SKILLMEM: COMPACT EXTERNAL SKILL MEMORY FOR LATENT MULTI-AGENT REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Latent multi-agent systems reduce inference cost by keeping intermediate reasoning and inter-agent communication in continuous representations. Reusable skills, however, are commonly supplied as text or stored as layer-wise Key-Value states or skill-specific model parameters, introducing repeated processing or substantial per-skill storage. We introduce **SkillMem**, which represents each reusable skill as a small external latent memory and makes it directly available during latent multi-agent reasoning. A role-conditioned cross-attention reader allows the same memory to guide different agents according to their roles and current hidden states. We train the skill memories and shared reader through behavioral distillation, matching the change in token predictions induced by a textual skill while keeping the backbone frozen. For our experiments, we obtain textual skills from paired successful and failed multi-agent trajectories, although existing skills can also be compiled directly. Across six benchmarks and Qwen3 models ranging from 4B to 14B parameters, SkillMem provides strong gains for smaller backbones while preserving efficient latent multi-agent inference across model scales. We use the same 8×256 memory configuration across all evaluated model scales, requiring 9.6 KB of skill-specific storage per skill, including its retrieval embedding. For a library of 100 skills, SkillMem reduces total storage by more than $38\times$ compared with alternative skill representations and achieves up to a $4.23\times$ end-to-end inference speedup. These results show that compact procedural memories can be reused without sacrificing the efficiency of latent multi-agent reasoning.

1 INTRODUCTION

Large language model (LLM) based multi-agent systems (MAS) assign different roles to agents, allowing them to plan, critique, and refine a solution across multiple reasoning steps. This division of work can improve performance over a single model response, yet role specialization alone does not prevent recurring errors. Some of these errors can be addressed by procedures such as checking assumptions or verifying intermediate results. Skills (Zhang et al., 2025) capture such reusable procedures from previous experience and make them available for similar problems. In text-based MAS, agents express their reasoning and exchange intermediate results in natural language. Latent reasoning and latent MAS replace this text-based reasoning and communication with continuous representations, reducing the need to generate and process intermediate text (Hao et al., 2024; Shen et al., 2025; Zou et al., 2026). A textual skill is typically inserted into an agent's prompt, so the model must prefill the skill tokens each time the skill is used. Across repeated uses, this extra prefilling can erode the inference speed gained from latent reasoning and communication. This raises a question: *how should agents reuse skills efficiently when their reasoning and communication remain in latent space?*

To avoid repeatedly processing skill text, we can store skills in other forms. For example, a cached key–value (KV) prefix avoids recomputing a skill prompt's activations, but its storage grows with the prefix length and the number of layers (Jin et al., 2026). A skill-specific LoRA adapter stores parameter updates for the layers it modifies (Zhang & Qi, 2026). Although each adapter is smaller than the full model, storing one for every skill can still require substantial storage as the library grows. These trade-offs motivate a compact but efficient, retrievable skill representation that agents can use directly during latent reasoning.

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

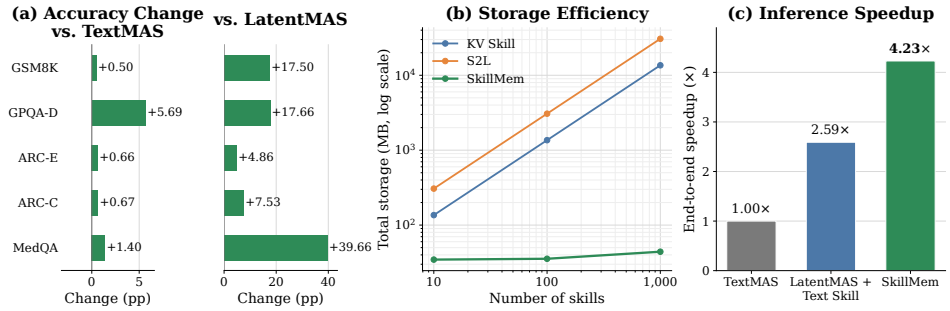

Figure 1: **Accuracy and efficiency comparison.** (a) Accuracy changes of SkillMem relative to TextMAS and LatentMAS using Qwen3-4B. (b) Total storage required by different skill representations as the skill library grows. (c) End-to-end inference speedup relative to TextMAS.

We introduce **SkillMem**, a framework that stores each reusable skill as a small set of learned latent memory vectors. The memory is external to the frozen backbone and can be retrieved independently of other skills. Given a task query, SkillMem selects a relevant skill and makes its memory available throughout latent multi-agent reasoning. Agents continue to exchange latent states, while the retrieved skill provides procedural guidance without adding its textual description to each agent's prompt. This separates the choice of a skill from the way an agent uses it.

Sharing a skill across agents requires more than giving every role the same memory vectors. A planner and a critic may use the same procedure at different stages and for different purposes. SkillMem therefore uses a role-conditioned **reader** to access the retrieved **memory embedding** at selected layers of the frozen backbone. The memory is stored once, while the reader uses the current agent's role to produce a role-dependent update to its hidden states. This allows agents to share procedural knowledge without requiring a separate skill representation for each role.

The remaining challenge is how to learn a useful skill memory. Rather than training the memory to reconstruct the skill description, we train it to preserve the skill's effect on model behavior. We first generate a response using the textual skill. We then measure how much the skill changes the probability of each token in that same response. Behavioral distillation trains the memory and shared reader to reproduce these changes while keeping the backbone frozen. In our experiments, the textual skills come from paired successful and failed multi-agent answer trajectories for the same task. Comparing these trajectories helps identify reusable procedures. The compilation step takes a textual skill as input, so it can use either a skill discovered from trajectories or an existing skill that has already proved useful.

Across six reasoning, question-answering, and code-generation benchmarks, SkillMem improves on the latent MAS baseline. On Qwen3-4B, accuracy rises from 17.97% to 35.63% on GPQA-Diamond and from 72.40% to 89.90% on GSM8K. Each skill requires only 9.6 KB of skill-specific storage. SkillMem achieves a 4.23× end-to-end speedup over TextMAS and a 1.63× speedup over LatentMAS with text skills. With a library of 100 skills, its total storage, including the shared reader, is 38.3× smaller than KV Skill and 86.6× smaller than LoRA Skill. These results show that SkillMem can provide reusable skills for latent multi-agent reasoning, as summarized in Figure 1.

