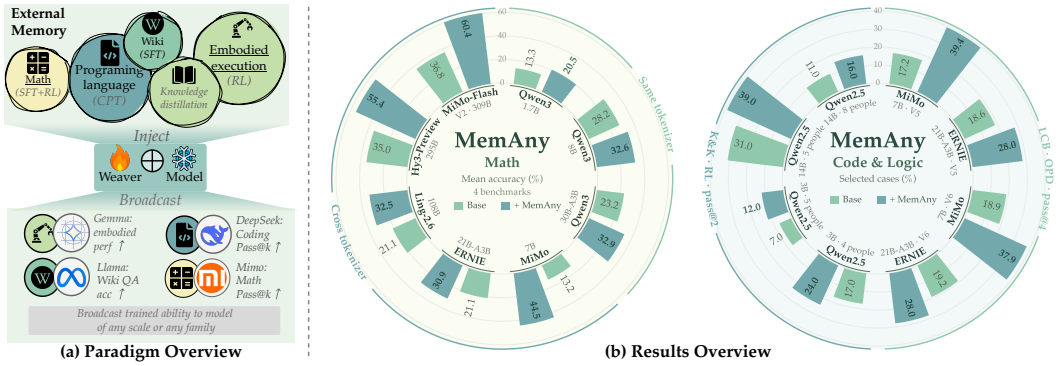
MEMANY: PLUG-AND-PLAY GENERATIVE LATENT MEMORY FOR ANY MODEL

Anonymous authors
Paper under double-blind review

ABSTRACT

Knowledge acquired through model training is typically tied to the model that learned it, requiring repeated training as new models emerge. We introduce `MemAny`, a framework that turns adapted knowledge into plug-and-play generative latent memory that can be trained once on a *host* model and reused across different *recipient* models. `MemAny` routes adaptation into an external memory weaver while keeping the host model frozen, and expresses its generated latent tokens as linear combinations of each recipient's own token embeddings. This latent superposition enables knowledge transfer across tokenizers through coefficients aligned by shared vocabulary anchors, without updating either the recipient or the learned memory. Experiments combining `MemAny` with various training algorithms, *e.g.*, supervised fine-tuning (SFT), reinforcement learning (RL), and on-policy distillation (OPD), demonstrate that memories learned with small host models yield transferable gains in mathematical/logical reasoning, code generation, and tool-use. In particular, OPD-trained memory from Qwen3-4B-Base transfers across model families and tokenizers to MiMo-V2-Flash-Base (309B) and HY3-Preview-Base (295B), improving their AIME24 accuracy by up to 33.3 points. These results establish portable generative memory as a practical interface for sharing acquired capabilities, paving the way for their efficient and rapid dissemination across an evolving model ecosystem.



(a) Paradigm Overview (b) Results Overview

Figure 1: **Train once, share across models.** **(a)** `MemAny` trains a memory weaver with a frozen host model and broadcasts it to new recipient models; the depicted applications are illustrative. **(b)** Broadcasting the memory trained on Qwen-series to various model families. Math: mean accuracy over four benchmarks; code and logic: LiveCodeBench pass@4 and K&K pass@2 results.

1 INTRODUCTION

Large language models (LLMs) are trained to encode parametric knowledge. After adaptation, a model may acquire new factual associations (Wang et al., 2024; Dou et al., 2024; Fang et al., 2025a), reasoning skills (DeepSeek-AI et al., 2025; Liu et al., 2025), or tool-use experiences (Zhang et al., 2026; Wang et al., 2026). Yet the resulting knowledge is rarely *portable*: it is absorbed into the particular parameter space of the model that was trained. A trained model therefore becomes

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.

a *container* of specialized knowledge, but not a *carrier* of it. When another model becomes the preferred model (no matter because it is larger, cheaper, safer, faster, or better aligned), the same knowledge must typically be learned again through another round of optimization (Jiang et al., 2026). If there were an interface through which adapted knowledge itself could flow across models, the cost of retraining would fall, and inter-model knowledge transfer would become frictionless.

This paper asks whether such an interface can exist. Suppose a model has acquired a capability through any training procedure, *e.g.*, continual pretraining (CPT) that injects world knowledge (Su et al., 2025; Team et al., 2026), reinforcement learning (RL) that boosts mathematical reasoning (Wang et al., 2025d; Dang & Ngo, 2026), supervised finetuning (SFT) that improves instruction following (Liu et al., 2026), or knowledge editing that inserts new factual associations (Fang et al., 2025a; Wang et al., 2025a). Can the parametric knowledge gained by that model (*host*) be transferred to another model (*recipient*) with no additional training? For exmaple, could a Qwen model trained for skill use share its acquired procedural knowledge with Gemma? Could a 4B model that has undergone factual editing pass the edited knowledge directly to a 300B-scale counterpart? To put the question more formally: *can specialized knowledge be trained once, detached from the model that learned it, and reused by models that were never optimized for it?*

Method. This work proposes `MemAny`, a framework for training portable generative memory to any models (Figure 1 (a)). `MemAny` trains an external memory module to capture the knowledge induced by adaptation, while keeping the language model itself frozen. At inference time, the memory module emits a short sequence of latent memory tokens that can be attached to a recipient model and used to condition its generation. In other words, `MemAny` turns the outcome of model training into a reusable continuous memory object: one that can be produced with one model, attached to another, and consumed without any additional optimization. This enables the same learned memory to operate across model scales, and, in the more challenging case, across model families.

Challenges. Realizing this requires solving two challenges: ❶ **Memory separation**: *how can adapted knowledge be separated from the model that acquired it?* `MemAny` addresses this by adopting a latent-weaving formulation: during adaptation, the model itself remains frozen, and optimization is routed into a separate memory module, also referred to as *memory weaver* following (Zhang et al., 2025a). The acquired knowledge is therefore concentrated outside the model parameters, and is later injected non-invasively during decoding through latent memory tokens; ❷ **Memory share**: *how can such memory tokens remain meaningful when attached to a different model*, possibly with a different tokenizer and vocabulary? `MemAny` enables knowledge transfer through *latent superposition* (Deng et al., 2025), expressing each memory token as a signed linear combination of the recipient model's own vocabulary embeddings. As a result, the emitted tokens are naturally tailored to the recipient's embedding geometry, enabling cross-model/tokenizer transfer without updating the recipient.

Contributions. `MemAny` meaningfully changes the lifecycle of model adaptation along the following three dimensions: ♣ **Shareability**: parametric memory can propagate independently of the model that produced it, so acquired capabilities need not remain locked inside a single parameter space. ♥ **Algorithm agnosticism**: the memory can be induced by any available training algorithms (SFT, RL, PT/CPT, *etc.*). ♠ **Amortizability**: the cost of acquiring knowledge is decoupled from the cost of deploying it; once trained, a memory can be attached to new models as they appear. This suggests a *memory market* for LLMs, where learned memories circulate as reusable assets rather than being repeatedly regenerated inside each model.

Experiment. We evaluate `MemAny` across logical and mathematical reasoning, code generation, and tool-use domains, highlighting two dimensions of memory portability. ① **Same- and cross-family transfer**: mathematical memory trained once with Qwen3-4B transfers to eight recipients, including Qwen3-1.7B/8B/30B-A3B within the same family and five models from other families, without retraining. Notably, this 4B-trained memory improves AIME24 accuracy by 31.7, 30.0, and 33.3 percentage points on Ling-2.6 (108B), HY3-Preview (295B), and MiMo-V2-Flash (309B), respectively, demonstrating that capabilities acquired with a small model can benefit models over 70× its size; ② **Multi-algorithm support**: `MemAny` supports transferable memory under SFT, OPD, and RL (GRPO). Together, these results show that portable memory can be acquired through different training algorithms and reused across model scales and families. Comparisons with leading explicit external memory (*e.g.*, ReasoningBank, ExpeL) further demonstrate stronger and more consistent cross-model transfer with `MemAny`, highlighting the advantages of plug-and-play generative memory.

2 Related Work

LLM & Agent Memory. Memory mechanisms for LLM/agents can be broadly categorized by their implementation form (Hu et al., 2026b; Huang et al., 2026): **(I) Parametric memory** injects skills, tool-use experience, or factual knowledge by training the model itself or additional trainable parameters (Wang et al., 2024; Wu et al., 2026; Feng et al., 2026a); **(II) Database memory** stores facts and experiences in external structures, such as vector databases (ByteDance, 2026; Hu et al., 2026a), hierarchical graphs (Fang et al., 2025b; Xu et al., 2025a), skill libraries (Lu et al., 2026), or hybrid repositories (Zhang et al., 2025b); **(III) Latent memory** represents memory through continuous latent tokens or hidden states rather than discrete text (Zhang et al., 2025a; Hou et al., 2026). `MemAny` lies at the intersection of (I) and (III): it internalizes knowledge through parametric training and communicates it through latent memory tokens. Unlike prior methods in these categories, however, it breaks the strong architectural coupling between memory and the model that produced it, enabling the same memory to be shared across different models.

Experience Learning. A broad range of parametric adaptation methods can be viewed as mechanisms for injecting experience into off-the-shelf agent backbones (Jiang et al., 2026). In practice, RLVR methods such as GRPO (DeepSeek-AI et al., 2025), DAPO (Yu et al., 2026), and GSPO (Zheng et al., 2025) are commonly used to internalize reasoning and agentic tool-use experience (Singh et al., 2025; Wei et al., 2026); SFT and its variants support instruction following (Li et al., 2025) and knowledge editing (Zhang et al., 2025c); continual pretraining injects domain-specific priors (Wang et al., 2025b; Shi et al., 2025); and on-policy distillation (OPD) has recently emerged as a competitive alternative for reasoning RL (Li et al., 2026; Song & Zheng, 2026). We regard these methods as different paradigms for experience learning. `MemAny` is orthogonal to them: rather than proposing another way to acquire experience, it changes how acquired experience is reused, turning model-bound adaptation into portable memory that can be shared across models.

3 Preliminary

Notation. We consider an LLM agent with host model π_{θ_m}, tokenizer $\mathcal{T}_m$, vocabulary $\mathcal{V}_m$, embedding matrix $\mathbf{E}_m \in \mathbb{R}^{|\mathcal{V}_m| \times d_m}$, and parameter space Θ_m. Under task distribution $\mathcal{D}$ and environment $\mathcal{E}$, a task $\mathbf{x} \sim \mathcal{D}$ induces a trajectory:

$$\boldsymbol{\tau} = \big(\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T\big),\ \mathbf{s}_{t+1} \sim \mathcal{E}(\cdot \mid \mathbf{s}_t, \mathbf{a}_t),\ R(\boldsymbol{\tau}) = \sum_{t=0}^{T} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t), \tag{1}$$

where r is the step reward and γ is the discount factor. Each action $\mathbf{a}_t = (z_{t,1}, \ldots, z_{t,L_t})$ is generated autoregressively; we write $\mathbf{z}_{t,<j}$ for the prefix before token j, $\mathbf{h}_{t,j}^{(m)} = F_{\theta_m}(\mathbf{s}_t, \mathbf{z}_{t,<j})$ for the decoding state, and $\mathbf{H}_{t,<j}^{(m)}$ for the collected prefix hidden states. Let $\mathcal{H}$ denote the training signal, such as expert trajectories (Yin et al., 2024; Du et al., 2025) or environment rewards (Xue et al., 2026; 2025), with sample $\boldsymbol{\xi} \sim \mathcal{H}$. A training algorithm $\mathcal{A}$ typically injects this signal by producing a host-specific parameter update:

$$\theta_m^{\mathcal{A}} = \theta_m + \Delta_m^{\mathcal{A}},\ \Delta_m^{\mathcal{A}} \in \arg\min_{\Delta \in \Omega_m} \mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{H}}\big[\mathcal{L}_{\mathcal{A}}\big(\pi_{\theta_m + \Delta}; \boldsymbol{\xi}\big)\big], \tag{2}$$

where $\Omega_m \subseteq \Theta_m$ is the trainable subspace and $\mathcal{L}_{\mathcal{A}}$ may be an SFT, RL, OPD, or other objective. The resulting $\Delta_m^{\mathcal{A}}$ is parametric memory, but it is expressed in the coordinate system of host m: for a recipient n with different vocabulary, hidden size, or transition function, $\theta_n + \Delta_m^{\mathcal{A}}$ is generally not type-compatible. Thus useful knowledge may be encoded in $\Delta_m^{\mathcal{A}}$, but the object is not portable.

Problem Formalization. Our goal is to replace this host-bound delta with a reusable memory object. Let $\Pi_{\theta_n}^{\mathcal{M}}$ be the policy obtained by conditioning a frozen recipient π_{θ_n} on memory $\mathcal{M}$, and let $\mathcal{U}_{\mathcal{A}}$ be the utility associated with the adaptation signal. `MemAny` seeks:

$$\mathcal{M}_{\mathcal{A}}^{\star} \in \arg\max_{\mathcal{M}} \mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{H}}\Big[\mathcal{U}_{\mathcal{A}}\Big(\Pi_{\theta_{m_s}}^{\mathcal{M}}; \boldsymbol{\xi}\Big)\Big],\ \forall n \in \mathcal{G}:\ \Pi_{\theta_n}^{\mathcal{M}_{\mathcal{A}}^{\star}} \text{ with } \nabla_{\theta_n} = 0,\ \nabla_{\mathcal{M}_{\mathcal{A}}^{\star}} = 0. \tag{3}$$

In words, memory is trained once on host m_s and then reused by frozen recipients without updating either the recipient model or the memory itself. The question is whether knowledge acquired through SFT, RL, OPD, pretraining, or other adaptation signals can become a shareable carrier rather than a model-specific parameter delta.

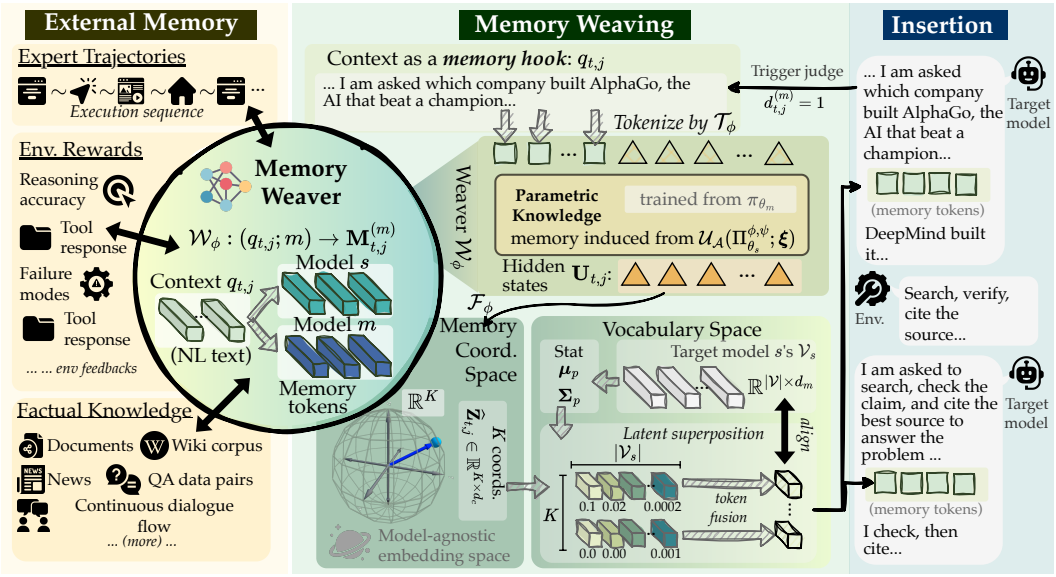


Figure 2: Overview of **MemAny**. The weaver learns from experience to generate context-dependent memory. Vocabulary alignment expresses memory as weighted token embeddings, which a trigger inserts into frozen recipient models to transfer capabilities across families.

4 METHODOLOGY

MemAny turns the outcome of model adaptation into a reusable memory intervention at inference time. Given a frozen recipient model, it uses a memory weaver to produce latent memory tokens and a trigger to decide where these tokens should enter the decoding process; the remaining subsections explain how the weaver makes such tokens compatible across vocabularies and how it internalizes information from arbitrary training algorithms.

