RECAP: RETRIEVAL-GUIDED CAPABILITY REUSE FOR MULTIMODAL CONTINUAL INSTRUCTION TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Multimodal continual instruction tuning (MCIT) aims to enable multimodal large language models to acquire new capabilities from sequential tasks while preserving previously learned knowledge. Existing methods primarily mitigate catastrophic forgetting by constraining parameter updates or separating task-specific adaptations. However, continual adaptation can also benefit from external knowledge that provides domain-specific information and reusable reasoning patterns for solving diverse instructions. For example, to answer "How many red cubes are to the left of the sphere?", domain knowledge can provide relevant concepts about objects and spatial relations, while reasoning knowledge can specify ordered operations such as object recognition, spatial filtering, and counting. Despite this potential, how to leverage external knowledge for continual adaptation remains largely unexplored in existing MCIT methods. To this end, we propose RECAP, a retrieval-guided framework that leverages external knowledge to guide capability reuse during continual adaptation. At each continual stage, RECAP uses external search and an LLM to incrementally build a knowledge base of domain, reasoning, and format knowledge based on the current-stage training data. For each instruction, retrieved domain knowledge guides generation, while retrieved reasoning knowledge selects and orders capability modules to form an instance-specific capability path. As these capability modules are reused across stages, subsequent adaptation can overwrite previously learned parameters. To enable stable cross-stage reuse, RECAP introduces adaptive subspace recycling, which parameterizes reusable capability modules with shared bases and stage-specific cores, protects historically important directions while recycling residual capacity. Extensive experiments on MCIT benchmarks show that RECAP achieves SOTA performance.

1 INTRODUCTION

Multimodal large language models (MLLMs) (Liu et al., 2023; Dai et al., 2023) acquire broad vision-language capabilities through instruction tuning (Zhang et al., 2026; Tong et al., 2025), yet real-world deployment is inherently dynamic. New visual domains, question types, and reasoning requirements continuously emerge after initial training, while retraining on all previously observed data is often unavailable or prohibitively expensive. Multimodal continual instruction tuning (MCIT) therefore aims to enable models to acquire new capabilities from sequential tasks while preserving previously learned knowledge (Chen et al., 2024). Existing MCIT methods primarily mitigate catastrophic forgetting by constraining parameter updates or introducing task-specific adaptations, including prompt-based methods (Zeng et al., 2025), low-rank modules (Chen et al., 2024), expandable components (Guo et al., 2025), and expert-based routing (Huai et al., 2025; Xie et al., 2026). While effective in reducing cross-task interference, these methods largely rely on model parameters and task-specific adaptation modules to encode and preserve what is learned across continual stages.

However, continual adaptation can also benefit from external knowledge. A natural way to access such knowledge is retrieval-augmented generation (RAG) (Guu et al., 2020; Lewis et al., 2020; Asai et al., 2024), which connects models to an editable external memory and allows knowledge beyond model parameters to be independently updated and retrieved. In MCIT, such knowledge can provide more than factual information, particularly for handling diverse and evolving task requirements.

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

Domain knowledge can supply relevant concepts, relations, and visual context for the current multimodal instruction, while reasoning knowledge can describe the ordered operations needed to solve it. For example, as illustrated in Figure 1, answering "Which category in the chart shows the largest increase from 2020 to 2022?" may require knowledge of chart conventions, axes, and legends, together with a reasoning process involving visual element recognition, value extraction, and comparison. Output conventions can further characterize the expected response form. Despite these benefits, existing MCIT methods have largely underexplored how external knowledge can support continual adaptation.

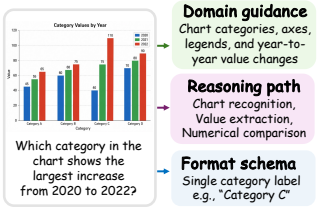


Figure 1: An example requiring chart understanding and comparison reasoning.

The key challenge, therefore, is how to make such external knowledge actionable for continual adaptation. Reasoning knowledge is particularly useful in this regard, as its ordered operations naturally reveal the capabilities required to solve an instruction. Vanilla RAG (Lewis et al., 2020) primarily conditions generation on retrieved information, but does not explicitly organize these operations into reusable capability units. Representing these operations as parameterized capability modules provides an explicit interface through which retrieval can select and compose reusable capabilities for each instruction, while allowing recurring capabilities to be reused across continual stages. However, such parameterized reuse introduces a new continual-learning challenge: adapting a recurring capability at later stages may overwrite previously learned behaviors. Effective knowledge-guided MCIT therefore requires both retrieval-guided capability composition and an adaptive mechanism for preserving reusable capabilities during continual adaptation.

Motivated by these considerations, we propose RECAP, a retrieval-guided framework for capability reuse in MCIT. At each continual stage, RECAP uses the current-stage training data, external search, and an LLM to incrementally build a structured knowledge base of domain, reasoning, and format cards. For each multimodal instruction, retrieved domain knowledge provides prompt guidance for generation, while retrieved reasoning knowledge identifies the required capability operations and organizes the corresponding parameterized modules into an instance-specific path. The selected modules are then composed sequentially along this path during the forward pass. Format knowledge provides response-form cues that further support reasoning retrieval and inference-time core routing. To support stable reuse of recurring capabilities across continual stages, RECAP introduces adaptive subspace recycling, which represents them with shared bases and stage-specific cores. Directions carrying high accumulated historical energy are protected from subsequent interference, while low-energy directions are recycled and new directions are appended to provide capacity for further adaptation. Finally, to enable task-free use of the retained stage-specific cores, retrieval-guided routing selects an appropriate core for each activated capability at inference.

In summary, our contributions are threefold: (1) we introduce retrieval-guided capability reuse for MCIT, where external knowledge provides generation guidance and organizes instance-level capability composition and reuse; (2) we propose adaptive subspace recycling for stable cross-stage capability reuse through shared bases, stage-specific cores, and protected historical directions; and (3) experiments on MCIT benchmarks demonstrate RECAP's effectiveness over representative baselines.

2 Related Work

Multimodal continual instruction tuning: MCIT extends continual learning from fixed-label prediction to unified vision–language instruction following. CoIN introduced a representative benchmark and mixture-of-expert LoRA adaptation for sequential multimodal tasks (Chen et al., 2024). Subsequent methods explore different strategies to balance stability and plasticity, including modality-aware prompting (Zeng et al., 2025), expert specialization and routing (Huai et al., 2025), expandable task-specific components with transferable representations (Guo et al., 2025), and joint stabilization of routing and parameter updates (Xie et al., 2026). Orthogonal low-rank updates provide another parameter-isolation strategy for reducing interference between continual updates (Wang et al., 2023). Meanwhile, LoRA-based adaptation (Hu et al., 2022) has become a common approach for storing compact task adaptations. These methods mainly focus on parameter-space solutions, either separating adaptations across tasks or constraining updates to preserve previous knowledge.

Retrieval-augmented generation: Retrieval-augmented generation combines a parametric model with an external, editable memory, enabling models to access and update knowledge beyond their parameters. Existing studies explore retrieval-enhanced pretraining (Borgeaud et al., 2022), knowledge-intensive generation, multi-passage fusion (Izacard & Grave, 2021), adaptation over external corpora (Izacard et al., 2022), query transformation (Gao et al., 2023), and retrieval-aware generation evaluation (Asai et al., 2024). These methods primarily focus on improving knowledge access, factual grounding, and generation quality. In contrast, the role of retrieval as a mechanism for organizing continual capability acquisition and reuse in MCIT remains largely unexplored.

3 PRELIMINARIES

3.1 MULTIMODAL CONTINUAL INSTRUCTION TUNING

Multimodal continual instruction tuning (MCIT) considers a sequence of multimodal instruction tuning tasks. Specifically, the training stream consists of T datasets:

$$\mathcal{D}_{1:T} = \{\mathcal{D}_t\}_{t=1}^{T}, \qquad \mathcal{D}_t = \{(\mathbf{v}_i^t, \mathbf{q}_i^t, \mathbf{y}_i^t)\}_{i=1}^{n_t}, \tag{1}$$

where $\mathbf{v}$, $\mathbf{q}$, and $\mathbf{y}$ denote an image, an instruction, and its target response, respectively. At stage t, the learner observes only the current dataset $\mathcal{D}_t$ and sequentially updates the model. After learning stage t, the model is evaluated on all observed tasks $\mathcal{D}_{1:t}$, requiring it to acquire new capabilities while preserving previously learned knowledge.

3.2 BASELINES IN MULTIMODAL CONTINUAL INSTRUCTION TUNING

To mitigate catastrophic forgetting, existing MCIT methods mainly follow two typical strategies: constraining parameter updates or separating task-specific adaptations.

Constraining Parameter Updates: Continually updating shared adaptation parameters may interfere with previously learned knowledge. Methods such as O-LoRA (Wang et al., 2023) and SAME (Xie et al., 2026) therefore restrict current-stage updates using information retained from previous stages. The constrained update is represented as:

$$\Delta\boldsymbol{\theta}_t^{\mathrm{c}} = \mathcal{C}\left(\Delta\boldsymbol{\theta}_t; \mathcal{H}_{<t}\right), \qquad \boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \Delta\boldsymbol{\theta}_t^{\mathrm{c}}, \tag{2}$$

where $\Delta\boldsymbol{\theta}_t$ denotes the update on the stage t, $\mathcal{H}_{<t}$ represents historical information from previous stages, and $\mathcal{C}$ constrains the current update according to this historical information. In this way, the model limits interference with previously learned knowledge while adapting to the current stage.

Separating Task-Specific Adaptations: To avoid overwriting previous adaptations, methods such as CL-MoE (Huai et al., 2025) and HiDe-LLaVA (Guo et al., 2025) maintain specialized or expandable components for different task requirements. Their forward computation is represented as:

$$\mathbf{o} = f_0(\mathbf{x}) + \sum_{m=1}^{M_t} r_m(\mathbf{x})\mathcal{A}_m(\mathbf{x}), \tag{3}$$

where f_0 denotes the frozen backbone, $\{\mathcal{A}_m\}_{m=1}^{M_t}$ denotes the adaptation components available after stage t, and $r_m(\mathbf{x})$ controls the selection or contribution of each component. In this way, task-specific adaptations are selectively activated to reduce interference across continual stages.