2 Related Work

Latent multi-agent reasoning and memory. LatentMAS (Zou et al., 2026) enables agents to reason and communicate through continuous states rather than exchanging natural-language messages. KVComm (Ye et al., 2025) and RecursiveMAS (Yang et al., 2026) further study efficient communication and computation in multi-agent systems. Building on latent collaboration, LatentMem (Fu et al., 2026) constructs agent-specific memories from retrieved interaction experience, while Agent Primitives (Jin et al., 2026) represents predefined multi-agent operations with KV-based latent states. SkillMem addresses a different unit of reuse: a task-level procedural skill stored as a persistent memory that can be retrieved and shared across agent roles.

Skill representations and behavioral distillation. Textual skills can be applied through prompting, but their instructions must be processed whenever they are inserted into an agent's context.

S2L (Zhang & Qi, 2026) and LatentSkill (Yu et al., 2026) compile textual skills into LoRA-based model adaptations. SkillSmith (Dery et al., 2026) studies the composition of learned KV-prefix skills and textual knowledge. SEED (Wu et al., 2026) distills skill-induced behavioral changes into the agent policy. SkillMem likewise learns from the behavioral effect of a textual skill, but stores that effect in an independently retrievable external memory. A reader shared across skills and roles accesses the memory while the backbone remains frozen. We compare against S2L as a parameter-based skill representation in our experiments.

Continuous representations and context compression. Prompt and prefix tuning (Lester et al., 2021; Li & Liang, 2021) learn continuous representations that condition a frozen model. Gisting and AutoCompressor (Mu et al., 2023; Chevalier et al., 2023) compress information from a textual context into continuous representations. These methods motivate compact continuous conditioning, while SkillMem learns a separate memory for each reusable procedure and accesses it through a shared, role-conditioned reader during latent multi-agent reasoning.

Experience-based skill learning and routing. Voyager (Wang et al., 2024) and ExpeL (Zhao et al., 2024) reuse knowledge from previous agent trajectories. SkillRL (Xia et al., 2026) uses successful and failed single-agent trajectories to build a hierarchical skill bank and jointly updates the skills and agent policy through reinforcement learning. We derive textual skills by comparing successful and failed multi-agent trajectories, retaining the roles involved in each reasoning stage. SkillRouter (Zheng et al., 2026) studies skill selection at scale. Our retriever selects a skill for the task, while SkillMem focuses on how the selected skill is stored and used across agents.

3 Preliminaries: Latent Multi-Agent Reasoning

Let f_θ denote a decoder-only Transformer with L layers. During autoregressive generation, the model retains previously computed KV states to avoid recomputing earlier positions. Latent reasoning instead uses continuous vectors in place of some intermediate natural-language tokens. Let t denote the current sequence position, and let $\boldsymbol{h}_t^{(L)}$ be its hidden state after the final Transformer layer. A mapping ϕ produces $\boldsymbol{z}_{t+1} = \phi(\boldsymbol{h}_t^{(L)})$, where $\boldsymbol{z}_{t+1}$ is used as the input embedding at position $t+1$. The model can therefore continue reasoning without decoding an intermediate natural language token and embedding it again.

LatentMAS extends this process to collaboration among agents (Zou et al., 2026). Consider a sequence of agents $\mathcal{A} = \{A_1, \ldots, A_R\}$ with corresponding roles $\mathcal{R} = \{\rho_1, \ldots, \rho_R\}$. Each agent A_r follows the instructions for its role ρ_r; for example, agents may serve as a planner, critic, refiner, or final-answer agent. Given a query q, each agent performs multiple latent reasoning steps. Let $\mathcal{C}_r = \left\{\boldsymbol{K}_r^{(\ell)}, \boldsymbol{V}_r^{(\ell)}\right\}_{\ell=1}^{L}$ denote the layer-wise KV cache available after agent A_r finishes. This cache contains the latent context passed from earlier agents together with the states added by A_r. Passing $\mathcal{C}_r$ to the next agent lets it continue from that context without decoding and re-encoding an intermediate message. Only the final agent generates a natural language answer. We use this latent communication pipeline as our base multi-agent system.

4 Method

SkillMem stores each textual skill s as a compact memory $\boldsymbol{Z}_s$ that can be retrieved independently (Section 4.1). Given a task query q, the system retrieves a skill s and makes its memory $\boldsymbol{Z}_s$ available to every agent in the latent MAS pipeline. Agents continue to communicate through KV states, while a role-conditioned reader uses $\boldsymbol{Z}_s$ to update their hidden states according to their roles (Section 4.2). We train the memories and reader to reproduce how textual skills change the frozen backbone's predictions (Section 4.3). Skills can be provided directly or discovered from past MAS trajectories; we use discovered skills in our experiments (Section 4.4). Figure 2 shows the full pipeline.

4.1 Skill Memory and Skill Retrieval

We define a *skill* as a reusable reasoning procedure that can guide agents on multiple problem instances. Let $\mathcal{S} = \{s_1, \ldots, s_N\}$ be a skill library, where $s_i = (n_i, d_i, p_i)$ contains a name, a description of when the skill is useful, and procedural instructions. Let $E(\cdot)$ denote a text encoder based

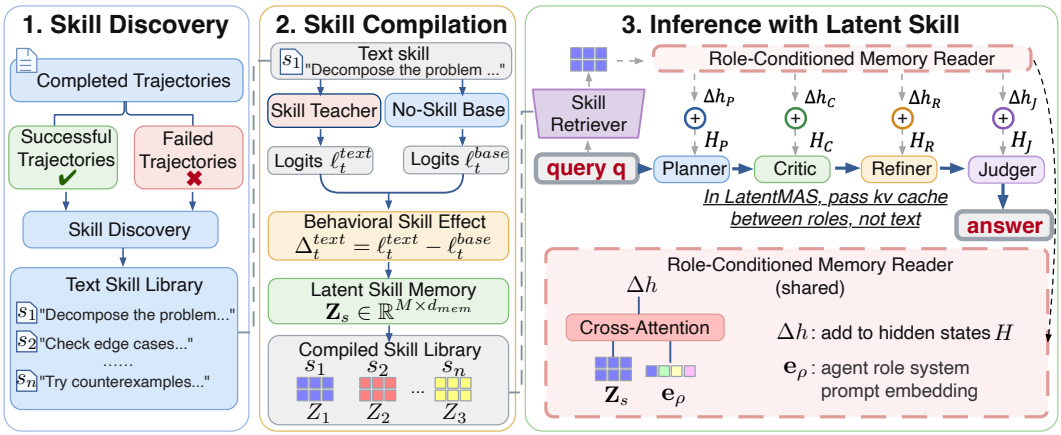

Figure 2: Overview of **SkillMem**. Reusable textual skills are discovered from paired multi-agent trajectories. Each skill is compiled into a compact latent memory by matching the behavioral effect of skill injection. At inference time, the retrieved memory is shared across agents and consumed through a role-conditioned reader while inter-agent communication remains entirely latent.