4.1 MEMORY-AUGMENTED DECODING

We first describe **MemAny** at the level of decoding, abstracting away the internal construction of latent memory tokens. The central design choice is to use *latent memory insertion* as the carrier of shareable memory: unlike a parameter delta $\Delta^{\mathcal{A}}_m$, it does not require modifying the recipient model, and unlike textual memory, it need not compress learned experience into discrete natural-language strings before the model can use it. Instead, memory is introduced directly into the model's ongoing computation as a compact continuous sequence. Consider a frozen recipient model π_{θ_m} and an ongoing generation at environment step t and token position j. A lightweight trigger $\mathcal{T}_\psi$, parameterized by ψ, observes the current prefix hidden states and decides whether memory should be invoked:

$$d^{(m)}_{t,j} = \mathcal{T}_\psi\left(\mathbf{H}^{(m)}_{t,<j}\right) \in \{0,1\}, \qquad \mathbf{H}^{(m)}_{t,<j} = \left(\mathbf{h}^{(m)}_{t,1}, \ldots, \mathbf{h}^{(m)}_{t,j-1}\right), \tag{4}$$

where $d^{(m)}_{t,j} \in \{0,1\}$ is a binary decision: $d^{(m)}_{t,j} = 1$ invokes memory and $d^{(m)}_{t,j} = 0$ leaves decoding unchanged. In practice, $\mathcal{T}_\psi$ can be evaluated sparsely, for example at the prompt boundary or delimiter-bounded decoding positions, so the intervention is selective rather than applied at every token. When invoked, the memory weaver $\mathcal{W}_\phi$, parameterized by ϕ, receives a query $q_{t,j}$ derived from the current context (*e.g.*, $q_{t,j} = (\mathbf{s}_t, \mathbf{z}_{t,<j})$) and emits a fixed-length sequence of K latent memory tokens for the recipient model:

$$\mathbf{M}^{(m)}_{t,j} = \mathcal{W}_\phi(q_{t,j}; m), \ \mathbf{M}^{(m)}_{t,j} = \left(\mathbf{m}^{(m)}_{t,j,1}, \ldots, \mathbf{m}^{(m)}_{t,j,K}\right), \ \mathbf{m}^{(m)}_{t,j,k} \in \mathbb{R}^{d_m}, \tag{5}$$

where the argument m indicates that the emitted memory lies in the input-embedding space of recipient π_{θ_m}; the next subsection explains how this compatibility is achieved. Let $\mathbf{X}^{(m)}_{t,<j}$ denote the input embeddings of the current textual prefix $q_{t,j}$, obtained from the recipient embedding matrix

$\mathbf{E}_m$. We append memory to this sequence before the first Transformer layer:

$$\widetilde{\mathbf{X}}^{(m)}_{t,<j} = \begin{cases} [\mathbf{X}^{(m)}_{t,<j}; \mathbf{M}^{(m)}_{t,j}], & d^{(m)}_{t,j} = 1, \\ \mathbf{X}^{(m)}_{t,<j}, & d^{(m)}_{t,j} = 0, \end{cases} \qquad z_{t,j} \sim \pi_{\theta_m}\left(\cdot \mid \widetilde{\mathbf{X}}^{(m)}_{t,<j}\right), \tag{6}$$

where $[\cdot;\cdot]$ denotes concatenation along the sequence dimension. The augmented input embeddings $\widetilde{\mathbf{X}}^{(m)}_{t,<j}$ pass through the frozen recipient's Transformer layers to condition subsequent decoding. This procedure can be applied once at the beginning of generation or repeatedly during decoding; in all cases, θ_m remains frozen and the textual prompt is not rewritten. The two questions left for **MemAny** are therefore the core of the method: (i) how can a single weaver produce latent memory tokens for models of different sizes and even different vocabularies (Section 4.2)? (ii) how can that weaver internalize the information supplied by arbitrary training signals (Section 4.3)?

4.2 Weaving Latent Memory Across Vocabularies

We now specify how the memory weaver $\mathcal{W}_\phi$ in Section 4.1 constructs latent memory. The weaver generates context-dependent vocabulary coefficients, while the recipient supplies the token embeddings that these coefficients combine. For clarity, we write $q = q_{t,j}$ and omit the indices below.

Latent Superposition. Let $\mathbf{E}_m \in \mathbb{R}^{V_m \times d_m}$ be the frozen embedding matrix of recipient m, where $V_m = |\mathcal{V}_m|$. For each of the K memory tokens, we assign a weight $a^{(m)}_{kv}(q)$ to each vocabulary token v and construct

$$\mathbf{m}^{(m)}_k(q) = \sum_{v \in \mathcal{V}_m} a^{(m)}_{kv}(q)\, \mathbf{E}_m[v], \qquad \mathbf{M}^{(m)}(q) = \mathbf{A}^{(m)}(q)\mathbf{E}_m, \tag{7}$$

where $\mathbf{A}^{(m)}(q) \in \mathbb{R}^{K \times V_m}$ collects the vocabulary coefficients. Their weighted sum is $\mathcal{W}_\phi(q; m) = \mathbf{M}^{(m)}(q)$ defined in Section 4.1, expressed directly in the recipient's embedding space.

Generating Vocabulary Coefficients. Given q, the weaver computes K internal memory representations, denoted by $\mathbf{Z}_\phi(q)$. The recipient-specific coefficient map $\mathcal{C}_m$ expresses these representations in the recipient's coefficient basis and assigns one weight to each vocabulary entry:

$$\mathbf{A}^{(m)}(q) = \mathcal{C}_m(\mathbf{Z}_\phi(q); \alpha_m) \in \mathbb{R}^{K \times V_m}, \qquad \sum_{v \in \mathcal{V}_m} a^{(m)}_{kv}(q) = 1. \tag{8}$$

This affine map includes a scale α_m that controls the resulting memory's magnitude around the recipient's mean embedding. Because embedding norms differ across models, this scale matches the strength of memory injection to the recipient's native token embeddings.

Coefficient Transfer. For each recipient, we calibrate $\mathcal{C}_m$ through shared vocabulary anchors that align the reference and recipient coefficient bases. The same learned memory representations can then produce coefficients over each recipient's vocabulary and combine its native embeddings. Alignment and scale calibration are computed once from frozen embedding statistics, without additional task data or training; details appear in Appendix A.

4.3 Internalizing Experience with Arbitrary Training Algorithms

We finally describe how **MemAny** turns ordinary training signals into portable memory. The important point is that **MemAny** does not replace the training algorithm. SFT, RL, OPD, or other adaptation procedures still provide their usual supervision, reward, or teacher distribution. What changes is where the acquired knowledge is stored and how it interacts with the host: instead of updating the host parameters, the training signal is routed into the trigger $\mathcal{T}_\psi$ and the parameters ϕ of the weaver $\mathcal{W}_\phi$, while the host π_{θ_s} remains frozen. The trigger learns when a context should call memory, and the weaver learns what memory coordinates should be emitted when that context acts as a hook.

Formally, let $\mathcal{U}_\mathcal{A}$ denote the utility induced by a training algorithm $\mathcal{A}$ over samples $\boldsymbol{\xi} \sim \mathcal{H}$, such as log-likelihood for SFT, task reward for RL, or teacher-student agreement for OPD. Training optimizes only the memory-side parameters:

$$(\phi^\star, \psi^\star) \in \arg\max_{\phi,\psi} \mathbb{E}_{\boldsymbol{\xi} \sim \mathcal{H}} \left[\mathcal{U}_\mathcal{A}\left(\Pi^{\phi,\psi}_{\theta_s}; \boldsymbol{\xi}\right)\right], \qquad \nabla_{\theta_s} = 0, \tag{9}$$

Table 1: SFT and OPD transfer results. The memory weaver is trained on QWEN3-4B-BASE with SFT or OPD and then attached to recipient base models without further recipient-side training. Math scores report accuracy (%), LCB-V5/V6 report pass@4 and Tool-use (When2Call) reports Macro-F1 and accuracy (%; 300 examples). See Appendix B.4 for metric details. Certain performance regression in code has been discussed in Appendix E.

Setting	Backbone	Config	MathReasoning				Code		Tool-use	
			AIME24	AIME25	Olympiad	MATH-500	LCB-V5	LCB-V6	Macro-F1	Acc.
Host	QWEN3-4B-BASE	Base	5.8±2.8	3.3±2.4	22.9±1.9	52.6	16.8	18.9	17.3	23.0
		+MemAny SFT	10.8±1.4 (↑ 5.0)	7.5±4.3 (↑ 4.2)	38.6±2.2 (↑ 15.8)	73.8 (↑ 21.2)	26.2 (↑ 9.3)	25.8 (↑ 6.8)	28.3 (↑ 11.0)	30.0 (↑ 7.0)
		+MemAny OPD	18.3±3.7 (↑ 12.5)	11.7±6.0 (↑ 8.3)	43.4±2.4 (↑ 20.5)	77.2 (↑ 24.6)	22.9 (↑ 6.1)	24.4 (↑ 5.5)	34.8 (↑ 17.5)	36.0 (↑ 13.0)
Same Tokenizer	QWEN3-1.7B	Base	0.8±1.4	0.8±1.4	10.6±0.6	41.0	9.3	11.5	12.1	17.3
		+MemAny SFT	4.2±2.8 (↑ 3.3)	3.3±2.4 (↑ 2.5)	22.6±1.9 (↑ 12.0)	58.8 (↑ 17.8)	12.9 (↑ 3.6)	13.2 (↑ 1.8)	15.7 (↑ 3.6)	18.7 (↑ 1.3)
		+MemAny OPD	3.3±2.4 (↑ 2.5)	2.5±2.8 (↑ 1.7)	26.2±0.9 (↑ 15.6)	50.0 (↑ 9.0)	12.5 (↑ 3.2)	14.5 (↑ 3.1)	20.1 (↑ 8.0)	25.0 (↑ 7.7)
	QWEN3-8B	Base	6.7±6.2	5.0±1.7	32.9±1.4	68.4	22.2	23.1	20.6	25.3
		+MemAny SFT	11.7±3.7 (↑ 5.0)	7.5±2.8 (↑ 2.5)	41.6±1.8 (↑ 8.8)	70.0 (↑ 1.6)	22.6 (↑ 0.4)	24.7 (↑ 1.5)	23.4 (↑ 2.8)	24.3 (↓ 1.0)
		+MemAny OPD	12.5±6.4 (↑ 5.8)	11.7±1.7 (↑ 6.7)	39.1±1.2 (↑ 6.2)	67.0 (↓ 1.4)	24.0 (↑ 1.8)	23.3 (↑ 0.2)	29.4 (↑ 8.8)	31.3 (↑ 6.0)
	QWEN3-30B-A3B	Base	5.0±3.7	0.8±1.4	27.6±1.0	59.4	30.1	29.7	26.6	30.0
		+MemAny SFT	12.5±6.4 (↑ 7.5)	5.0±3.7 (↑ 4.2)	22.4±14.7 (↓ 5.2)	60.8 (↑ 1.4)	28.0 (↓ 2.2)	27.8 (↓ 2.0)	29.8 (↑ 3.2)	32.0 (↑ 2.0)
		+MemAny OPD	16.7±6.2 (↑ 11.7)	4.2±1.4 (↑ 3.3)	42.1±2.3 (↑ 14.5)	68.6 (↑ 9.2)	30.8 (↑ 0.7)	30.2 (↑ 0.4)	31.3 (↑ 4.7)	34.3 (↑ 4.3)
Cross Tokenizer	MIMO-7B	Base	3.3±3.3	2.5±1.4	8.6±0.4	38.4	17.2	18.9	30.3	35.3
		+MemAny OPD	32.5±7.2 (↑ 29.2)	25.8±2.8 (↑ 23.3)	43.6±1.8 (↑ 35.0)	76.2 (↑ 37.8)	39.4 (↑ 22.2)	37.9 (↑ 18.9)	33.1 (↑ 2.8)	36.0 (↑ 0.7)
	ERNIE-21B-A3B	Base	5.8±4.3	3.3±5.8	20.2±1.8	55.2	18.6	19.2	15.1	17.3
		+MemAny OPD	10.0±4.7 (↑ 4.2)	5.8±2.8 (↑ 2.5)	36.6±1.9 (↑ 16.4)	71.2 (↑ 16.0)	28.0 (↑ 9.3)	28.0 (↑ 8.8)	30.2 (↑ 15.1)	33.3 (↑ 16.0)
	LING-2.6-108B	Base	4.2±2.8	4.2±3.6	20.2±1.4	55.8	55.2	48.7	45.9	44.3
		+MemAny OPD	35.8±8.6 (↑ 31.7)	25.8±6.4 (↑ 21.7)	25.0±0.9 (↑ 4.8)	43.4 (↓ 12.4)	43.4 (↓ 11.8)	41.2 (↓ 7.5)	54.1 (↑ 8.2)	55.0 (↑ 10.7)
	HY3-PREVIEW	Base	6.7±4.1	13.3±4.1	42.0±1.6	78.0	53.4	48.5	42.7	43.3
		+MemAny OPD	36.7±6.7 (↑ 30.0)	43.3±5.3 (↑ 30.0)	52.5±2.1 (↑ 10.5)	89.0 (↑ 11.0)	51.3 (↓ 2.2)	45.4 (↓ 3.1)	45.7 (↑ 3.0)	46.0 (↑ 2.7)
	MIMO-FLASH-V2	Base	16.7±5.3	16.7±4.7	43.0±1.7	71.0	59.1	53.1	48.9	50.0
		+MemAny OPD	50.0±7.5 (↑ 33.3)	41.7±5.5 (↑ 25.0)	61.2±2.3 (↑ 18.2)	88.8 (↑ 17.8)	58.1 (↓ 1.1)	52.0 (↓ 1.1)	59.0 (↑ 10.1)	60.3 (↑ 10.3)

where $\Pi_{\theta_s}^{\phi,\psi}$ is the frozen host augmented by the trigger and weaver through the memory insertion mechanism in Section 4.1. For example, under SFT, the target sequence likelihood is improved by learning latent memories that help the frozen host predict demonstrations; under RL, trajectory reward updates the trigger-weaver pair rather than the host; under OPD, the memory learns to steer the frozen student host toward a teacher-conditioned distribution. After training, the portable memory is the learned trigger-weaver pair:

$$\mathcal{M}_{\mathcal{A}}^{\star} = (\phi^{\star}, \psi^{\star}), \qquad \mathbf{M}_{t,j}^{(m)} = \mathcal{W}_{\phi^{\star}}(q_{t,j}; m) \text{ if } \mathcal{T}_{\psi^{\star}}(\mathbf{H}_{t,<j}^{(m)}) = 1. \tag{10}$$

The same trained memory therefore transfers through the vocabulary coefficient head $\mathcal{C}_m$ and the recipient's embeddings. `MemAny` keeps the training algorithm intact and stores its acquired knowledge in a portable module that supplies the recipient with latent memory tokens.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Experimental Settings. We evaluate `MemAny` under three training settings: ❶ **Qwen3+SFT**: we learn memory with Qwen3-4B (Yang et al., 2025) on IneqMath (relation subset) (Lu et al., 2025). We evaluate Qwen3-1.7B/4B/8B in base and thinking modes on 100 sampled training problems (Seen100) and 50 development-set relation problems (dev50). ❷ **Qwen3+OPD**: following Lightning-OPD (Wu et al., 2026), we use Qwen3-8B with thinking disabled as the teacher and Qwen3-4B-Base as the host model. We train separate memories for math, code, and tool use on 10,000 prompts from DAPO-Math-17k (Yu et al., 2026), CodeContests-O, and 2,000 examples from NVIDIA's When2Call (Ross et al., 2025), respectively; When2Call primarily adapts BFCL questions. We evaluate transfer across model scales and families on AIME24/25, OlympiadBench, MATH-500, LiveCodeBench (LCB) V5/V6 (Jain et al., 2024), and the `llm_judge` test set of When2Call. ❸ **Qwen2.5+RL**: we train a Qwen3-4B-Base weaver with a Qwen2.5-7B-Instruct reasoner using GRPO on Knights-and-Knaves (KK) puzzles with 3–7 people, and transfer the memory to Qwen2.5-3B/14B-Instruct. Evaluation covers 2–8 people, including problem sizes outside the training range. Dataset construction is detailed in Appendix B.2.

Baselines. We compare ■ **vanilla models** with ■ **memory-augmented models**. To compare `MemAny` with explicit text-based memory, we also evaluate ExpeL (Zhao et al., 2024), Dynamic Cheatsheet (Suzgun et al., 2025), and ReasoningBank (Ouyang et al., 2026). For math, these methods accumulate experience from the same 10,000 DAPO problems used in the OPD stage of `MemAny`. Evaluation protocols and prompt templates are detailed in Appendices B.1, B.5 and B.6.

