KNOWLEDGE IS NOT ENOUGH: INJECTING RL SKILLS FOR CONTINUAL ADAPTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) face the "knowledge cutoff" challenge, where their frozen parametric memory prevents direct internalization of new information. While Supervised Fine-Tuning (SFT) is commonly used to update model knowledge, it often updates factual content without reliably improving the model's ability to use the newly incorporated information for question answering or decision-making. Reinforcement Learning (RL) is essential for acquiring reasoning skills; however, its high computational cost makes it impractical for efficient online adaptation. We empirically observe that the parameter updates induced by SFT and RL are nearly orthogonal. Based on this observation, we propose **Parametric Skill Transfer (PaST)**, a framework that supports modular skill transfer for efficient and effective knowledge adaptation. By extracting a domain-agnostic **Skill Vector** from a source domain, we can linearly inject knowledge manipulation skills into a target model after it has undergone lightweight SFT on new data. Experiments on knowledge-incorporation QA (SQuAD, LooGLE) and agentic tool-use benchmarks (ToolBench) demonstrate the effectiveness of our method. On SQuAD, PaST outperforms the state-of-the-art self-editing SFT baseline by up to 9.9 points. PaST further scales to long-context QA on LooGLE with an 8.0-point absolute accuracy gain, and improves zero-shot ToolBench success rates by +10.3 points on average with consistent gains across tool categories, indicating strong scalability and cross-domain transferability of the Skill Vector.

1 INTRODUCTION

Large Language Models (LLMs, Vaswani et al. (2017); Brown et al. (2020)) have demonstrated remarkable capabilities in **static** benchmarks, yet their utilization in real-world scenarios is constrained by the "knowledge cutoff" problem (Cheng et al., 2024)—the inherent limitation that their parametric memory remains **frozen** after pre-training, preventing them from natively internalizing new information or tools on the fly (Ouyang et al., 2022; Touvron et al., 2023). Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) attempts to mitigate this by injecting external context at inference time; however, it often struggles with long-range dependency modeling over large corpora and incurs substantial inference-time overhead due to repeated processing of retrieved contexts(Shao et al., 2023). Consequently, recent research has shifted towards *Parametric Knowledge Updating*, aiming to efficiently internalize new information directly into model weights. Techniques such as Knowledge Editing (Meng et al., 2022; Yao et al., 2023; Mao et al., 2025) and Test-Time Training (TTT) (Liu et al., 2021; Osowiechi et al., 2023; Gandelsman et al., 2022; Hong et al., 2023) have emerged as promising directions, attempting to keep the model's parametric memory synchronized with the evolving world.

However, a critical limitation of existing adaptation paradigms is the functional disconnect between *knowledge* and *skills*. Prevailing methods largely rely on Supervised Fine-Tuning (SFT) to inject new domain knowledge. Recent work (Chu et al., 2025) highlighted a fundamental distinction in optimization dynamics: SFT **memorizes**, RL **generalizes**. Supported by our experiments, SFT tends to induce surface-level memorization of the training distribution, without explicitly teaching the model how to reason over the acquired knowledge in downstream tasks. While Reinforcement Learning (RL) is essential for acquiring robust reasoning and execution skills (Guo et al., 2025), it remains a bottleneck for efficient online adaptation to novel scenarios. The high cost of collecting

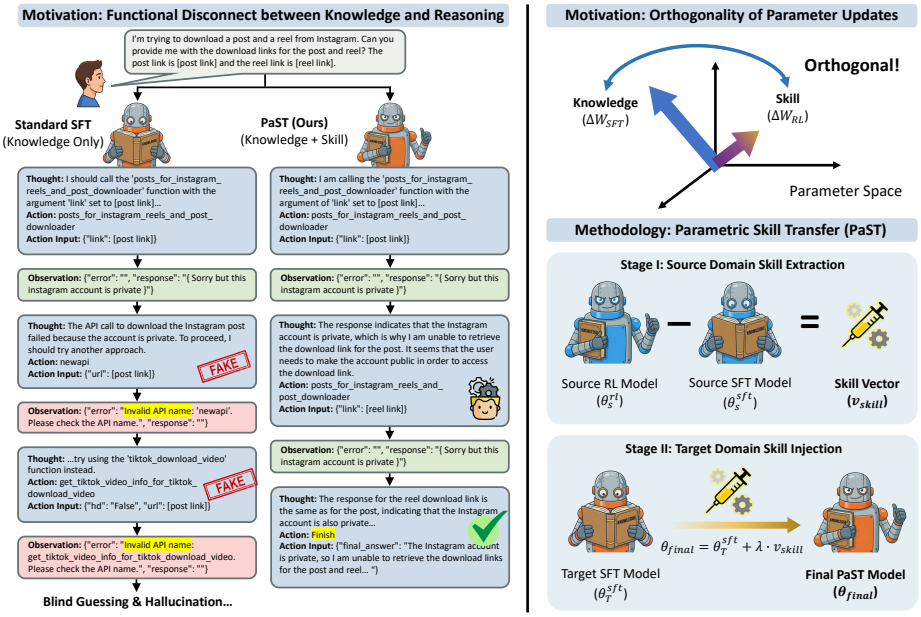

Figure 1: **Overview of Parametric Skill Transfer (PaST).** The motivation (left) illustrates how standard SFT fails to handle environmental errors, leading to hallucinations, while PaST enables robust execution by incorporating reasoning skills. Our approach is based on the empirical finding (top right) that parameter updates for knowledge (ΔW_{SFT}) and skills (ΔW_{RL}) are nearly orthogonal and reside in disentangled subspaces. PaST first extracts a domain-agnostic skill vector $v_{skill} = \theta_S^{rl} - \theta_S^{sft}$ from a source domain and then linearly injects it into a target model via $\theta_{final} = \theta_T^{sft} + \lambda \cdot v_{skill}$, enabling efficient and effective knowledge adaptation without requiring expensive reinforcement learning in the target domain.

interaction data and the computational burden of on-policy exploration make it infeasible to perform RL for every new environment the model encounters.

To bridge this gap, we propose **Parametric Skill Transfer (PaST)**, a modular framework that injects RL-optimized reasoning capabilities into models adapted to new knowledge, without explicitly performing RL on the new knowledge. Our approach is driven by the empirical observation that the parameter updates induced by SFT and RL occupy nearly orthogonal spaces. Therefore, we hypothesize that the SFT and RL updates are natively decoupled, where the RL-learned skills need not be bound to specific SFT knowledge but can be transferred to new domains. Subsequently, we introduce a mechanism to extract the **domain-agnostic** *skill vector* by subtracting the parameters of an SFT-anchored model from its RL-refined counterpart in a source domain. This vector, which captures the "gradient" direction of reasoning improvement, can then be linearly added to a target model during the adaptation phase, immediately after it has undergone lightweight SFT on new target data, which avoids the expensive RL process in the new domain. Figure 1 illustrates the idea.

We empirically evaluate PaST across two primary capability domains: Knowledge Incorporation (covering both short- and long-context scenarios via SQuAD (Rajpurkar et al., 2016) and LooGLE (Li et al., 2024)) and Closed-Book Tool Use (via ToolBench (Qin et al., 2023; Guo et al., 2024)). Extensive experiments demonstrate the efficacy of our framework. First, on the SQuAD knowledge incorporation task, PaST achieves 56.9% accuracy, surpassing the state-of-the-art self-adapting baseline SEAL (47.0%, Zweiger et al. (2025)) by a substantial margin. Second, on the LooGLE long-context benchmark, we demonstrate that our approach scales to massive documentation (over 24k tokens), enabling more precise information retrieval from parametric memory than standard SFT. Finally, in ToolBench cross-domain evaluation, PaST enables zero-shot transfer of tool-use skills to RL-unseen categories, successfully activating execution capabilities in target domains.

Our contributions are summarized as follows:

- We identify the *Reasoning-Knowledge Disconnect* in knowledge adaptation, highlighting the insufficiency of SFT for transferring procedural logic to new domains.
- We provide empirical evidence that parameter updates induced by skill learning (via RL) and knowledge acquisition (via SFT) are nearly orthogonal and reside in disentangled subspaces of the parameter landscape, enabling separate optimization and linear composition.
- We propose Parametric Skill Transfer (PaST), a novel method that utilizes task vector arithmetic to transfer RL-learned skills from a source domain to a target domain, bypassing the need for test-time RL.
- We empirically demonstrate the effectiveness of PaST across diverse tasks, including knowledge-intensive QA and agentic tool use, proving that knowledge manipulation and execution skills can be effectively decoupled and transferred to enable robust adaptation in data-scarce target domains.

2 MOTIVATION

2.1 THE DISCONNECT BETWEEN KNOWLEDGE AND REASONING

Current adaptation methods (Yehudai et al., 2024; Mao et al., 2025) predominantly rely on SFT to introduce new domain data. While SFT effectively lowers perplexity on domain documents, we hypothesize that it often fails to instill the execution logic required to manipulate that knowledge. As a result, models may "know" the facts (e.g., document content) without being able to dynamically utilize them, especially in complex settings.

To visualize this, we compare a standard Target SFT model against a Skill-Injected model (adapted using our proposed PaST framework, detailed in Section 3). We present a case study (Figure 1 (left), full execution trajectory along with additional case studies is provided in Appendix B) on a Closed-Book Tool Use task (Schick et al., 2023; Li et al., 2023), where the model must rely entirely on its parametric memory to recall API usage given only the API names. In this instance, the user requests to download an Instagram post, but the target account is private, triggering an API error. The SFT model correctly recalls the API name, but its reasoning collapses upon encountering the error, leading to the hallucination of non-existent tools. In contrast, the Skill-Injected model demonstrates robust execution logic despite sharing the same knowledge base. This observation highlights that knowledge storage (via SFT) and knowledge manipulation (via RL) are distinct capabilities. SFT alone anchors the model in the domain semantics but leaves it functionally fragile. This necessitates a method to explicitly inject robust manipulation patterns—precisely the role of our Skill Vector.

2.2 ORTHOGONALITY OF PARAMETER UPDATES

While the behavioral analysis in Section 2.1 highlights the functional difference between SFT and RL models, a fundamental question remains regarding their internal mechanics: Do knowledge acquisition and skill learning interfere with each other in the parameter space?