on the frozen backbone. We index each skill using $\boldsymbol{e}_i = E([n_i; d_i; p_i])$. Given a task query q, we compute $\boldsymbol{e}_q = E(q)$ and retrieve $s_q = \arg\max_{s_i \in \mathcal{S}} \mathrm{sim}(\boldsymbol{e}_q, \boldsymbol{e}_i)$, where sim is cosine similarity. The selected skill is used by every agent in the multi-agent run.

We store each skill s as a persistent external memory $\boldsymbol{Z}_s \in \mathbb{R}^{M \times d_{\mathrm{mem}}}$. This memory contains M learned vectors for that skill, each of dimension d_{mem}. After selecting s, the system loads $\boldsymbol{Z}_s$ for use across agent roles. Each skill has its own memory, requiring $\mathcal{O}(M d_{\mathrm{mem}})$ skill-specific storage.

Inserting a textual skill into an agent's prompt requires processing its tokens when the skill is used. Caching the skill's layer-wise KV states avoids repeated processing, but stores states for every skill token at every Transformer layer. For a skill of T_s tokens, KV storage scales as $\mathcal{O}(L T_s d_{\mathrm{kv}})$, where L is the number of layers and d_{kv} is the KV dimension. The storage of $\boldsymbol{Z}_s$ does not depend on T_s.

4.2 Role-Conditioned Memory Reader

The retrieved memory $\boldsymbol{Z}_s$ is shared across agents, but its effect should depend on the agent's role ρ and current reasoning state. We therefore insert a role-conditioned *cross-attention reader* at selected Transformer layers ℓ. For **each token**, the reader forms **queries from its hidden state** and the **agent's role**, and forms **keys and values from $\boldsymbol{Z}_s$**. The query is matched against the memory keys to determine which memory vectors are relevant to the current token, and the corresponding values provide the information read from memory. In addition, a role-conditioned gate $\boldsymbol{g}_\ell^\rho$ controls how much of this memory readout is added to the token's hidden state. The reader has its own trainable projections $\boldsymbol{W}_q^{(\ell)}$, $\boldsymbol{W}_k^{(\ell)}$, $\boldsymbol{W}_v^{(\ell)}$, $\boldsymbol{W}_o^{(\ell)}$, and $\boldsymbol{W}_g^{(\ell)}$; the backbone's weights remain frozen.

We derive the role input from each agent's system instruction. Let π_ρ denote each role's system instruction. For each role, we feed its system instruction π_ρ into the frozen backbone and take the token representations from a middle Transformer layer. We combine these representations into a single role vector $\boldsymbol{e}_\rho$ using attention pooling, and $\boldsymbol{e}_\rho$ remains fixed during memory training.

At an insertion layer ℓ, let $\boldsymbol{H}_\ell$ contain the agent's hidden states and define $\boldsymbol{U}_\ell^\rho = [\mathrm{LN}_\ell(\boldsymbol{H}_\ell); \boldsymbol{e}_\rho]$, where we concatenate the role vector $\boldsymbol{e}_\rho$ with the hidden state at each token position. LN_ℓ denotes layer normalization. The reader computation is defined in Eq. 1, where d_r denotes the dimension of the reader's attention space.

$$\boldsymbol{Q}_\ell^\rho = \boldsymbol{U}_\ell^\rho \boldsymbol{W}_q^{(\ell)}, \quad \boldsymbol{K}_{s,\ell} = \boldsymbol{Z}_s \boldsymbol{W}_k^{(\ell)}, \quad \boldsymbol{V}_{s,\ell} = \boldsymbol{Z}_s \boldsymbol{W}_v^{(\ell)}, \quad \boldsymbol{A}_\ell^\rho = \mathrm{softmax}\left(\frac{\boldsymbol{Q}_\ell^\rho \boldsymbol{K}_{s,\ell}^\top}{\sqrt{d_r}} \right), \quad (1)$$

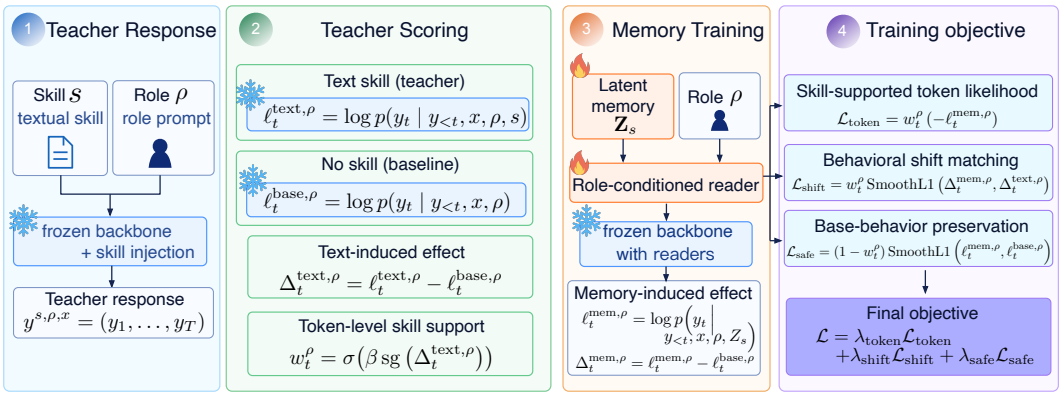

Figure 3: Behavioral skill distillation. For each skill, agent role, and calibration task, the frozen backbone generates a teacher response with the textual skill included in the role prompt. We score the same response with and without the textual skill to measure its token-level effect. We then replace the skill text with its memory and train the memory and role-conditioned reader to reproduce the skill-supported behavior while preserving the base behavior at positions with limited skill support.