Table 2: Transfer results on K&K logic puzzles. The QWEN3-4B-BASE memory weaver is trained with QWEN2.5-7B-INSTRUCT as the host and then attached to recipient models without further training. We report Avg@2 and pass@2 for 2–8-person puzzles. The 2/8-person puzzles are OOD.

Setting	Backbone	Config	Metric	Difficulty by Number of People							Overall
				2 OOD	3	4	5	6	7	8 OOD	
Host	QWEN2.5-7B-INSTRUCT	Base	Avg@2	43.5	24.0	18.0	11.0	8.0	4.0	2.5	15.9
			pass@2	65.0	41.0	30.0	21.0	16.0	7.0	5.0	26.4
		+MemAny	Avg@2	44.0 (↑0.5)	33.0 (↑9.0)	20.0 (↑2.0)	13.0 (↑2.0)	11.5 (↑3.5)	5.0 (↑1.0)	3.5 (↑1.0)	18.6 (↑2.7)
			pass@2	67.0 (↑2.0)	46.0 (↑5.0)	34.0 (↑4.0)	24.0 (↑3.0)	20.0 (↑4.0)	10.0 (↑3.0)	6.0 (↑1.0)	29.6 (↑3.1)
Same Tokenizer	QWEN2.5-3B	Base	Avg@2	27.0	16.0	9.5	3.5	2.5	1.0	1.5	8.7
			pass@2	44.0	30.0	17.0	7.0	5.0	2.0	3.0	15.4
		+MemAny	Avg@2	31.0 (↑4.0)	16.0 (→0.0)	13.0 (↑3.5)	6.5 (↑3.0)	4.0 (↑1.5)	3.0 (↑2.0)	2.0 (↑0.5)	10.8 (↑2.1)
			pass@2	45.0 (↑1.0)	28.0 (↓2.0)	24.0 (↑7.0)	12.0 (↑5.0)	8.0 (↑3.0)	6.0 (↑4.0)	4.0 (↑1.0)	18.1 (↑2.7)
	QWEN2.5-14B	Base	Avg@2	55.0	46.0	33.5	22.0	17.5	10.5	6.5	27.3
			pass@2	74.0	62.0	50.0	31.0	29.0	20.0	11.0	39.6
		+MemAny	Avg@2	55.5 (↑0.5)	47.0 (↑1.0)	33.0 (↓0.5)	25.5 (↑3.5)	17.0 (↓0.5)	13.5 (↑3.0)	8.5 (↑2.0)	28.6 (↑1.3)
			pass@2	74.0 (→0.0)	64.0 (↑2.0)	50.0 (→0.0)	39.0 (↑8.0)	29.0 (→0.0)	25.0 (↑5.0)	16.0 (↑5.0)	42.4 (↑2.9)

Implementation Details. We train the weaver with $K = 8$ prompt memory tokens and rank-128 LoRA adapters (Hu et al., 2022). In all experiments, we insert memory tokens once after the prompt's input embeddings, before the first Transformer layer and the start of decoding, without separately training the trigger. Nevertheless, the framework also supports trigger training following (Zhang et al., 2025a). Under the standard evaluation protocol, AIME and OlympiadBench report Avg@4, MATH-500 reports accuracy (pass@1), LCB reports pass@4, KK reports Avg@2 and pass@2, and tool use reports Macro-F1 and accuracy. We use DeepSeek-V4-Flash-0731 as the judge for tool-use evaluation. Detailed training hyperparameters, decoding settings, and scoring rules are provided in Appendices B.3 and B.4.

5.2 MAIN RESULTS

Across OPD, SFT, and RL, `MemAny` turns training on a host model into reusable memory that improves frozen recipient models. The results demonstrate transfer across model scales and tokenizers.

`MemAny` + SFT. SFT-trained memory transfers across tasks and model scales (Table 1). On QWEN3-4B, it raises MATH-500 accuracy from 52.6% to 73.8%, LCB-V5 pass@4 from 16.8% to 26.2%, and tool-use Macro-F1 from 17.3% to 28.3%. Transfer raises MATH-500 from 41.0% to 58.8% on Qwen3-1.7B, Olympiad from 32.9% to 41.6% on Qwen3-8B, and AIME24 from 5.0% to 12.5% on Qwen3-30B-A3B. On IneqMath, dev50 Avg@4 improves across all six model–mode settings by 7 percentage points on average (Figure 3). Qwen3-1.7B improves from 22% to 30% in base model and from 62% to 76% in thinking model.

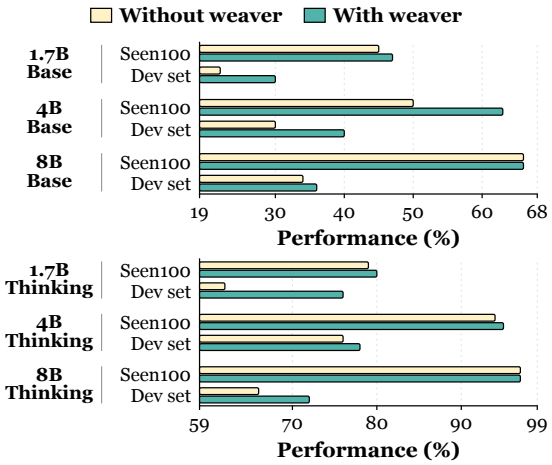


Figure 3: SFT transfer from Qwen3-4B across Qwen3 scales in base (top) and thinking (bottom) modes. We report Seen100 accuracy and dev50 Avg@4 for IneqMath.

`MemAny` + OPD. Following Lightning-OPD (Wu et al., 2026), we first perform SFT warmup and then OPD; memories learned at both stages transfer directly to frozen recipients (Table 1). On Qwen3-1.7B, Olympiad accuracy progresses from 10.6% (Base) to 22.6% (SFT) and 26.2% (OPD), while tool-use Macro-F1 on Qwen3-8B follows 20.6% → 23.4% → 29.4%. Across tokenizers, OPD-trained memory raises MiMo-7B's MATH-500 accuracy from 38.4% to 76.2% and LCB-V5/V6 pass@4 from 17.2%/18.9% to 39.4%/37.9%, extending transfer to both math and code. Transfer also reaches larger recipients: Ling-2.6-108B gains 31.7 points on AIME24, while MiMo-Flash-V2 improves from 16.7% to 50.0% on AIME24 and from 43.0% to 61.2% on Olympiad. We also observe code performance regressions on larger recipients (*e.g.*, HY3) in Table 1, which we attribute to a limitation of OPD: a teacher offers limited benefit in a domain where the recipient is already comparably strong (as discussed in Appendix E).

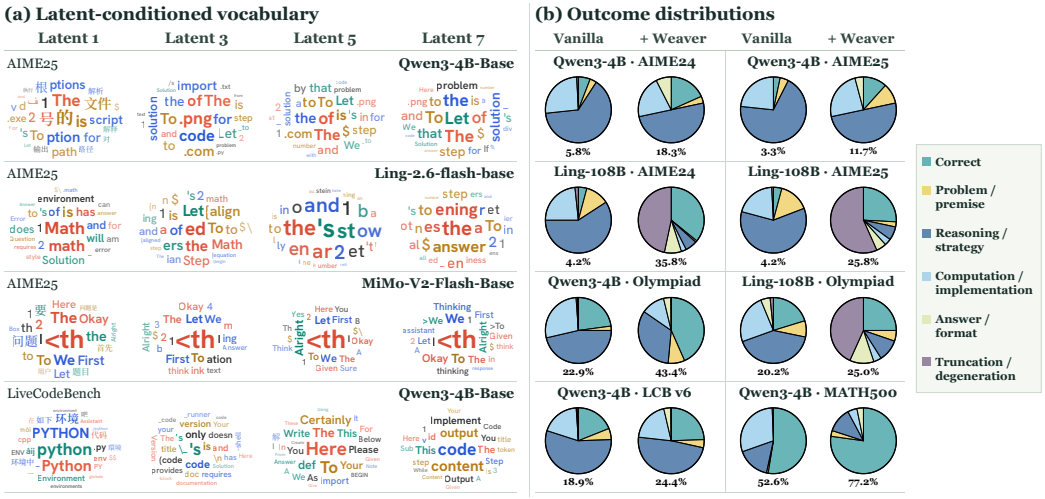


Figure 4: Latent memory and transferred reasoning. **(a)** Latent-conditioned vocabulary on AIME25 (top three rows) and LiveCodeBench (bottom row), with columns showing latent positions 1, 3, 5, and 7. Larger words indicate higher vocabulary probabilities induced by the latent memory in Equation (7). **(b)** Outcome distributions before and after adding the weaver, with one model–benchmark combination per pair. Correct sectors and labels report mathematical accuracy and LCB pass@4; + Weaver denotes +MemAny OPD. Additional results appear in Appendices D and E.

`MemAny` + RL. RL-trained memory improves reasoning on K&K logic puzzles (Table 2), raising the host QWEN2.5-7B-INSTRUCT's overall pass@2 from 26.4% to 29.6% and Avg@2 from 15.9% to 18.6%. After transfer, overall pass@2 rises from 15.4% to 18.1% for QWEN2.5-3B and from 39.6% to 42.4% for QWEN2.5-14B, supporting reuse by both smaller and larger models. The corresponding overall Avg@2 scores increase from 8.7% to 10.8% and from 27.3% to 28.6%, so the gains hold under both evaluation metrics. On 5-person puzzles, pass@2 improves from 7% to 12% on the 3B model and from 31% to 39% on the 14B model, whose 7-person score also rises from 20% to 25%. These benefits extend to unseen puzzle sizes: on OOD 8-person puzzles, the 14B model improves from 11% to 16% in pass@2 and from 6.5% to 8.5% in Avg@2, showing that transferred memory supports reasoning beyond the training distribution.

5.3 LATENT MEMORY VISUALIZATION

At latent positions 1, 3, 5, and 7 (Figure 4(a)), Qwen's LiveCodeBench clouds highlight `python`, `code`, and `import`, while Ling's AIME25 clouds include `Math` and `Solution`. In MIMO-V2-FLASH-BASE, the literal fragment `<th` is the highest-probability next token at all four displayed AIME25 positions, with probability rising from 21.4% at the first position to 93.2% at the fifth, alongside cues such as `First`, `Let`, and `Thinking`. We hypothesize that the weaver activates reasoning patterns acquired from thinking-style pretraining data, making these capabilities accessible through latent memory injection. More visualizations appear in Appendix D, including AIME24, Olympiad, and K&K, whose clouds combine participant names with reasoning cues across the 3B, 7B, and 14B recipients.

5.4 FAILURE-MODE ANALYSIS

Building on these results, we analyze paired generations with and without the weaver (Figure 4(b)) using five failure categories—problem/premise, reasoning/strategy, computation/implementation, answer/format, and truncation/degeneration—with definitions, annotation details, and the full atlas in Appendix E. On LiveCodeBench v6, QWEN3-4B-BASE corrects 128 previously incorrect responses, including 62 reasoning/strategy and 58 computation/implementation failures, together accounting for 93.8% of the corrected responses. For LING-2.6-108B-BASE, these two categories account for 32/39 (82.1%) and 25/30 (83.3%) of corrected responses on AIME24 and AIME25, where the correctness values shown in Figure 4(b) rise from 4.2% to 35.8% and 25.8%, respectively. Together with Qwen's MATH500 gain from 52.6% to 77.2%, these patterns highlight improved strategy selection and execution across recipient models.

Table 3: **Ablation Study.** Accuracy (%) on AIME24/25 and OlympiadBench (Olymp.).

	Setting	AIME24	AIME25	Olymp.	Mean
Qwen3-4B	No memory	5.8	3.3	22.9	10.7
	<think> prefix	7.5	5.0	23.6	12.0
	CoT prompt	8.3	5.8	24.4	12.8
	Untrained weaver	2.5	1.7	13.1	5.8
	Fixed soft prompt	1.7	2.5	16.8	7.0
	MemAny	**18.3**	**11.7**	**43.4**	**24.5**
MiMo-7B	No memory	3.3	2.5	8.6	4.8
	<think> prefix	3.3	3.3	9.8	5.5
	CoT prompt	4.2	3.3	11.0	6.2
	Untrained weaver	2.5	2.5	3.9	3.0
	Fixed soft prompt	2.5	2.5	2.4	2.5
	MemAny	**32.5**	**25.8**	**43.6**	**34.0**
HY3-Preview	No memory	6.7	13.3	42.0	20.7
	<think> prefix	8.3	14.2	42.9	21.8
	CoT prompt	9.2	15.0	43.6	22.6
	Untrained weaver	2.5	4.2	24.5	10.4
	Fixed soft prompt	1.7	5.0	20.6	9.1
	MemAny	**36.7**	**43.3**	**52.5**	**44.2**

Table 4: Performance (%). Explicit-memory math: Avg@4; **MemAny**: MATH500 accuracy and Olympiad Avg@4.

	Method	Math		Tool use	
		MATH500	Olympiad	Macro-F1	Acc.
Qwen3-4B	ExpeL	39.2 (↓ 13.4)	14.2 (↓ 8.6)	19.6 (↑ 2.3)	24.0 (↑ 1.0)
	Dyn. Cheatsheet	44.2 (↓ 8.4)	14.6 (↓ 8.2)	21.2 (↑ 3.9)	23.0 (→ 0.0)
	ReasoningBank	65.1 (↑ 12.5)	29.2 (↑ 6.4)	24.8 (↑ 7.5)	25.3 (↑ 2.3)
	MemAny	77.2 (↑ 24.6)	43.4 (↑ 20.5)	34.8 (↑ 17.5)	36.0 (↑ 13.0)
Qwen3-8B	ExpeL	47.9 (↓ 20.5)	20.0 (↓ 12.9)	26.4 (↑ 5.8)	28.3 (↑ 3.0)
	Dyn. Cheatsheet	55.8 (↓ 12.6)	25.6 (↓ 7.2)	22.5 (↑ 1.9)	23.7 (↓ 1.7)
	ReasoningBank	53.7 (↓ 14.7)	20.4 (↓ 12.5)	25.9 (↑ 5.3)	27.3 (↑ 2.0)
	MemAny	67.0 (↓ 1.4)	39.1 (↑ 6.2)	29.4 (↑ 8.8)	31.3 (↑ 6.0)
MiMo-7B	ExpeL	58.3 (↑ 19.9)	23.6 (↑ 15.0)	24.0 (↓ 6.3)	28.0 (↓ 7.3)
	Dyn. Cheatsheet	64.1 (↑ 25.7)	28.9 (↑ 20.2)	30.1 (↓ 0.2)	34.0 (↓ 1.3)
	ReasoningBank	68.9 (↑ 30.5)	35.4 (↑ 26.8)	27.1 (↓ 3.2)	26.3 (↓ 9.0)
	MemAny	76.2 (↑ 37.8)	43.6 (↑ 35.0)	33.1 (↑ 2.8)	36.0 (↑ 0.7)

5.5 Framework Analysis

Ablation Study. We ablate the components of **MemAny** to assess their effectiveness (Table 3). A <think> prefix or CoT instruction yields only modest gains over the no-memory model; **MemAny** exceeds the stronger baseline by 11.6, 27.8, and 21.6 percentage points in mean accuracy on Qwen3-4B, MiMo-7B, and HY3-Preview, respectively, demonstrating the value of learned memory. Replacing the trained weaver with an untrained one, with the eight-latent budget, recipient-side bridge, and scale unchanged, reduces performance by 18.7–33.8 points across the three recipients, confirming the importance of weaver training. Finally, we optimize a fixed soft prompt of eight tokens shared across all questions; input-dependent memory outperforms this variant by 17.5–35.1 points, highlighting the importance of tailoring memory to the current problem.

Efficiency Analysis. **MemAny** requires only 89.9 minutes for math and 138.7 minutes for code to complete the SFT and OPD training loops on a single NVIDIA L40S server, demonstrating low training cost. At inference, **MemAny** elicits longer generated responses alongside substantial accuracy gains, raising Qwen3-4B's AIME24 accuracy from 5.8% to 18.3% and HY3-Preview's AIME25 accuracy from 13.3% to 43.3%. We have provided a detailed efficiency analysis on training/inference wall-clock time and token cost. in Appendix C.