Discussion: Equations 2 and 3 represent two typical parameter-space strategies for mitigating catastrophic forgetting in MCIT. Constraining shared parameter updates helps preserve previously learned knowledge, but may restrict the flexibility required to acquire new capabilities. Separating task-specific adaptations reduces direct interference, but can duplicate reusable knowledge across different components. More importantly, both strategies rely primarily on model parameters and adaptation modules to organize continual learning, without exploiting external knowledge that can provide domain information and reasoning operations for individual instructions. Such knowledge can further reveal capabilities that are reusable within and across tasks. Hence, an effective MCIT framework should leverage external knowledge to guide capability acquisition and reuse while preserving previously learned behaviors across continual stages.

4 METHOD

RECAP connects external knowledge with continual parameter adaptation through reusable capabilities. We first incrementally build a structured knowledge base, where retrieved domain knowledge provides generation guidance and reasoning knowledge provides ordered capability operations for composing capability-specific modules. Afterward, to enable stable reuse of recurring capabilities, RECAP introduces adaptive subspace recycling, which protects historically important directions while retaining residual capacity for subsequent adaptation.

4.1 KNOWLEDGE BASE CONSTRUCTION AND RETRIEVAL

Knowledge Card Construction: At each continual stage j, we construct a stage-local knowledge shard from the current training instructions in $\mathcal{D}_j$. We first group instructions by their inferred answer schema, then encode them with a frozen embedding model ϕ and cluster semantically similar instructions within each group. Representative instructions from each cluster, together with external search results that provide visual context and background information, are then summarized by an LLM. This process produces three types of knowledge cards: domain cards contain relevant concepts, relations, and concise *prompt guidance*; reasoning cards contain ordered capability operations; and format cards specify the expected answer schema. Detailed construction prompts and card schemas are provided in Appendix B. Let $\mathcal{K}_c^j$ denote the stage-j cards of type $c \in \{\mathrm{dom}, \mathrm{rea}, \mathrm{fmt}\}$. At stage t, the available knowledge is accumulated only from the observed stages:

$$\mathcal{K}_c^{1:t} = \bigcup_{j=1}^{t} \mathcal{K}_c^j, \qquad c \in \{\mathrm{dom}, \mathrm{rea}, \mathrm{fmt}\}. \tag{4}$$

Accordingly, training and evaluation at stage t use only $\mathcal{K}_c^{1:t}$, without access to future-stage instructions or cards. For subsequent retrieval, each domain and reasoning card is associated with one or more cluster prototypes obtained by averaging the embeddings of its source instructions. The same frozen embedding model ϕ is used to encode retrieval queries.

Knowledge Retrieval: For each instruction $\mathbf{q}$, RECAP first retrieves a domain card and a format card, whose information is then used as additional cues for reasoning-card retrieval. For a candidate domain or reasoning card k, the retrieval score combines similarity to its associated prototypes with type-specific card-content signals capturing semantic, lexical, and structured compatibility:

$$s_{\mathrm{proto}}(\mathbf{q}, k) = \max_{\mathbf{p} \in \mathcal{P}(k)} \cos(\phi(\mathbf{q}), \mathbf{p}), \tag{5}$$

$$s_c(\mathbf{q}, k) = \alpha_c s_{\mathrm{proto}}(\mathbf{q}, k) + (1 - \alpha_c) \sum_{m \in \mathcal{M}_c} \lambda_{c,m} s_m(\mathbf{q}, k), \qquad c \in \{\mathrm{dom}, \mathrm{rea}\}. \tag{6}$$

Here $\mathcal{P}(k)$ contains the prototypes associated with card k, while $\mathcal{M}_c$ indexes the type-specific card-content signals used for card type c. The coefficient α_c balances prototype similarity and content matching, and $\lambda_{c,m}$ weights each normalized signal s_m. Domain retrieval uses card-text semantic similarity, keyword overlap, and concept overlap. Format cards are retrieved separately using keyword and answer-schema matching, with explicit response requirements prioritized. Reasoning retrieval further combines semantic, keyword, and concept matching with compatibility to the retrieved domain and format information. We retain the highest-scoring card of each type. All retrieval signals and fixed coefficients are detailed in Appendix C. The left panel of Figure 2 illustrates this stage-wise knowledge construction and instruction-conditioned retrieval process.

4.2 RETRIEVAL-GUIDED CAPABILITY COMPOSITION

After knowledge retrieval, we keep the pretrained backbone frozen and maintain a shared capability set $\mathcal{S}$. Each capability $s \in \mathcal{S}$ represents a canonical reusable operation shared across continual stages, such as visual recognition, spatial reasoning, or counting, and is implemented by a capability-specific parameterized module in the adapted LLM layers. Retrieved knowledge determines which capabilities are activated for each instance and how they are composed.

Domain Guidance: Given the retrieved domain card k_{dom}^*, its *prompt guidance* field is concatenated with the original instruction:

$$\widetilde{\mathbf{q}} = \mathrm{concat}(\mathrm{guidance}(k_{\mathrm{dom}}^*), \mathbf{q}). \tag{7}$$

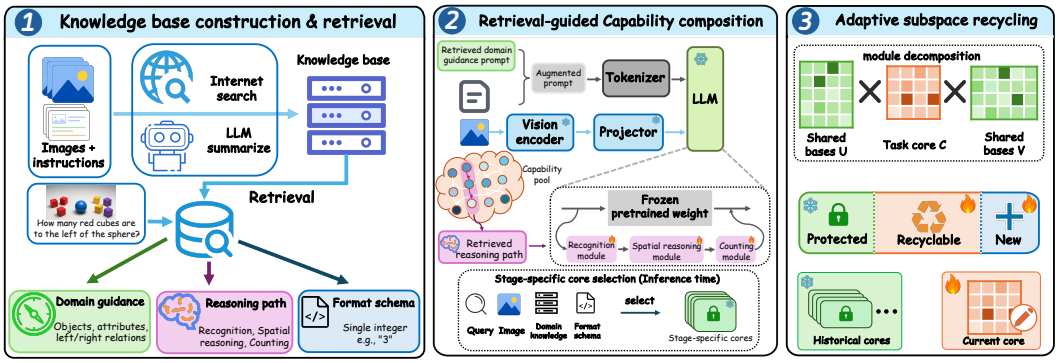


Figure 2: Overview of RECAP. (1) At each continual stage, a structured knowledge base is incrementally constructed from the current training instructions with external search and LLM summarization; the current instruction retrieves domain guidance, a reasoning path, and a format schema. (2) Domain guidance augments the model prompt, while the retrieved reasoning path selects and sequentially composes parameterized modules from a shared capability pool. At inference time, instruction, retrieved knowledge, visual, and path-transition cues select a stage-specific core for each activated capability. (3) Adaptive subspace recycling represents recurring capabilities with shared bases and stage-specific cores, protecting high-energy historical directions, recycling low-energy directions, and appending new directions for subsequent adaptation.

Capability Path: The retrieved reasoning card specifies an ordered sequence of capability operations, each mapped to a canonical capability in $\mathcal{S}$ during path construction. The resulting capabilities define an instance-specific capability path:

$$\pi(\mathbf{q}) = (s_1, s_2, \ldots, s_L), \qquad s_j \in \mathcal{S}, \tag{8}$$

where L is the path length and each s_j denotes a reusable capability. For example, an instruction involving object identification, spatial filtering, and counting may yield visual recognition $\rightarrow$ spatial reasoning $\rightarrow$ counting. During training at stage t, capability s_j at adapted layer ℓ is parameterized by $A^{\ell}_{s_j,t}$. At adapted layer ℓ, the stage-t parameterization of capability s_j is denoted by $A^{\ell}_{s_j,t}$. Given an input representation $\mathbf{x}$, the selected capability modules are applied sequentially along the retrieved path:

$$\mathbf{z}_0 = \mathbf{x}, \qquad \mathbf{z}_j = \mathbf{z}_{j-1} + A^{\ell}_{s_j,t}\mathbf{z}_{j-1}, \quad j = 1, \ldots, L. \tag{9}$$

The adapted layer output is:

$$\mathbf{o} = W^{\ell}_0\mathbf{x} + \mathbf{z}_L - \mathbf{z}_0, \tag{10}$$

where W^{ℓ}_0 is the frozen layer weight. Each capability operates on the representation produced by preceding capabilities, preserving the ordered computation specified by the retrieved reasoning path. The middle panel of Figure 2 illustrates this retrieval-guided capability composition process.

4.3 ADAPTIVE SUBSPACE RECYCLING

The same capability may recur across continual stages. Using a separate module for each recurrence prevents sharing, whereas updating a single shared module risks overwriting previously learned behaviors. To support stable reuse, RECAP represents each recurring capability with shared basis parameters and stage-specific cores, and applies adaptive subspace recycling to protect historically important directions while retaining trainable capacity for subsequent adaptation.

Core Factorization: For capability s, we define its parameterization at layer ℓ and stage t as $A^{\ell}_{s,t} = U^{\ell}_s C^{\ell}_{s,t} V^{\ell}_s$, where $A^{\ell}_{s,t} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, $U^{\ell}_s \in \mathbb{R}^{d_{\text{out}} \times K_s}$, $C^{\ell}_{s,t} \in \mathbb{R}^{K_s \times K_s}$, and $V^{\ell}_s \in \mathbb{R}^{K_s \times d_{\text{in}}}$. Here d_{in} and d_{out} denote the input and output dimensions, and K_s is the rank. U^{ℓ}_s and V^{ℓ}_s are shared across stages, while $C^{\ell}_{s,t}$ is the stage-specific core of capability s.