To answer this, we analyze the weight updates. We utilize 5 documents from the LooGLE dataset to train a model sequentially via SFT and GRPO (refer to Section 4.1.2 for settings), and then compute the layer-wise cosine similarity between the parameter update matrices induced by each stage. Figure 2 visualizes the cosine similarity across all layers and modules. We observe a consistent trend: the correlation between ΔW_{SFT} and ΔW_{RL} is remarkably close to zero across almost all depths and components (for a contrast with the significantly higher similarity between different SFT updates, see Appendix C). This orthogonality provides strong evidence that knowledge and manipulation skills correspond to disentangled subspaces within the high-dimensional parameter landscape.

To understand why this parameter-level property ensures the coexistence of skills and knowledge, we analyze the signal propagation in the activation space. As detailed in Appendix D, assuming the input activations x follow a quasi-isotropic distribution (facilitated by `LayerNorm`, Ba et al. (2016)), the expected inner product of the signals generated by these updates approximates the inner product of the weight matrices $\langle \Delta W_{\text{SFT}}, \Delta W_{\text{RL}} \rangle_F$. High-dimensional concentration of measure (Vershynin, 2018) further ensures that this overlap remains minimal for individual inputs. Given our empirical finding of near-orthogonal updates, knowledge and skill signal remain functionally disentangled, preventing destructive interference during inference and allowing downstream components to route these streams distinctively (Elhage et al., 2022).

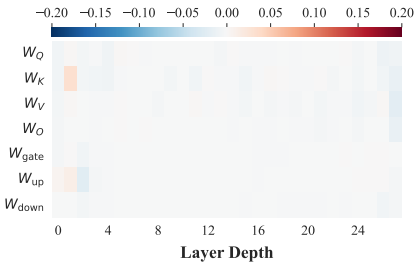

Figure 2: We visualize the layer-wise cosine similarity between the weight changes induced by SFT (ΔW_{SFT}) and RL (ΔW_{RL}) on the LooGLE task. The dominant near-zero values indicate that knowledge acquisition and skill learning modify the model parameters along nearly orthogonal subspaces.

This finding implies that the "manipulation skill" acquired during RL is separable from the domain-specific knowledge learned via SFT, existing as an independent and extractable parameter vector ΔW_{RL}. This separability motivates our approach: the skill vector can be extracted from a source domain and transferred to a target domain, enabling efficient adaptation without target-side RL.

3 Parametric Skill Transfer

Leveraging the theoretical insight of *parameter orthogonality* established in Section 2.2, we propose **Parametric Skill Transfer (PaST)**, a framework that explicitly disentangles and recombines knowledge and skills. As visualized in the right panel of Figure 1, our approach treats the skill component as a portable vector extracted from a source domain and linearly injected into a target knowledge base.

3.1 Problem Formulation

We consider a knowledge updating scenario involving a Source Domain $\mathcal{D}_S = \{\mathcal{C}_S, \mathcal{T}_S\}$ and a Target Domain $\mathcal{D}_T = \{\mathcal{C}_T\}$. Here, $\mathcal{C}$ denotes unstructured knowledge documents and $\mathcal{T} = \{(x, y)\}$ represents a set of successful interaction pairs that exemplify the desired task-solving behaviors. While $\mathcal{D}_S$ is enriched with both knowledge and behavioral demonstrations, $\mathcal{D}_T$ contains only raw documents without task-specific labels. Our goal is to derive a target policy π_{target} that maximizes performance on $\mathcal{D}_T$ by leveraging the skills from $\mathcal{T}_S$ and the knowledge in $\mathcal{C}_T$, eliminating the need for expensive on-policy exploration in the target domain.

3.2 Methodology: Decoupled Skill Transfer

Stage I: Source Skill Distillation. We first anchor the base model θ_{base} to the source knowledge by fine-tuning on a corpus $\mathcal{C}_S$, yielding θ_S^{sft}. Subsequently, we apply Reinforcement Learning on trajectories $\mathcal{T}_S$ to internalize reasoning policies, resulting in θ_S^{rl}. Leveraging our finding that RL updates occupy a subspace orthogonal to the knowledge manifold (Sec. 2.2), we isolate the procedural expertise by extracting the **Skill Vector**: $\mathbf{v}_{\text{skill}} = \theta_S^{\text{rl}} - \theta_S^{\text{sft}}$. This subtraction neutralizes domain-specific declarative patterns while retaining the sparse parameter residuals responsible for internal knowledge manipulation capabilities.

Stage II: Target Adaptation via Vector Composition. To adapt to the target domain without expensive on-policy RL, we adopt a "Compose-and-Go" strategy. We first perform lightweight SFT on target-specific documents $\mathcal{C}_T$ to obtain θ_T^{sft}. While this model captures target facts, it lacks the necessary reasoning logic. We then inject the source-distilled skills directly into the target parameters as $\theta_{\text{final}} = \theta_T^{\text{sft}} + \lambda \cdot \mathbf{v}_{\text{skill}}$, where λ is a scaling coefficient (set to 1 in all experiments for simplicity). This linear composition grafts the source-learned reasoning geometry onto the target knowledge manifold, enabling zero-shot execution of complex tasks in the new domains.

3.3 Iterative Skill Refinement

A potential risk in single-round extraction is that the skill vector might overfit to the specific content distribution of the sampled source data, rather than capturing purely domain-agnostic reasoning patterns. To mitigate this, we propose an Iterative Bootstrapping Strategy. We partition $\mathcal{D}_S$ into K disjoint subsets $\{\mathcal{D}_S^{(k)}\}_{k=1}^K$ and refine the vector iteratively. For each round k, we first obtain $\theta_{S,k}^{\text{sft}}$ via SFT on the current subset, then initialize the model for reinforcement learning as $\theta_{\text{init},k} = \theta_{S,k}^{\text{sft}} + \mathbf{v}_{k-1}$ (where $\mathbf{v}_{k-1}$ is the skill vector from the last round, and $\mathbf{v}_0 = \mathbf{0}$). The purpose is to inject the previously extracted skills as a warm-start for the RL in this round. Subsequent RL training on $\mathcal{D}_S^{(k)}$ yields $\theta_{S,k}^{\text{rl}}$, from which we extract the updated skill vector as the residual $\mathbf{v}_k = \theta_{S,k}^{\text{rl}} - \theta_{S,k}^{\text{sft}}$. This iterative process ensures that the skill vector is progressively refined across diverse knowledge contexts, forcing the optimization to converge towards content-invariant. We empirically validate the benefit of this strategy in Section 4.3.1.

4 Experiments

We empirically evaluate PaST across two distinct tasks: **Knowledge-based QA** (Roberts et al., 2020) and **Agentic Tool Use** (Li et al., 2023). Our experiments are designed to verify: (1) **Effectiveness** on standard benchmarks (SQuAD, Rajpurkar et al. (2016)) against strong baselines like SEAL (Zweiger et al., 2025); (2) **Scalability** to complex, long-context scenarios (LooGLE, Li et al. (2024)); and (3) **Generalization** to RL-unseen tool categories (ToolBench, Qin et al. (2023); Guo et al. (2024)). Section 4.1 details the QA results, while Section 4.2 presents the cross-domain tool-use evaluation. Finally, in Section 4.3, we conduct ablation studies to analyze the impact of our **iterative skill refinement strategy** and validate the architectural necessity of our post-hoc injection method by comparing it against alternative transfer paradigms.

4.1 Knowledge-based Question Answering

4.1.1 Knowledge Incorporation on SQuAD

To investigate the effectiveness of our proposed framework, we benchmark PaST on the task of Knowledge Incorporation using the SQuAD dataset. Unlike the original SQuAD evaluation where the passage is provided alongside the question, we adopt the closed-book setting from SEAL (Zweiger et al., 2025). This task requires the model to first memorize the specific passage through test-time weight updates and then answer downstream questions by retrieving facts directly from its new parameter state, rather than reading them from the context window.

We utilize `Qwen2.5-7B` (Qwen et al., 2025) as our base model and compare against a comprehensive set of baselines shown originally in SEAL: (1) Base Model (zero-shot); (2) Passage-Only SFT (standard fine-tuning); (3) SFT with Synthetic Data (augmenting passages with model-generated implications); (4) SFT with GPT-4.1 Data; and (5) SEAL, the current state-of-the-art method that integrates knowledge via *self-edits* generated by meta-trained models. We evaluate performance under three different regimes: Single Passage updating, Continued Pretraining (CPT) on $n = 200$ documents, and large-scale CPT on the full validation set ($n = 2067$).

Following the Iterative Skill Refinement strategy in Section 3.3, we conduct two rounds of training (using the same training data as SEAL) and extract the skill vector via the parameter residual between the final RL-tuned model and its SFT counterpart. We use the same SFT data generation paradigm as the "Train on Passage + Synthetic" baseline for both Source Skill Distillation and Target Adaptation; it serves as our direct baseline to test the skill vector's contribution. The RL phase utilizes GRPO (Shao et al., 2024) with GPT-4.1 as the reward evaluator. Detailed training configurations are provided in Appendix E.

As shown in Table 1, standard SFT on passages (33.5%) confirms that raw text training is insufficient for knowledge retention. By injecting our skill vector onto the "Train on Passage + Synthetic" baseline (39.7%), **performance surges to 56.9%**. This **+17.2%** absolute improvement demonstrates the contribution of the skill vector, proving that while synthetic data provides the factual content, the skill vector provides the essential inference logic for answering questions accurately. Notably, **PaST significantly outperforms both SEAL (47.0%) and GPT-4.1 (46.3%)**. While

Table 1: Mean accuracy on SQuAD (no-context) across different adaptation regimes. Values for baselines are taken from SEAL (Zweiger et al., 2025). The values in parentheses denote the absolute improvement over the "Train on Passage + Synthetic" baseline.

Method	**Single Passage** (n = 1; LoRA)	**CPT** (n = 200; full-FT)	**CPT** (n = 2067; full-FT)
Base Model	32.7	32.7	29.0
Train on Passage	33.5	36.0	31.2
Train on Passage + Synthetic	39.7	50.6	43.4
Train on Passage + GPT-4.1 Synthetic	46.3	**59.4**	**49.2**
SEAL	47.0	58.2	46.4
PaST 50 (Ours)	50.8 (+11.1)	58.9 (+8.3)	47.4 (+4.0)
PaST 50 × 2 (Ours)	**56.9** (+17.2)	58.7 (+8.1)	**49.2** (+5.8)

SEAL focuses on synthesizing higher-quality training data (e.g., self-edits or implications) through costly meta-training, our method achieves superior results by directly transferring intrinsic procedural skills. This suggests that the bottleneck in knowledge incorporation may not be the quality of the SFT data itself, but rather the model's underlying ability to utilize its incorporated knowledge. This trend holds in Continued Pretraining settings ($n = 200/2067$), where PaST consistently achieves superior performance over SEAL. These results demonstrate that our skill-centric transfer paradigm is robust and scalable; it does not degrade when the underlying knowledge base expands from a single document to hundreds or thousands.