5.6 Comparison with Explicit Memory

A natural question is whether explicit skills and external memory banks can also support capability transfer across models. We compare **MemAny** with ExpeL (Zhao et al., 2024), Dynamic Cheatsheet (Suzgun et al., 2025), and ReasoningBank (Ouyang et al., 2026) on mathematical reasoning and tool use with Qwen3-4B, Qwen3-8B, and MiMo-7B as recipients (Table 4). Explicit-memory transfer varies substantially across recipients and tasks: ReasoningBank improves MATH500 by 12.5 points on Qwen3-4B but reduces it by 14.7 points on Qwen3-8B, while all three explicit-memory methods reduce MiMo-7B tool-use accuracy by 1.3–9.0 points. **MemAny** leads all 12 reported model–metric comparisons, exceeding the strongest explicit-memory baseline by 2.0–14.1 points. These results demonstrate stronger and more consistent capability transfer through generative latent memory across model scales and families.

6 Conclusion

We introduced **MemAny**, a framework that turns model adaptation into portable generative memory, allowing acquired capabilities to be trained once and reused across models for various sizes and families. By routing learning into an external memory weaver and expressing its outputs through each recipient model's own vocabulary embedding space, **MemAny** separates capability acquisition from the model that ultimately uses it. Experiments combining **MemAny** with SFT, RL, and OPD demonstrate transferable gains in mathematical and logical reasoning, code generation and tool-use, including transfer from Qwen3-4B to 200B+ recipients (MIMO-V2, HY3-Preview) across model families and tokenizers without retraining the memory or model. This work points toward a future in which knowledge learned by one model can benefit many, turning advances in individual models into lasting, shared progress across generations of AI systems.

AI USE STATEMENT

In this work, we used generative AI tools, including OpenAI Codex, for literature research, manuscript polishing, figure preparation, code development, and refinement of methodological descriptions. LLM-based training supervision and evaluation are documented in Appendices B.2, B.4 and E. We have reviewed all AI-assisted work for correctness and consistency and take responsibility for the final content, including text, claims, code, and figures produced with generative AI assistance.

ETHICS STATEMENT

This work evaluates capability transfer through latent memory on academic benchmarks. As with other LLM-based systems, deployment should account for unreliable outputs, bias, privacy risks, and misuse when reusing capabilities across models.

REPRODUCIBILITY STATEMENT

Our codebase is anonymously available at https://anonymous.4open.science/r/memany-anony-60CC/, including the complete **MemAny** training and cross-model transfer pipelines. We will also release our trained weaver checkpoints publicly upon acceptance to help the community rapidly share and reuse learned capabilities through plug-and-play memory. Methodological and experimental details are provided in Section 4 and Appendices A and B.

REFERENCES

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token Merging: Your ViT But Faster, 2022. URL https://arxiv.org/abs/2210.09461.

ByteDance. GitHub - volcengine/OpenViking: OpenViking is an open-source context database designed specifically for AI Agents(such as openclaw). OpenViking unifies the management of context (memory, resources, and skills) that Agents need through a file system paradigm, enabling hierarchical context delivery and self-evolving. — github.com. https://github.com/volcengine/{O}pen{V}iking, 2026.

Quy-Anh Dang and Chris Ngo. Reinforcement learning for reasoning in small llms: What works and what doesn't, 2026. URL https://arxiv.org/abs/2503.16219.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying

Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Jingcheng Deng, Liang Pang, Zihao Wei, Shicheng Xu, Zenghao Duan, Kun Xu, Yang Song, Huawei Shen, and Xueqi Cheng. LLM Latent Reasoning as Chain of Superposition, 2025. URL https://arxiv.org/abs/2510.15522.

Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Jun Zhao, Wei Shen, Yuhao Zhou, Zhiheng Xi, Xiao Wang, Xiaoran Fan, Shiliang Pu, Jiang Zhu, Rui Zheng, Tao Gui, Qi Zhang, and Xuanjing Huang. Loramoe: Alleviate world knowledge forgetting in large language models via moe-style plugin, 2024. URL https://arxiv.org/abs/2312.09979.

Weihua Du, Pranjal Aggarwal, Sean Welleck, and Yiming Yang. Agentic-r1: Distilled dual-strategy reasoning, 2025. URL https://arxiv.org/abs/2507.05707.

Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Shi Jie, Xiang Wang, Xiangnan He, and Tat seng Chua. Alphaedit: Null-space constrained knowledge editing for language models, 2025a. URL https://arxiv.org/abs/2410.02355.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory. arXiv preprint arXiv:2508.06433, 2025b.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. Advances in Neural Information Processing Systems, 38:46375–46408, 2026a.

Shiyuan Feng, Huan-ang Gao, Haohan Chi, Hanlin Wu, Zhilong Zhang, Zheng Jiang, Bingxiang He, Wei-Ying Ma, Ya-Qin Zhang, and Hao Zhou. Weak-to-strong generalization via direct on-policy distillation. arXiv preprint arXiv:2607.05394, 2026b. URL https://arxiv.org/abs/2607.05394.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. MiniLLM: Knowledge distillation of large language models. In International Conference on Learning Representations, 2024. URL https://arxiv.org/abs/2306.08543.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space, 2024. URL https://arxiv.org/abs/2412.06769.

Yubo Hou, Zhisheng Chen, Tao Wan, and Zengchang Qin. Flashmem: Distilling intrinsic latent memory via computation reuse. arXiv preprint arXiv:2601.05505, 2026.

Chuanrui Hu, Xingze Gao, Zuyi Zhou, Dannong Xu, Yi Bai, Xintong Li, Hui Zhang, Tong Li, Chong Zhang, Lidong Bing, et al. Evermemos: A self-organizing memory operating system for structured long-horizon reasoning. arXiv preprint arXiv:2601.02163, 2026a.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. ICLR, 1(2):3, 2022.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, Senjie Jin, Jiejun Tan, Yanbin Yin, Jiongnan Liu, Zeyu Zhang, Zhongxiang Sun, Yutao Zhu, Hao Sun, Boci Peng, Zhenrong Cheng, Xuanbo Fan, Jiaxin Guo, Xinlei Yu, Zhenhong Zhou, Zewen Hu, Jiahao Huo, Junhao Wang, Yuwei Niu, Yu Wang, Zhenfei Yin, Xiaobin Hu, Yue Liao, Qiankun Li, Kun Wang, Wangchunshu Zhou, Yixin Liu, Dawei Cheng, Qi Zhang, Tao Gui, Shirui Pan, Yan Zhang, Philip Torr, Zhicheng Dou, Ji-Rong Wen, Xuanjing Huang, Yu-Gang Jiang, and Shuicheng Yan. Memory in the age of ai agents, 2026b. URL https://arxiv.org/abs/2512.13564.

Wei-Chieh Huang, Weizhi Zhang, Yueqing Liang, Yuanchen Bei, Yankai Chen, Tao Feng, Xinyu Pan, Zhen Tan, Yu Wang, Tianxin Wei, Shanglin Wu, Ruiyao Xu, Liangwei Yang, Rui Yang, Wooseong Yang, Chin-Yuan Yeh, Hanrong Zhang, Haozhen Zhang, Siqi Zhu, Henry Peng Zou, Wanjia Zhao, Song Wang, Wujiang Xu, Zixuan Ke, Zheng Hui, Dawei Li, Yaozu Wu, Langzhou He, Chen Wang, Xiongxiao Xu, Baixiang Huang, Juntao Tan, Shelby Heinecke, Huan Wang, Caiming Xiong, Ahmed A. Metwally, Jun Yan, Chen-Yu Lee, Hanqing Zeng, Yinglong Xia, Xiaokai Wei, Ali Payani, Yu Wang, Haitong Ma, Wenya Wang, Chenguang Wang, Yu Zhang, Xin Wang, Yongfeng Zhang, Jiaxuan You, Hanghang Tong, Xiao Luo, Xue Liu, Yizhou Sun, Wei Wang, Julian McAuley, James Zou, Jiawei Han, Philip S. Yu, and Kai Shu. Rethinking memory mechanisms of foundation agents in the second half: A survey, 2026. URL https://arxiv.org/abs/2602.06052.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code, 2024. URL https://arxiv.org/abs/2403.07974.

Pengcheng Jiang, Jiacheng Lin, Zhiyi Shi, Zifeng Wang, Luxi He, Yichen Wu, Ming Zhong, Peiyang Song, Qizheng Zhang, Heng Wang, Xueqiang Xu, Hanwen Xu, Pengrui Han, Dylan Zhang, Jiashuo Sun, Chaoqi Yang, Kun Qian, Tian Wang, Changran Hu, Manling Li, Quanzheng Li, Hao Peng, Sheng Wang, Jingbo Shang, Chao Zhang, Jiaxuan You, Liyuan Liu, Pan Lu, Yu Zhang, Heng Ji, Yejin Choi, Dawn Song, Jimeng Sun, and Jiawei Han. Adaptation of agentic ai: A survey of post-training, memory, and skills, 2026. URL https://arxiv.org/abs/2512.16301.

Tianjie Ju, Yubin Zheng, Hanyi Wang, Haodong Zhao, and Gongshen Liu. Is continuous prompt a combination of discrete prompts? towards a novel view for interpreting continuous prompts. In Findings of the association for computational linguistics: ACL 2023, pp. 7804–7819, 2023.

Weizhen Li, Jianbo Lin, Zhuosong Jiang, Jingyi Cao, Xinpeng Liu, Jiayu Zhang, Zhenqiang Huang, Qianben Chen, Weichen Sun, Qiexiang Wang, et al. Chain-of-agents: End-to-end agent foundation models via multi-agent distillation and agentic rl. arXiv preprint arXiv:2508.13167, 2025.

Yaxuan Li, Yuxin Zuo, Bingxiang He, Jinqian Zhang, Chaojun Xiao, Cheng Qian, Tianyu Yu, Huanang Gao, Wenkai Yang, Zhiyuan Liu, et al. Rethinking on-policy distillation of large language models: Phenomenology, mechanism, and recipe. arXiv preprint arXiv:2604.13016, 2026.

Youwei Liang, Chongjian Ge, Zhan Tong, Yibing Song, Jue Wang, and Pengtao Xie. Not All Patches are What You Need: Expediting Vision Transformers via Token Reorganizations, 2022. URL https://arxiv.org/abs/2202.07800.

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. Prorl: Prolonged reinforcement learning expands reasoning boundaries in large language models, 2025. URL https://arxiv.org/abs/2505.24864.

Weize Liu, Minghui Liu, Sy-Tuyen Ho, Souradip Chakraborty, Xiyao Wang, and Furong Huang. Agentic critical training, 2026. URL https://arxiv.org/abs/2603.08706.

Pan Lu, Jiayi Sheng, Luna Lyu, Jikai Jin, Tony Xia, Alex Gu, and James Zou. Solving inequality proofs with large language models, 2025. URL https://arxiv.org/abs/2506.07927.

Zhengxi Lu, Zhiyuan Yao, Jinyang Wu, Chengcheng Han, Qi Gu, Xunliang Cai, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Skill0: In-context agentic reinforcement learning for skill internalization. arXiv preprint arXiv:2604.02268, 2026.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In International Conference on Learning Representations, 2026. URL https://arxiv.org/abs/2509.25140.

Hayley Ross, Ameya Sunil Mahabaleshwarkar, and Yoshi Suhara. When2Call: When (not) to call tools. In Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the

Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pp. 3391–3409. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.174. URL https://aclanthology.org/2025.naacl-long.174/.

Michael S. Ryoo, AJ Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. TokenLearner: What Can 8 Learned Tokens Do for Images and Videos?, 2021. URL https://arxiv.org/abs/2106.11297.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. Codi: Compressing chain-of-thought into continuous space via self-distillation, 2025. URL https://arxiv.org/abs/2502.21074.

Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, Zifeng Wang, Sayna Ebrahimi, and Hao Wang. Continual learning of large language models: A comprehensive survey. ACM Computing Surveys, 58(5):1–42, 2025.

Joykirat Singh, Raghav Magazine, Yash Pandya, and Akshay Nambi. Agentic reasoning and tool integration for llms via reinforcement learning. arXiv preprint arXiv:2505.01441, 2025.

Mingyang Song and Mao Zheng. A survey of on-policy distillation for large language models. arXiv preprint arXiv:2604.00626, 2026.

Liangcai Su, Zhen Zhang, Guangyu Li, Zhuo Chen, Chenxi Wang, Maojia Song, Xinyu Wang, Kuan Li, Jialong Wu, Xuanzhong Chen, Zile Qiao, Zhongwang Zhang, Huifeng Yin, Shihao Cai, Runnan Fang, Zhengwei Tao, Wenbiao Yin, Chenxiong Qian, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. Scaling agents via continual pre-training, 2025. URL https://arxiv.org/abs/2509.13310.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory, 2025. URL https://arxiv.org/abs/2504.07952.

Tongyi DeepResearch Team, Baixuan Li, Bo Zhang, Dingchu Zhang, Fei Huang, Guangyu Li, Guoxin Chen, Huifeng Yin, Jialong Wu, Jingren Zhou, Kuan Li, Liangcai Su, Litu Ou, Liwen Zhang, Pengjun Xie, Rui Ye, Wenbiao Yin, Xinmiao Yu, Xinyu Wang, Xixi Wu, Xuanzhong Chen, Yida Zhao, Zhen Zhang, Zhengwei Tao, Zhongwang Zhang, Zile Qiao, Chenxi Wang, Donglei Yu, Gang Fu, Haiyang Shen, Jiayin Yang, Jun Lin, Junkai Zhang, Kui Zeng, Li Yang, Hailong Yin, Maojia Song, Ming Yan, Minpeng Liao, Peng Xia, Qian Xiao, Rui Min, Ruixue Ding, Runnan Fang, Shaowei Chen, Shen Huang, Shihang Wang, Shihao Cai, Weizhou Shen, Xiaobin Wang, Xin Guan, Xinyu Geng, Yingcheng Shi, Yuning Wu, Zhuo Chen, Zijian Li, and Yong Jiang. Tongyi deepresearch technical report, 2026. URL https://arxiv.org/abs/2510.24701.

Changyue Wang, Weihang Su, Qingyao Ai, Yichen Tang, and Yiqun Liu. Knowledge editing through chain-of-thought. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pp. 10673–10693, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.540. URL https://aclanthology.org/2025.emnlp-main.540/.

Chloe Wang, Haotian Cui, Andrew Zhang, Ronald Xie, Hani Goodarzi, and Bo Wang. scgpt-spatial: Continual pretraining of single-cell foundation model for spatial transcriptomics. biorxiv, pp. 2025–02, 2025b.

Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Wise: Rethinking the knowledge memory for lifelong model editing of large language models, 2024. URL https://arxiv.org/abs/2405.14768.

Yaoxiang Wang, Haoling Li, Xin Zhang, Jie Wu, Xiao Liu, Wenxiang Hu, Zhongxin Guo, Yangyu Huang, Ying Xin, Yujiu Yang, et al. Epicoder: Encompassing diversity and complexity in code generation. arXiv preprint arXiv:2501.04694, 2025c.

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

Yiping Wang, Qing Yang, Zhiyuan Zeng, Liliang Ren, Liyuan Liu, Baolin Peng, Hao Cheng, Xuehai He, Kuan Wang, Jianfeng Gao, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. Reinforcement learning for reasoning in large language models with one training example, 2025d. URL https://arxiv.org/abs/2504.20571.

Tianxin Wei, Ting-Wei Li, Zhining Liu, Xuying Ning, Ze Yang, Jiaru Zou, Zhichen Zeng, Ruizhong Qiu, Xiao Lin, Dongqi Fu, et al. Agentic reasoning for large language models. arXiv preprint arXiv:2601.12538, 2026.

Yecheng Wu, Song Han, and Hai Cai. Lightning opd: Efficient post-training for large reasoning models with offline on-policy distillation. arXiv preprint arXiv:2604.13010, 2026.

Wujiang Xu, Kai Mei, Hang Gao, Juntao Tan, Zujie Liang, and Yongfeng Zhang. A-mem: Agentic memory for llm agents, 2025a. URL https://arxiv.org/abs/2502.12110.

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. Softcot: Soft chain-of-thought for efficient reasoning with llms, 2025b. URL https://arxiv.org/abs/2502.12134.

Xiangyuan Xue, Yifan Zhou, Guibin Zhang, Zaibin Zhang, Yijiang Li, Chen Zhang, Zhenfei Yin, Philip Torr, Wanli Ouyang, and Lei Bai. Comas: Co-evolving multi-agent systems via interaction rewards. arXiv preprint arXiv:2510.08529, 2025.