Let $\boldsymbol{R}_\ell^\rho = \boldsymbol{A}_\ell^\rho \boldsymbol{V}_{s,\ell} \boldsymbol{W}_o^{(\ell)}$ denote the projected memory readout. The hidden-state updates are

$$\boldsymbol{g}_\ell^\rho = \sigma\left(\boldsymbol{U}_\ell^\rho \boldsymbol{W}_g^{(\ell)}\right), \qquad \Delta \boldsymbol{H}_\ell^\rho = \boldsymbol{g}_\ell^\rho \odot \boldsymbol{R}_\ell^\rho, \qquad \boldsymbol{H}_\ell \leftarrow \boldsymbol{H}_\ell + \Delta \boldsymbol{H}_\ell^\rho. \tag{2}$$

Here, $\sigma(\cdot)$ is the sigmoid function, which maps each element of $\boldsymbol{g}_\ell^\rho$ to $(0, 1)$, and $\odot$ denotes elementwise multiplication. Thus, $\boldsymbol{g}_\ell^\rho$ controls the contribution of each element in the memory readout $\boldsymbol{R}_\ell^\rho$ to the hidden-state update $\Delta \boldsymbol{H}_\ell^\rho$. Each insertion layer has its own reader parameters, shared across all skills and agent roles at that layer. Therefore, each skill requires only one stored memory $\boldsymbol{Z}_s$. The receiving agent's hidden states and role determine which parts of that memory are read and how strongly they update the hidden states. Agents continue to communicate through latent states.

4.3 Behavioral Skill Distillation

A compact memory cannot be expected to reproduce the wording of a textual skill. Instead, we train it to reproduce the effect of that skill on the model's predictions. For each skill, role, and calibration task, we first generate a fixed response using the textual skill. We then score the same response under three conditions: with the textual skill, without the skill, and with the skill memory. The change caused by the textual skill provides the training target for the memory. During this process, we train the skill memories and reader while keeping the backbone frozen. Figure 3 illustrates this procedure.

Teacher responses and behavioral targets. Consider a textual skill s, an agent role ρ, and a calibration task x. We add the textual skill to the prompt for role ρ and use the frozen backbone to generate a teacher response $y^{s,\rho,x} = (y_1, \ldots, y_T)$, where T is the response length and $t \in \{1, \ldots, T\}$ indexes a token position. For readability, we omit the task index x from the notation below.

Intermediate agents normally communicate through latent states and do not generate text. For these roles, we generate a textual response only to construct a training target. The generation starts from the role prompt without the KV states passed from preceding agents. These teacher responses are used only during training and do not change the latent communication process at inference time.

We score each teacher response under three conditions. The *text* condition includes the textual skill in the prompt. The *base* condition uses the same prompt without the skill. The *memory* condition also removes the skill text but provides $\boldsymbol{Z}_s$ through the role-conditioned reader. All three conditions use teacher forcing: when scoring token y_t, the model receives the same preceding teacher tokens $y_{<t}$. This ensures that the three scores differ only in how the skill is provided.

Let $\ell_t^{\text{text},\rho}$, $\ell_t^{\text{base},\rho}$, and $\ell_t^{\text{mem},\rho}$ denote the log-probability assigned to y_t under the three conditions. We define the text-induced and memory-induced changes relative to the base model as $\Delta_t^{\text{text},\rho} = \ell_t^{\text{text},\rho} - \ell_t^{\text{base},\rho}$, $\Delta_t^{\text{mem},\rho} = \ell_t^{\text{mem},\rho} - \ell_t^{\text{base},\rho}$. A positive $\Delta_t^{\text{text},\rho}$ means that the textual skill increases the model's probability of the teacher token y_t. We train the memory so that $\Delta_t^{\text{mem},\rho}$ follows $\Delta_t^{\text{text},\rho}$.

Token-level skill support. A teacher response contains both skill-related reasoning and generic tokens that the model could predict without the skill. We therefore give more training weight to positions where the textual skill increases the probability of the teacher token, and preserve the base model's behavior where the skill provides little support. Specifically, we convert the text-induced change at each position into a soft weight: $w_t^\rho = \sigma\big(\beta\ \text{sg}\big(\Delta_t^{\text{text},\rho}\big)\big)$. Here, $\sigma(\cdot)$ is the sigmoid function, β controls the sharpness of the weighting, and $\text{sg}(\cdot)$ denotes stop-gradient. A larger w_t^ρ indicates that the textual skill provides stronger support for token y_t. We use w_t^ρ to emphasize skill-supported behavior and $1 - w_t^\rho$ to preserve the base behavior at positions with limited skill support.

Training objective. The token weight w_t^ρ acts as a soft switch between imitating the textual skill and preserving the base behavior. When w_t^ρ is large, the memory-conditioned model should follow the text-conditioned model; when it is small, the memory should have little effect. We optimize

$$\mathcal{L}_{\text{token}} = \mathbb{E}_{s,\rho,x,t}\left[w_t^\rho\big(-\ell_t^{\text{mem},\rho}\big)\right], \tag{3}$$

$$\mathcal{L}_{\text{shift}} = \mathbb{E}_{s,\rho,x,t}\left[w_t^\rho\ \text{SmoothL1}\left(\Delta_t^{\text{mem},\rho}, \Delta_t^{\text{text},\rho}\right)\right], \tag{4}$$

$$\mathcal{L}_{\text{safe}} = \mathbb{E}_{s,\rho,x,t}\left[(1 - w_t^\rho)\ \text{SmoothL1}\left(\ell_t^{\text{mem},\rho}, \ell_t^{\text{base},\rho}\right)\right]. \tag{5}$$

The expectation is taken over skills, roles, calibration tasks, and teacher-response tokens. $\mathcal{L}_{\text{token}}$ increases the likelihood of skill-supported teacher tokens, $\mathcal{L}_{\text{shift}}$ matches the strength of the memory-induced and text-induced changes, and $\mathcal{L}_{\text{safe}}$ keeps the model close to its base behavior at positions with limited skill support. Here, SmoothL1 is quadratic for small errors and linear for larger errors.