4.1.2 Scalability to Long-Context Reasoning

While SQuAD validates the efficacy of our method on standard-length paragraphs, real-world adaptation often requires processing extensive documentation where reasoning is complicated by the sheer volume of information. To evaluate the scalability of our framework, we conduct experiments on LooGLE, a benchmark designed for long-context understanding, consisting of realistic documents with an average length exceeding 21k tokens.

Using `Qwen2.5-7B-Instruct` as the base model, we construct a source set using the last 10 documents from the LooGLE Short Dependency QA dataset, while reserving the first 50 documents exclusively for evaluation. Using the source set, we perform 2 rounds of iterative skill acquisition. In each round, we sample a batch of 5 documents and apply a specialized two-stage SFT curriculum to enforce deep knowledge encoding: (1) context memorization via multi-task training (text modeling, expansion and compression) and (2) synthetic QA training. This is followed by GRPO to distill the retrieval logic. Full details are in Appendix F.

As shown in Table 2, applying the skill vector yields a significant improvement over standard Target SFT baseline which is trained using the same two-stage curriculum. Injecting the skill vector extracted from just 5 source documents (Round 1) immediately boosts accuracy to **35.0% (+4.9%)**, while **the Round 2 elevates performance to 38.1%, resulting in a cumulative gain of +8.0%**. These results confirm that our "how-to-recall" skill is highly transferable and successfully mitigates hallucination by transforming the model from a passive container of facts into a focused expert capable of precise parametric retrieval.

Table 2: **Long-Context QA Performance on LooGLE (Short Dependence QA).** Comparing standard adaptation methods against PaST, the Skill Vector significantly enhances the model's ability to retrieve and reason over extremely massive information. All results are averaged over three independent runs.

Method	**Accuracy**
SFT	30.1
PaST 5	35.0 (+4.9)
PaST 5 × 2	**38.1** (+8.0)

4.2 Cross-Domain Generalization in Tool Use

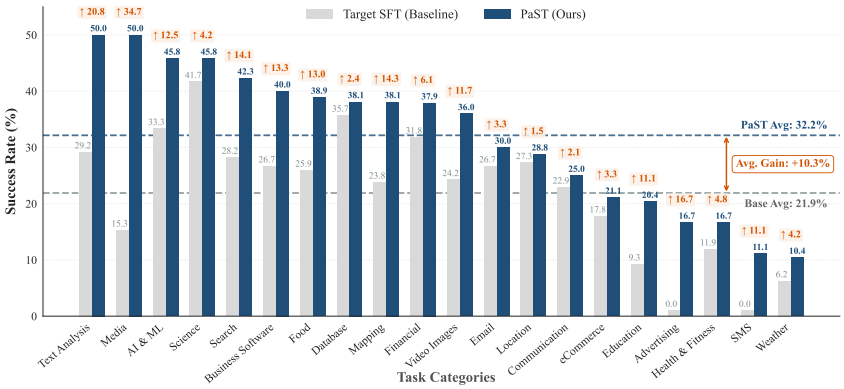

Figure 3: **Zero-shot cross-domain generalization on StableToolBench.** Success Rate across 20 RL-unseen target categories using a skill vector trained solely on *Movies*. PaST (dark blue) raises the average success rate by **+10.3%** over the Target SFT baseline (grey). All results are averaged over three independent runs.

We extend our evaluation to the domain of Agentic Tool Use, a task requiring the model to precisely index and utilize internalized API schemas. Unlike QA, this necessitates a mechanism of accurate parametric retrieval and multi-round execution, which we hypothesize is domain-agnostic and transferable to other tool categories.

Task Definition: Closed-Book Execution. Standard tool-use evaluations often provide full API documentation within the context window (Li et al., 2023). However, for massive libraries with thousands of APIs, retrieving and injecting complete schemas into the context is computationally prohibitive and introduces high inference latency and token costs. To simulate this realistic constraint, we adopt a **Closed-Book Execution** setting: the model is provided only with the names of the APIs, devoid of their detailed parameter definitions or descriptions.

Dataset Construction and Split. We utilize ToolBench, featuring 3,451 tools and 16,000+ APIs spanning 50 distinct categories. We designate a single representative category, *Movies*, as the Source Domain for skill acquisition due to its representative complexity and data richness. For the Target Domain, we identify 20 distinct categories entirely unseen during RL, filtered for data sufficiency and API count to ensure a balanced evaluation (see Appendix G for details). For testing, we utilize the curated solvable queries from StableToolBench, covering both single-tool (G1) and intra-category multi-tool (G2) scenarios.

Training Setup. Using `Qwen2.5-7B-Instruct`, we first establish a robust mapping between API names and functionalities on source domain via SFT on a composite dataset: raw API schemas, natural language transcriptions, and bidirectional QA pairs (training the model to predict usage from API names and vice versa). Additionally, we perform a format-alignment SFT using the initial steps of the trajectories in ToolBench data to instill the ReAct convention. To refine the execution policy, we employ PPO (Schulman et al., 2017) implemented by adapting the Search-R1 (Jin et al., 2025) framework. During training, we employ GPT-4o-mini as an environment simulator to generate realistic API return values. The reward signal is a composite of format rewards (JSON syntax and ReAct format), execution rewards (successful API returns), and solution rewards (judged by GPT-4.1 for intent resolution). Details are provided in Appendix G.

Target Adaptation and Results. Following the two-stage adaptation (SFT as described above, followed by skill vector injection), PaST significantly outperforms the Target SFT baseline. As visualized in Figure 3, **our method increases the average success rate from 21.9% to 32.2%.** Notably, PaST achieves zero-shot activation in domains where the baseline fails completely (*Advertising* 0% $\rightarrow$ 16.7% and *SMS* 0% $\rightarrow$ 11.1%). Remarkably, PaST outperforms the baseline in all 20 evaluated categories, demonstrating robust positive transfer across diverse domains.

4.3 Ablation Studies

4.3.1 Impact of Iterative Skill Refinement

We first validate the effectiveness of the **Iterative Skill Refinement** by comparing it against Single-Round baselines on SQuAD and LooGLE, maintaining equal total optimization steps and data volume. As shown in Table 3, simply doubling the source data in a single round (e.g., $N = 100$ or 10) often yields marginal gains or even performance degradation, suggesting that reasoning logic becomes overfitted to specific source content. In contrast, our Iterative strategy consistently achieves the highest accuracy. This confirms that iterative refinement forces the skill vector to capture content-invariant execution logic, preventing the reasoning policy from being inextricably bound to a specific set of source facts.

Table 3: **Ablation on Iterative Refinement.** We compare Single-Round training (with half and full data) against our Iterative strategy on SQuAD and LooGLE. $N = (\text{SQuAD}/\text{LooGLE})$ denotes the number of *source training documents* in each round.

Training Rounds	SQuAD			LooGLE
	Single	CPT (200)	CPT (2K)	
1 ($N = 50/5$)	50.8	**58.9**	47.4	41.7
1 ($N = 100/10$)	49.9	58.3	47.1	42.9
2 ($N = 50/5$)	**56.9**	58.7	**49.2**	**44.6**

4.3.2 Impact of Transfer Strategy

Finally, we analyze the optimal stage for skill injection by comparing our Post-hoc Composition against two alternative paradigms on the LooGLE benchmark: (1) **Sequential Fine-Tuning**, where the source RL model θ_S^{rl} is directly fine-tuned on target documents; and (2) **Pre-Injection**, where $\mathbf{v}_{\text{skill}}$ is added to θ_{base} before target SFT. Following the setup in Section 4.1.2, we report the results on the first 10 documents of the test set in Table 4. Interestingly, Sequential Fine-Tuning (30.3) performs slightly worse than the standard Target SFT baseline (32.9). This suggests that directly optimizing for new knowledge on top of RL parameters may induce optimization conflicts that potentially disrupt the delicate reasoning circuitry learned during RL. Pre-Injection achieves moderate performance (36.5) but lags behind our method, likely because subsequent SFT shifts the weight manifold and misaligns the pre-injected skills. Post-hoc Composition (Ours) yields the highest accuracy (44.6) by anchoring the declarative knowledge first, ensuring execution logic is grafted onto a stable knowledge representation without being distorted by the SFT optimization trajectory.

Table 4: **Ablation on Transfer Strategy.** We evaluate different methods of combining source skills with target knowledge on LooGLE. "Post-hoc Composition" (Ours) significantly outperforms sequential training or pre-injection methods.

Transfer Strategy	LooGLE Accuracy
Target SFT	32.9
Sequential FT	30.3
Pre-Injection	36.5
Post-hoc Composition	**44.6**

5 Conclusion

In this paper, we introduced Parametric Skill Transfer (PaST) to bridge the functional disconnect between knowledge acquisition and reasoning skills in LLMs. By identifying that SFT and RL parameter updates are nearly orthogonal, we developed a modular framework to extract a domain-agnostic Skill Vector (v_{skill}) from source tasks and linearly inject it into models adapted to new data. Our evaluations on SQUAD, LooGLE, and ToolBench demonstrate that PaST significantly enhances a model's ability to manipulate newly internalized knowledge. Ultimately, PaST offers a computationally efficient and scalable alternative to on-policy RL, enabling effective knowledge adaptation.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Sheng Cao, Mingrui Wu, Karthik Prasad, Yuandong Tian, and Zechun Liu. Param delta for direct mixing: Post-train large language model at zero cost. In *The Thirteenth International Conference on Learning Representations*.

Daixuan Cheng, Shaohan Huang, and Furu Wei. Adapting large language models to domains via reading comprehension. *arXiv preprint arXiv:2309.09530*, 2023.

Jeffrey Cheng, Marc Marone, Orion Weller, Dawn Lawrie, Daniel Khashabi, and Benjamin Van Durme. Dated data: Tracing knowledge cutoffs in large language models. *arXiv preprint arXiv:2403.12958*, 2024.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. *arXiv preprint arXiv:2501.17161*, 2025.

Guodong Du, Xuanning Zhou, Junlin Li, Zhuo Li, Zesheng Shi, Wanyu Lin, Ho-Kin Tang, Xiucheng Li, Fangming Liu, Wenya Wang, et al. Knowledge grafting of large language models. *arXiv preprint arXiv:2505.18502*, 2025.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

Yossi Gandelsman, Yu Sun, Xinlei Chen, and Alexei Efros. Test-time training with masked autoencoders. *Advances in Neural Information Processing Systems*, 35:29374–29385, 2022.

Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. *arXiv preprint arXiv:2403.07714*, 2024.

Junyuan Hong, Lingjuan Lyu, Jiayu Zhou, and Michael Spranger. Mecta: Memory-economic continual test-time model adaptation. In *2023 International Conference on Learning Representations*, 2023.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Andrew K Lampinen, Arslan Chaudhry, Stephanie CY Chan, Cody Wild, Diane Wan, Alex Ku, Jörg Bornschein, Razvan Pascanu, Murray Shanahan, and James L McClelland. On the generalization of language models from in-context learning and finetuning: a controlled study. *arXiv preprint arXiv:2505.00661*, 2025.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Jiaqi Li, Mengmeng Wang, Zilong Zheng, and Muhan Zhang. Loogle: Can long-context language models understand long contexts? In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16304–16333, 2024.

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. *arXiv preprint arXiv:2304.08244*, 2023.

Yuejiang Liu, Parth Kothari, Bastien Van Delft, Baptiste Bellot-Gurlet, Taylor Mordan, and Alexandre Alahi. Ttt++: When does self-supervised test-time training fail or thrive? *Advances in Neural Information Processing Systems*, 34:21808–21820, 2021.

Yansheng Mao, Yufei Xu, Jiaqi Li, Fanxu Meng, Haotong Yang, Zilong Zheng, Xiyuan Wang, and Muhan Zhang. Lift: Improving long context understanding of large language models through long input fine-tuning. *arXiv preprint arXiv:2502.14644*, 2025.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372, 2022.

Kevin Meng, Arnab Sen Sharma, Alex J Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=MkbcAHIYgyS.

Sagnik Mukherjee, Lifan Yuan, Dilek Hakkani-Tür, and Hao Peng. Reinforcement learning finetunes small subnetworks in large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=0NdS4xCngO.

David Osowiechi, Gustavo A Vargas Hakim, Mehrdad Noori, Milad Cheraghalikhani, Ismail Ben Ayed, and Christian Desrosiers. Tttflow: Unsupervised test-time training with normalizing flow. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2126–2134, 2023.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, 2022.

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

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*, 2023.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? *arXiv preprint arXiv:2002.08910*, 2020.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. *arXiv preprint arXiv:2305.15294*, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Idan Shenfeld, Jyothish Pari, and Pulkit Agrawal. Rl's razor: Why online reinforcement learning forgets less. *arXiv preprint arXiv:2509.04259*, 2025.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Roman Vershynin. *High-dimensional probability: An introduction with applications in data science*, volume 47. Cambridge university press, 2018.

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, et al. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning. *arXiv preprint arXiv:2505.16421*, 2025.

Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. Editing large language models: Problems, methods, and opportunities. *arXiv preprint arXiv:2305.13172*, 2023.

Asaf Yehudai, Boaz Carmeli, Yosi Mass, Ofir Arviv, Nathaniel Mills, Eyal Shnarch, and Leshem Choshen. Achieving human parity in content-grounded datasets generation. In *International Conference on Learning Representations*, 2024.

Mohammad Zbeeb, Hasan Abed Al Kader Hammoud, and Bernard Ghanem. Reasoning vectors: Transferring chain-of-thought capabilities via task arithmetic. *arXiv preprint arXiv:2509.01363*, 2025.

Adam Zweiger, Jyothish Pari, Han Guo, Ekin Akyürek, Yoon Kim, and Pulkit Agrawal. Self-adapting language models. *arXiv preprint arXiv:2506.10943*, 2025.

A Related Work

A.1 Knowledge Updating

Many works aim to inject new knowledge into pre-trained Large Language Models (LLMs) by directly updating their parameters. Some works (Meng et al., 2022; 2023) seek to precisely locate and modify specific neurons or weight matrices responsible for storing entity relationships. Others focus on text-based adaptation, where meaningful implications or synthetic Question-Answer (QA) pairs are generated from new documents to fine-tune the models (Yehudai et al., 2024; Lampinen et al., 2025; Mao et al., 2025). SEAL (Zweiger et al., 2025) advanced this direction by optimizing the generation of self-editing data through meta-training. Our work generally follows the latter paradigm, while advancing previous methods with a critical insight: beyond merely improving fine-tuning data, enabling the model to effectively utilize the injected knowledge is more important.

A.2 Reinforcement Learning for LLMs

Reinforcement learning (RL) is a critical post-training paradigm for LLMs. Recent advances, such as RL with verifiable rewards, have demonstrated the ability to elicit reasoning behaviors: DeepSeekMath introduces GRPO and improves mathematical reasoning (Shao et al., 2024), and DeepSeek-R1 further demonstrates that large-scale RL can yield strong reasoning capability with only limited cold-start (Guo et al., 2025). Beyond single-turn reasoning, end-to-end RL is increasingly explored for training agentic LLMs that must plan over multi-turn interactions in external environments, including web search (Wei et al., 2025) and tool-use agents (Qian et al., 2025). Recent analyses also suggest that RL induces favorable update dynamics—e.g., parameter updates concentrate in relatively small subnetworks (Mukherjee et al., 2025), generalize better than SFT under distribution shift (Chu et al., 2025), and exhibit reduced catastrophic forgetting (Shenfeld et al., 2025). These properties make RL a natural candidate for injecting reusable procedural skills; however, the need for on-policy rollouts makes RL expensive to rerun for each knowledge update. We aim to reconcile this conflict: keeping RL's benefits without sacrificing the efficiency required for continual adaptation.

A.3 Task Vectors

Recent studies on task arithmetic view fine-tuning updates as vectors in weight space that can be composed to transfer capabilities. Concretely, a task vector is the parameter delta between a fine-tuned model and its base model, and can be added or subtracted to steer model behavior (Ilharco et al., 2022). Building on this idea, some works (Du et al., 2025; Cao et al.) treat such deltas as modular "patches" to transplant instruction-following or other reusable skills across compatible checkpoints without full fine-tuning. Zbeeb et al. (2025) propose *Reasoning Vectors*, extracted as the residual between **parallel** SFT and RL branches, and show that injecting this residual improves chain-of-thought capabilities. However, while their parallel extraction focuses on enhancing static base models, we address the specific challenge in *knowledge adaptation*. As Cheng et al. (2023) observe, training directly on raw domain corpora effectively injects factual knowledge but often impairs the model's prompting ability for question answering. Our work shows that composing RL-derived skill vectors with test-time SFT updates strengthens the model's ability to use newly incorporated knowledge for question answering.

B Detailed Case Studies

B.1 Full Execution Trajectory: Instagram Post Task

In this section, we provide the complete interaction trace for the example discussed in Section 4.2. This comparison highlights the difference in reasoning logic when facing a "Private Account" error. The full interaction trajectories on both models are shown in Table 5 and 6.

B.2 Parametric Knowledge Retrieval: SQuAD Case Study

We present a comparison on the SQuAD dataset in Table 7, involving a specific legal context regarding EU Directives. The SFT baseline suffers from knowledge fallback, ignoring the internalized document and providing a generic answer about legal liability based on its pre-training priors. In contrast, our model demonstrates precise parametric retrieval, successfully locating the specific term "Directives" within its weights and accurately synthesizing the supporting explanation (e.g., lack of "horizontal direct effect") as presented in the hidden context.

C Additional Visualization: Orthogonality Control Experiment

In Section 2, we argued that the parameter updates for knowledge acquisition (ΔW_{SFT}) and skill learning (ΔW_{RL}) are structurally disentangled, evidenced by their near-zero cosine similarity. A potential counter-argument is that in high-dimensional parameter spaces (e.g., $d_{\text{model}} \gg 1$), random vectors naturally tend to be orthogonal. To rule out the possibility that our observation is merely a statistical artifact of high dimensionality, we conducted a control experiment to measure the similarity between two updates of the *same* modality (i.e., Knowledge vs. Knowledge).

Experimental Setup. Using the same LooGLE dataset setting as the main experiment, we performed two consecutive rounds of Supervised Fine-Tuning (SFT) on disjoint data subsets. We then computed the layer-wise cosine similarity $\text{Sim}(\Delta W_{\text{SFT1}}, \Delta W_{\text{SFT2}}) = \frac{\langle \Delta W_{\text{SFT1}}, \Delta W_{\text{SFT2}} \rangle_F}{||\Delta W_{\text{SFT1}}||_F \cdot ||\Delta W_{\text{SFT2}}||_F}$ where $\langle A, B \rangle_F = \text{Tr}(AB^\top) = \sum_{i,j} A_{ij} B_{ij}$ denotes the Frobenius inner product (Golub & Van Loan, 2013).

Result Analysis. Figure 4 presents the resulting heatmap. In stark contrast to the SFT-RL comparison (Figure 2 in the main text), the SFT-SFT heatmap exhibits a distinct positive correlation (indicated by the prevalent orange/red hues) across most layers. This comparison provides two critical insights:

1. **Manifold Alignment of Knowledge:** Tasks of the same nature (injecting declarative facts) tend to modify the model parameters along a shared or aligned subspace, resulting in non-zero cosine similarity.
2. **Validation of Disentanglement:** The fact that ΔW_{SFT} vs. ΔW_{SFT} shows correlation while ΔW_{RL} vs. ΔW_{SFT} does not confirms that the orthogonality observed in our main result is a genuine property of the Knowledge-Skill decomposition, rather than a geometric triviality.

D Theoretical Proof of Functional Disentanglement

In Section 2.2, we empirically observed that the parameter update matrices for knowledge acquisition (ΔW_{SFT}) and skill learning (ΔW_{RL}) are nearly orthogonal in terms of the Frobenius inner product. Here, we provide a formal derivation showing why this parameter-level orthogonality guarantees functional disentanglement in the activation space.

D.1 Preliminaries and Assumptions

Let $x \in \mathbb{R}^{1 \times d}$ be the input activation vector at a given layer, where d is the model dimension (e.g., 4096). Let $A = \Delta W_{\text{SFT}} \in \mathbb{R}^{d \times d}$ and $B = \Delta W_{\text{RL}} \in \mathbb{R}^{d \times d}$ be the weight update matrices. The signals generated by these updates are $u = xA$ and $v = xB$, respectively.