Energy-Ordered Reparameterization: Directly protecting individual basis coordinates is unreliable because equivalent low-rank factorizations can represent the same capability parameterization using different coordinates. For a recurring capability s at one adapted layer, we therefore first construct an orthonormal coordinate system. For clarity, we omit the layer superscript ℓ and write U and V for the active shared bases. Reduced QR decomposition gives:

$$U = Q_U R_U, \qquad V^{\top} = Q_V R_V. \tag{11}$$

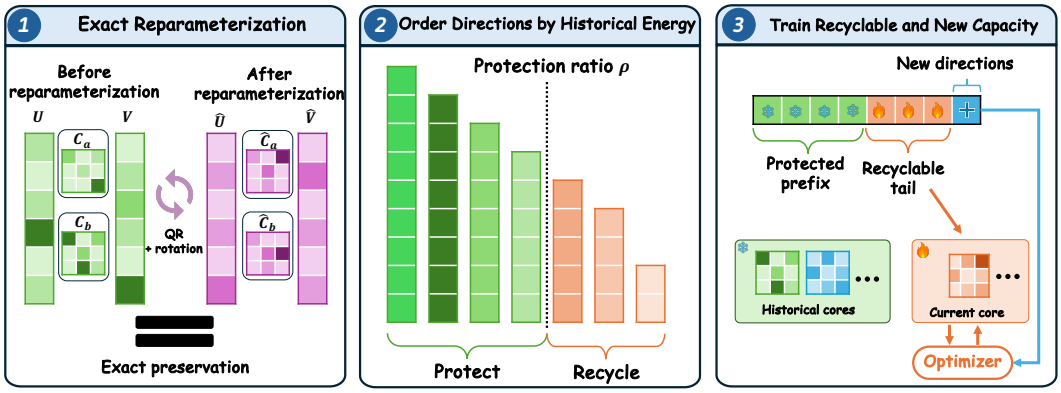


Figure 3: Adaptive subspace recycling. Historical capability parameterizations are re-expressed in energy-ordered coordinates without changing their represented functions. High-energy directions are protected, low-energy directions are recycled as trainable capacity, and new directions are appended for subsequent adaptation.

Let $\mathcal{T}_s^{t-1}$ denote the previous stages in which capability s has a stored core. For each $\tau \in \mathcal{T}_s^{t-1}$, the historical core is expressed in the QR coordinates as $B_{s,\tau} = R_U C_{s,\tau} R_V^\top$. We then aggregate historical energy across these cores along the left and right coordinates:

$$M_U = \sum_{\tau \in \mathcal{T}_s^{t-1}} B_{s,\tau} B_{s,\tau}^\top = P_U \Lambda_U P_U^\top, \qquad M_V = \sum_{\tau \in \mathcal{T}_s^{t-1}} B_{s,\tau}^\top B_{s,\tau} = P_V \Lambda_V P_V^\top. \tag{12}$$

Here P_U and P_V contain the corresponding eigenvectors, while $\Lambda_U = \mathrm{diag}(\lambda_{U,1}, \ldots)$ and $\Lambda_V = \mathrm{diag}(\lambda_{V,1}, \ldots)$ contain nonnegative eigenvalues sorted in descending order. Each eigenvalue measures the accumulated historical energy along the corresponding left or right direction. We rotate the shared bases and historical cores into these energy-ordered coordinates:

$$\widetilde{U} = Q_U P_U, \qquad \widetilde{V} = P_V^\top Q_V^\top, \qquad \widetilde{C}_{s,\tau} = P_U^\top B_{s,\tau} P_V. \tag{13}$$

Proposition E.1 shows that the QR reparameterization and energy-based rotation leave every historical capability parameterization unchanged. These operations therefore only establish an energy-ordered coordinate system for determining which directions to protect and recycle.

Protection and Recycling: Let $\rho \in (0, 1]$ denote the fraction of historical energy to preserve. Since the eigenvalues are ordered by historical energy, the protected ranks p_U and p_V are chosen as the smallest values satisfying:

$$p_U = \min\left\{ p : \frac{\sum_{j=1}^{p} \lambda_{U,j}}{\sum_j \lambda_{U,j}} \geq \rho \right\}, \qquad p_V = \min\left\{ p : \frac{\sum_{j=1}^{p} \lambda_{V,j}}{\sum_j \lambda_{V,j}} \geq \rho \right\}. \tag{14}$$

The prefixes $\widetilde{U}[:, 1{:}p_U]$ and $\widetilde{V}[1{:}p_V, :]$ capture the dominant historical directions and are frozen during subsequent training. The remaining pre-existing directions form the recyclable subspace: they retain their current values but remain trainable for the new stage. Proposition E.2 shows that the historical energy outside each protected subspace is at most a fraction $1-\rho$ of the corresponding total historical energy. A newly observed capability is initialized with active rank K_0. When capability s recurs, let K^- denote its active rank before expansion and set $K^+ = \min(K^- + \delta, K_{\max})$. Historical cores and protected basis prefixes remain frozen, while the current-stage core and basis slices $\widetilde{U}[:, p_U{+}1{:}K^+]$ and $\widetilde{V}[p_V{+}1{:}K^+, :]$ are trainable. These trainable slices comprise the recyclable low-energy directions within the previous rank K^- and up to δ newly appended directions. Historical cores are zero-padded after rank expansion, which preserves their represented parameterizations before further training as shown in Proposition E.1. Thus, recycling reuses low-energy directions as trainable capacity, while expansion provides additional directions when needed. The right panel of Figure 2 illustrates this protection, recycling, and expansion process.

4.4 Summary

During continual training, each capability in the retrieved path uses its current-stage cores $C_{s,t}^{\ell}$. After stage t, these cores are retained as a stage-specific realization of capability s, together with routing signatures based on the instruction, retrieved domain and format information, visual input, and capability-transition context. At inference time, RECAP retrieves the knowledge cards and

Table 1: Comparison with existing methods on UCIT. Task-wise results report final performance $\mathcal{A}_{i,T}$. Best and second-best values among the main comparison methods are bolded and underlined, respectively; Vanilla RAG variants are shown for reference.

Method	ImageNet-R	ArxivQA	VizWiz	IconQA	CLEVR	Flickr30k	$\overline{\mathcal{A}}\uparrow$
LoRA-FT (Hu et al., 2022)	58.03	77.63	44.39	67.40	61.77	**58.22**	61.24
O-LoRA (Wang et al., 2023)	77.50	78.07	44.50	63.13	<u>64.73</u>	<u>58.16</u>	64.35
MoELoRA (Chen et al., 2024)	70.07	77.70	44.69	50.03	54.03	57.34	58.98
ModalPrompt (Zeng et al., 2025)	51.07	87.27	48.11	39.23	46.57	42.93	52.53
CL-MoE (Huai et al., 2025)	66.33	77.00	44.78	51.87	53.53	57.42	58.49
HiDe-LLaVA (Guo et al., 2025)	<u>84.03</u>	90.73	44.43	58.93	41.37	54.25	62.29
SEFE (Chen et al., 2025)	80.83	78.00	47.01	**69.63**	**65.83**	57.92	66.54
SAME (Xie et al., 2026)	83.83	<u>91.40</u>	<u>51.33</u>	65.27	53.50	57.43	<u>67.12</u>
LoRA-FT + Vanilla RAG	37.33	79.06	44.57	71.13	63.07	58.07	58.87
SAME + Vanilla RAG	82.93	87.60	52.61	60.64	53.33	57.32	65.74
RECAP (Ours)	**85.57**	**92.60**	**60.56**	<u>68.53</u>	60.57	56.09	**70.65**

obtains the capability path $\pi(\mathbf{q}) = (s_1, \ldots, s_L)$. Let $\mathcal{T}_s^t$ denote the stages up to t that contain a stored realization of capability s. Each candidate $\tau \in \mathcal{T}_s^t$ is scored as:

$$r_{s,\tau} = \frac{\sum_{g \in \mathcal{G}(\mathbf{q},\mathbf{v})} \beta_g r_g(s,\tau)}{\sum_{g \in \mathcal{G}(\mathbf{q},\mathbf{v})} \beta_g}, \tag{15}$$

where $\mathcal{G}(\mathbf{q}, \mathbf{v})$ contains the available routing cues, including instruction similarity, retrieved domain and format information, visual similarity, and capability-transition compatibility. Here $r_g(s,\tau)$ is the normalized score for cue g, and β_g is its fixed weight. We select $\widehat{\tau}_s = \arg\max_{\tau \in \mathcal{T}_s^t} r_{s,\tau}$ and use the corresponding per-layer cores $C^{\ell}_{s,\widehat{\tau}_s}$ to recover the selected stage-specific realization of capability s. The selected capability modules are then composed along $\pi(\mathbf{q})$, enabling historical capability reuse.

5 Experiments

5.1 Experimental Setup

Datasets: We evaluate on the CoIN (Chen et al., 2024) and UCIT (Guo et al., 2025) benchmarks. CoIN contains eight heterogeneous vision–language tasks: ScienceQA (Lu et al., 2022), TextVQA (Singh et al., 2019), ImageNet (Deng et al., 2009), GQA (Hudson & Manning, 2019), VizWiz (Gurari et al., 2018), Grounding (RefCOCO) (Kazemzadeh et al., 2014; Mao et al., 2016), VQAv2 (Goyal et al., 2017), and OCR-VQA (Mishra et al., 2019). We follow the CoIN task order and evaluate after each continual stage. UCIT contains six tasks in the order ImageNet-R (Hendrycks et al., 2021), ArxivQA (Li et al., 2024), VizWiz-Caption (Gurari et al., 2018), IconQA (Lu et al., 2021), CLEVR (Lindström & Abraham, 2022), and Flickr30k (Plummer et al., 2015); we likewise evaluate after every stage.

Metrics: Following Chen et al. (2024), $\mathcal{A}_{i,t}$ denotes the performance on task i after stage t. Final average performance is defined as $\overline{\mathcal{A}} = \frac{1}{T}\sum_{i=1}^{T} \mathcal{A}_{i,T}$. For $t > 1$, average forgetting is measured by $B_t = \frac{1}{t-1}\sum_{i=1}^{t-1}(\mathcal{A}_{i,i} - \mathcal{A}_{i,t})$. Higher $\overline{\mathcal{A}}$ and lower B_t are preferred, while negative B_t indicates backward transfer.

Baselines: We compare with representative continual adaptation methods. LoRA-FT (Hu et al., 2022) provides a standard parameter-efficient baseline, while O-LoRA (Wang et al., 2023) and MoELoRA (Chen et al., 2024) study orthogonalized updates and mixtures of low-rank experts. ModalPrompt (Zeng et al., 2025), CL-MoE (Huai et al., 2025), HiDe-LLaVA (Guo et al., 2025), SEFE (Chen et al., 2025), and SAME (Xie et al., 2026) cover prompt-based adaptation, modular composition, and mechanisms for preserving or routing previously learned knowledge.