Xiangyuan Xue, Yifan Zhou, Zidong Wang, Shengji Tang, Philip Torr, Wanli Ouyang, Lei Bai, and Zhenfei Yin. Strata: Incentivizing agentic reinforcement learning with strategic trajectory abstraction. arXiv preprint arXiv:2605.06642, 2026.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-Math technical report: Toward mathematical expert model via self-improvement. arXiv preprint arXiv:2409.12122, 2024. URL https://arxiv.org/abs/2409.12122.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, and Bo Zheng et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. Agent lumos: Unified and modular training for open-source language agents, 2024. URL https://arxiv.org/abs/2311.05657.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. Advances in Neural Information Processing Systems, 38:113222–113244, 2026.

Guibin Zhang, Muxin Fu, and Shuicheng Yan. Memgen: Weaving generative latent memory for self-evolving agents, 2025a. URL https://arxiv.org/abs/2509.24704.

Guibin Zhang, Haotian Ren, Chong Zhan, Zhenhong Zhou, Junhao Wang, He Zhu, Wangchunshu Zhou, and Shuicheng Yan. Memevolve: Meta-evolution of agent memory systems, 2025b. URL https://arxiv.org/abs/2512.18746.

Kai Zhang, Xiangchao Chen, Bo Liu, Tianci Xue, Zeyi Liao, Zhihan Liu, Xiyao Wang, Yuting Ning, Zhaorun Chen, Xiaohan Fu, Jian Xie, Yuxuan Sun, Boyu Gou, Qi Qi, Zihang Meng, Jianwei Yang, Ning Zhang, Xian Li, Ashish Shah, Dat Huynh, Hengduo Li, Zi Yang, Sara Cao, Lawrence Jang, Shuyan Zhou, Jiacheng Zhu, Huan Sun, Jason Weston, Yu Su, and Yifan Wu. Agent learning via early experience, 2026. URL https://arxiv.org/abs/2510.08558.

Mengqi Zhang, Xiaotian Ye, Qiang Liu, Pengjie Ren, Zhumin Chen, et al. Uncovering overfitting in large language model editing. In International Conference on Learning Representations, volume 2025, pp. 79550–79568, 2025c.

Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, Shuohang Wang, Yelong Shen, and Xin Eric Wang. Soft Thinking: Unlocking the Reasoning Potential of LLMs in Continuous Concept Space, 2025d. URL https://arxiv.org/abs/2505.15778.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners, 2024. URL https://arxiv.org/abs/2308.10144.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. arXiv preprint arXiv:2507.18071, 2025.

A Vocabulary Coefficients: Construction and Transfer

The coefficient map $\mathcal{C}_m$ is part of the memory weaver $\mathcal{W}_\phi$ introduced in Section 4.1. It converts context-dependent memory representations into coefficients over the recipient's vocabulary; these coefficients determine the weighted embedding sum in Equation (7).

A.1 Affine Vocabulary Coefficients

The weaver uses K learned query positions to produce internal memory representations $\mathbf{Z}_\phi(q) \in \mathbb{R}^{K\times d_c}$, with $d_c = 2{,}048$. These representations are normalized using LayerNorm followed by row-wise ℓ_2 normalization with 10^{-6} added to the norm. Let $\mathbf{Z}_m$ denote their expression in the recipient's coefficient basis. The map assigns vocabulary coefficients through an affine transformation:

$$\mathbf{A}^{(m)}(q) = \mathcal{C}_m(\mathbf{Z}_\phi(q); \alpha_m) = \frac{1}{V_m}\mathbf{1}_K\mathbf{1}_{V_m}^\top + \frac{\alpha_m}{V_m}\mathbf{Z}_m\widetilde{\mathbf{B}}_m^\top. \tag{11}$$

Here, $\widetilde{\mathbf{B}}_m$ supplies a fixed coefficient basis derived from the recipient's frozen embeddings. Each coefficient combines a uniform offset with a context-dependent adjustment; centering the basis makes the coefficients sum to one, and α_m controls the adjustment's strength.

Constructing the coefficient basis. For model p, let $\boldsymbol{\mu}_p$ be the mean token embedding and $\mathbf{X}_p = \mathbf{E}_p - \mathbf{1}_{V_p}\boldsymbol{\mu}_p^\top$ the centered embeddings. We retain the leading $c_p = \min(d_c, d_p)$ eigenvectors $\mathbf{Q}_{p,c}$ and eigenvalues $\mathbf{\Lambda}_{p,c}$ of the covariance $\mathbf{X}_p^\top\mathbf{X}_p/V_p$, and set $\widetilde{\mathbf{B}}_p = \mathbf{X}_p\mathbf{Q}_{p,c}\mathbf{\Lambda}_{p,c}^{-1/2}$, using a pseudoinverse for zero eigenvalues. The coefficients are therefore affine in the weaver's memory representations, with vocabulary geometry fixed by the recipient. Their recipient coordinates $\mathbf{Z}_m$ are obtained through the alignment below.

A.2 Recipient Coefficient Alignment

All cross-model transfers calibrate the coefficient map through shared token anchors. The reference r is the weaver backbone, which is the Qwen3-4B-Base host for math and code; anchors pair identical token strings in its vocabulary and the recipient's vocabulary. For each model, we center the anchor embeddings, project them onto $\mathbf{Q}_{p,c}$, divide coordinate i by $\sqrt{\lambda_{p,i}} + 10^{-6}$, and normalize each row with a norm floor of 10^{-12}, obtaining $\mathbf{G}_p$. A thin singular value decomposition gives the alignment:

$$\mathbf{G}_r^\top\mathbf{G}_m = \mathbf{U}\mathbf{S}\mathbf{V}^\top, \qquad \mathbf{R}_{r\to m} = \mathbf{U}\mathbf{V}^\top, \qquad \mathbf{Z}_m = \mathbf{Z}_\phi(q)^{[:c_r]}\mathbf{R}_{r\to m}. \tag{12}$$

Substituting $\mathbf{Z}_m$ into Equation (11) yields coefficients over the recipient vocabulary without training the weaver or recipient further. This is orthogonal Procrustes alignment for equal dimensions and a rectangular map otherwise. Shared token strings identify corresponding anchors; the fitted map resolves the orientation of independently computed bases, including their sign ambiguity. The identity map is used only when the recipient reuses the reference embedding basis. Anchor normalization estimates the alignment, while coefficient computation retains the unnormalized basis $\widetilde{\mathbf{B}}_m$.

A.3 Coefficient Scale and Efficient Evaluation

The scale α_m controls how strongly the context-dependent coefficients move memory away from the recipient's mean embedding. Multiplying the coefficients by $\mathbf{E}_m$ gives the compact expression

$$\mathbf{M}^{(m)}(q) = \mathcal{C}_m(\mathbf{Z}_\phi(q); \alpha_m)\mathbf{E}_m = \mathbf{1}_K \boldsymbol{\mu}_m^\top + \alpha_m \mathbf{Z}_m \boldsymbol{\Lambda}_{m,c}^{1/2} \mathbf{Q}_{m,c}^\top. \tag{13}$$

This is the same coefficient-weighted memory as Equation (7), evaluated without materializing the full $K \times V_m$ coefficient matrix. The mean and projection are cached; the aligned projection is $\mathbf{L}_m = \mathbf{Q}_{m,c}\boldsymbol{\Lambda}_{m,c}^{1/2}\mathbf{R}_{r\to m}^\top$.

Host training learns a memory scale. For transfer, calibration sets a recipient-specific value computed once from frozen embedding statistics: the mean norm of centered recipient embeddings divided by the mean projected norm $\|\mathbf{u}_i\mathbf{L}_m^\top\|_2$ of 4,096 normalized standard Gaussian vectors $\mathbf{u}_i \in \mathbb{R}^{c_r}$. This sets the memory scale without task examples or gradient updates. Vocabulary statistics and PCA are computed in float32 on CPU and cached in the recipient's computation dtype. At inference, the prompt is tokenized separately for the weaver and recipient. We concatenate the $K = 8$ generated memory vectors after the recipient's prompt input embeddings and pass the augmented sequence through the first Transformer layer and all subsequent layers before ordinary autoregressive decoding.

B Experiment Details

B.1 Baseline Setup

We compare each recipient model with and without the learned memory. In both conditions, the reasoner parameters remain fixed. Transfer reuses the source-trained weaver; the recipient's token embeddings determine its alignment and memory scale, without further optimization on task examples.

We evaluate each system with the prompt format appropriate to its training stage. In the OPD setting, the unmodified mathematical baseline uses fixed four-shot CoT demonstrations and the code baseline uses LCB's `GenericBase` one-shot template; **`MemAny`** uses direct mathematical instructions and the code instruction template. Providing demonstrations is important for a fair base-model evaluation: they elicit completion-based reasoning and establish the expected answer format, following established base-model evaluation practice (Yang et al., 2024; Jain et al., 2024). OPD matches the teacher's response distribution on student-generated trajectories, thereby teaching the student how to follow task instructions and structure its answers alongside task-solving behavior, as illustrated by instruction-following distillation in MiniLLM (Gu et al., 2024). In **`MemAny`**, this behavior is learned through the memory weaver while the reasoner remains frozen. Direct-instruction evaluation therefore tests a capability targeted by training and follows the instruction-based post-training evaluations used in Lightning-OPD, Direct-OPD, and MiniLLM (Wu et al., 2026; Feng et al., 2026b; Gu et al., 2024).

KK uses identical task instructions and the same native chat template in both conditions; complete prompts appear in Appendix B.6.

B.2 Training Dataset Setup

Supervised initialization. We use separate supervised corpora for mathematical reasoning, code generation, and KK. For the standalone IneqMath SFT setting, rejection sampling from Qwen3-4B-Instruct yields 1.2K accepted trajectories for memory training. The OPD math warm start uses verified solutions to OpenThoughts3 problems generated by QWEN3-8B with thinking disabled. To reconstruct this corpus, we sample eight candidates per problem, verify their final boxed answers against the reference, and retain the shortest correct solution between 100 and 8,000 tokens. Problems without an accepted solution are excluded. For code, we sample 30,000 examples from EpiCoder-func-380k (Wang et al., 2025c) and use their reference programs, normalized to Python code blocks. The math and code weavers are initialized independently. The KK training set contains 4,500 puzzles, balanced across 3–7 people. Sampling two teacher responses per puzzle and retaining at most one correct, correctly formatted response produces 2,004 supervised examples.

Training with OPD and GRPO. After supervised initialization, math OPD uses 10,000 prompts from DAPO-Math-17k (Yu et al., 2026), and code OPD uses CodeContests-O. For each task, we sample responses from the initialized student and obtain token-level log-probabilities from the QWEN3-8B teacher. Responses that exhaust the generation budget are removed; the retained trajectories and teacher scores are cached for optimization. KK GRPO uses all 4,500 training puzzles, including those without an accepted supervised response. Data-generation and rollout parameters are listed in Table 7.

Tool-use data. We use NVIDIA's When2Call: When (not) to Call Tools, whose questions are primarily adapted from BFCL. Tool-use memory training uses 2,000 examples, and evaluation uses the 300 questions in the dataset's `llm_judge` test set. The three gold action labels and the judging protocol are described in Appendix B.4.

SFT transfer evaluation. The standalone SFT study uses only IneqMath's relation-prediction subset and evaluates Qwen3-1.7B/4B/8B in base and thinking modes (Figure 3). Seen100 comprises 100 problems sampled from the training set and reports single-response accuracy; dev50 comprises 50 relation problems from the development set and reports Avg@4, the mean correctness of four responses per problem. For each problem, the official `build_query` output is supplied as the user message through the evaluated Qwen model's native chat template, with no additional system message, demonstrations, or reference answer. Both conditions receive the same zero-shot question; **`MemAny`** injects latent memory without adding textual hints. Base models use greedy decoding and thinking models use sampling, with the settings in Table 7; responses reaching the 16,000-token limit remain in official grading and are separately flagged as `length_limited`.

B.3 PARAMETER CONFIGURATIONS

Memory module. For math, code and tool-call, both the host reasoner and the weaver are initialized from QWEN3-4B-BASE. For KK, the host is QWEN2.5-7B-INSTRUCT and the weaver remains QWEN3-4B-BASE. Training updates the query vectors and their normalization parameters, projection head, memory scale, and LoRA adapters, while freezing the reasoner and the weaver's original backbone parameters. Table 5 summarizes the architecture and adapter settings. Evaluation uses the prompt-query bank only, inserting $K = 8$ memory tokens before response generation. Recipient coefficient calibration uses shared vocabulary anchors and embedding statistics; the construction and its vocabulary-weighted interpretation are given in Appendix A.

Table 5: Architecture and adaptation parameters of the memory weaver.

Settings	Hyperparameters
Memory	Prompt queries: 8; intermediate-generation queries: 8. Canonical dimension: 2,048; weaver hidden dimension: 2,560. Projection head: $2{,}560 \to 2{,}560 \to 2{,}048$, with GELU. Output normalization: LayerNorm followed by ℓ_2 normalization. Initial memory scale: $\sqrt{2{,}048}$.
LoRA	Rank: 128; scaling parameter: 256; dropout: 0.05. Target modules: `q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, and `down_proj`. Bias adaptation: disabled.
Implementation	Computation precision: bfloat16. Frozen: all reasoner parameters, weaver backbone and token embeddings. Trainable: LoRA adapters, queries and their normalization, projection head and scale.

Optimization. Supervised initialization minimizes cross-entropy on the completion tokens. Following Wu et al. (2026), OPD uses the cached student trajectories and teacher scores, with clipped importance ratios; the log-probability temperature in its objective is 1.0 and is independent of the rollout sampling temperature. For KK, GRPO normalizes rewards within each prompt's response

group and assigns zero advantages to groups with no reward variation. It applies an importance-ratio clipping objective and a squared log-ratio penalty relative to the rollout policy. Table 6 gives the stage-specific optimization parameters. The SFT hyperparameters in this table refer to the OPD and KK supervised initialization stages.

Table 6: Training hyperparameters. Within the SFT block, task-specific triples follow the order math / code / KK. OPD settings shared by math and code are stated once; differing settings are listed explicitly.

Settings	Hyperparameters
Training (SFT)	Optimizer: AdamW; learning rate: 2×10^{-5}; epochs: 2. Schedule: cosine decay; warmup fraction: 0.05. Microbatch per device: 1; gradient accumulation: 4 / 8 / 4. Effective batch size: 32 / 64 / 64. Sequence length: 8,192; loss: completion-only cross-entropy. Prompt/padding loss: masked; incomplete batches: dropped. Gradient checkpointing: enabled, non-reentrant. KK optimizer: $\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$, weight decay $= 0$, maximum gradient norm $= 1.0$.
Training (OPD)	Optimizer: AdamW; learning rate: 2×10^{-6}; epochs: 2. $\beta = (0.9, 0.98)$; $\epsilon = 10^{-8}$; weight decay: 0.1. Schedule: constant; warmup fraction: 0. Microbatch per device: 1; accumulation: 8 (math), 32 (code). Ratio clipping: $[0.8, 1.2]$; maximum gradient norm: 1.0. Prompt / response limits: 3,072 / 8,192 tokens. Teacher context: 12,288 tokens; loss temperature: 1.0. Entropy bonus: 0; reference-model KL coefficient: 0. Gradient checkpointing: non-reentrant (math), disabled (code).
Training (GRPO)	Optimizer: AdamW; learning rate: 5×10^{-7}; updates: 4,000. $\beta = (0.9, 0.999)$; $\epsilon = 10^{-8}$; weight decay: 0.01. Schedule: constant; warmup fraction: 0. Prompt groups per update: 2; responses per group: 6. Policy microbatch: 1; maximum sequence length: 8,192. Ratio clipping: $[0.8, 1.2]$; maximum gradient norm: 1.0. Squared log-ratio penalty coefficient: 0.01. Group-normalization stabilizer: 10^{-6}.

KK reward. For a response with a valid outer format, a fully correct assignment receives an answer reward of 2. For a parseable but incorrect assignment, the reward is $-1.5 + f$, where f is the fraction of correctly assigned identities. A nonempty but unparseable answer receives -2, an empty answer block receives -2.5, and a response that fails the format checks receives -4. We additionally subtract $0.001 \max(0, L - 1600)$, where L is the completion length in characters. Thus, the reward encourages correct assignments and concise, well-formed responses. The response-format requirements are given in Appendix B.4.

B.4 Evaluation Setup

Benchmarks and metrics. Table 8 lists the evaluation sets and their metrics. For a dataset of N problems, let $c_{ij} \in \{0, 1\}$ denote whether the jth response to problem i is correct. We compute

$$\text{Avg@}\, n = \frac{1}{Nn} \sum_{i=1}^{N} \sum_{j=1}^{n} c_{ij}, \qquad \text{pass@}\, n = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\left[\sum_{j=1}^{n} c_{ij} > 0 \right].$$

For $n = 1$, both reduce to answer accuracy. All reported scores are percentages. All tabulated performance scores, standard deviations, and percentage-point differences are rounded to one decimal place using round-half-to-even. All aggregates and differences are computed from unrounded values before the final rounding step. Decoding settings are summarized in Table 7; prompt templates and demonstrations are given in Appendix B.6.