Because both shifts use the same base condition, $\Delta_t^{\text{mem},\rho} - \Delta_t^{\text{text},\rho} = \ell_t^{\text{mem},\rho} - \ell_t^{\text{text},\rho}$. Thus, the shift loss is algebraically equivalent to matching the memory-conditioned and text-conditioned log-probabilities. We retain the shift notation to express both effects relative to the no-skill behavior; the same base condition is also used to compute w_t^ρ and $\mathcal{L}_{\text{safe}}$. The full objective is $\mathcal{L} = \lambda_{\text{token}}\mathcal{L}_{\text{token}} + \lambda_{\text{shift}}\mathcal{L}_{\text{shift}} + \lambda_{\text{safe}}\mathcal{L}_{\text{safe}}$. Teacher responses from all roles supervise the same $\boldsymbol{Z}_s$; the skill memories and shared reader are trained jointly while the backbone remains frozen.

4.4 Skill Discovery

The preceding method can compile any textual skill into a latent memory. To construct the skill libraries used in our experiments, we discover such skills using the same backbone model that is subsequently evaluated. For each benchmark $\mathcal{D}$, we reserve a calibration set $\mathcal{D}_{\text{cal}}$ of 20 examples. We run the backbone in the base multi-agent pipeline multiple times on each calibration task. A completed *multi-agent trajectory* contains the successive outputs produced by the planner, critic, refiner, and judger. For each task instance x_i, we retain a matched pair $\mathcal{P}_i = (\tau_i^+, \tau_i^-)$, where the final answer of τ_i^+ is correct and that of τ_i^- is incorrect. The resulting discovery set is $\mathcal{P} = \{(\tau_i^+, \tau_i^-)\}_{i=1}^{|\mathcal{D}_{\text{cal}}|}$. We then prompt the same backbone to compare the two trajectories in each pair and extract a reusable procedure. Because the paired trajectories solve the same task, their differences can reveal reasoning patterns that contribute to success or failure. Their role-specific outputs further indicate whether a pattern concerns planning, critique, refinement, or final verification. We remove candidates that contain instance-specific entities, values, or solution details, and deduplicate the remaining candidates by name. This produces a skill library $\mathcal{S} = \{s_1, \ldots, s_N\}$ for each benchmark–backbone setting. All calibration trajectories are disjoint from the evaluation examples. Further details and per-benchmark skill counts are provided in Appendix A, and representative skills are listed in Appendix A.1.

Skill discovery provides the textual skills used in our experiments, but it is not required by the compilation method. If an existing textual skill is already available, it can be added to the library and compiled directly into $\boldsymbol{Z}_s$ using the procedure in Section 4.3. We also explored directly deriving latent skill memories from successful and failed trajectories without textual discovery or distillation; these zero-training variants were substantially less reliable (see Appendix F).

5 WHY TEXT-SUPERVISED SKILLS TRANSFER TO LATENT REASONING

SkillMem is trained from textual teacher responses but used during latent reasoning. The key observation is that it does not reconstruct the skill text itself; instead, it learns the *behavioral effect* induced by the skill on the frozen backbone and applies that effect through hidden-state readers.

Behavioral effect as a likelihood ratio. Let $\boldsymbol{c}_t^\rho = (x, \rho, y_{<t})$ denote the prediction context. Recall that $\Delta_t^{\text{text},\rho} = \ell_t^{\text{text},\rho} - \ell_t^{\text{base},\rho}$. Equivalently,

$$\Delta_t^{\text{text},\rho} = \log \frac{p_\theta(y_t \mid \boldsymbol{c}_t^\rho, s)}{p_\theta(y_t \mid \boldsymbol{c}_t^\rho)}, \qquad \Delta_t^{\text{mem},\rho} = \log \frac{p_{\theta,\psi}(y_t \mid \boldsymbol{c}_t^\rho, \boldsymbol{Z}_s)}{p_\theta(y_t \mid \boldsymbol{c}_t^\rho)}.$$

Thus, behavioral distillation matches the change in model behavior caused by the textual skill rather than its textual representation.

Aligned latent inputs. LatentMAS maps a final-layer latent state back toward the model input space as $\tilde{\boldsymbol{e}}_{t+1} = \boldsymbol{h}_t^{(L)} \boldsymbol{W}_a$, with $\boldsymbol{W}_a \approx \boldsymbol{W}_{\text{out}}^\dagger \boldsymbol{W}_{\text{in}}$, where $\boldsymbol{W}_{\text{in}}$ and $\boldsymbol{W}_{\text{out}}$ are the input and output embedding matrices (Zou et al., 2026). Let Φ_ℓ denote the frozen mapping from the model input to a SkillMem insertion layer ℓ, and define $\boldsymbol{u}_{\text{text}} = \Phi_\ell(\boldsymbol{e})$ and $\boldsymbol{u}_{\text{lat}} = \Phi_\ell(\tilde{\boldsymbol{e}})$. If Φ_ℓ is locally L_Φ-Lipschitz, then $\|\boldsymbol{u}_{\text{lat}} - \boldsymbol{u}_{\text{text}}\| \le L_\Phi \|\tilde{\boldsymbol{e}} - \boldsymbol{e}\|$. Hence, the LatentMAS input-output alignment also controls the hidden-state mismatch seen by the SkillMem reader.

Proposition 1 (Text-to-latent behavioral transfer).

Let $F_s^\rho(\boldsymbol{u})$ and $G_s^\rho(\boldsymbol{u})$ denote the behavioral effects induced by the textual skill and SkillMem, respectively, at reader state $\boldsymbol{u}$. Assume both are locally Lipschitz with constants L_F and L_G, and that behavioral distillation satisfies $|G_s^\rho(\boldsymbol{u}_{\text{text}}) - F_s^\rho(\boldsymbol{u}_{\text{text}})| \le \delta$. If $\|\tilde{\boldsymbol{e}} - \boldsymbol{e}\| \le \epsilon_{\text{align}}$, then

$$|G_s^\rho(\boldsymbol{u}_{\text{lat}}) - F_s^\rho(\boldsymbol{u}_{\text{lat}})| \le \delta + (L_F + L_G) L_\Phi \epsilon_{\text{align}}.$$

The bound separates the transfer error into the behavioral distillation error δ and the representation gap introduced by latent recurrence. LatentMAS reduces the latter through input-output alignment before the hidden states reach the SkillMem reader. SkillMem therefore transfers the *behavioral effect* of a textual skill through a shared hidden-state interface rather than reproducing the skill text itself. A full derivation is provided in Appendix E.