Implementation details: We use LLaVA-v1.5-7B with the CLIP ViT-L/14-336 visual encoder (Liu et al., 2023; Radford et al., 2021). We adapt the attention projections while keeping the backbone frozen. Each stage is trained for one epoch with a batch size of 8 and a learning rate of 2×10^{-4}. The factorized capability modules are initialized with rank $K_0 = 8$ and expanded by $\delta = 1$ whenever a capability recurs, with $\rho = 0.99$; the maximum active rank is 13 on UCIT and 15 on CoIN. Qwen3-Embedding-4B encodes instructions and structured knowledge cards. For each instruction, retrieval selects one domain, reasoning, and format card. The knowledge base is constructed offline using external search and an LLM, and at stage t, only shards from observed stages $1{:}t$ are available during training and evaluation. Retrieval and core-routing coefficients are provided in Appendix C.

Table 2: Comparison with existing methods on CoIN. Task-wise results report final performance $\mathcal{A}_{i,T}$. Best and second-best values among the main comparison methods are bolded and underlined, respectively; Vanilla RAG variants are shown for reference.

Method	ScienceQA	TextVQA	ImageNet	GQA	VizWiz	Grounding	VQAv2	OCR-VQA	$\overline{\mathcal{A}}\uparrow$
LoRA-FT (Hu et al., 2022)	26.00	25.38	28.51	33.07	26.52	0.10	40.00	52.92	29.06
O-LoRA (Wang et al., 2023)	75.40	52.89	71.85	47.30	37.35	7.10	61.85	61.20	51.87
MoELoRA (Chen et al., 2024)	62.02	52.05	37.21	53.12	43.32	33.22	57.92	<u>65.75</u>	50.58
ModalPrompt (Zeng et al., 2025)	68.42	56.40	41.13	61.11	50.13	36.69	66.90	59.68	55.06
CL-MoE (Huai et al., 2025)	73.28	59.94	31.80	60.22	46.98	<u>64.48</u>	**67.36**	62.39	58.31
SEFE (Chen et al., 2025)	75.35	58.66	83.10	54.25	48.85	16.75	65.35	**66.25**	58.57
HiDe-LLaVA (Guo et al., 2025)	73.20	56.92	69.28	<u>61.33</u>	50.76	59.18	<u>67.12</u>	64.76	62.82
SAME (Xie et al., 2026)	<u>78.35</u>	<u>60.69</u>	<u>90.21</u>	**61.70**	<u>54.13</u>	59.87	66.04	63.59	<u>66.82</u>
LoRA-FT + Vanilla RAG	24.16	26.14	18.69	39.60	24.85	6.74	48.09	50.88	29.89
SAME + Vanilla RAG	66.51	60.26	90.25	61.55	61.23	59.79	65.98	60.70	65.78
RECAP (Ours)	**79.70**	**61.89**	**96.89**	61.12	**58.12**	**71.18**	65.97	57.90	**69.10**

5.2 BENCHMARK COMPARISON

Tables 1 and 2 report final task-wise performance on UCIT and CoIN, respectively. On UCIT, RECAP achieves the highest final average performance of 70.65, outperforming SAME by 3.53 points, with the best results on ImageNet-R, ArxivQA, and VizWiz. Although several baselines remain stronger on individual tasks such as IconQA, CLEVR, and Flickr30k, RECAP delivers the strongest overall performance across the heterogeneous task stream. On the longer CoIN stream, RECAP achieves an average of 69.10, exceeding SAME by 2.28 points, and obtains the best results on ScienceQA, TextVQA, ImageNet, VizWiz, and visual grounding. Other methods remain competitive on GQA, VQAv2, and OCR-VQA. Overall, the consistent gains across both benchmarks demonstrate the effectiveness of RECAP across diverse continual tasks.

Additionally, we investigate whether the gains of RECAP can be attributed merely to access to external knowledge. To isolate this factor, we augment LoRA-FT and SAME with a vanilla RAG pipeline (Lewis et al., 2020), giving them access to the same knowledge base as RECAP during both training and evaluation. Despite this matched knowledge access, RECAP continues to substantially outperform both RAG-augmented baselines. This result suggests that the improvement does not stem simply from retrieving additional external information. Rather, the advantage lies in how RECAP operationalizes the retrieved knowledge: instead of treating retrieval as auxiliary context, RECAP uses it to identify, compose, and reuse instance-relevant capabilities across continual stages. In this way, retrieved knowledge serves not only as additional evidence for prediction, but also as a structured interface for guiding capability selection and transfer throughout continual learning.

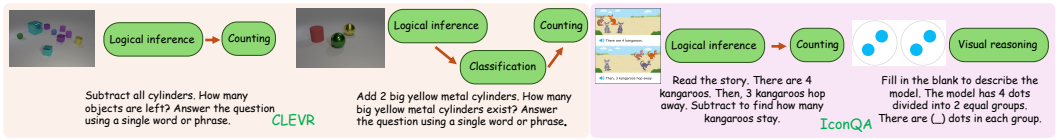


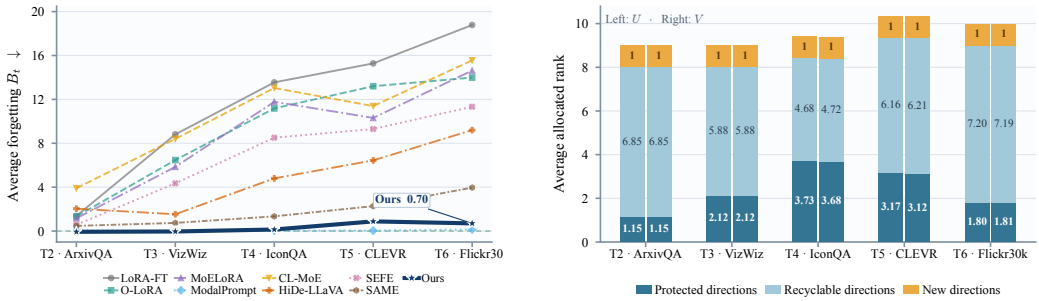
Figure 4: Instance-level capability paths for examples sampled from the UCIT test set. Samples within the same task may require different capability compositions, while shared capabilities can be reused across tasks.

5.3 FURTHER ANALYSIS

Instance-Level Capability Paths: Figure 4 illustrates capability paths derived from retrieved reasoning knowledge. The two CLEVR examples show that different instructions within the same task can require different capability compositions: one follows logical inference → counting, while the other additionally requires classification before counting. The IconQA example shares the logical inference → counting path with CLEVR, showing that capability compositions can also be reused across different tasks. These examples illustrate how reasoning knowledge provides instance-level structure for capability composition and reuse in RECAP.

Continual Retention: Figure 5a shows that RECAP maintains consistently low forgetting throughout the UCIT stream. It exhibits slight backward transfer after stages 2 and 3, while average forgetting remains only 0.14, 0.88, and 0.70 after stages 4, 5, and 6, respectively. Compared with the substantially larger forgetting accumulated by most baselines, these results indicate that previously learned capabilities remain stable as new stages are introduced.

Subspace Allocation: Figure 5b summarizes the realized subspace allocation across the five updates after the initial stage, averaged over recurring capability-projection pairs. Since the protection rule in Equation 14 is applied independently to each capability and adapted projection, the protected rank varies across updates rather than using a fixed coordinate count. Despite the high protection ra-



(a) Average forgetting across stages.

(b) Subspace allocation under adaptive protection.

Figure 5: Continual retention and adaptive subspace allocation on UCIT. Panel (a) reports average forgetting after each stage, where lower values indicate stronger retention. Panel (b) visualizes the protected, recyclable, and newly appended basis directions averaged over recurring capability-projection pairs.

tio $\rho = 0.99$, the protected prefixes occupy only about 14%–44% of the pre-expansion rank, leaving roughly 56%–86% of the existing directions recyclable before new capacity is added. This shows that historical energy is concentrated in a compact subspace, allowing RECAP to preserve dominant historical directions while retaining substantial capacity for subsequent adaptation.

Ablation Study: Table 3 evaluates each component of RECAP. Removing domain guidance and eliminating reasoning-guided ordering by replacing the sequential path with parallel capability composition both reduce performance, confirming the complementary benefits of generation guidance and ordered capability execution. Disabling adaptive subspace recycling leads to substantial degradation, demonstrating the importance of preserving historical capability information while retaining sufficient capacity for subsequent adaptation. Removing stage-specific core routing and forcing activated capabilities to use final-stage cores further degrades performance, as cores capture distinct stage-specific realizations, highlighting the need for effective routing in task-free capability reuse.

Protection Ratio Sensitivity: Table 4 evaluates the sensitivity of RECAP to the historical-energy protection ratio ρ on UCIT using a held-out validation set, with the realized protected rank indicating how much subspace is preserved under each setting. Lower protection ratios ($\rho \leq 0.90$) retain only 1.01–1.34 directions on average and yield lower performance than the default setting. Increasing ρ to 0.99 raises the protected rank to 2.53 and achieves the best average performance of 70.67, indicating that most historical energy is concentrated within a compact subspace. In contrast, full protection at $\rho = 1.0$ increases the protected rank sharply to 8.37 and reduces performance to 66.52, as substantially less residual capacity remains for new adaptation. These results favor a high but non-complete protection ratio that balances historical retention and plasticity.

Table 3: Component ablations of RECAP on UCIT. Higher is better.

Variant	$\overline{\mathcal{A}}\uparrow$
RECAP	**70.65**
w/o Domain Guidance	70.02
w/o Reasoning-Guided Ordering	70.08
w/o Adaptive Subspace Recycling	49.88
w/o Stage-specific Core Routing	31.50

Table 4: Sensitivity to the historical-energy ratio ρ, with the resulting protected rank.

ρ	Protected Rank	$\overline{\mathcal{A}}\uparrow$
0.50	1.01	67.49
0.70	1.10	66.91
0.90	1.34	69.33
0.99 (default)	2.53	**70.67**
1.00	8.37	66.52

6 CONCLUSION

We presented RECAP, a retrieval-guided framework for multimodal continual instruction tuning that leverages external knowledge to organize capability composition and reuse across continual stages. Retrieved domain knowledge provides generation guidance, while reasoning knowledge specifies ordered capability operations for instance-level composition, with format knowledge further supporting retrieval and routing. To enable stable cross-stage reuse, adaptive subspace recycling represents recurring capabilities through shared bases and stage-specific cores, protecting high-energy historical directions while retaining residual capacity for subsequent adaptation. Experiments on UCIT and CoIN show that RECAP consistently outperforms representative baselines while maintaining strong continual retention and effective adaptation to newly observed tasks.

Limitations. The current knowledge base represents external information as concise textual cards and does not model richer sources such as tables, diagrams, and structured knowledge graphs. Extending knowledge construction and retrieval to these forms is a natural direction for future work.