We make two standard assumptions regarding the statistical properties of deep neural networks:

1. **Parameter Orthogonality:** Based on our empirical observations, we assume $\langle A, B \rangle_F = \text{Tr}(AB^\top) \approx 0$.
2. **Isotropic Inputs:** We assume the input activations x are zero-centered and quasi-isotropic, with covariance proportional to the identity matrix. This is a common property in Transformers facilitated by `LayerNorm` (Ba et al., 2016):

$$\mathbb{E}[x^\top x] = \sigma^2 I \tag{1}$$

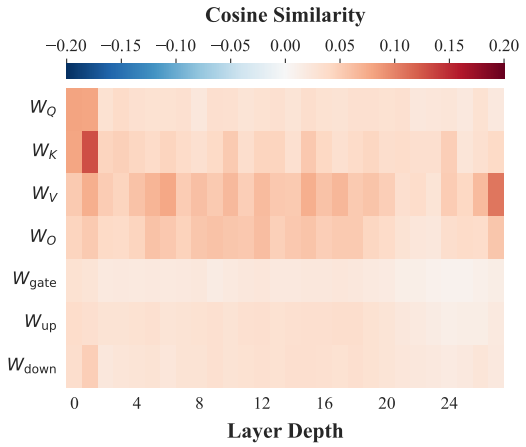

Figure 4: **Control Experiment: Similarity between two SFT updates.** We visualize the cosine similarity between parameter updates induced by two different rounds of SFT (ΔW_{SFT21} vs. ΔW_{SFT2}) on LooGLE. Unlike the SFT-RL comparison, these updates show a clear positive correlation (red regions), indicating that knowledge injection tasks operate within a shared parameter subspace.

where σ^2 is the variance of the activations.

D.2 Derivation of Signal Orthogonality

We investigate the interference between the knowledge signal (u) and the skill signal (v) by examining their inner product $\langle u, v \rangle$.

1. Expectation of Signal Overlap. The expected inner product of the generated signals over the data distribution is:

$$\begin{aligned}\mathbb{E}[\langle u, v \rangle] = \mathbb{E}[uv^\top] &= \mathbb{E}[(xA)(xB)^\top] \\ &= \mathbb{E}[xAB^\top x^\top] \end{aligned} \tag{2}$$

Using the property that the trace of a scalar is the scalar itself ($\text{Tr}(c) = c$) and the cyclic property of the trace ($\text{Tr}(XYZ) = \text{Tr}(YZX)$):

$$\begin{aligned}\mathbb{E}[xAB^\top x^\top] &= \mathbb{E}[\text{Tr}(xAB^\top x^\top)] \\ &= \mathbb{E}[\text{Tr}(AB^\top x^\top x)] \\ &= \text{Tr}(AB^\top \mathbb{E}[x^\top x]) \end{aligned} \tag{3}$$

Substituting the isotropic assumption $\mathbb{E}[x^\top x] = \sigma^2 I$:

$$\mathbb{E}[\langle u, v \rangle] = \text{Tr}(AB^\top \cdot \sigma^2 I) = \sigma^2 \langle A, B \rangle_F \tag{4}$$

Conclusion 1: Since $\langle A, B \rangle_F \approx 0$, the expected overlap between the knowledge and skill signals is zero ($\mathbb{E}[\langle u, v \rangle] \approx 0$).

2. Concentration in High Dimensions. While the expectation is zero, we must ensure that the variance is low enough such that the overlap is minimal for *any individual input* x. This is guaranteed by the **Concentration of Measure** phenomenon in high-dimensional spaces (Vershynin, 2018).

Let $M = AB^\top$. Consider the quadratic form $Y = xMx^\top$. For a random vector x with independent sub-Gaussian components (a reasonable approximation for normalized representations), the Hanson-Wright inequality bounds the deviation from the mean:

$$\begin{aligned}&P(|\langle u, v \rangle - \mathbb{E}[\langle u, v \rangle]| \geq t) \\ &\leq 2 \exp\left(-c \min\left(\frac{t^2}{\sigma^4 ||M||_F^2}, \frac{t}{\sigma^2 ||M||_2}\right)\right) \end{aligned} \tag{5}$$

where c is a universal constant. In modern LLMs where $d \gg 1$, the Frobenius norm $||M||_F$ grows with $\sqrt{d}$, but the concentration probability improves exponentially. This implies that for the vast majority of input samples, the actual inner product $\langle u, v \rangle$ will be tightly concentrated around its expectation (zero).

D.3 Functional Implication

This derivation proves that parameter-level orthogonality ($\Delta W_{\text{SFT}} \perp \Delta W_{\text{RL}}$) translates directly to signal-level orthogonality in the activation space. Consequently, the "knowledge signal" and "skill signal" propagate through the network as functionally independent components, preventing destructive interference and enabling the subsequent layers (e.g., attention heads) to attend to them distinctively.

E SQuAD Experiments

Task definition. We follow the closed-book SQuAD knowledge incorporation paradigm of SEAL (Zweiger et al., 2025). For each SQuAD context (document) d, the model first performs test-time weight updates on d (knowledge incorporation), and is then evaluated on its associated question set $\{q\}$ without providing the context in the prompt. We report the mean answer correctness rate judged by GPT-4.1 as the evaluation metric.

E.1 PaST Training Pipeline on Closed-Book SQuAD

Data used for skill distillation. To distill a domain-specific *procedural* skill for parametric knowledge retrieval, we construct a source corpus $\mathcal{D}^{\text{src}}$ consisting of $K = 2$ rounds of SQuAD contexts, each with $N = 50$ documents, matching the data budget used in SEAL. We denote the k-th round documents as $\mathcal{D}_k^{\text{src}}$. All rounds use the same base model θ_{base} (Qwen2.5-7B) as the SFT initialization.

Two-round iterative refinement (PaST-50×2). Algorithm E.1 summarizes the pipeline. Each round performs: (i) knowledge injection via SFT on the documents, then (ii) skill acquisition via GRPO on the corresponding closed-book QA pairs. Crucially, the RL-induced parameter residual (skill vector) from the previous round is injected into the next round *after* SFT and *before* RL, encouraging the learned skill to be content-invariant rather than overfitting to a single batch.

Notation (SQuAD)

- θ_{base}: base model parameters.
- $\mathcal{D}_k^{\text{src}}$: source documents in round k (each document is a SQuAD context).
- $\mathcal{Q}(\mathcal{D})$: the closed-book QA pairs attached to documents in $\mathcal{D}$.
- θ_k^{sft}: SFT model trained on $\mathcal{D}_k^{\text{src}}$ (always initialized from θ_{base}).
- v_k: skill vector after round k (parameter residual capturing RL-induced procedural skill).

[t] PaST iterative skill distillation on SQuAD (PaST-50×2).

1. Initialize skill vector $v_0 \leftarrow 0$.
2. For each round $k = 1, \ldots, K$ (here $K = 2$):
   (a) **(Knowledge injection / SFT)** Train $\theta_k^{\text{sft}} \leftarrow \text{SFT}(\theta_{\text{base}}, \mathcal{D}_k^{\text{src}})$.
   (b) **(Skill carryover)** Initialize RL policy $\theta_k^{\text{init}} \leftarrow \theta_k^{\text{sft}} + v_{k-1}$.
   (c) **(Skill acquisition / RL)** Train $\theta_k^{\text{rl}} \leftarrow \text{GRPO}(\theta_k^{\text{init}}, \mathcal{Q}(\mathcal{D}_k^{\text{src}}))$, where rewards are computed by GPT-4.1 judging answer correctness (Appendix E.4).
   (d) **(Skill extraction)** Update $v_k \leftarrow \theta_k^{\text{rl}} - \theta_k^{\text{sft}}$.
3. Output final skill vector $v_\star \leftarrow v_K$.

E.2 SFT Hyperparameters (Following SEAL)

Synthetic data generation and packing. We generate implications for each passage and train on *passage + implications*. In the single-passage regime, we split the generated implication text by newlines into multiple training sequences; in the multi-passage regime, we keep the full generation as one training document. We sample $K = 5$ synthetic generations per passage when forming the CPT training corpus.

Hyperparameters. We do not perform hyperparameter tuning on SQuAD. For all SFT-based knowledge incorporation regimes, we directly adopt the best-performing hyperparameter configurations reported in SEAL. Table 8 summarizes the settings used in our implementation.

Compute and hardware. All experiments are run on NVIDIA A100 80GB GPUs. For **single-passage** knowledge incorporation with LoRA, we use **two** A100 80GB GPUs: one GPU hosts a `vLLM` inference server for fast generation, while the other GPU performs the inner-loop LoRA updates. For **CPT** settings (n=200/2067), we run full fine-tuning on a single A100 80GB GPU.

E.3 GRPO (RL) Hyperparameters (Our Implementation)

We implement GRPO training using `verl` with a custom reward function that queries an LLM judge to score answer correctness. The effective hyperparameters are listed in Table 9

E.4 Prompt Templates

E.4.1 Implication generation prompt (for SFT data)

Implications prompt

```
Let's read the following passage and produce a
list of implications derived directly or
indirectly from the content.
Passage:
{passage}
Implications:
```

E.4.2 Closed-book QA prompt (actor rollout / evaluation)

Closed-book QA prompt

```
Let's answer a question directly and concisely.
Question: {question}
Answer:
```

E.4.3 LLM-judge reward prompt (binary correctness)

Judge prompt (returns yes/no)

```
You are a grading assistant. Your job is to
determine whether a student's answer correctly
answers the question based solely on the
provided gold answer. Do not use any outside
knowledge. The student answer can include
additional information, but it must at least
fully convey the gold answer and must not
contradict it. Ignore style, phrasing, or
extra details that do not affect correctness.
Respond ONLY with 'yes' or 'no'.
Question: {question}
Gold answer: {gold}
```

```
Student answer: {pred}
Is the student answer correct based solely
on the gold answer? Respond 'yes' or 'no'.
```

F Implementation Details for LooGLE Experiments

In this section, we provide comprehensive implementation details for our experiments on the LooGLE benchmark, including data selection, synthetic data generation pipelines, training hyperparameters, and prompt templates.

F.1 Data Selection and Preprocessing

Dataset Source. We utilize the **Short Dependency QA** subset of the LooGLE benchmark (Li et al., 2024), which consists of long-context documents with an average length exceeding 21k tokens.

Data Splitting. To rigorously evaluate generalization, we implement a strict split:

- **Source Domain (Training):** We select the **last 10 documents** (indices 100-104 for Round 1, indices 95-99 for Round 2) from the dataset. These documents are used solely for constructing the Skill Vector and are never seen during evaluation.
- **Target Domain (Evaluation):** We reserve the **first 50 documents** (indices 0-49) exclusively for testing.