Table 7: Decoding parameters for data construction, training rollouts, and evaluation. n denotes responses per problem; generation limits apply to each response.

Setting	n	Temperature	Top-p	Top-k	Max. tokens
Data construction and training					
Math rejection sampling	8	0.8	0.95	50	8,192
KK rejection sampling	2	0.8	0.95	Off	6,144
OPD student rollouts	1	0.7	0.90	–	8,192
GRPO student rollouts	6	0.9	0.95	Off	4,096
Evaluation					
OPD mathematical reasoning	1 or 4	0.6	0.95	Off	16,384
IneqMath SFT: base	1 or 4	0	1	–	16,000
IneqMath SFT: thinking	1 or 4	0.6	0.95	20	16,000
Code generation	4	0.6	0.95	Off	8,192
KK puzzles	2	0.7	0.95	Off	8,192

"Off" denotes sampling without top-k truncation; "–" denotes no explicit top-k override. The memory-augmented decoder uses a minimum of five new tokens. IneqMath base-mode evaluation uses greedy decoding; Seen100 uses one response per problem and dev50 uses four. The OPD mathematical evaluation sample count depends on the dataset (Table 8).

Table 8: Evaluation benchmarks and reporting protocols. The mathematical protocols listed here apply to the base models and **`MemAny`**. KK includes 100 puzzles at each of seven problem sizes.

Benchmark	Problems	Responses	Metric
Mathematical reasoning			
AIME 2024	30	4	Avg@4
AIME 2025	30	4	Avg@4
OlympiadBench subset	200	4	Avg@4
MATH-500	500	1	Accuracy
IneqMath Seen100 (SFT)	100	1	Accuracy
IneqMath dev50: relation (SFT)	50	4	Avg@4
Code generation			
LiveCodeBench V5	279	4	pass@4
LiveCodeBench V6	454	4	pass@4
Logical reasoning			
KK (2–8 people)	700	2	Avg@2, pass@2
Tool use			
When2Call (`llm_judge`)	300	1	Macro-F1, Accuracy

Mathematical reasoning. The OPD comparison uses fixed four-shot CoT demonstrations for the unmodified base model and direct instructions for the trained `MemAny` system. This choice follows the standard distinction between few-shot base-model evaluation (Yang et al., 2024) and instruction-based evaluation after OPD (Wu et al., 2026; Feng et al., 2026b). We regard this as the appropriate comparison: demonstrations give the base model the reasoning and answer-format examples needed to express its existing capabilities, while direct instructions assess the teacher-guided response behavior acquired through memory training (Appendix B.1). Our mathematical reasoning protocol aims to reproduce the models' officially reported performance as closely as possible and primarily follows the Qwen2.5-Math evaluation setup.[1] On MATH500, we obtain near-identical scores to the official reports across the evaluated model families. The remaining discrepancy concerns AIME: we could not reproduce the MiMo series' reported AIME scores, and its original evaluation suite was unavailable. We therefore report the AIME scores reproduced under the standard Qwen-math protocol, using standard generation-length limits, answer parsing, and correctness evaluation consistent with those used for the Qwen series.

For OPD evaluation, AIME and OlympiadBench use four independent sampling passes; we report the mean and population standard deviation of accuracy across passes. MATH-500 uses one sampled response per problem. The grader extracts the last `\boxed{...}` expression and checks symbolic equivalence with the reference using `math_verify`. Missing or unbalanced boxed answers are incorrect; exact-string comparison is used if the symbolic verifier is unavailable. Generation terminates at EOS or the token limit. The four-shot baseline also stops when a new `Problem:` marker begins, preventing continuation into another example.

Code generation. The base-model baseline uses LCB's `GenericBase` one-shot prompt to establish the expected code-completion format (Jain et al., 2024). After memory training, `MemAny` uses the instruction template, following Lightning-OPD's instruction-based code evaluation (Wu et al., 2026). As in math, the demonstration elicits the base model's completion ability, while the instruction template evaluates the response behavior learned through distillation; full templates appear in Appendix B.6. LCB V5 contains problems released between August 1, 2024 and February 1, 2025; V6 extends this interval to May 1, 2025. The V5 problems form a subset of V6, and both scores reuse the same generated solutions. Each problem has four candidate programs. The official LCB execution harness checks all public and private tests with a six-second test timeout; a candidate succeeds only if every test passes. We evaluate the final fenced code block for the instruction prompt and the raw continuation for `GenericBase`. For instruction-format responses, a missing fenced code block is a failure. Execution errors and timeouts are failures in both conditions.

Knights-and-Knaves. We evaluate 100 held-out puzzles at each size from 2 to 8 people. The 2- and 8-person splits measure generalization beyond the 3–7-person training range. A response is correct only if every identity matches the reference and the answer satisfies the required format: one `<answer>...</answer>` block, one identity per line in the reference order, and no trailing text. The scorer rejects disallowed markup and non-ASCII/control characters, apart from ordinary whitespace. It also imposes a 4,096-character response limit, which is distinct from the token budget used during generation. We report Avg@2 and pass@2 for each problem size, with an unweighted average across the seven sizes.

Tool use. We use DeepSeek-V4-Flash-0731 as the judge for the 300-question When2Call `llm_judge` test set. For each generated response, the judge assigns one of four action labels: `tool_call`, `request_for_info`, `cannot_answer`, or `direct_answer`. The gold labels comprise the first three classes; a `direct_answer` prediction is therefore incorrect. Tool-use Macro-F1 is the unweighted mean of the F1 scores for the three gold classes, $\text{Macro-F1} = \frac{1}{3}\sum_{c\in\mathcal{C}} \text{F1}_c$, where $\mathcal{C}$ is the set of gold labels. Tool-use accuracy is the proportion of test questions for which the judge-assigned label exactly matches the gold label, $\text{Accuracy} = \frac{1}{300}\sum_{i=1}^{300} \mathbb{I}[\hat{y}_i = y_i]$. Both metrics are reported as percentages in Tables 1 and 4.

[1] https://github.com/QwenLM/Qwen2.5-Math

B.5 EXPLICIT-MEMORY TRANSFER PROTOCOL

Experience accumulation. The mathematical explicit-memory protocol uses frozen QWEN3-4B-INSTRUCT-2507 to accumulate experience for ExpeL, Dynamic Cheatsheet, and ReasoningBank from the same 10,000 DAPO-Math-17k problems used in the mathematical OPD stage of **MemAny**. Problems follow a shared deterministic order; all problems use the preceding snapshot, after which solution trajectories and correctness feedback are distilled into memory. The final memory bank and retrieval index are frozen after all 10,000 problems. Evaluation uses this final snapshot, without validation-based checkpoint selection.

Memory retrieval and evaluation. ExpeL retrieves textual rules and successful examples; Dynamic Cheatsheet combines a global cheatsheet with relevant training summaries; ReasoningBank retrieves structured experience entries. ExpeL and Dynamic Cheatsheet use EvolveLab adaptations, while ReasoningBank uses a separate adapter. At test time, recipient models read retrieved experience and generate answers with fixed weights; the memory bank remains read-only, and Dynamic Cheatsheet retrieval uses the stored cheatsheet and summaries directly. The supplied Qwen3-4B protocol allows up to 2,048 tokens of retrieved experience under the recipient tokenizer and uses a shared question template across the three explicit-memory methods. The comparison in Table 4 organizes results by recipient model and benchmark.

Reported mathematical scores. The frozen-memory results cover Qwen3-4B-Base, Qwen3-8B, and MiMo-7B; the explicit-memory protocol uses a 16,000-token generation cap for all methods.

B.6 PROMPT TEMPLATES

We provide the prompts used for mathematical reasoning, code generation, and KK below. OPD math and code prompts are supplied as raw text to the base reasoners. KK instructions are formatted with the recipient's native Qwen2.5 chat template, without an additional system message. Braced placeholders denote the test problem or its supplied starter code. For IneqMath SFT evaluation, we use the official `build_query` prompt in a single user message with the model's native chat template. We use only relation problems, which require `The answer is (Letter) Symbol`, selecting from the six A–F relation choices. No system prompt, few-shot demonstrations, or reference answers are added, and the textual input is identical for the baseline and **MemAny**.

Four-shot mathematical reasoning. For the unmodified base-model baseline, a shot consists of a complete demonstration problem and its worked solution, ending with a boxed answer. Every test problem receives the same four demonstrations in the same order; there is no example retrieval or problem-dependent selection. The examples come from the `math-evaluation-harness` chain-of-thought template and cover a geometric series, a circle-area calculation, an absolute-value bound, and a self-inverse rational function. The model is given all four problem–solution pairs, followed by the test problem and an empty `Solution:` field. It generates only the test solution.

The following box reproduces the complete prompt, including the four demonstration solutions. Adjacent demonstrations, and the final query, are separated by three newline characters. The test problem is substituted for {`problem`}; the demonstration answers remain fixed. This four-shot context supplies the unmodified base model with the task demonstrations needed for completion-based evaluation. The trained memory-augmented system instead uses the instruction prompt shown immediately afterward, following the post-training evaluation approach of Lightning-OPD (Wu et al., 2026).

Mathematics: complete four-shot baseline prompt

```
Problem:
Kevin Kangaroo begins hopping on a number line at 0. He wants to get
    to 1, but he can hop only $\frac{1}{3}$ of the distance. Each hop
    tires him out so that he continues to hop $\frac{1}{3}$ of the
    remaining distance. How far has he hopped after five hops?
    Express your answer as a common fraction.
Solution:
```

```
Let's think step by step
Kevin hops $1/3$ of the remaining distance with every hop.
His first hop takes $1/3$ closer.
For his second hop, he has $2/3$ left to travel, so he hops forward
    $(2/3)(1/3)$.
For his third hop, he has $(2/3)^2$ left to travel, so he hops
    forward $(2/3)^2(1/3)$.
In general, Kevin hops forward $(2/3)^{k-1}(1/3)$ on his $k$th hop.
We want to find how far he has hopped after five hops.
This is a finite geometric series with first term $1/3$, common ratio
    $2/3$, and five terms.
Thus, Kevin has hopped
    $\frac{\frac{1}{3}\left(1-\left(\frac{2}{3}\right)^5\right)}{1-\frac{2}{3}}
    = \frac{211}{243}$.
So the final answer is $\boxed{\frac{211}{243}}$.


Problem:
What is the area of the region defined by the equation $x^2+y^2 - 7 =
    4y-14x+3$?
Solution:
Let's think step by step
We rewrite the equation as $x^2 + 14x + y^2 - 4y = 10$ and then
    complete the square,
resulting in  $(x+7)^2-49 + (y-2)^2-4=10$,
or $(x+7)^2+(y-2)^2=63$.
This is the equation of a circle with center $(-7, 2)$ and radius
    $\sqrt{63},$
so the area of this region is $\pi r^2 = 63\pi$.
So the final answer is $\boxed{63\pi}$.


Problem:
If $x^2+y^2=1$, what is the largest possible value of $|x|+|y|$?
Solution:
Let's think step by step
If $(x,y)$ lies on the circle,
so does $(x,-y),$ $(-x,-y),$ and $(-x,-y),$ (which all give the same
    value of $|x| + |y|$),
so we can assume that $x \ge 0$ and $y \ge 0.$
Then $|x| + |y| = x + y.$  Squaring, we get
\[(x + y)^2 = x^2 + 2xy + y^2 = 1 + 2xy.\]
Note that $(x - y)^2 \ge 0.$
Expanding, we get $x^2 - 2xy + y^2 \ge 0,$ so $2xy \le x^2 + y^2 = 1.$
Hence,\[1 + 2xy \le 2,\]which means $x + y \le \sqrt{2}.$
Equality occurs when $x = y = \frac{1}{\sqrt{2}},$
so the maximum value of $|x| + |y|$ is $\sqrt{2}.$
So the final answer is $\boxed{\sqrt{2}}$.


Problem:
If $f(x)=\frac{ax+b}{cx+d}, abcd\not=0$ and $f(f(x))=x$ for all $x$
    in the domain of $f$, what is the value of $a+d$?
Solution:
Let's think step by step
The condition $f(f(x))$ means that $f$ is the inverse of itself,
so its graph is symmetrical about the line $y = x$.
With a rational function of this form, we will have two asymptotes:
a vertical one at $x=-d/c$ if $cx+d$ does not divide $ax+b$,
and a horizontal one at $y=a/c$,
if we take the limit of $f(x)$ as $x$ goes to $\pm\infty$.
In order for $f$ to be its own inverse, the intersection of the
    asymptotes must lie on the line $y=x$
```

```
so that it and its asymptotes reflect onto themselves.
This means that $-d/c=a/c$,
and therefore $-d=a$ and $a+d=0$.
So the final answer is $\boxed{0}$.


Problem:
{problem}
Solution:
```

Mathematics: memory-augmented instruction

```
Solve the following math problem:

{problem}

Reason step by step, then put your final answer in \boxed{}.
```

Code generation. The unmodified model uses LCB's `GenericBase` one-shot template. We prepend the first official function-completion example (counting unequal triplets) when starter code is supplied, or the first standard-input/output example (equalizing gifts) otherwise. The demonstration contains both the question and its reference program; the test problem then appears under the same `### Question`, optional `### Starter Code`, and `### Answer` headings. The model continues directly with code.

The memory-augmented model uses the instruction template below. For function completion, the format instruction is "You will use the following starter code to write the solution to the problem and enclose your code within delimiters." For standard-input/output tasks, it is "Read the inputs from stdin solve the problem and write the answer to stdout (do not directly test on the sample inputs). Enclose your code within delimiters as follows. Ensure that when the python program runs, it reads the inputs, runs the algorithm and writes output to STDOUT." Starter code is inserted verbatim; when none is provided, the placeholder is `# YOUR CODE HERE`. EpiCoder supervised initialization uses the same structure with the format instruction "Write a complete Python solution and enclose your code within delimiters as follows."

Code generation: memory-augmented instruction

````
You are an expert Python programmer. You will be given a question
    (problem specification) and will generate a correct Python
    program that matches the specification and passes all tests.

### Question:
{problem}

### Format: {format_instruction}
```python
{starter_code_or_placeholder}
```

### Answer: (use the provided format with backticks)
````

Knights-and-Knaves. Both conditions receive the same instruction and formatting example. The optional rationale precedes the answer block and contains no markup. Each identity occupies one line; after the closing answer tag, the model must stop. The weaver receives the corresponding plain-text task content.

Knights-and-Knaves: shared task instruction

```
Solve this logical reasoning problem carefully. You may give a
    concise plain-English rationale before the final answer, but use
    no markup in the rationale. End the response with exactly one
    <answer> </answer> block containing every person's identity, and
    write nothing after </answer>.
List each person exactly once, one per line, for example:
<answer>
(1) Zoey is a knight
(2) Oliver is a knight
</answer>

{quiz}
```

When2Call tool-use decision. We evaluate whether a model decides to call an available tool, ask the user for missing information, or decline a request that cannot be completed with the provided capabilities. Each test instance contains a user request and a list of available tools. We use a fixed three-shot prompt: one demonstration each for `tool_call`, `request_for_info`, and `cannot_answer`. Each demonstration contains its tool list, user request, and a representative assistant response. For the test instance, the model receives the supplied tools and user request and may either emit a native function call or respond in text. No answer options or gold labels are shown to the model.

We score generated responses using When2Call's LLM-as-a-Judge protocol. The judge receives the available tools, user request, and model response, and assigns one of four labels: `tool_call`, `request_for_info`, `cannot_answer`, or `direct_answer`. We report macro-F1 and accuracy over the first three gold classes. The judge measures the decision category only; it does not verify tool arguments or factual correctness of a direct textual answer.

