42**±0.22 | $\mathbf{-4.42 \pm 0.22}$ |
| EWC-DR | 70.76±0.12 | 64.96±0.17 | 59.98±0.53 | 54.93±0.96 | 51.89±0.26 | 60.50±0.30 | 29.44±0.39 | $-29.44 \pm 0.39$ |
| PromptDSI | 50.30±7.50 | 53.15±2.71 | 50.12±0.90 | 47.87±0.65 | 46.38±1.01 | 49.57±2.09 | 21.94±5.07 | $-21.94 \pm 5.07$ |
| SPMR (ours) | 61.93±2.91 | 70.94±2.31 | 71.85±1.92 | 71.48±1.67 | 70.72±1.53 | 69.38±1.95 | 9.17±1.07 | $-9.17 \pm 1.07$ |

Table 7: Per-task results on ToolRet (NDCG@5).

| Method | $A_0$ | $A_1$ | $A_2$ | $A_3$ | $A_4$ | AIP | FT | BWT |
|---|---|---|---|---|---|---|---|---|
| L2P | 62.85±0.25 | 56.36±0.92 | 56.36±1.19 | 55.47±1.42 | 54.31±0.75 | 57.07±0.74 | 15.82±1.32 | $-15.82 \pm 1.32$ |
| InsCL | 68.96±0.56 | 63.47±0.59 | 60.93±0.53 | 57.09±0.23 | 49.95±0.71 | 60.08±0.25 | 31.65±1.06 | $-29.41 \pm 1.05$ |
| PGP | 54.79±0.22 | 52.20±3.32 | 52.18±1.63 | 51.00±1.26 | 49.46±1.83 | 51.92±1.64 | 11.16±2.34 | $-11.16 \pm 2.34$ |
| DSI++ | 74.60±0.03 | 68.04±0.15 | 59.35±0.08 | 53.32±0.16 | 48.78±0.21 | 60.82±0.05 | 30.91±0.41 | $-30.91 \pm 0.41$ |
| CODA | **75.39**±0.21 | **77.01**±0.22 | **75.10**±0.28 | **73.06**±0.25 | **71.22**±0.25 | **74.36**±0.24 | **10.98**±0.24 | $\mathbf{-10.98 \pm 0.24}$ |
| EWC-DR | 56.34±0.09 | 52.04±0.17 | 46.66±0.19 | 41.54±0.78 | 38.80±0.35 | 47.08±0.16 | 31.07±0.43 | $-31.07 \pm 0.43$ |
| PromptDSI | 41.00±7.39 | 46.11±3.62 | 42.24±1.70 | 39.62±1.41 | 37.82±1.00 | 41.36±2.97 | 26.09±4.94 | $-26.09 \pm 4.94$ |
| SPMR (ours) | 48.02±2.31 | 58.55±1.74 | 58.60±1.42 | 57.25±1.28 | 55.97±1.12 | 55.68±1.49 | 16.78±0.52 | $-16.78 \pm 0.52$ |

## C TOOL LIFECYCLE EDITING

The prompt-indexed memory also supports local edits to the tool inventory after training. Starting from the task3 checkpoint on ToolBench, we construct a controlled post-hoc benchmark that deletes 20 tools and updates 10. A deleted tool is retargeted to a box-local abstention label `NO_TOOL_L2`$_k$, where $k$ is the tool's L2 region, and an updated tool is retargeted to a replacement `tool_v2` that inherits its L2 assignment. The edit freezes the backbone, router, and query projection, and updates only the prompt rows and classifier rows touched by these changes. Table 9 reports the effect under oracle L2 evaluation, which isolates the local edit from router errors.

Updating a tool is effective: the edited model maps old-tool queries to the replacement in 73.21% of cases and eliminates old-tool leakage entirely. Deleting is weaker but still works, reaching 47.22% safe fallback while cutting leakage from 39.81% to 14.81%, and retain accuracy drops by only

Table 8: Full path-budget sweep on ToolBench (Recall@5), with all four evaluation sets retrieved from the same final memory. "Paths" is the total number of prompt paths executed over the four sets; macro is the mean of the four columns. **Best** per column.

| Budget | Mean $p$ | Paths ($\times 10^3$) | base | task1 | task2 | task3 | macro |
|---|---|---|---|---|---|---|---|
| Fixed $p$=1 | 1.00 | 36.4 | 24.30 | 60.38 | 63.64 | 68.55 | 54.22 |
| Fixed $p$=2 | 2.00 | 72.8 | 36.09 | 66.10 | 70.31 | 74.14 | 61.66 |
| Fixed $p$=3 | 3.00 | 109.1 | 41.13 | 68.21 | 72.48 | 76.42 | 64.56 |
| Fixed $p$=4 | 4.00 | 145.5 | 43.62 | 68.24 | 72.54 | 76.79 | 65.30 |
| Fixed $p$=5 | 5.00 | 181.9 | 44.73 | 68.24 | 72.73 | **77.22** | 65.73 |
| Fixed $p$=6 | 6.00 | 218.2 | 45.49 | 68.24 | 73.04 | 76.72 | 65.87 |
| Fixed $p$=7 | 7.00 | 254.6 | 46.07 | 67.84 | 72.98 | 76.79 | 65.92 |
| Fixed $p$=8 | 8.00 | 291.0 | 46.53 | 67.87 | 72.82 | 76.29 | 65.88 |
| Fixed $p$=9 | 9.00 | 327.4 | 46.81 | 67.81 | 72.88 | 76.16 | 65.91 |
| Fixed $p$=10 | 10.00 | 363.8 | **47.10** | 67.74 | 72.67 | 76.12 | 65.91 |
| Adaptive $p$ | 6.01 | 218.7 | 45.24 | **68.71** | **73.10** | 77.12 | **66.04** |

Table 9: Post-hoc tool lifecycle editing under oracle L2 evaluation. "Leak" is the fraction of edited queries still routed to the deprecated label.

| Metric | Before | After | $\Delta$ |
|---|---|---|---|
| Update success | 0.00 | 73.21 | +73.21 |
| Update old-tool leakage | 55.36 | 0.00 | $-55.36$ |
| Delete success / safe fallback | 0.00 | 47.22 | +47.22 |
| Delete old-tool leakage | 39.81 | 14.81 | $-25.00$ |
| Retain accuracy | 72.13 | 71.41 | $-0.72$ |

0.72 points, so the edit does not materially damage unrelated tools. Under router-selected prompts the end-to-end numbers are much lower, limited by an L2 routing accuracy of about 27%, which indicates that the bottleneck for lifecycle editing is reaching the correct prompt region rather than the local edit itself.