6 EXPERIMENTS

6.1 EXPERIMENTAL SETUP

Backbones and benchmarks. We evaluate SKILLMEM using Qwen3-4B, Qwen3-8B, and Qwen3-14B (Yang et al., 2025) as frozen backbones. The evaluation includes GSM8K (Cobbe et al., 2021), ARC-Easy and ARC-Challenge (Clark et al., 2018), MBPP+ (Austin et al., 2021; Liu et al., 2023), MedQA (Jin et al., 2021), and GPQA-Diamond (Rein et al., 2024), covering mathematical reasoning, QA, medical reasoning, and code generation. Calibration examples used for skill discovery and distillation are disjoint from the evaluation examples. Unless otherwise stated, **results are averaged over three evaluation seeds**. The full evaluation protocol is provided in Appendix C.

SkillMem configuration. Each skill memory contains $M = 8$ vectors of dimension $d_{\text{mem}} = 256$. Memory readers are inserted at layers $\{L/4, L/2, 3L/4, L-1\}$, and each agent performs eight latent reasoning steps. We train the skill memories and shared reader for 200 steps using AdamW, with $\beta = 5.0$ and $(\lambda_{\text{token}}, \lambda_{\text{shift}}, \lambda_{\text{safe}}) = (1.0, 0.5, 0.1)$. Details are provided in Appendix D.

Baselines. TextMAS performs text-based multi-agent reasoning without skills, while **LatentMAS** is the corresponding latent multi-agent baseline. **KV Skill** represents each retrieved skill using its cached layer-wise KV states. Soft Prompt (**SP**) (Lester et al., 2021; Li & Liang, 2021) learns skill-specific continuous prompt vectors and inserts them at the input layer. It uses the same behavioral distillation objective as SKILLMEM, but does not use the memory reader. Skill-to-LoRA (**S2L**) (Zhang & Qi, 2026) stores each skill in a skill-specific LoRA adapter. **SkillMem** stores each skill as a compact external memory accessed through the shared role-conditioned reader. All skill-based methods use the same skill library and retrieved skill for each evaluation example. S2L settings are provided in Appendix D.

Table 1: **Main results across model scales.** We report task accuracy (%, "Acc."), average output tokens ("Token"), and inference time in seconds ("Speed"). "Text" and "Latent" denote TextMAS and LatentMAS, respectively, and "SP" denotes Soft Prompt. SkillMem accuracy is bolded when it exceeds either TextMAS or LatentMAS.

Tasks	Metrics	Qwen3-4B						Qwen3-8B						Qwen3-14B					
		Text	Latent	KV	SP	S2L	SkillMem	Text	Latent	KV	SP	S2L	SkillMem	Text	Latent	KV	SP	S2L	SkillMem
GSM8K	Acc. ↑	89.40	72.40	84.90	89.20	84.13	**89.90**	90.80	92.33	92.60	92.20	92.27	**93.07**	94.00	94.30	94.30	93.70	94.00	92.60
	Token ↓	229	215	205	208	185	169	240	196	206	195	223	202	198	188	195	195	187	175
	Speed ↓	19.1	6.8	6.2	6.1	7.6	5.5	35.1	6.0	12.2	6.0	7.9	6.6	26.8	9.4	11.8	9.7	9.7	8.9
GPQA-D	Acc. ↑	29.94	17.97	22.46	30.24	27.54	**35.63**	33.53	36.53	40.72	35.33	34.33	**37.72**	37.72	31.74	28.14	35.33	34.13	**35.33**
	Token ↓	488	222	268	380	366	420	495	398	408	427	407	374	501	445	456	365	428	476
	Speed ↓	54.5	9.0	9.6	10.3	12.6	12.1	49.7	15.2	22.7	16.2	14.1	13.5	64.9	21.2	28.4	18.0	23.1	22.6
ARC-E	Acc. ↑	96.87	92.67	96.27	97.27	93.00	**97.53**	97.60	98.90	98.40	98.20	98.53	**97.80**	98.40	98.70	99.00	98.80	98.60	**98.90**
	Token ↓	220	171	162	162	147	92	304	163	168	177	191	170	261	169	187	164	119	172
	Speed ↓	22.0	5.4	6.5	5.0	5.4	3.3	27.4	7.4	5.3	7.8	8.0	5.7	34.1	9.8	10.9	9.7	7.2	8.8
ARC-C	Acc. ↑	90.53	83.67	86.27	89.80	82.67	**91.20**	91.80	92.40	92.80	92.60	92.47	91.80	93.40	94.20	94.40	93.33	93.00	**93.93**
	Token ↓	294	190	186	177	193	162	341	183	191	205	195	163	315	189	210	153	137	222
	Speed ↓	26.8	5.9	8.5	5.2	6.9	5.2	25.2	8.2	11.2	9.1	7.7	5.5	37.1	11.2	10.3	9.7	7.7	11.4
MedQA	Acc. ↑	60.10	21.84	33.57	55.63	33.65	**61.50**	64.56	67.84	68.55	66.43	67.14	**69.49**	70.42	71.60	69.25	68.08	69.01	69.72
	Token ↓	360	190	166	259	252	215	416	250	267	279	298	263	406	183	182	214	90	54
	Speed ↓	44.1	7.7	11.4	8.3	10.5	6.8	49.0	11.5	8.5	10.6	10.6	8.6	61.3	11.7	10.2	13.3	6.5	4.8
MBPP+	Acc. ↑	67.75	32.74	40.12	60.96	31.07	**65.39**	65.11	70.23	70.23	69.65	68.72	**71.56**	66.86	72.09	70.93	71.61	70.64	**67.25**
	Token ↓	187	122	121	96	163	78	92	76	79	78	77	68	121	87	109	102	118	129
	Speed ↓	36.5	9.0	8.3	6.2	11.3	5.6	19.9	3.1	3.2	3.2	3.7	3.1	37.2	4.6	8.4	5.1	6.1	8.6

6.2 Main Results

Table 1 compares SkillMem with text-based and latent multi-agent baselines, as well as KV, Soft Prompt, and LoRA skill representations, across three Qwen3 model scales.