F.2 Synthetic Data Generation Pipeline

We employ a two-stage data generation strategy to create high-quality training signals for both SFT and RL. The training data for both stages is generated by the base model `Qwen2.5-7B-Instruct` itself.

Stage 1: Multi-Task SFT Data Generation. To ensure the model deeply encodes the document content, we generate a diverse mixture of training tasks beyond simple text modeling. The SFT dataset $\mathcal{D}_{\text{SFT}}$ consists of three components mixed with specific ratios:

- **Summarization (Ratio 50%):** The model is tasked with compressing text chunks into concise summaries.
- **Recall/Expansion (Ratio 50%):** The model is tasked with reconstructing detailed text from summaries (inverse of summarization).

This data is generated using a sliding window approach with chunk sizes of $\{1024, 2048, 4096\}$ tokens and an overlap of 256 tokens. We generate 16 data points for each chunk. The generation temperature is set to 1.0.

Stage 2: QA Generation for RL. For the RL stage, we generate synthetic Question-Answer pairs.

- **Granularity:** We process the text in smaller chunks of $\{128, 256, 512\}$ tokens with an overlap of 16 tokens to capture fine-grained details.
- **Density:** We generate 8 pairs per chunk.
- **Prompt Diversity:** We employ a set of 6 distinct "Proposer" prompts (detailed in Sec. F.3) to ensure questions cover various aspects: factual details, reasoning (why/how), definitions, comparisons, lists, and significance.

F.3 Prompt Templates

To ensure the reproducibility of our synthetic data generation pipeline, we provide the exact content of the prompt templates used.

QA Generation Prompts. We utilize 6 distinct variations of prompts to generate diverse Question-Answer pairs. All variations share a common template to enforce strict formatting (XML tags) and specificity requirements. The full content of the instruction template and the 6 variations are detailed in Table 10 and 11.

Multi-Task SFT Prompts. For the summarization and expansion tasks in Stage 1 SFT, we randomly sample from a set of templates to prevent overfitting to a specific instruction format. The templates for Summarization and Recall/Expansion are listed in Table 12.

F.4 Training Hyperparameters

We perform the training using 8 NVIDIA A100 GPUs. The Source Domain training (Stage 1 in our method) is further divided into two sub-phases to ensure stability:

1. **SFT Phase 1 (Knowledge Encoding):** High learning rate training on the mixed dataset (Summarization, Recall, Verbatim) to enforce document memorization.
2. **SFT Phase 2 (QA Adaptation):** Lower learning rate training specifically on the synthetic QA pairs to bridge the gap to the RL format.
3. **RL Phase (Skill Sharpening):** GRPO training to refine the retrieval logic.

Checkpoint Selection Strategy. To avoid overfitting to the source documents, we employ an independent validation set consisting of LooGLE documents with indices 90-94. We evaluate checkpoints every 40 steps. Based on the validation accuracy, we selected the checkpoint at **Step 120** for Round 1 and **Step 160** for Round 2 as the final models for skill extraction.

Table 13 details the specific hyperparameters used in each phase.

F.5 Evaluation

Given the open-ended nature of the generated answers in the LooGLE benchmark, standard metrics like Exact Match or ROUGE are often insufficient to capture semantic correctness. Therefore, we employ a Model-Based Evaluation paradigm using `GPT-4.1` as an impartial judge to determine if the predicted answer matches the ground truth.

Judge Prompts. To ensure the evaluation is strictly binary and easy to parse, we enforce a strict output format. The exact prompts used for the judge model are provided below:

Evaluation Prompt

```
You are a precise evaluator. Your task is to
deter-mine if the 'Predicted Answer' is
semantically the same as the 'Ground Truth' for
the given 'Question'. Your entire response MUST
be only the single word 'True' or the single word
'False'. Do not provide any explanation or
punctuation.
Question: \{question\}
Ground Truth: \{reference\}
Predicted Answer: \{pred\}
```

Robustness via Multi-Pass Sampling. To account for generation stochasticity and ensure the robustness of our reported metrics, we perform 3 independent generation runs with a temperature of $T = 1.0$ for every question in the test set.This protocol ensures that our results reflect the model's consistent capability rather than lucky generations.

G TOOLBENCH EXPERIMENTS

G.1 CATEGORY FILTERING AND SELECTION CRITERIA

To ensure a robust and balanced evaluation of cross-domain generalization in agentic tool use, we perform a multi-stage filtering process on the original ToolBench and StableToolBench datasets. We apply the following constraints to select the evaluation categories:

- **Data Sufficiency:** We exclude categories with fewer than 3 solvable queries in the StableToolBench test set to ensure that the evaluation results are statistically meaningful.
- **Category Complexity:** We select categories with an API count between 75 and 350. This range ensures that the domain is sufficiently complex to require parametric internalizing (avoiding trivial domains with too few APIs) while remaining manageable for the environment simulator.

Based on these criteria, 21 categories were retained. We designate *Movies* as the **Source Domain** for skill acquisition because its API count (111) represents the median complexity of the filtered set, and it contains a relatively high number of test queries (35), allowing for stable monitoring of the RL training progress. The remaining 20 categories serve as the **Target Domains** for zero-shot generalization testing.

Statistics of Evaluated Categories Table 14 summarizes the statistics for the selected domains. The "APIs" column denotes the number of unique API schemas the model must internalize, and the "Test Queries" column denotes the number of solvable scenarios used for final evaluation.

G.2 IMPLEMENTATION DETAILS FOR SFT

Our SFT process is divided into two stages: **Knowledge Internalization** (Stage 1) and **Format Alignment** (Stage 2). All experiments are conducted using the `verl` framework with FSDP2 strategy on 8×A100 (80GB) GPUs.

G.2.1 DATA GENERATION FOR STAGE 1

To internalize the parametric knowledge of tools, we use the base model `Qwen2.5-7B-Instruct` to transform raw JSON schemas into diverse natural language (NL) descriptions and QA pairs.

Teacher Prompts. We use three distinct templates to generate descriptions from JSON schemas to ensure linguistic diversity.

Teacher Generation Prompts

Transform the following API JSON into a coherent, natural language paragraph describing how to use it. CRITICAL: You MUST explicitly mention the API name ("{API_NAME}") and its purpose at the beginning.

You are a technical documentation expert. Convert the provided API JSON definition into a comprehensive technical reference. Rules: 1. Identity: Explicitly state the API Name. 2. Purpose: Explain functionality. 3. Inputs: Detail parameters.

Convert the provided API definition into a structured natural language summary. Format: 1. Tool Identifier: The exact API name string. 2. Intent: User goal. 3. Action: Parameters.

Training Pair Construction. Based on the generated descriptions, we construct four types of training pairs to build a robust mapping between API names, intents, and schemas:

- **Type A (Name → Usage):** Queries like "How do I use the {name} API?" mapped to NL descriptions.

- **Type B (Intent → Usage):** Queries like "Identify the API defined by: {description} and explain its parameters."
- **Type C (Intent → Raw JSON):** Requesting the underlying JSON schema based on a description.
- **Type D (Name → Raw JSON):** Directly mapping the API name to its original JSON definition.

G.2.2 Training Configurations and Hyperparameters

We utilize different optimization strategies for the two stages. Stage 1 focuses on broad knowledge acquisition with a larger batch size and higher learning rate, while Stage 2 performs fine-grained format alignment. The detailed configurations are shown in Table 15.

G.3 Details of Reinforcement Learning for Tool Use

We implement the reinforcement learning phase based on the Search-R1 (Jin et al., 2025) framework, extending it to support multi-turn agentic tool-use trajectories and environment interactions.

Agent Prompting. The model is prompted to rely on its internalized knowledge acquired during the SFT phase. The system prompt specifies the ReAct format (`Thought`, `Action`, `Action Input`) and lists only the names of available APIs. The exact prompt template is provided in Table 16.

Environment Simulator. Since real-world API execution can be unstable or costly during RL, we employ `gpt-4o-mini` as an API Simulator. The simulator is provided with the full API documentation and examples (from the original ToolBench) and is tasked to validate the agent's generated `Action Input` against the ground-truth schema. It returns either a realistic JSON response or a specific error message (e.g., missing required parameters, type mismatch). The simulator's instructions are detailed in Table 17.

G.3.1 Reward Design

The reward signal R for a trajectory is composed of three components, designed to encourage format adherence, successful tool invocation, and task resolution.

Format and Execution Reward. For each intermediate turn j, we assign a step-wise reward $R_{\text{step},j}$ based on the validity of the generated action and its execution outcome. Specifically: (i) a positive reward of **+0.1** is granted if the action follows the correct ReAct format and the API call is successfully executed by the simulator; (ii) a penalty of -0.1 is applied if the format is correct but the API call fails (e.g., due to missing required parameters or type mismatches); (iii) a heavier penalty of -0.2 is imposed if the model fails to follow the output format or invokes a hallucinated API name not present in the available toolset.

Termination Reward. To prevent infinite loops and encourage concise solutions, we apply a maximum of 5 turns and apply a reward at the final turn where the model calls the `Finish` tool:

- **Active Finish:** If the model successfully calls `Finish`, it receives $+0.2$.
- **Forced Termination:** If the model reaches the maximum turn limit without calling `Finish`, it receives -0.5.

Solution Reward. The final Pass Reward is determined by a GPT-4.1 judge, which evaluates the entire trajectory. The judge assigns one of three statuses based on the rules in Table 18: `Solved` ($+1.0$), `Partially Solved` ($+0.5$), or `Unsolved` (0.0).

G.4 Hyperparameters for Reinforcement Learning

The reinforcement learning phase is conducted using the PaST framework on 4×A100 GPUs. Table 19 summarizes the detailed hyperparameter settings used for the ToolBench experiments.

H DECLARATION OF AI USAGE

Generative AI tools were used for grammar refinement and language polishing to enhance the readability of the manuscript. AI assistance was also employed during the coding and implementation phases of the project. All AI-assisted outputs were reviewed by the authors to ensure the technical quality and accuracy of the final paper.

User Query

I'm trying to download a post and a reel from Instagram. Can you provide me with the download links for the post and reel? The post link is [post link] and the reel link is [reel link].