When2Call: fixed three-shot tool-use prompt

```
You are an assistant with access only to the tools supplied for
  each request.
Decide whether to call a supplied tool, ask the user for missing
  information,
or state that the request cannot be completed with the available
  capabilities.

### Example 1
Available tools:
{tool_call_demo_tools}

User request:
{tool_call_demo_question}

Assistant:
<tool_call>
{"name": "{tool_call_demo_name}",
 "arguments": {tool_call_demo_arguments}}
</tool_call>


### Example 2
Available tools:
{request_info_demo_tools}

User request:
{request_info_demo_question}

Assistant:
```

```
{request_info_demo_response}


### Example 3
Available tools:
{cannot_answer_demo_tools}

User request:
{cannot_answer_demo_question}

Assistant:
{cannot_answer_demo_response}


### Task
Available tools:
{tools}

User request:
{question}

Assistant:
```

C Efficiency and Computational Cost

Training and inference configurations follow the preceding appendices: memory construction and recipient calibration are specified in Appendix A, training data and optimization settings in Appendices B.2 and B.3, and evaluation protocols, decoding settings, and prompt templates in Appendices B.1, B.4 and B.6. We refer to these shared configurations throughout and report only resource usage, timing boundaries, and output-length statistics here.

Training costs are measured on one NVIDIA L40S server (8×48 GB), and inference costs on one NVIDIA A100 server (8×80 GB). Timings are from single runs. For an elapsed time of t minutes on G allocated GPUs, aggregate GPU-hours are computed as

$$\text{GPU-hours} = G \, \frac{t}{60}. \tag{14}$$

Here, $G = 8$ for both training and inference. GPU-hours count all allocated GPUs over the reported timing window, irrespective of instantaneous utilization; they therefore measure allocated accelerator time.

Training cost. Table 9 summarizes the source training costs of the SFT initialization and subsequent OPD stages under the configurations in Appendices B.2 and B.3. Training time covers the training loop and checkpointing, excluding data preparation, rollout generation, teacher scoring, and evaluation. Each exported adapter-and-memory checkpoint occupies 1.007 GiB, excluding the frozen backbone. The stage-specific GPU-hours use Equation (14); the total training-loop cost of an OPD-trained memory is the sum of its SFT and OPD stages.

Table 9: Source training cost on one NVIDIA L40S server (eight GPUs).

	Math		Code	
Metric	**SFT**	**OPD**	**SFT**	**OPD**
Training time (min)	23.29	66.59	107.57	31.14
L40S GPU-h	3.105	8.879	14.343	4.152

Paired inference time. Given the detailed discussion of code transfer to larger recipients in Appendix E, we omit the LiveCodeBench inference timing and output-length results for MiMo-V2-Flash and HY3-Preview here, while retaining those for smaller recipients.

Table 10 compares elapsed evaluation time and aggregate GPU-hours for Base and `+MemAny` on AIME24/25 and LCB-V6. Model and memory configurations, prompting, decoding, and stopping settings follow Appendices B.1, B.3, B.4 and B.6.

For math, the timing window covers generation, grading, and result writing, excluding engine initialization and memory preparation. For code, it includes recipient initialization, memory preparation, generation, and result writing, while excluding prior weaver loading and calibration, shutdown, and CPU scoring. GPU-hours are computed from the corresponding elapsed evaluation window using Equation (14). These measurements compare the complete evaluation configurations; they do not isolate the cost of latent-memory insertion from differences in prompting, stopping, or generated output length.

Table 10: Paired inference time on one NVIDIA A100 server (eight GPUs). Math: AIME24/25; code: LCB-V6.

	Eval time (min)		A100 GPU-h	
Recipient	**Base**	**+MemAny**	**Base**	**+MemAny**
(a) Math: AIME24/25				
QWEN3-4B	3.82	5.46	0.509	0.728
MIMO-7B	14.67	21.28	1.956	2.837
HY3-PREVIEW	50.08	68.04	6.677	9.072
MIMO-V2-FLASH	7.54	10.25	1.006	1.367
(b) Code: LCB-V6				
QWEN3-4B	9.73	15.86	1.297	2.115
MIMO-7B	17.42	26.35	2.323	3.513

Mathematical output length. Table 11 reports mean mathematical output lengths under the evaluation and prompting configurations in Appendices B.4 and B.6. Counts exclude inputs and latent memory tokens; N denotes the number of generated responses per condition. Lengths use native generated-token counts for Qwen3-4B, Qwen3-8B, and MiMo-7B, and retokenized returned text for Hy3-Preview. Qwen3-4B's pooled length-limit rate increases from 3.85% to 7.88% with memory, so its output lengths also reflect unfinished generation. Relative length changes are computed as $100 \times (L_{\text{MemAny}}/L_{\text{Base}} - 1)$, where L is the mean output length.

Table 11: Mean math output length (tokens) under the shared evaluation protocol. N is the number of responses per condition. Changes are relative to Base.

Recipient	**Task**	N	**Base**	**+MemAny**	**Change**
QWEN3-4B	AIME24	120	2,351.8	3,387.3	+44.0%
	AIME25	120	2,027.0	3,156.5	+55.7%
	Olympiad	800	1,254.4	1,688.6	+34.6%
QWEN3-8B	AIME24	120	2,450.9	2,604.1	+6.3%
	AIME25	120	2,514.6	1,522.8	−39.4%
	Olympiad	800	1,179.3	1,218.8	+3.4%
MIMO-7B	AIME24	120	3,027.8	3,254.8	+7.5%
	AIME25	120	2,305.4	4,561.9	+97.9%
	Olympiad	800	2,818.0	4,438.6	+57.5%
HY3-PREVIEW	AIME24	120	2,042.5	2,303.5	+12.8%
	AIME25	120	1,463.9	3,723.4	+154.4%
	Olympiad	800	1,004.5	1,779.9	+77.2%

The output-length statistics provide additional context for the accuracy gains in Table 1. On AIME24, Qwen3-4B's mean response length increases from 2,351.8 to 3,387.3 tokens (+44.0%), alongside an accuracy improvement from 5.8% to 18.3%. Similarly, HY3-Preview's mean response length on AIME25 grows from 1,463.9 to 3,723.4 tokens (+154.4%), while accuracy rises from 13.3% to 43.3%. These observations are consistent with more extended solution development contributing to the gains in these settings.

Code output length. Table 12 reports mean code output lengths on LCB-V5/V6 under the evaluation and prompting configurations in Appendices B.4 and B.6. Lengths exclude inputs and latent memory tokens and are computed by retokenizing all returned text, including text outside executable code. The response counts and relative changes follow the same conventions as the mathematical output-length analysis. These statistics reflect the combined effects of prompting, stopping, and memory on generated outputs.

Table 12: Mean code output length (tokens) on LCB-V5/V6. N is the number of responses per condition under the shared evaluation protocol. Changes are relative to Base.

Recipient	Task	N	Base	+MemAny	Change
QWEN3-4B	LCB-V5	1,116	532.6	641.8	+20.5%
	LCB-V6	1,816	578.3	704.6	+21.8%
QWEN3-8B	LCB-V5	1,116	486.2	563.9	+16.0%
	LCB-V6	1,816	524.7	616.8	+17.6%
MIMO-7B	LCB-V5	1,116	1,864.7	2,187.6	+17.3%
	LCB-V6	1,816	2,038.5	2,436.8	+19.5%

D ADDITIONAL LATENT VOCABULARY VISUALIZATIONS

The visualizations span AIME24, AIME25, Olympiad, LiveCodeBench, and K&K, covering 672 displayed clouds from 29 questions. Figure 4(a) shows 16 clouds at latent positions 1, 3, 5, and 7; the ten figures below provide the remaining 656, with eight latent positions per row.



Figure 5: Additional latent-conditioned vocabulary (1/10). Each row identifies a question and recipient model; columns align latent positions 1–8. Font size reflects relative probability within each cloud.

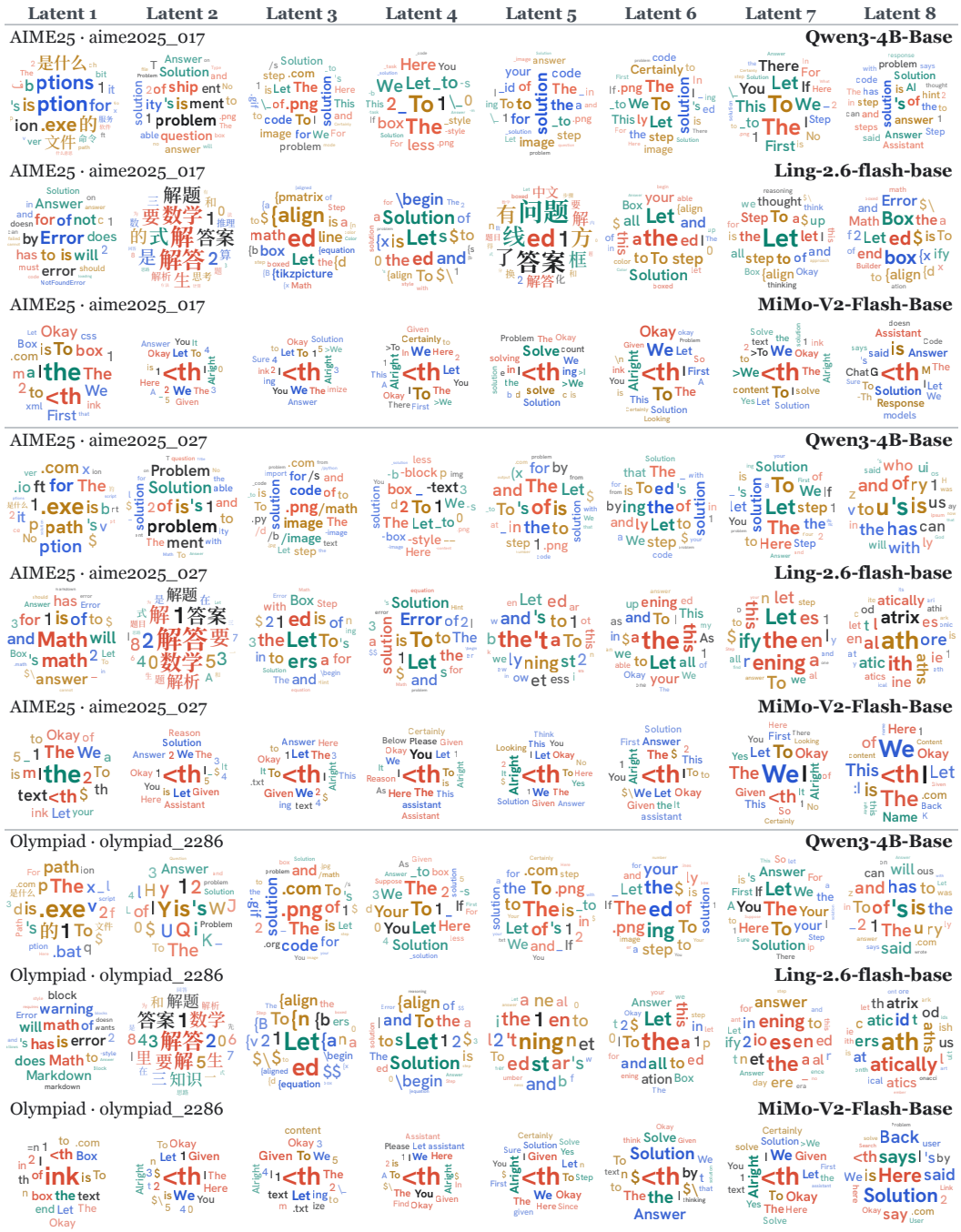


Figure 6: Additional latent-conditioned vocabulary (2/10). Each row identifies a question and recipient model; columns align latent positions 1–8. Font size reflects relative probability within each cloud.



Figure 7: Additional latent-conditioned vocabulary (3/10). Each row identifies a question and recipient model; columns align latent positions 1–8. Font size reflects relative probability within each cloud.



Figure 8: Additional latent-conditioned vocabulary (4/10). Each row identifies a question and recipient model; columns align latent positions 1–8. Font size reflects relative probability within each cloud.



Figure 9: Additional latent-conditioned vocabulary (5/10). Each row identifies a question and recipient model; columns align latent positions 1–8. Font size reflects relative probability within each cloud.

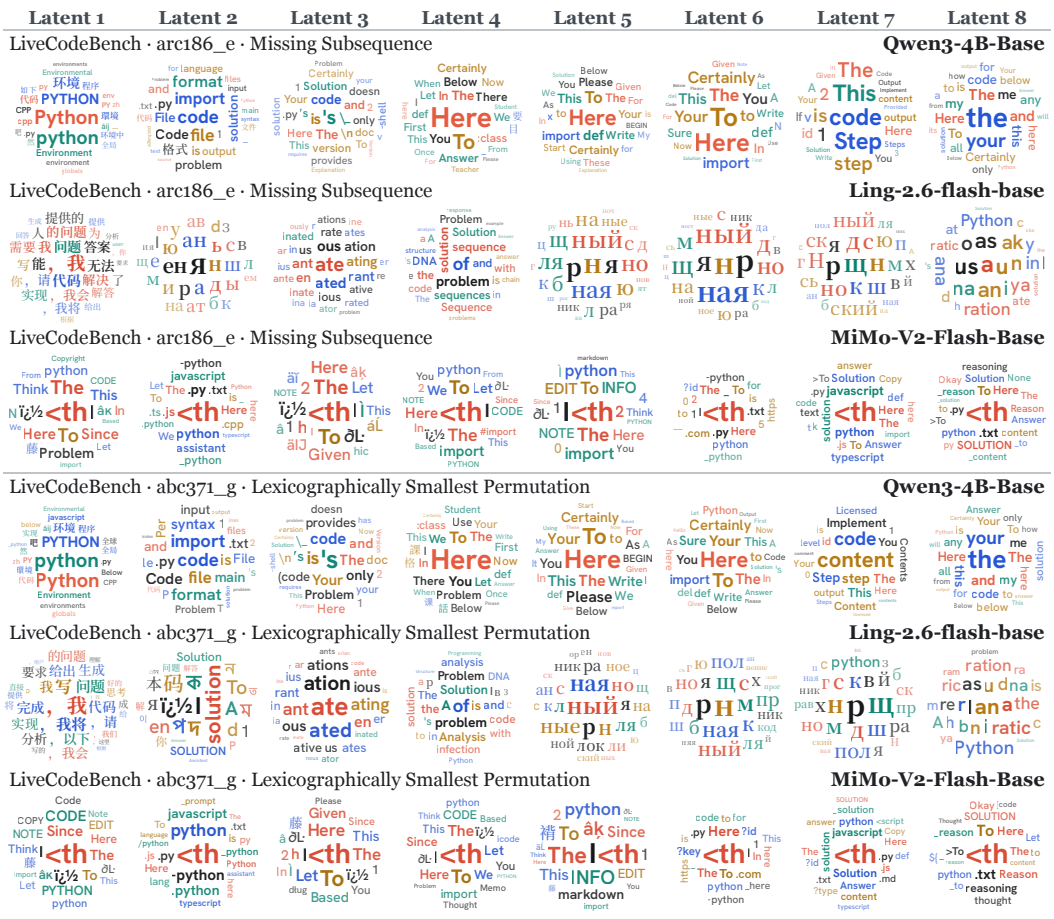


Figure 10: Additional latent-conditioned vocabulary (6/10). Each row identifies a question and recipient model; columns align latent positions 1–8. Font size reflects relative probability within each cloud.

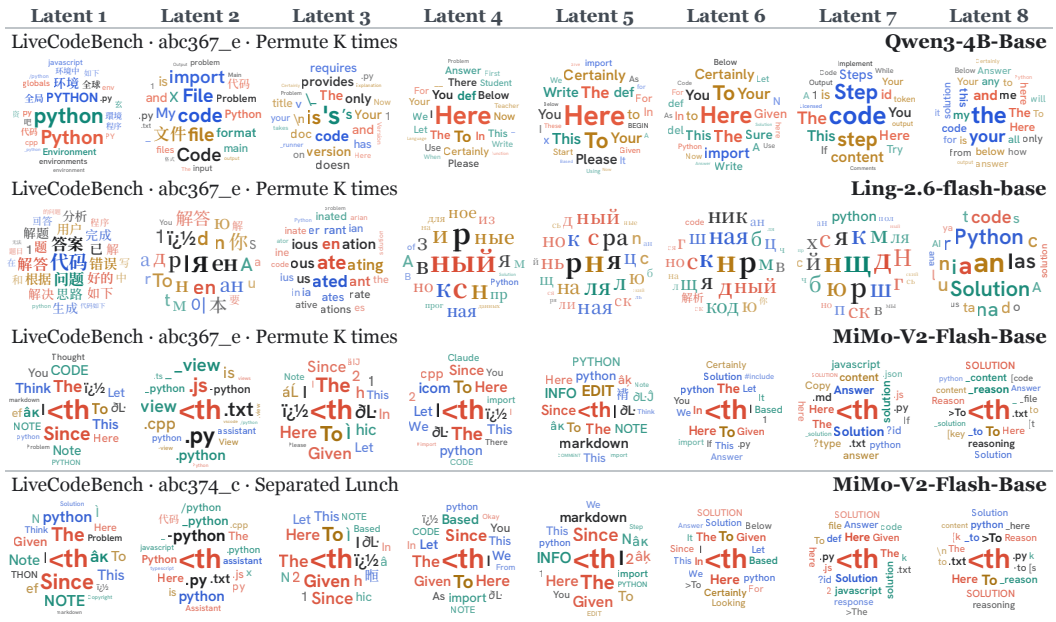


Figure 11: Additional latent-conditioned vocabulary (7/10). Each row identifies a question and recipient model; columns align latent positions 1–8. Font size reflects relative probability within each cloud.