AI USE STATEMENT

In this work, we used generative AI tools for research execution. Specifically, GPT-4o mini was used within RECAP for knowledge-base construction, including knowledge-card generation, capability canonicalization, domain-card consolidation, and prompt-guidance generation. Additionally, we used generative AI tools to check language, notation consistency, and presentation clarity throughout the manuscript. We reviewed and verified all AI-assisted work. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*, 2024.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George B. M. van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2206–2240. PMLR, 2022.

Cheng Chen, Junchen Zhu, Xu Luo, Hengtao Shen, Lianli Gao, and Jingkuan Song. CoIN: A benchmark of continual instruction tuning for multimodel large language model. *arXiv preprint arXiv:2403.08350*, 2024.

Jinpeng Chen, Runmin Cong, Yuzhi Zhao, Hongzheng Yang, Guangneng Hu, Horace Ho Shing Ip, and Sam Kwong. Sefe: Superficial and essential forgetting eliminator for multimodal continual instruction tuning. *arXiv preprint arXiv:2505.02486*, 2025.

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale N Fung, and Steven Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning. *Advances in neural information processing systems*, 36:49250–49267, 2023.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1762–1777. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.99.

Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6904–6913, 2017.

Haiyang Guo, Fanhu Zeng, Ziwei Xiang, Fei Zhu, Da-Han Wang, Xu-Yao Zhang, and Cheng-Lin Liu. HiDe-LLaVA: Hierarchical decoupling for continual instruction tuning of multimodal large language model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, 2025.

Danna Gurari, Qing Li, Abigale J Stangl, Anhong Guo, Chi Lin, Kristen Grauman, Jiebo Luo, and Jeffrey P Bigham. Vizwiz grand challenge: Answering visual questions from blind people. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3608–3617, 2018.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: Retrieval-augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 3929–3938, 2020.

Dan Hendrycks, Steven Basart, Norman Mu, Saurav Kadavath, Frank Wang, Evan Dorundo, Rahul Desai, Tyler Zhu, Samyak Parajuli, Mike Guo, et al. The many faces of robustness: A critical analysis of out-of-distribution generalization. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 8340–8349, 2021.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Tianyu Huai, Jie Zhou, Xingjiao Wu, Qin Chen, Qingchun Bai, Ze Zhou, and Liang He. CL-MoE: Enhancing multimodal large language model with dual momentum mixture-of-experts for continual visual question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6700–6709, 2019.

Gautier Izacard and Edouard Grave. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pp. 874–880. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.eacl-main.74.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Few-shot Learning with Retrieval Augmented Language Models. 2022. URL `http://arxiv.org/abs/2208.03299`.

Jushaan Singh Kalra, Xinran Zhao, To Eun Kim, Fengyu Cai, Fernando Diaz, and Tongshuang Wu. Mor: Better handling diverse queries with a mixture of sparse, dense, and human retrievers. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 11982–12001, 2025.

Sahar Kazemzadeh, Vicente Ordonez, Mark Matten, and Tamara Berg. Referitgame: Referring to objects in photographs of natural scenes. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 787–798, 2014.

Dohyeon Lee, Seung-won Hwang, Kyungjae Lee, Seungtaek Choi, and Sunghyun Park. On complementarity objectives for hybrid retrieval. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13357–13368, 2023.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474, 2020.

Lei Li, Yuqi Wang, Runxin Xu, Peiyi Wang, Xiachong Feng, Lingpeng Kong, and Qi Liu. Multimodal arxiv: A dataset for improving scientific comprehension of large vision-language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14369–14387, 2024.

Adam Dahlgren Lindström and Savitha Sam Abraham. Clevr-math: A dataset for compositional language, visual and mathematical reasoning. *arXiv preprint arXiv:2208.05358*, 2022.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Pan Lu, Liang Qiu, Jiaqi Chen, Tony Xia, Yizhou Zhao, Wei Zhang, Zhou Yu, Xiaodan Liang, and Song-Chun Zhu. Iconqa: A new benchmark for abstract diagram understanding and visual language reasoning. *arXiv preprint arXiv:2110.13214*, 2021.

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in Neural Information Processing Systems*, 35:2507–2521, 2022.

Junhua Mao, Jonathan Huang, Alexander Toshev, Oana Camburu, Alan L Yuille, and Kevin Murphy. Generation and comprehension of unambiguous object descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 11–20, 2016.

Anand Mishra, Shashank Shekhar, Ajeet Kumar Singh, and Anirban Chakraborty. Ocr-vqa: Visual question answering by reading text in images. In *2019 international conference on document analysis and recognition (ICDAR)*, pp. 947–952. IEEE, 2019.

OpenAI. GPT-4o mini: Advancing cost-efficient intelligence. `https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/`, July 2024.

Bryan A Plummer, Liwei Wang, Chris M Cervantes, Juan C Caicedo, Julia Hockenmaier, and Svetlana Lazebnik. Flickr30k entities: Collecting region-to-phrase correspondences for richer image-to-sentence models. In *Proceedings of the IEEE international conference on computer vision*, pp. 2641–2649, 2015.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021.

Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. Towards vqa models that can read. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8317–8326, 2019.

Shengbang Tong, David Fan, Jiachen Li, Yunyang Xiong, Xinlei Chen, Koustuv Sinha, Michael Rabbat, Yann LeCun, Saining Xie, and Zhuang Liu. Metamorph: Multimodal understanding and generation via instruction tuning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17001–17012, 2025.

Xiao Wang, Tianze Chen, Qiming Ge, Han Xia, Rong Bao, Rui Zheng, Qi Zhang, Tao Gui, and Xuan-Jing Huang. Orthogonal subspace learning for language model continual learning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 10658–10671, 2023.

Zhen-Hao Xie, Jun-Tao Tang, Yu-Cheng Shi, Han-Jia Ye, De-Chuan Zhan, and Da-Wei Zhou. SAME: Stabilized mixture-of-experts for multimodal continual instruction tuning. In *International Conference on Machine Learning*, 2026.

Fanhu Zeng, Fei Zhu, Haiyang Guo, Xu-Yao Zhang, and Cheng-Lin Liu. ModalPrompt: Towards efficient multimodal continual instruction tuning with dual-modality guided prompt. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025.

Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Guoyin Wang, et al. Instruction tuning for large language models: A survey. *ACM Computing Surveys*, 58(7):1–36, 2026.

APPENDIX

A TRAINING AND INFERENCE ALGORITHMS

Algorithms 1 and 2 summarize the complete procedure for stage-wise training and task-free inference.

Algorithm 1 Stage-wise training of RECAP

Require: Stage data $\mathcal{D}_t$, knowledge shard $\mathcal{K}^t$, and parameters retained from stages $1{:}t-1$
Update the available knowledge: $\mathcal{K}^{1:t} \leftarrow \mathcal{K}^{1:t-1} \cup \mathcal{K}^t$
for all $(\mathbf{v}, \mathbf{q}, \mathbf{y}) \in \mathcal{D}_t$ **do**
  Retrieve one domain, format, and reasoning card
  Form $\widetilde{\mathbf{q}}$ and obtain the capability path $\pi(\mathbf{q}) = (s_1, \ldots, s_L)$
end for
Collect the capabilities activated by the current-stage paths
for all capabilities s activated at stage t **do**
  if s has a stored historical realization **then**
    Apply the QR reparameterization and energy-based rotation in equation 11–equation 13
    Compute p_U, p_V and expand the active rank by up to δ
  else
    Initialize the shared bases with active rank K_0
  end if
  Initialize the current-stage cores $C^{\ell}_{s,t}$
end for
Optimize the current-stage cores and trainable basis directions using equation 21 and equation 22
Keep historical cores and protected basis directions fixed
Store the current capability realizations and their routing signatures

Algorithm 2 Task-free inference with RECAP

Require: Query–image pair $(\mathbf{q}, \mathbf{v})$, available knowledge $\mathcal{K}^{1:t}$, and stored capability realizations
Retrieve the domain, format, and reasoning cards
Form $\widetilde{\mathbf{q}}$ and obtain the capability path $\pi(\mathbf{q}) = (s_1, \ldots, s_L)$
for $j = 1, \ldots, L$ **do**
  Score the stored realizations of s_j using equation 15
  Select $\widehat{\tau}_{s_j}$ and use the corresponding per-layer cores $C^{\ell}_{s_j, \widehat{\tau}_{s_j}}$
end for
Compose the selected capability modules in path order and decode the response

B KNOWLEDGE BASE CONSTRUCTION

Overview: For each continual stage j, we construct a knowledge shard from the training instructions in $\mathcal{D}_j$. Instructions are first grouped by their inferred answer schemas and then clustered by semantic similarity within each group using the embedding model ϕ. For each cluster, representative instructions and externally retrieved evidence, which may provide visual context and background information, are summarized by an LLM into domain, reasoning, and format cards. Domain cards contain reusable knowledge and prompt guidance, reasoning cards specify ordered capability operations and their applicable contexts, and format cards define answer schemas and output constraints. Generated capabilities are canonicalized against the existing capability registry to promote reuse, while semantically compatible domain cards are consolidated to reduce redundancy. Algorithm 3 summarizes the complete procedure.

Algorithm 3 Construction of the knowledge base.