Baseline: SFT Model

```
Thought: I should call the
'posts_for_instagram_reels_and_post_downloader' function with the
argument 'link' set to [post link]. This will allow me to download
the Instagram post URL [post link] and retrieve the download links
for the reels and posts from the specified Instagram media URL.
Action: posts_for_instagram_reels_and_post_downloader
Action Input: {"link": [post link]}

Observation: {"error": "", "response": "{ Sorry but this instagram
account is private }"}

Thought: The API call to download the Instagram post failed because
the account is private. To proceed, I should try another approach.
Action: newapi
Action Input: {"url": [post link]}

Observation: {"error": "Invalid API name: 'newapi'. Please check the
API name.", "response": ""}

Thought: The API call failed because the API name is invalid. This
indicates that the 'newapi' function is not available. To proceed, I
can try using the 'tiktok_download_video' function instead.
Action: get_tiktok_video_info_for_tiktok_download_video
Action Input: {"hd": "False", "url": [post link]}

Observation: {"error": "Invalid API name:
'get_tiktok_video_info_for_tiktok_download_video'. Please check the API
name.", "response": ""}

Thought: The API call failed because the API name is invalid. This
suggests that the 'get_tiktok_video_info_for_tiktok_download_video'
function is not available. As a last resort, I can use the
'cheap_youtube_api' to download the Instagram post.
Action: trending_music_gaming_news_movies_for_cheap_youtube_api
Action Input: {"category": "Instagram", "region": "US", "limit": 1}

Observation: {"error": "Invalid API name:
'trending_music_gaming_news_movies_for_cheap_youtube_api'. Please check
the API name.", "response": ""}

Thought: Since the tool call limit has been reached, I now need to
summarize my thoughts and call 'Finish' to end the task. I tried
using different APIs to download the Instagram post and retrieve the
download links, but none of them worked. It seems that the
'instagram_reels_and_post_downloader' function is not functioning
properly. To handle this, I can provide the user with an alternative
solution. I can suggest using a third-party app or website that
specifically caters to downloading Instagram posts and reels.
However, I should also mention that the user may need to provide the
post URL and the download links separately.
Action: Finish
Action Input: {"final_answer": "The current API does not have a valid
function available for downloading Instagram posts and reels. As an
alternative, you can try using third-party apps or websites
specifically designed for this purpose. When using these apps, make
sure to provide the Instagram post URL and the download links
separately. This will help you download the desired Instagram media
efficiently. Sorry for the inconvenience caused by this issue."}
```


Table 5: Case Study: Model comparison on tool-use trajectories (Part 1). Links are masked due to privacy concerns.

Ours

```
Thought: I am calling the
'posts_for_instagram_reels_and_post_downloader' function with the
argument of 'link' set to [post link]. This is because the user wants
to download a post from Instagram and has provided the post link. By
using this function, I can retrieve the download link for the
specified post.
Action: posts_for_instagram_reels_and_post_downloader
Action Input: {"link": [post link]}

Observation: {"error": "", "response": "{ Sorry but this instagram
account is private }"}

Thought: The response indicates that the Instagram account is
private, which is why I am unable to retrieve the download link for
the post. It seems that the user needs to make the account public in
order to access the download link.
Action: posts_for_instagram_reels_and_post_downloader
Action Input: {"link": [reel link]}

Observation: {"error": "", "response": "{ Sorry but this instagram
account is private }"}

Thought: The response for the reel download link is the same as for
the post, indicating that the Instagram account is also private.
Therefore, the user needs to make the account public in order to
access the download links for both the post and reel.
Action: Finish
Action Input: {"final_answer": "The Instagram account is private, so
I am unable to retrieve the download links for the post and reel. The
user needs to make the account public in order to access the download
links."}
```

Table 6: Case Study: Model comparison on tool-use trajectories (Part 2). Links are masked due to privacy concerns.

Context (Not shown to the model)

While the Treaties and Regulations will have direct effect (if clear, unconditional and immediate), Directives do not generally give citizens (as opposed to the member state) standing to sue other citizens. In theory, this is because TFEU article 288 says Directives are addressed to the member states and usually ̈leave to the national authorities the choice of form and methodsd̈o implement. In part this reflects that directives often create minimum standards, leaving member states to apply higher standards. For example, the Working Time Directive requires that every worker has at least 4 weeks paid holidays each year, but most member states require more than 28 days in national law. However, on the current position adopted by the Court of Justice, citizens have standing to make claims based on national laws that implement Directives, but not from Directives themselves. Directives do not have so called ḧorizontaldirect effect (i.e. between non-state parties). This view was instantly controversial, and in the early 1990s three Advocate Generals persuasively argued that Directives should create rights and duties for all citizens. The Court of Justice refused, but there are five large exceptions.

Question

What generally does not allow citizens to sue other citizens?

Ground Truth

Directives.

Baseline: SFT Model

In general, citizens cannot directly sue other citizens in most legal systems. However, there are exceptions where private individuals can be held liable for certain types of harm, such as intentional torts or breaches of contract.

Ours

Directives do not create rights and duties for citizens. They are addressed to Member States and usually have indirect effect, meaning that they do not create rights and duties for citizens. Directives usually contain exemptions and allow Member States to choose the form in which they are implemented. This means that there is often no direct effect between citizens and in most cases, citizens cannot sue other citizens based on Directives. However, Directives often create higher standards than those that exist in national laws, so if a national law is less strict than a Directive, the latter can be used to argue that a law is invalid. In summary, citizens generally cannot sue other citizens based on Directives, but Directives can create higher standards than national laws, which can be used to challenge the validity of national laws.

Table 7: Case Study: Model comparison on SQuAD.

Table 8: **Adopted SFT hyperparameters on SQuAD.** We do *not* tune hyperparameters; instead, we directly reuse the best-performing configurations reported in SEAL for each regime.

Hyperparameter	**Single Passage** (n=1, **LoRA update**)	**CPT** (n=200/2067, **Full FT**)
Base model	`Qwen2.5-7B`	
Training data	*passage* + *synthetic implications* (same construction as SEAL)	
Max sequence length	2048	2048
Update type	LoRA (test-time update)	Full fine-tuning
LoRA rank r	32	–
LoRA alpha α	64	–
LoRA dropout	0	–
LoRA target modules	as in SEAL implementation	–
Training epochs	10	1
Learning rate	1×10^{-3}	7×10^{-5}
Per-device batch size	1	4
Gradient accumulation	1	2

Table 9: GRPO hyperparameters used in our training script.

Category	**Value**
Algorithm	GRPO (`algorithm.adv_estimator=grpo`)
Epochs	15
Train batch size	32 (`data.train_batch_size`)
Actor LR	1×10^{-6} (`actor.optim.lr`)
Max prompt length	512 (`data.max_prompt_length`)
Max response length	1024 (`data.max_response_length`)
Rollout backend	vLLM (`actor_rollout_ref.rollout.name=vllm`)
#rollouts per prompt (n)	5 (`actor_rollout_ref.rollout.n`)
Sampling	temperature $= 0.7$, top-$p = 0.9$, top-$k = 50$
Max new tokens	512 (`actor_rollout_ref.rollout.max_new_tokens`)
KL regularization	enabled (`use_kl_loss`), coef $= 0.001$, type `low_var_kl`
PPO mini/micro batch	mini $= 64$, micro per GPU $= 8$
Precision	rollout dtype `float16`

Shared Instruction Template (Common to all QA Prompts)

```
You are an AI assistant tasked with generating a single, high-quality
question-answer pair from a given text.

**Your instructions are critical:**
1.  **Generate one Q&A pair** based *only* on the provided text.
2.  **Specificity is Key:** The question *must* be self-contained and
unambiguous. It must include specific names or key terms from the
text (e.g., "What is 'Project Helios'?" instead of "What is this
project?").
3.  **Format:** Your output *must* use XML-style tags: <question>Your
question here</question> <answer>Your answer here</answer>. Do not
include any other text before or after the tags.
4. [Specific Task Instruction: Refer to Variations 1-6 below]

**Example of a Good, Specific Q&A:**
[Task-specific Example: Refer to Variations 1-6 below]

**Text Fragment:**
{text_segment}

**Your Output:**
```

Variation 1: Factual Detail

```
4.  **Task:** Focus on a specific, factual detail: a name, a date, a
key term, or a specific component.

**Example of a Good, Specific Q&A:**
<question>
What consensus mechanism does the 'Helios' architecture pioneer?
</question>
<answer>
It pioneers a decentralized consensus mechanism called
'Proof-of-History' (PoH).
</answer>
```

Variation 2: Reasoning (Why/How)

```
4.  **Task:** Focus on the *reason* (Why) or the *method* (How)
behind a concept or event described in the text.

**Example of a Good, Specific Q&A:**
<question>
Why does the 'Proof-of-History' (PoH) mechanism successfully reduce
latency?
</question>
<answer>
Because it creates a verifiable, sequential record of time, which
avoids the need for solving complex computational puzzles like in
'Proof-of-Work'.
</answer>
```

Table 10: **QA Generation Prompts (Part 1).** The shared system instruction and the first two task variations used to generate synthetic QA pairs for LooGLE.

Variation 3: Definitions

```
4.  **Task:** Generate a "What is..." or "What does... stand for?"
question about a key concept.

**Example of a Good, Specific Q&A:**
<question>
What is 'Proof-of-History' (PoH) as described in the context of the
'Helios' architecture?
</question>
<answer>
It is a decentralized consensus mechanism that creates a verifiable,
sequential record of time.
</answer>
```

Variation 4: Comparisons

```
4.  **Task:** Focus on the *difference* or *similarity* between two
specific concepts, methods, or entities in the text.

**Example of a Good, Specific Q&A:**
<question>
What is the key difference between the 'Proof-of-History' (PoH)
mechanism and 'Proof-of-Work' (PoW)?
</question>
<answer>
'Proof-of-History' (PoH) creates a verifiable, sequential record of
time, whereas 'Proof-of-Work' (PoW) relies on solving complex
computational puzzles.
</answer>
```

Variation 5: Lists/Processes

```
4.  **Task:** Generate a question that asks to list steps,
components, or stages of a specific system or method.

**Example of a Good, Specific Q&A:**
<question>
What are the three main stages of the 'Project Nova' deployment
pipeline?
</question>
<answer>
The three main stages are 'Build', 'Test', and 'Verify'.
</answer>
```

Variation 6: Significance/Impact

```
4.  **Task:** Generate a "What is the significance of..." or "What is
the main advantage of..." question.

**Example of a Good, Specific Q&A:**
<question>
What is the main advantage of the 'Helios' architecture's
'Proof-of-History' mechanism?
</question>
<answer>
Its main advantage is a drastic reduction in latency.
</answer>
```

Table 11: **QA Generation Prompts (Part 2).** Additional task variations (3-6) used to ensure diversity in the synthetic LooGLE training data.