Overall performance. With Qwen3-4B, SkillMem outperforms LatentMAS and all three alternative skill representations on every benchmark, and achieves the highest overall accuracy on five of the six tasks. With Qwen3-8B, it achieves the best results on GSM8K, MedQA, and MBPP+. Overall, the results show that compact external memory can preserve useful skill behavior without storing layer-wise KV states or skill-specific model parameters.

Effect of model scale. The gains are strongest and most consistent with Qwen3-4B, become task dependent with Qwen3-8B, and show no consistent advantage with Qwen3-14B. This reduction is partly consistent with a ceiling effect: on GSM8K and ARC, the 14B baselines already achieve accuracy above 90%, leaving less room for additional skill conditioning. Overall, SkillMem provides the clearest benefit when the backbone has more room to improve from external procedural guidance.

Table 2: (a) Skill storage scaling with bank size. (b) Inference and training efficiency.

(a) Skill Storage Scaling

Method	Per-Skill	N = 10	N = 100	N = 1000
Text Skill	173 B	1.7 KB	17.3 KB	173 KB
LoRA Skill	30.7 MB	306.9 MB	3.07 GB	30.69 GB
KV Skill	13.0 MB	135.9 MB	1.36 GB	13.59 GB
SkillMem: Reader	*shared*	34.5 MB	34.5 MB	34.5 MB
SkillMem: Memory	9.6 KB	96 KB	0.96 MB	9.6 MB
SkillMem: Total	–	**34.6 MB**	**35.5 MB**	**44.1 MB**
vs. KV	–	**3.9×**	**38.3×**	**308×**
vs. LoRA	–	**8.9×**	**86.6×**	**696×**

(b) Inference and Training Efficiency

Inference			
Method	**Latency ↓**	**Speedup ↑**	**Mem. ↓**
TextMAS	33.65	1.00×	34.02
LatentMAS	8.18	4.11×	33.83
Text Skill	12.98	2.59×	36.13
KV Skill	12.01	2.80×	34.97
SkillMem	**7.95**	**4.23×**	**33.86**

Training (10-skill MedQA bank)		
Method	**Scheme**	**Time**
LoRA Skill	Per-skill Adapters	110 min
SkillMem	**Shared Reader**	**11.7 min**

6.3 Storage and Efficiency

Table 2 compares the efficiency of different skill representations on Qwen3-8B.

Storage. Text Skill has negligible storage but must be processed at each use. KV Skill and LoRA Skill require 13.0 MB and 30.7 MB per skill, respectively, while SkillMem uses only 9.6 KB per skill plus one shared reader. From 10 to 1000 skills, SkillMem is 3.9×–308× smaller than KV Skill.

Table 3: **Ablation studies of SKILLMEM.** Results report task accuracy (%).

(a) Role-Conditioned Memory Access

Method	GSM8K	GPQA-D	ARC-E	ARC-C	MedQA	Avg.
Qwen3-4B						
SKILLMEM	89.90	35.63	97.53	91.20	61.50	**75.15**
Role Agnostic	75.70	32.94	97.87	89.73	40.61	67.37
Constant Emb.	87.40	34.14	97.53	90.07	59.39	73.71
Shuffled Emb.	85.00	34.13	97.33	87.80	57.75	72.40
Per-role Memory	83.90	32.94	96.60	90.27	58.92	72.52
Qwen3-8B						
SKILLMEM	93.07	37.72	97.80	91.80	69.48	**77.97**
Role Agnostic	88.93	35.33	98.80	91.20	68.23	76.50
Constant Emb.	92.27	40.12	98.40	92.20	66.04	77.81
Shuffled Emb.	92.87	38.32	98.30	91.40	68.54	77.89
Per-role Memory	90.87	35.93	98.40	91.60	66.98	76.76

(b) Memory Capacity

Task–Model	$M{=}1$	$M{=}2$	$M{=}4$	$M{=}8$	$M{=}16$
GSM8K-4B	87.67	87.80	82.00	**89.90**	85.87
GSM8K-8B	**93.47**	92.47	92.67	93.07	89.40
GPQA-D-4B	34.53	**36.13**	32.14	35.63	29.74
GPQA-D-8B	**38.12**	**38.12**	37.52	37.72	35.33

(c) Distillation Objective

Task–Model	Full	w/o $\mathcal{L}_{\text{token}}$	w/o $\mathcal{L}_{\text{shift}}$	w/o $\mathcal{L}_{\text{safe}}$
GSM8K-4B	**89.90**	87.33	86.13	88.00
GSM8K-8B	**93.07**	92.00	92.33	92.87
GPQA-D-4B	**35.63**	33.73	32.53	31.34
GPQA-D-8B	37.72	**37.92**	35.73	33.33
MedQA-4B	**61.50**	59.31	60.56	60.56
MedQA-8B	**69.49**	67.29	67.29	68.70
Avg.	**64.55**	62.93	62.43	62.47

Inference efficiency. SkillMem achieves 7.95 s end-to-end latency, comparable to LatentMAS at 8.18 s and lower than Text Skill and KV Skill at 12.98 s and 12.01 s. Its fixed eight-vector memory adds neither skill tokens nor skill-specific KV prefixes, preserving efficient latent inference.

Training efficiency. For 10 MedQA skills, SkillMem trains its shared reader in 11.7 minutes, versus 110 minutes for separate LoRA adapters, a $9.4\times$ reduction. This advantage comes from sharing the reader across the skill bank, whereas S2L trains a separate adapter for each skill.

6.4 ABLATION STUDIES

Table 3 evaluates the role-conditioned reader, memory capacity, and distillation objective while keeping the retrieved skills and evaluation examples fixed.

Role conditioning. Table 3(a) compares the full reader with role-agnostic, constant-embedding, shuffled-embedding, and per-role memory variants. Removing role conditioning causes the largest drop, reducing average accuracy from 75.15% to 67.37% on Qwen3-4B and from 77.97% to 76.50% on Qwen3-8B. Separate memories for each role provide no advantage over one shared memory, supporting role-conditioned access to shared skill knowledge.

Memory capacity. Table 3(b) shows a non-monotonic relationship between memory size and accuracy. The default $M{=}8$ achieves the best average accuracy of 64.08%, while $M{=}2$ remains close at 63.63% with one quarter as many memory vectors. Increasing the capacity to $M{=}16$ reduces accuracy, indicating that only a small memory is needed to capture useful skill behavior.