Require: Training sets $\{\mathcal{D}_j\}_{j=1}^{T}$, embedding model ϕ, external search, LLM $\mathcal{L}$, capability registry $\mathcal{R}$, allowed answer schemas $\mathcal{Y}$
Ensure: Knowledge shards $\{\mathcal{K}_{\text{dom}}^{j}, \mathcal{K}_{\text{rea}}^{j}, \mathcal{K}_{\text{fmt}}^{j}\}_{j=1}^{T}$
for $j = 1, \ldots, T$ **do**
  Collect instructions $\mathcal{Q}_j$ from $\mathcal{D}_j$
  Infer answer schemas and group instructions accordingly
  Encode instructions with ϕ and cluster semantically similar instructions within each schema group
  for each instruction cluster $\mathcal{C}_a$ **do**
    Summarize the cluster intent and relevant concepts
    Retrieve external evidence $\mathcal{E}_a$ using queries constructed from the task name, cluster intent, and concepts
    Generate domain, reasoning, and format cards using Prompt A
    Canonicalize generated capabilities against $\mathcal{R}$ using Prompt B
    Update $\mathcal{R}$ with retained new capabilities
    Validate grounding, schema consistency, and path completeness
  end for
  Identify semantically compatible domain-card groups
  Consolidate compatible domain cards using Prompt C
  Generate domain-card prompt guidance using Prompt D
  Encode domain and reasoning card representations with ϕ
  for each instruction cluster $\mathcal{C}_a$ **do**
    $\mathbf{p}_a \leftarrow \text{norm}\left(\frac{1}{|\mathcal{C}_a|}\sum_{\mathbf{q}_i \in \mathcal{C}_a} \phi(\mathbf{q}_i)\right)$
    Associate $\mathbf{p}_a$ with its resulting domain and reasoning cards
  end for
  Consolidated domain cards retain the prototypes of all member clusters
  Store the resulting cards in $\mathcal{K}_{\text{dom}}^{j}$, $\mathcal{K}_{\text{rea}}^{j}$, and $\mathcal{K}_{\text{fmt}}^{j}$
end for
At stage t, expose only $\mathcal{K}_c^{1:t} = \bigcup_{j=1}^{t} \mathcal{K}_c^{j}$, $c \in \{\text{dom}, \text{rea}, \text{fmt}\}$

B.1 Knowledge-Card Generation

External Search: For each instruction cluster, search queries are constructed from the task name, cluster intent, and concepts. The retrieved textual evidence provides relevant background information and may include descriptions of visual context associated with the task and concepts.

Card Generation: The instruction cluster, retrieved evidence, currently available capabilities, and answer-schema constraints are provided to GPT-4o mini (OpenAI, 2024). Prompt A generates structured domain, reasoning, and format cards, while Prompts B-D support capability canonicalization and subsequent card refinement when applicable.

System prompt.

```
You construct source-grounded knowledge cards for multimodal continual learning.
Generate reusable knowledge and a complete reasoning path grounded in the provided evidence
      and instruction cluster.
Do not answer any sample instruction.
Return compact valid JSON only.
Prefer an existing allowed capability whenever it performs the same reusable operation.
Introduce a new capability only when the required operation is not covered by the existing
     capability list.
Capabilities must be atomic, reusable operations rather than task-specific procedures.
Decompose composite reasoning processes into separate capability steps.
Every answer schema must exactly match the allowed schema list.
```

User prompt template.

```
Task: <task_name>
Reasoning intent: <intent>
Schema hint: <schema_hint>
Concept hints: <concept_hints>

Instruction cluster:
- <instruction_1>
- ...
```

```
- <instruction_n>

Allowed capabilities:
<capability_list>

Allowed schemas:
<schema_list>

Retrieved evidence:
<source_records>

Return JSON with the following structure:
{
  "domain_cards": [
    {
      "name": "",
      "knowledge": "",
      "concepts": [],
      "keywords": [],
      "source_refs": []
    }
  ],
  "reasoning_cards": [
    {
      "name": "",
      "domains": [],
      "schemas": [],
      "pattern": [
        {
          "capability": "",
          "operation": ""
        }
      ],
      "source_refs": []
    }
  ],
  "format_cards": [
    {
      "schema": "",
      "format_template": "",
      "validation_rule": ""
    }
  ]
}

Requirements:
- Generalize to reusable knowledge rather than instruction-specific facts.
- Do not include final answers to any instruction.
- Domain and reasoning cards must be supported by the retrieved evidence.
- The reasoning path must contain only atomic capabilities.
- The reasoning path must be sufficient to complete the requested reasoning intent.
- Reuse an existing capability whenever its core operation already covers a required step.
- At least one reasoning card and one format card must match the requested schema.
- Include only operations supported by the instruction cluster and retrieved evidence.
- Use the shortest complete reasoning path.
```

B.2 Capability Canonicalization

Reasoning cards from different instruction clusters may describe the same reusable operation using different names. We therefore compare newly generated capabilities with the existing registry and reuse a canonical capability only when their core operations and output roles are equivalent.

Prompt B: capability canonicalization. System prompt.

```
Determine whether each generated capability represents the same reusable operation as an
     existing capability.
Use REUSE only when the core operation and output role are substantially equivalent.
Related, broader, narrower, or task-specific operations must remain separate.
Return valid JSON only.
For each candidate, return its capability, decision, canonical_capability, and confidence.
The decision must be either REUSE or KEEP_SEPARATE.
```

Input/output template.

```
Input:
{
  "candidates": [
```

```
    {
      "capability": "<generated_capability>",
      "operation": "<operation_description>",
      "schemas": <schemas>
    }
  ],
  "existing_capabilities": [
    {
      "capability": "<canonical_capability>",
      "operation": "<operation_description>"
    }
  ]
}

Output:
{
  "decisions": [
    {
      "capability": "",
      "decision": "",
      "canonical_capability": "",
      "confidence": 0.0
    }
  ]
}
```

Only high-confidence `REUSE` decisions whose canonical capability already exists in the registry are applied. This canonicalization provides persistent capability identities across knowledge cards constructed at different continual stages.

B.3 Domain-Card Consolidation

Domain cards from different instruction clusters may encode equivalent reusable knowledge. We first form conservative candidate groups using semantic and embedding compatibility, and then use the LLM to determine whether cards within each group express the same knowledge rule. Cards with distinct rules remain separate even when they share similar terminology or answer formats.

Prompt C: domain-card consolidation. System prompt.

```
Merge Domain knowledge cards only when they express the same semantic intent and the same
     reusable knowledge rule.
A shared answer schema, reasoning path, name, or superficial vocabulary is not sufficient
     for merging.
Use representative instructions as the primary evidence when the knowledge description is
     ambiguous.
Do not invent new knowledge.
Every input card must appear in exactly one output group.
Return valid JSON only.
For each group, return member_uids, name, knowledge, concepts, and keywords.
```

Input/output template.

```
Input:
{
  "cards": [
    {
      "id": "",
      "knowledge": "",
      "concepts": [],
      "semantic_intent": "",
      "representative_instructions": []
    }
  ]
}

Output:
{
  "groups": [
    {
      "member_ids": [],
      "name": "",
      "knowledge": "",
      "concepts": [],
      "keywords": []
    }
```

```
  ]
}
```

This consolidation reduces redundant domain knowledge while preserving distinct semantic rules that may share similar terminology or response formats.

B.4 PROMPT-GUIDANCE GENERATION

Each consolidated domain card is converted into concise operational guidance that can be prepended to the retrieved instruction through equation 7. Prompt D restricts the guidance to card-grounded concepts and rules, without introducing sample-specific answers or output-format instructions.

Prompt D: domain-card guidance. System prompt.

```
Create prompt_guidance for every supplied Domain knowledge card.
Ground the guidance only in the supplied knowledge and concepts.
Rewrite the card as one concise operational instruction describing the evidence, concepts,
     distinctions, or stable rules the model should use.
Do not reveal an answer, rationale, option, or instruction-specific fact.
Do not include answer-format instructions.
Avoid generic advice and use concrete operations only when supported by the card.
Return one concise English paragraph for each card in valid JSON.
```

Input/output template.

```
Input:
{
  "cards": [
    {
      "id": "",
      "knowledge": "",
      "concepts": []
    }
  ]
}

Output:
{
  "items": [
    {
      "id": "",
      "prompt_guidance": ""
    }
  ]
}
```

B.5 RETRIEVAL REPRESENTATIONS

The final knowledge store maintains card-text representations together with instruction-cluster prototypes. Domain-card representations contain names, knowledge, concepts, and keywords, while reasoning-card representations contain applicable domains, schemas, and ordered capability operations. Both are encoded by the frozen embedding model ϕ.

For each instruction cluster $\mathcal{C}_a$, its prototype is obtained by averaging the instruction embeddings followed by ℓ_2-normalization:

$$\mathbf{p}_a = \mathrm{norm}\left(\frac{1}{|\mathcal{C}_a|}\sum_{\mathbf{q}_i \in \mathcal{C}_a} \phi(\mathbf{q}_i)\right), \tag{16}$$

where $\mathrm{norm}(\cdot)$ denotes ℓ_2-normalization. Each domain or reasoning card retains the prototypes of its source instruction clusters; a consolidated domain card may therefore contain multiple prototypes. We denote this set by $\mathcal{P}(k)$. Retrieval combines prototype similarity with the card-content signals described in Section 4.1 and Appendix C.

C Fixed Retrieval and Core-Selection Coefficients

Table 5 summarizes the fixed coefficients used by ReCAP across all continual tasks and stages. Domain and reasoning retrieval combine instruction-cluster prototype similarity with card-content matching using $\alpha_{\mathrm{dom}} = \alpha_{\mathrm{rea}} = 0.80$ in equation 6, while format retrieval is performed separately using lexical and answer-schema matching. The same coefficients are used throughout the continual stream.

Retrieval Signals: At stage t, retrieval accesses only the knowledge accumulated from observed stages $1{:}t$. For a domain or reasoning card k, the prototype score is defined as:

$$s_{\mathrm{proto}}(\mathbf{q}, k) = \max_{\mathbf{p} \in \mathcal{P}(k)} \cos(\phi(\mathbf{q}), \mathbf{p}), \tag{17}$$

where $\mathcal{P}(k)$ contains the instruction-cluster prototypes associated with card k. In addition to prototype similarity, each card type uses a weighted combination of normalized content-matching signals:

$$h_c(\mathbf{q}, k) = \sum_{m \in \mathcal{M}_c} \lambda_{c,m} s_m(\mathbf{q}, k), \qquad c \in \{\mathrm{dom}, \mathrm{rea}, \mathrm{fmt}\}, \tag{18}$$

where $\mathcal{M}_c$ denotes the signals used for card type c, and $\lambda_{c,m}$ denotes their fixed weights.

For domain retrieval, the content signals include card-text semantic similarity, keyword overlap, and concept overlap. Card-text similarity captures overall instruction–card relevance, while keyword and concept matching provide lexical and structured compatibility. The resulting content score is combined with prototype similarity according to equation 6, and the highest-scoring domain card is retained.

Format retrieval uses keyword overlap and answer-schema compatibility. Keyword matching captures explicit response cues in the instruction, while schema matching measures compatibility with the answer form represented by the format card. When an instruction explicitly specifies an answer schema, matching format cards are given priority.