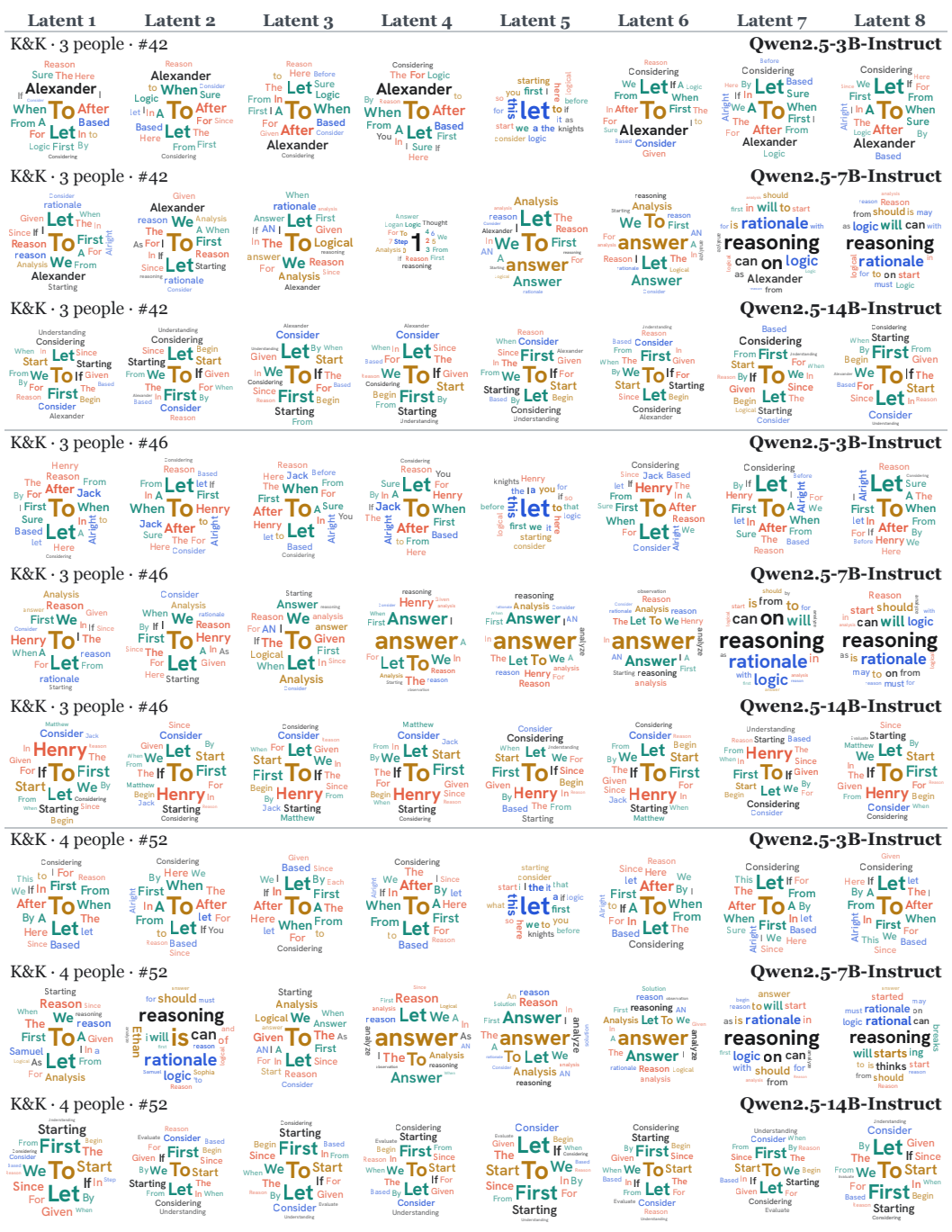


Figure 12: Additional latent-conditioned vocabulary (8/10). Each row identifies a question and recipient model; columns align latent positions 1–8. Font size reflects relative probability within each cloud.

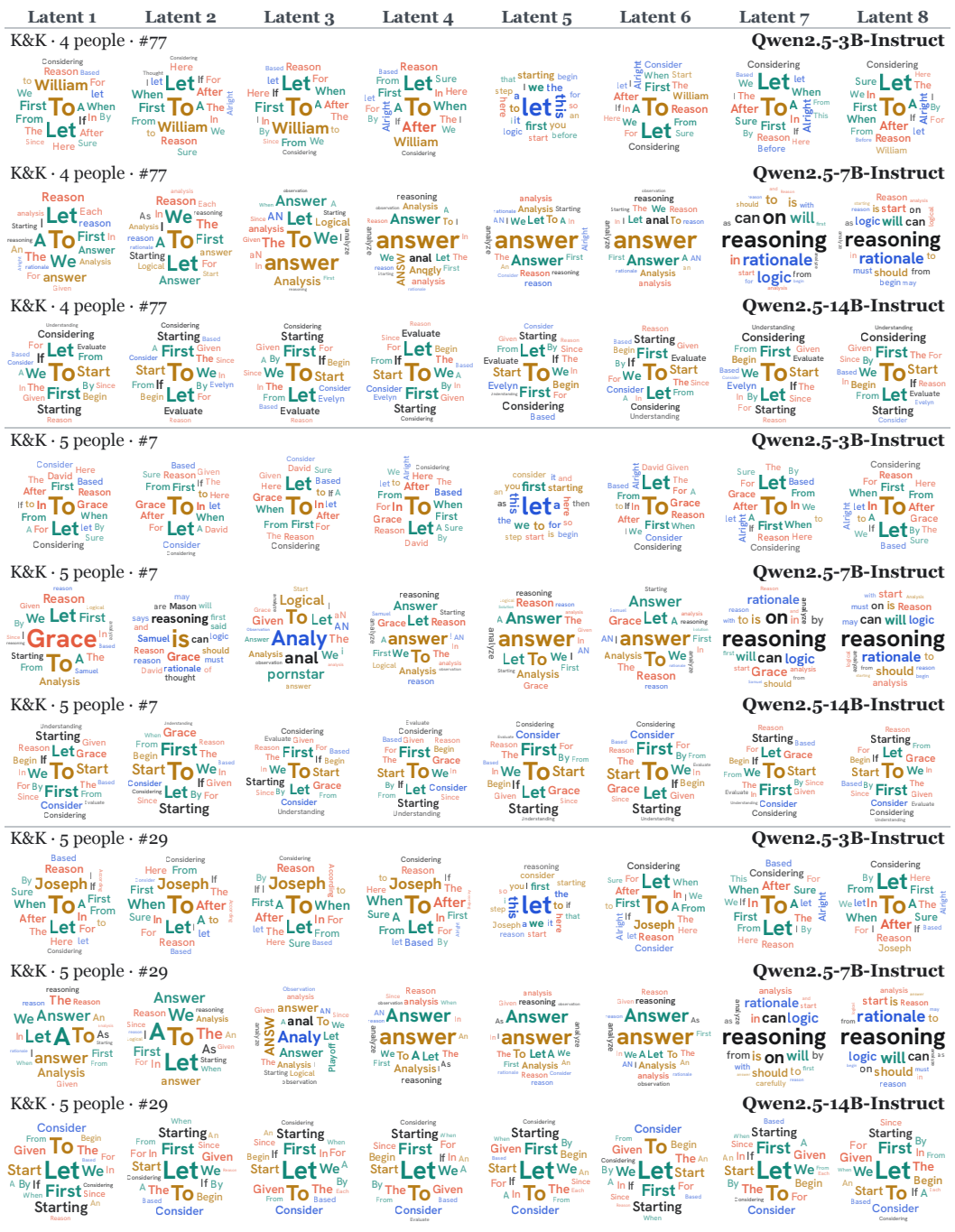


Figure 13: Additional latent-conditioned vocabulary (9/10). Each row identifies a question and recipient model; columns align latent positions 1–8. Font size reflects relative probability within each cloud.



Figure 14: Additional latent-conditioned vocabulary (10/10). Each row identifies a question and recipient model; columns align latent positions 1–8. Font size reflects relative probability within each cloud.

E Additional Failure-Mode Analysis

Annotation protocol. The audit covers Qwen3-4B-Base and Ling-2.6-108B-Base, each evaluated with and without the weaver. Official benchmark scorers determine correctness, and each failed response is submitted with its problem to GPT-5.5 at temperature zero, with model, condition, and checkpoint identities hidden. The judge assigns one failure category in a single pass, identifying the earliest root cause along task representation, strategy, execution, finalization, and degeneration. Paired analyses match question and sample indices across conditions; the corrected-response counts in Section 5.4 use observed failure labels.

Code transfer to larger recipients. The code regressions contrast with substantial mathematical gains on the same recipients: MiMo-V2-Flash's LCB-V5/V6 pass@4 changes from $59.1\%/53.1\%$ to $58.1\%/52.0\%$, and HY3-Preview's from $53.4\%/48.5\%$ to $51.3\%/45.4\%$ (Table 1). For MiMo, the reasoning/strategy plus computation/implementation share falls from $35.8\%/41.1\%$ to $15.3\%/18.0\%$ on LCB-V5/V6, while answer/format rises from $1.1\%/1.2\%$ to $16.4\%/18.4\%$ and truncation/degeneration from $1.4\%/1.5\%$ to $7.9\%/9.0\%$. HY3 follows the same pattern: strategy and implementation errors decrease from $41.0\%/45.4\%$ to $18.3\%/20.9\%$, while format failures increase from $1.0\%/1.0\%$ to $18.8\%/20.7\%$ and truncation/degeneration from $1.5\%/1.6\%$ to $9.1\%/10.1\%$. These distributions suggest that answer delivery merits particular attention when transferring coding memory to larger recipients.

Closer inspection suggests a teacher–recipient capability mismatch: Qwen3-8B reports $76.0\%/67.3\%$ on AIME24/25 in (Yang et al., 2025), versus $16.7\%/16.7\%$ for MiMo-V2-Flash-Base and $6.7\%/13.3\%$ for HY3-Preview-Base, while their LCB-V5 scores are numerically close (57.5% versus 59.1% and 53.4%). When the teacher offers little additional coding capability, OPD has limited scope to improve an already strong recipient. **We therefore interpret these regressions as a limitation of OPD with an insufficiently strong teacher, rather than of `MemAny`'s memory-transfer mechanism.** We nevertheless report these results in full for transparency, despite attributing them to the OPD teacher rather than `MemAny` itself.

Failure categories. Problem/premise denotes misreading the task or its assumptions; reasoning/strategy denotes an invalid plan or inference; computation/implementation covers arithmetic and coding errors; answer/format covers incorrect extraction or presentation of the final answer; and truncation/degeneration denotes unfinished or repetitive generation. Each failed response receives the category corresponding to its earliest root cause.

F Discussion of Related Work

Continuous reasoning and vocabulary superposition. Coconut (Hao et al., 2024) feeds the previous hidden state back as the next input embedding, establishing a direct route to reasoning through continuous representations. CODI (Shen et al., 2025) learns compact latent reasoning through self-distillation from explicit chains of thought. SoftCoT (Xu et al., 2025b) uses an assistant to generate input-dependent soft thoughts and a learned projection to condition a frozen language model. More directly related to our representation, Soft Thinking (Zhang et al., 2025d) constructs continuous concept tokens as probability-weighted mixtures of vocabulary embeddings. Latent-SFT (Deng et al., 2025) formalizes latent tokens as vocabulary-space superpositions and trains models to generate compact latent reasoning chains, highlighting the value of grounding continuous representations in pretrained token embeddings.

Token aggregation and linear weighting. In vision, TokenLearner (Ryoo et al., 2021) uses input-dependent spatial weights to pool features into a small set of informative tokens. EViT (Liang et al., 2022) retains attentive tokens and fuses the remaining tokens through attention-weighted aggregation. Token Merging (ToMe) (Bolya et al., 2022) matches similar tokens and combines them using averages weighted by token size, reducing computation while retaining information from merged tokens. These works provide complementary examples of how weighted combinations can preserve useful information in compact representations, with the aggregation coefficients determining which content is carried forward.

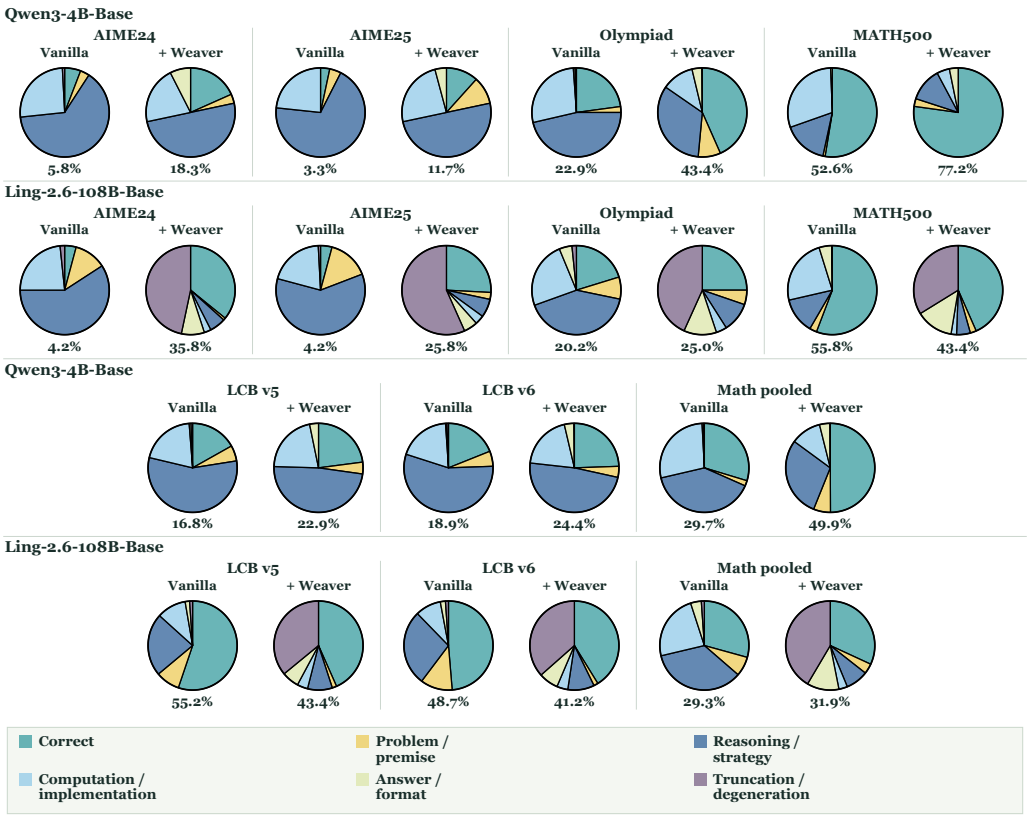


Figure 15: Full failure-mode atlas. Each pair compares Vanilla and +MemAny OPD (+ Weaver); labels report correctness. Correct sectors use mathematical accuracy and question-level LCB pass@4 (at least one success in four generations). Math pooled aggregates 1,540 responses across AIME24, AIME25, Olympiad, and MATH500 using unrounded correct counts.

From latent superposition to transferable memory. We gratefully acknowledge these works for inspiring our view of latent linear superposition as a carrier of transferable knowledge. Closely related, Ju et al. (2023) interpret continuous prompts as weighted combinations of discrete token embeddings and explore their cross-model transfer without additional training signals on the target model. `MemAny` develops this perspective into a coefficient-based memory interface: the weaver learns what to combine, while each recipient supplies the native vocabulary embeddings used to express that combination (Equations (7) and (8)). Shared vocabulary anchors align the coefficient bases, allowing the same trained weaver to construct memory in different recipients' embedding spaces without further training. This connects continuous representation learning with knowledge transfer: capabilities acquired through SFT, RL, or OPD can be stored in a reusable memory module and communicated across model scales and families.