Summarization Templates

```
Create a concise and objective summary of the text below. Focus on
the main ideas and most important information, presenting them
clearly in your own words.\n\nText to
summarize:\n{text_segment}\n\nNow generate your summary:

Distill the following text to its absolute essence. Provide a
one-paragraph summary that captures the core argument and key
takeaways. Avoid any fluff or secondary details.\n\nOriginal
Text:\n{text_segment}\n\nYour distilled summary:

Break down the following text for someone completely unfamiliar with
the topic. Your summary should be simple, clear, and use
easy-to-understand language, as if you were explaining it to a 5th
grader. Focus on the main concepts without getting lost in technical
jargon.\n\nText:\n{text_segment}\n\nYour simple explanation:

Provide a summary of the following passage. Your summary should not
only capture the key information but also reflect the original
author's tone and style (e.g., formal, persuasive, humorous,
critical). \n\nOriginal Passage:\n{text_segment}\n\nYour summary, in
the original tone:

Provide a strictly neutral and objective summary of the provided
text. Your goal is to recount the main points and arguments without
injecting any personal opinion, interpretation, or evaluative
language. Simply state what the text
says.\n\nText:\n{text_segment}\n\nNeutral Summary:
```

Recall/Expansion Templates

```
Below is a summary of a larger text. Your task is to expand on this
summary, creating a detailed and comprehensive paragraph that could
have been the original source. Flesh out the main points with
explanations, smooth transitions, and supporting
details.\n\nSummary:\n{summary}\n\nNow generate the detailed original
text:

You are given a concise summary below. Your objective is to generate
a more detailed and fully-formed text based on it. Elaborate on the
ideas presented in the summary, ensuring the resulting text is
coherent, well-structured, and reads
naturally.\n\nSummary:\n{summary}\n\nGenerated Text:

The following summary is a high-level overview of a piece of
information. Your task is to reconstruct a plausible original text by
filling in the necessary details, examples, and explanations that
might have been removed during
summarization.\n\nSummary:\n{summary}\n\nReconstructed Text with
Details:

Consider the summary below as a set of conclusions or key statements.
Your goal is to write a text that logically leads to these points.
Build a coherent argument or explanation that culminates in the
information provided in the summary.\n\nKey Points /
Summary:\n{summary}\n\nText with Logical Development:

Take the core ideas presented in the summary below and weave them
into a single, seamless paragraph. Focus on creating smooth
transitions between the points so they flow together as a unified
piece of writing, rather than a list of facts.\n\nCore
Ideas:\n{summary}\n\nCoherent Paragraph:
```


Table 12: **Multi-Task SFT Prompts.** Templates used for the Summarization and Recall tasks in Stage 1 training to enhance knowledge encoding.

Table 13: **Detailed Hyperparameters for LooGLE Experiments.** We employ a multi-stage curriculum: SFT Phase 1 enforces deep encoding of the 21k-token documents via mixed tasks; SFT Phase 2 adapts the model to the QA format; and the RL stage (GRPO) refines the retrieval logic using a closed-book setting (short prompt, long generation).

Hyperparameter	**SFT Phase 1** *(Knowledge Encoding)*	**SFT Phase 2** *(QA Adaptation)*	**RL (GRPO)** *(Skill Sharpening)*
General Configuration			
Base Model		Qwen2.5-7B-Instruct	
Precision	bf16	bf16	bf16
Gradient Checkpointing	True	True	True
Number of GPUs	8	8	8
Optimization			
Optimizer	AdamW	AdamW	AdamW
Learning Rate	1e-4	2e-5	1e-6
LR Scheduler	Cosine	Cosine	Constant
Global Batch Size	16	64	128
Micro Batch Size (per GPU)	2	2	8
Total Epochs	3	2	10 (Selected Best Step)
Data & Context			
Max Sequence Length	8192	8192	-
Max Prompt Length	-	-	128 (Closed-Book)
Max Response Length	-	-	1024
Data Source	Mixed (Sum/Recall)	Synthetic QA	Synthetic QA
RL Specifics (GRPO)			
Group Size (G)	-	-	5
KL Coefficient (β)	-	-	0.001
KL Reference Model	-	-	SFT Phase 2 Checkpoint
Reward Function	-	-	GPT-4.1 Judge

Table 14: Statistics of the 21 selected categories from ToolBench. The Source Domain is used for RL training, while Target Domains are used for zero-shot evaluation.

Category	**Domain Role**	**# APIs**	**# Test Queries**
Movies	**Source**	111	35
Advertising	Target	118	3
AI & ML	Target	108	4
Business Software	Target	199	5
Communication	Target	250	8
Database	Target	260	7
Education	Target	214	9
Email	Target	143	5
Financial	Target	224	11
Food	Target	208	9
Health and Fitness	Target	90	7
Location	Target	328	11
Mapping	Target	113	7
Media	Target	159	12
SMS	Target	75	3
Science	Target	99	4
Search	Target	103	13
Text Analysis	Target	98	4
Video Images	Target	207	44
Weather	Target	199	8
eCommerce	Target	342	15

Table 15: Hyperparameters for the two stages of SFT in ToolBench.

Hyperparameter	**Stage 1: Internalization**	**Stage 2: Alignment**
Base Model	Qwen2.5-7B-Instruct	
Max Sequence Length	8192	8192
Optimizer	AdamW	
Precision	BF16	
Learning Rate	5×10^{-5}	2×10^{-5}
Total Batch Size	64	32
Micro Batch Size (per GPU)	2	2
Total Epochs	3	3
Parallel Strategy	FSDP2 (verl)	
Liger Kernel	Enabled	

Table 16: The system prompt for the agent in ToolBench Experiments.

Agent System Prompt

```
You are an intelligent agent designed to handle real-time user
queries using a variety of tools.

First, you will receive a task description. Then, you will enter a
loop of reasoning and acting to complete the task.

At each step, follow this process:
1. **Thought**: Analyze the current status and determine the next
logical step.
2. **Action**: Select the appropriate tool to execute that step and
output the function name directly.
3. **Action Input**: Provide the arguments for the tool as a STRICT
valid JSON object.

Output Format:
Thought: <your reasoning>
Action: <function_name>
Action Input: <function_arguments_as_a_valid_JSON_object>

After the action is executed, you will receive the result
('Observation: <observation>'). Based on the new state, continue the
loop until the task is complete.

Constraints & Rules:
1. **Action Field**: The "Action" output must be the EXACT name of
the function. Do NOT include parentheses `()`, words like "call" or
"use", or any punctuation.
2. **Finishing**: You MUST call the "Finish" function to submit your
final answer.

Available Tools:
1. **General Tools**: You have been trained on a specific set of
APIs: {api_names}. You must rely on your **internal knowledge** to
recall the correct parameter schemas for these tools.
2. **Termination Tool**: You MUST use the following tool to finish
the task. Its definition is provided below:
{"name": "Finish", "description": "If you believe that you have
obtained a result that can answer the task, please call this function
to provide the final answer. Remember: you must ALWAYS call this
function at the end of your attempt, and the only part that will be
shown to the user is the final answer, so it should contain
sufficient information.", "parameters": {"properties": {
"final_answer": {"type": "string", "description": "The final answer
you want to give the user."}}}, "required": ["final_answer"],
"optional": []}
```

Table 17: The simulator's instructions in ToolBench Experiments.

API Simulator Instructions

```
You are an advanced API Simulator and Validator. Your role is to act
as a real API server, strictly adhering to the provided documentation
to process requests.

### 1. Input Structure Explanation
The user will provide input in the following specific format:
API Documentation:
Contains the API's URL, method, description, and parameter
definitions (required/optional).
API Examples:
Contains reference calls (Note: This section is truncated to 2048
chars and may be incomplete; use it for style reference but rely on
Documentation for logic).
API Input:
The specific arguments/payload you need to process.

### 2. Processing Logic
1. **Analyze:** Read the `API Documentation` to understand the schema
and constraints (types, required fields).
2. **Validate:** Check the `API Input` against the `API
Documentation`.
- Are all `required_parameters` present?
- Do the data types match (e.g., string vs int)?
3. **Execute:**
- **If Valid:** Generate a realistic, rich JSON response.
- **If Invalid:** Generate a JSON response where `error` describes
the specific validation failure.

### 3. Output Format
You must output ONLY a valid JSON object. No Markdown code blocks. No
conversational text.

**JSON Schema:**
{
    "error": "String describing the error (if any), otherwise empty
string",
    "response": <The_Simulated_Response_Object_or_Null>
}

### 4. Behavior Rules
- **Length Constraints:** - Keep the response **concise and
lightweight**. Keep the entire JSON output shorter than 300 words. Do
not generate excessively large payloads. If the API returns a list or
array, **limit it to maximum 1-2 items**.
- **Source of Truth:** Do not blindly copy "API Examples" if they
contradict the "API Input". The "API Input" is your priority.
```

Table 18: The evaluation prompt in ToolBench experiments.

GPT-4.1 Judge Prompt

```
Giving the query and the corresponding execution trajectory
(including thoughts, tool calls, and observations), evaluate the
`answer_status` based on these rules:

1. **Solved**: The tool calls were successful. The final answer is
strictly grounded in the real "Observation" data and fully addresses
the query.
2. **Partially Solved**: The model used real "Observation" data, but
the task is only halfway finished or the final answer missed some
details from the observations.
3. **Unsolved**:
    - The model fabricated information not found in the Observations.
    - The tool calls failed, and the model failed to solve the query
or made up a result.
    - The answer is incorrect or irrelevant.

Output a JSON object with the following fields:
- "reason": A very brief explanation (less than 20 words).
- "answer_status": One of ["Solved", "Partially Solved", "Unsolved"].

<Target_Query>
{query}
</Target_Query>

<Model_Execution_Trajectory>
{trajectory}
</Model_Execution_Trajectory>
```

Table 19: Detailed hyperparameters for PPO training in ToolBench experiments.

Category	**Hyperparameter**	**Value**
Data & Environment	Max Prompt Length	8192
	Max Response Length	1024
	Max Simulation Turns	5
Optimization	Actor Learning Rate	1×10^{-6}
	Critic Learning Rate	1×10^{-5}
	Actor LR Warmup Ratio	0.285
	Critic LR Warmup Ratio	0.015
	LR Scheduler	Constant
	Optimizer	AdamW
PPO Algorithm	Advantage Estimator	GAE
	PPO Epochs per Batch	1
	Mini-batch Size	64
	Clip Range (ϵ)	0.2
	KL Penalty Coefficient (β)	0.001
	Entropy Coefficient	0.001
Training Schedule	Total Training Steps	180
	Compute Precision	BF16