After retrieving the domain and format cards, their information is used as additional cues for reasoning retrieval. Reasoning-card matching uses card-text semantic similarity, keyword overlap, domain compatibility, schema compatibility, and concept overlap. The domain and schema signals measure whether a candidate reasoning procedure is applicable to the retrieved context, while the remaining signals measure its semantic and lexical relevance to the instruction. The resulting content score is combined with prototype similarity according to equation 6, and the highest-scoring reasoning card is retained.

Stage-Specific Core Selection: For each activated capability s, ReCAP scores the stored stage-specific realizations $\tau \in \mathcal{T}_s^t$ using four cues in $\mathcal{G}(\mathbf{q}, \mathbf{v})$. The instruction cue measures similarity between the current instruction and the stored instruction signature of a candidate realization; the metadata cue measures compatibility with the retrieved domain and format information; the image cue measures visual similarity to its stored visual signature; and the path-transition cue measures compatibility with the realization selected for the preceding capability.

Let $h = (s, \tau)$ denote a stage-specific realization and h_{prev} the realization selected for the preceding capability. For non-initial positions in a capability path, the transition cue is computed from the empirical transition frequency as:

$$P_{\mathrm{trans}}(h \mid h_{\mathrm{prev}}) = \frac{N(h_{\mathrm{prev}} \to h)}{\sum_{h'} N(h_{\mathrm{prev}} \to h')}. \tag{19}$$

Any unavailable cue is omitted and the remaining coefficients are renormalized. The highest-scoring realization $\widehat{\tau}_s$ is selected, and its corresponding per-layer cores $C^{\ell}_{s,\widehat{\tau}_s}$ are used for capability execution.

Coefficient Design: The coefficients are fixed globally rather than adapted to individual tasks or continual stages. Their relative magnitudes follow a simple evidence hierarchy based on how directly each signal matches the current instance to a knowledge card or stored realization. Motivated by prior hybrid-retrieval studies showing that semantic and lexical signals provide complementary relevance evidence (Lee et al., 2023; Kalra et al., 2025), we use semantic matching as the primary

Table 5: Fixed card-content matching and stage-specific core-selection coefficients used by RECAP. Signals are listed in the same order as their coefficients.

Component	Signals	Coefficients
Domain card	card-text semantic, keyword overlap, concept overlap	0.58, 0.32, 0.10
Reasoning card	card-text semantic, keyword overlap, domain match, schema match, concept overlap	0.48, 0.27, 0.10, 0.10, 0.05
Format card	keyword overlap, schema-rule match	0.60, 0.40
Core selection	instruction cue, domain/format metadata, image cue, path transition	0.60, 0.15, 0.20, 0.05

retrieval signal while retaining lexical and structured cues for disambiguation. Prototype similarity receives the dominant interpolation weight $\alpha = 0.80$ because it directly compares the query with the source instruction clusters from which a card is constructed. Within card-content matching, semantic and keyword signals receive larger weights, while concept, domain, and schema signals provide auxiliary compatibility constraints. For core selection, instruction similarity serves as the primary task-free cue, visual similarity provides complementary instance-level evidence, and retrieved metadata and path-transition statistics provide additional context. This shared configuration reflects the heterogeneous instructions, response formats, and visual content encountered in MCIT, and its robustness to alternative weight configurations is evaluated on a held-out validation set in Section C.1.

C.1 Sensitivity to Retrieval and Routing Coefficients

We further examine the sensitivity of RECAP to the fixed retrieval and stage-specific core-selection coefficients on UCIT using a held-out validation set. Rather than varying individual coefficients exhaustively, we compare the default configuration with several representative alternatives while keeping the underlying retrieval and routing signals unchanged. For retrieval, we consider balanced weighting across the available evidence and a semantic-emphasized configuration; for core selection, we compare balanced cue weighting with increased emphasis on instruction similarity.

Table 6: Sensitivity to retrieval and stage-specific core-selection coefficients on UCIT. Weight tuples follow the signal order in Table 5, with $\alpha = \alpha_{\mathrm{dom}} = \alpha_{\mathrm{rea}}$. Routing accuracy is computed over all routed capability positions.

Setting	α	Domain	Reasoning	Format	Core Selection	Routing Acc. (%)	$\overline{\mathcal{A}}\uparrow$
Default	0.80	0.58, 0.32, 0.10	0.48, 0.27, 0.10, 0.10, 0.05	0.60, 0.40	0.60, 0.15, 0.20, 0.05	99.304	70.67
Balanced retrieval	0.50	0.33, 0.33, 0.34	0.20, 0.20, 0.20, 0.20, 0.20	0.50, 0.50	default	66.354	67.41
Semantic-emphasized retrieval	0.90	0.65, 0.25, 0.10	0.55, 0.20, 0.10, 0.10, 0.05	default	default	99.141	70.34
Balanced routing	default	default	default	default	0.25, 0.25, 0.25, 0.25	99.761	70.51
Instruction-emphasized routing	default	default	default	default	0.70, 0.10, 0.15, 0.05	98.982	70.22

Routing accuracy is computed over all routed capability positions. For each position, we compare the stage-specific realization selected by the router with the reference realization associated with the corresponding task and capability. A position is counted as correct only when the two realization identifiers match. If either realization is unavailable, the position is marked as *unknown* and treated as an error. The reported value is the micro accuracy over all routed positions, with *unknown* positions included in the denominator. Task identity is used only to construct this evaluation reference and is never provided to the router.

As shown in Table 6, RECAP remains stable under most variations of the retrieval and core-selection coefficients, with high routing accuracy and only minor changes in downstream performance. The main exception is Balanced retrieval, which assigns approximately equal importance to the available retrieval signals and leads to lower routing accuracy and average performance. Further inspection shows that this degradation is largely associated with *unknown* positions, where the retrieved reasoning path contains a capability for which no corresponding stage-specific realization is available under the evaluation reference. This suggests that the primary failure arises from changes in capability-path retrieval rather than from stage-specific core selection itself. Overall, these results indicate that RECAP is robust to reasonable coefficient variations, while appropriate weighting of retrieval signals remains important for reliable capability-path construction.

D KNOWLEDGE INTERFACE AND STAGE-WISE EXECUTION

D.1 STORED FIELDS AND FORWARD CONSUMERS

Table 7 summarizes the typed interface. Only Domain Guidance modifies the language-model input; reasoning and format records remain structured control state.

Table 7: Knowledge-card fields and their consumers.

Card	Principal stored fields	Forward consumer	In prompt?
Domain	concepts, keywords, knowledge, *prompt guidance*	generator	prompt guidance only
Reasoning	domains, schemas, ordered operations	capability-path constructor	no
Format	schema, template, validation rule	reasoning retrieval, core router	no

For nonempty domain guidance g_{dom}, the augmented instruction is

$$\widetilde{\mathbf{q}} = \texttt{[GUIDANCE: } g_{\mathrm{dom}} \texttt{]}\ \mathbf{q}. \tag{20}$$

Retrieval uses the original $\mathbf{q}$, and the prefix contains neither a sample answer nor a dataset identifier. Removing it therefore leaves retrieval and routing unchanged. An explicit format request in $\mathbf{q}$ is not duplicated by a retrieved template.

D.2 TRAINING AND INFERENCE ORDER

At stage t, RECAP (1) adds the validated shard to $\mathcal{K}^{1:t}$; (2) retrieves and canonicalizes capability paths; (3) re-expresses recurring adapters, protects high-energy prefixes, expands their rank, and initializes current cores; (4) optimizes current cores and unprotected basis slices; and (5) stores multi-signal core-selection prototypes.

After stage t, inference uses the corresponding cumulative collection $\mathcal{K}^{1:t}$. It retrieves the three card types, optionally prepends Domain Guidance, maps the Reasoning Path to capabilities, selects a historical core independently for every capability, and activates the selected updates in path order before autoregressive generation. Neither this procedure nor the core score in equation 15 receives a task label.

For completeness, the stage-wise training objective and parameter masks used by the algorithms below are

$$\mathcal{L}_t = -\sum_{(\mathbf{v},\mathbf{q},\mathbf{y})\in\mathcal{D}_t} \log p_{\Theta,\mathcal{U}^{1:t}_{\pi(\mathbf{q})}}\big(\mathbf{y} \mid \mathbf{v}, \widetilde{\mathbf{q}}\big), \tag{21}$$

where Θ is the frozen backbone and $\mathcal{U}^{1:t}_{\pi(\mathbf{q})}$ contains the factorized updates activated by the ordered path. For a recurring capability, the masks are

$$\begin{aligned} &C_{c,t}[1{:}K^+, 1{:}K^+] && \text{trainable},\\ &U_c[:, p_U{+}1{:}K^+], \quad V_c[p_V{+}1{:}K^+, :] && \text{trainable},\\ &\{C_{c,\tau}\}_{\tau<t}, \quad U_c[:, 1{:}p_U], \quad V_c[1{:}p_V, :] && \text{frozen}. \end{aligned} \tag{22}$$

Only capabilities activated by the current path receive gradients; inactive capability realizations remain unchanged.

E THEORETICAL PROPERTIES OF ADAPTIVE SUBSPACE RECYCLING

We establish two properties of adaptive subspace recycling. First, we show that the QR reparameterization, energy-based rotation, and zero-padded rank expansion preserve every historical capability parameterization exactly before subsequent optimization. Second, we show that the protection rule bounds the historical energy exposed to the recyclable subspace. Throughout this section, we follow the notation of Section 4.3. For a fixed capability s and layer ℓ, we omit the capability and layer superscripts on the shared bases U_s^ℓ and V_s^ℓ for clarity. Here, $\tau \in \mathcal{T}_s^{t-1}$ indexes the previous stages in which capability s has a stored core.

E.1 Exact Preservation under Reparameterization

Low-rank factorizations are not coordinate-unique. For any invertible matrices $G_U, G_V \in \mathbb{R}^{K_s \times K_s}$:

$$UC_{s,\tau}V = (UG_U)\left(G_U^{-1}C_{s,\tau}G_V^{-1}\right)(G_V V). \tag{23}$$

Thus, directly protecting individual coordinates of the original bases would depend on the particular factorization. The QR reparameterization and energy-based rotation in equation 11–equation 13 instead establish an orthonormal, energy-ordered coordinate system without changing the represented capability parameterization.

Proposition E.1 (Exact preservation under reparameterization)**.** *For every historical realization $\tau \in \mathcal{T}_s^{t-1}$, the QR reparameterization and energy-based rotation preserve the capability parameterization exactly:*

$$\widetilde{U}\widetilde{C}_{s,\tau}\widetilde{V} = UC_{s,\tau}V = A^{\ell}_{s,\tau}. \tag{24}$$

Appending new basis directions and zero-padding the historical core also leave $A^{\ell}_{s,\tau}$ unchanged.

Proof. Using equation 11–equation 13 and the orthogonality of P_U and P_V:

$$\begin{aligned}
\widetilde{U}\widetilde{C}_{s,\tau}\widetilde{V} &= Q_U P_U \left(P_U^\top B_{s,\tau} P_V\right) P_V^\top Q_V^\top && (25)\\
&= Q_U B_{s,\tau} Q_V^\top && (26)\\
&= Q_U R_U C_{s,\tau} R_V^\top Q_V^\top && (27)\\
&= UC_{s,\tau}V = A^{\ell}_{s,\tau}. && (28)
\end{aligned}$$

Hence, the QR reparameterization and energy-based rotation change only the coordinate representation.

For the expansion from K^- to K^+, define:

$$U^+ = \begin{bmatrix}\widetilde{U} & U_{\text{new}}\end{bmatrix}, \qquad V^+ = \begin{bmatrix}\widetilde{V} \\ V_{\text{new}}\end{bmatrix}, \qquad C^+_{s,\tau} = \begin{bmatrix}\widetilde{C}_{s,\tau} & 0 \\ 0 & 0\end{bmatrix}. \tag{29}$$

Then:

$$U^+ C^+_{s,\tau} V^+ = \widetilde{U}\widetilde{C}_{s,\tau}\widetilde{V} = A^{\ell}_{s,\tau}. \tag{30}$$

Therefore, both coordinate reparameterization and zero-padded rank expansion preserve every historical capability parameterization exactly. □

E.2 Residual-Energy Bound

The protection rule in equation 14 selects the smallest prefixes that retain at least a fraction ρ of the accumulated historical energy. Define the corresponding projection matrices:

$$\Pi_U = P_U[:, 1{:}p_U]P_U[:, 1{:}p_U]^\top, \qquad \Pi_V = P_V[:, 1{:}p_V]P_V[:, 1{:}p_V]^\top. \tag{31}$$

Proposition E.2 (Bounded residual historical energy)**.** *The aggregate historical energy outside the protected left and right subspaces is bounded by a fraction $1-\rho$ of the corresponding total historical energy:*

$$\sum_{\tau \in \mathcal{T}_s^{t-1}} \|(I - \Pi_U)B_{s,\tau}\|_F^2 \le (1-\rho) \sum_{\tau \in \mathcal{T}_s^{t-1}} \|B_{s,\tau}\|_F^2, \tag{32}$$

$$\sum_{\tau \in \mathcal{T}_s^{t-1}} \|B_{s,\tau}(I - \Pi_V)\|_F^2 \le (1-\rho) \sum_{\tau \in \mathcal{T}_s^{t-1}} \|B_{s,\tau}\|_F^2. \tag{33}$$

Proof. From equation 12:

$$\operatorname{tr}(M_U) = \sum_{\tau \in \mathcal{T}_s^{t-1}} \|B_{s,\tau}\|_F^2 = \sum_j \lambda_{U,j}. \tag{34}$$

Since Q_U and Q_V have orthonormal columns:

$$\|B_{s,\tau}\|_F = \|Q_U B_{s,\tau} Q_V^\top\|_F = \|A^{\ell}_{s,\tau}\|_F, \tag{35}$$

so the energy measured in the QR coordinates equals that of the corresponding historical capability parameterization.

For the left subspace:

$$\sum_{\tau \in \mathcal{T}_s^{t-1}} \|(I - \Pi_U) B_{s,\tau}\|_F^2 = \operatorname{tr}\left[(I - \Pi_U) M_U\right] \tag{36}$$

$$= \sum_{j > p_U} \lambda_{U,j}. \tag{37}$$

By the definition of p_U in equation 14:

$$\sum_{j > p_U} \lambda_{U,j} \leq (1 - \rho) \sum_j \lambda_{U,j}, \tag{38}$$

which proves equation 32. Applying the same argument to M_V proves equation 33. □

Proposition E.2 gives a direct interpretation of ρ: each protected subspace captures at least a fraction ρ of the accumulated historical energy, leaving at most a fraction $1 - \rho$ exposed to recycling. Adaptive subspace recycling therefore preserves dominant historical directions while retaining low-energy directions as trainable capacity for subsequent adaptation.

F Sensitivity to Rank Expansion

Table 8: Sensitivity to rank expansion on UCIT. $K_{\text{real}}^{\max}$ denotes the largest realized active rank, and Trainable Params reports the trainable parameter count under each setting. Δ is relative to the default $\delta = 1$. Best task-wise and average results are bolded.

δ	$K_{\text{real}}^{\max}$	Trainable Params (M)	ImgNet-R	ArxivQA	VizWiz	IconQA	CLEVR	Flickr30k	$\overline{\mathcal{A}}\uparrow$	Δ
0	8	22.176	**85.87**	**93.00**	60.70	65.53	59.27	55.97	70.06	−0.61
1 (default)	11	25.340	85.57	92.60	60.56	68.53	60.57	56.09	70.67	0.00
2	14	28.491	85.77	92.80	60.23	69.27	61.87	56.40	71.06	+0.39
3	17	31.879	85.47	92.67	60.66	70.33	62.23	**56.53**	71.32	+0.65
4	20	35.145	85.80	92.87	**60.86**	**71.10**	**62.47**	56.38	**71.58**	+0.91

To examine the effect of rank expansion in adaptive subspace recycling, we vary the number of newly appended basis directions $\delta \in \{0, 1, 2, 3, 4\}$ on UCIT using a held-out validation set. As shown in Table 8, increasing δ consistently improves average validation performance while increasing both the realized active rank and trainable parameter count. The gains are mainly observed on later tasks such as IconQA and CLEVR, while performance on earlier tasks remains relatively stable, suggesting that additional rank capacity primarily supports subsequent adaptation without substantially compromising previously learned capabilities. Although larger expansion budgets achieve further gains, they also incur higher parameter costs. We therefore use $\delta = 1$ as the default to balance continual adaptation performance and parameter efficiency.

G Computational Efficiency and Overhead

G.1 Training Efficiency

We evaluate the computational efficiency of RECAP on UCIT. For a focused comparison, we include LoRA-FT as a standard parameter-efficient fine-tuning baseline, together with SAME, HiDe-LLaVA, and SEFE, which are the three strongest baselines in terms of final average performance on UCIT. We compare trainable parameter footprint and training overhead using device-hours per task and per-device training throughput. The trainable parameter count is measured at each continual stage and averaged over all stages. Device-hours per task are computed as

$$\text{Device}-\text{h}/\text{task} = \frac{\sum_{t=1}^{T} N_t T_t^{\text{train}}}{3600T}, \tag{39}$$

Table 9: Trainable-parameter footprint and training efficiency on UCIT. Lower is better for trainable-parameter count and device-hours, whereas higher is better for throughput.

Method	Trainable Params (M) ↓	Device-h/task ↓	Train samples/s/dev. ↑
SAME (Xie et al., 2026)	73.286	7.12	1.391
HiDe-LLaVA (Guo et al., 2025)	29.360	3.94	2.510
SEFE (Chen et al., 2025)	340.795	8.30	1.192
LoRA-FT (Hu et al., 2022)	239.862	3.62	2.735
RECAP (Ours)	25.340	6.48	1.529

Table 10: Inference efficiency on UCIT. Higher throughput and lower TTFT and peak memory are better.

Method	Tokens/s ↑	TTFT (ms) ↓	Samples/s ↑	Peak memory (GB) ↓
Zeroshot	11.4597	198.89	2.0963	14.23
LoRA-FT (Hu et al., 2022)	6.2338	246.08	1.4529	15.48
O-LoRA (Wang et al., 2023)	8.8487	223.00	2.0312	28.33
MoELoRA (Chen et al., 2024)	1.8979	619.01	0.4898	15.44
ModalPrompt (Zeng et al., 2025)	9.9822	269.88	1.7951	29.06
CL-MoE (Huai et al., 2025)	2.4209	499.94	0.5511	14.83
HiDe-LLaVA (Guo et al., 2025)	2.4283	594.72	0.6441	15.44
SEFE (Chen et al., 2025)	6.8335	230.57	1.3403	15.69
SAME (Xie et al., 2026)	3.0351	461.36	0.7373	29.43
RECAP (Ours)	5.3755	291.92	1.2086	14.73

where T is the number of continual stages, N_t is the number of devices used at stage t, and T_t^{train} is the corresponding training time in seconds. Per-device training throughput is computed over the complete continual-learning stream using the accumulated device time across all stages.

As shown in Table 9, RECAP has the smallest trainable parameter footprint among all compared methods, optimizing only 25.340M parameters on average across continual stages. In particular, among the three strongest baselines on UCIT, RECAP uses fewer trainable parameters than HiDe-LLaVA, SAME, and SEFE. It also requires fewer device-hours and achieves higher training throughput than SAME and SEFE, while HiDe-LLaVA remains more efficient in terms of training time and throughput. LoRA-FT similarly trains faster, but does so with a substantially larger trainable parameter footprint. These results show that RECAP does not uniformly minimize every measure of training cost; rather, its main efficiency advantage lies in substantially reducing the number of optimized parameters while maintaining moderate training-time overhead.

G.2 Inference Efficiency

We compare inference efficiency with baselines on UCIT using token throughput, time to first token (TTFT), sample throughput, and peak memory. Token and sample throughput measure decoding and sample-level processing speed, respectively, while TTFT reflects response latency and peak memory records the maximum memory usage during inference. As shown in Table 10, RECAP achieves 5.38 tokens/s and 1.21 samples/s, with a TTFT of 291.92 ms and peak memory usage of 14.73 GB. Among the strongest-performing baselines, RECAP is more efficient than SAME across all four metrics, while SEFE achieves higher throughput and lower latency but requires more memory. RECAP also outperforms HiDe-LLaVA, MoELoRA, and CL-MoE in both throughput and latency. Several other baselines, including LoRA-FT, O-LoRA, and ModalPrompt, achieve higher throughput or lower latency, but with higher peak memory usage. Notably, RECAP has the lowest peak memory usage among all evaluated continual-learning methods. Overall, these results show that RECAP maintains competitive inference efficiency with low memory overhead.