Distillation objective. Removing any loss term reduces the macro-average accuracy from 64.55% to between 62.43% and 62.93% in Table 3(c). The largest drop occurs without $\mathcal{L}_{\text{shift}}$, supporting direct matching of the behavioral change induced by the textual skill. The full objective performs best overall, showing that the three losses provide complementary training signals.

7 CONCLUSION AND FUTURE WORK

We introduced SKILLMEM, which represents each reusable skill as a compact external memory for latent multi-agent reasoning. A role-conditioned reader allows agents with different roles to share the same memory, while behavioral distillation transfers the effect of a textual skill without modifying the frozen backbone. Across six benchmarks and three model scales, the results show that compact skill memories can provide useful procedural guidance with low skill-specific storage and efficient latent inference.

Future work can extend SKILLMEM to retrieve and compose multiple skills within the same reasoning process, including methods for resolving conflicting guidance. Retrieval could also incorporate the agent's role and current reasoning state in addition to the task query. Finally, skill memories could be updated online as agents encounter new tasks and interaction patterns.

AI Assistance Disclosure

Large language models were used to assist with language editing and manuscript polishing, including improvements to grammar, clarity, and readability. They were also used to assist with implementing parts of the research code. All model-assisted code was reviewed, tested, and verified by the authors. The research ideas, methodology, experimental design, analysis, and scientific conclusions were developed and validated by the authors, who take full responsibility for the content of the paper.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021. URL https://arxiv.org/abs/2108.07732.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3829–3846. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.232. URL https://aclanthology.org/2023.emnlp-main.232/.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018. URL https://arxiv.org/abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. URL https://arxiv.org/abs/2110.14168.

Lucio M. Dery, Benedict Aaron Tjandra, Siavash Samiei, Adhiguna Kuncoro, Zohar Yahav, Jiajun Shen, and Arthur Szlam. SkillSmith: Learning to compose parametric skills and textual knowledge. *arXiv preprint arXiv:2607.27497*, 2026. URL https://arxiv.org/abs/2607.27497.

Muxin Fu, Xiangyuan Xue, Yafu Li, Zefeng He, Siyuan Huang, Xiaoye Qu, Yu Cheng, and Yang Yang. LatentMem: Customizing latent memory for multi-agent systems. *arXiv preprint arXiv:2602.03036*, 2026. URL https://arxiv.org/abs/2602.03036.

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. *arXiv preprint arXiv:2412.06769*, 2024. URL https://arxiv.org/abs/2412.06769.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421, 2021. doi: 10.3390/app11146421. URL https://www.mdpi.com/2076-3417/11/14/6421.

Haibo Jin, Kuang Peng, Ye Yu, Xiaopeng Yuan, and Haohan Wang. Agent primitives: Reusable latent building blocks for multi-agent systems. *arXiv preprint arXiv:2602.03695*, 2026. URL https://arxiv.org/abs/2602.03695.

Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 3045–3059. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.243. URL https://aclanthology.org/2021.emnlp-main.243/.

Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 4582–4597. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353/.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html.

Jesse Mu, Xiang Li, and Noah Goodman. Learning to compress prompts with gist tokens. In *Advances in Neural Information Processing Systems*, volume 36, 2023. doi: 10.52202/075280-0848. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/3d77c6dcc7f143aa2154e7f4d5e22d68-Abstract.html.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A graduate-level google-proof Q&A benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. CODI: Compressing chain-of-thought into continuous space via self-distillation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 677–693. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.36. URL https://aclanthology.org/2025.emnlp-main.36/.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024. URL https://openreview.net/forum?id=ehfRiF0R3a.

Jinyang Wu, Shuo Yang, Zhengxi Lu, Fan Zhang, Yuhao Shen, Lang Feng, Haoran Luo, Zheng Lian, Shuai Zhang, Zhengqi Wen, and Jianhua Tao. SEED: Self-evolving on-policy distillation for agentic reinforcement learning. *arXiv preprint arXiv:2607.14777*, 2026. URL https://arxiv.org/abs/2607.14777.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving agents via recursive skill-augmented reinforcement learning. *arXiv preprint arXiv:2602.08234*, 2026. URL https://arxiv.org/abs/2602.08234.

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

Xiyuan Yang, Jiaru Zou, Rui Pan, Ruizhong Qiu, Pan Lu, Shizhe Diao, Jindong Jiang, Hanghang Tong, Tong Zhang, Markus J. Buehler, Jingrui He, and James Zou. Recursive multi-agent systems. *arXiv preprint arXiv:2604.25917*, 2026. URL https://arxiv.org/abs/2604.25917.

Hancheng Ye, Zhengqi Gao, Mingyuan Ma, Qinsi Wang, Yuzhe Fu, Ming-Yu Chung, Yueqian Lin, Zhijian Liu, Jianyi Zhang, Danyang Zhuo, and Yiran Chen. KV-COMM: Online cross-context KV-cache communication for efficient LLM-based multi-agent systems. In *Advances in Neural Information Processing Systems*, volume 38,

2025. URL https://papers.nips.cc/paper_files/paper/2025/hash/1a074a28c3a6f2056562d00649ae6416-Abstract-Conference.html.

Aofan Yu, Chenyu Zhou, Tianyi Xu, Zihan Guo, Rong Shan, Zhihui Fu, Jun Wang, Weiwen Liu, Yong Yu, Weinan Zhang, and Jianghao Lin. LatentSkill: From in-context textual skills to in-weight latent skills for LLM agents. *arXiv preprint arXiv:2606.06087*, 2026. URL https://arxiv.org/abs/2606.06087.

Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping agents for the real world with agent skills. Anthropic, October 2025. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills.

Tianyi Zhang and Zhonghao Qi. Skill-to-LoRA: From using skills to learning behaviors for token-efficient LLM agents. *arXiv preprint arXiv:2606.16769*, 2026. URL https://arxiv.org/abs/2606.16769.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

YanZhao Zheng, ZhenTao Zhang, Chao Ma, YuanQiang Yu, JiHuan Zhu, Baohua Dong, and Hangcheng Zhu. SkillRouter: Retrieve-and-rerank skill selection for LLM agents at scale. *arXiv preprint arXiv:2603.22455*, 2026. URL https://arxiv.org/abs/2603.22455.

Jiaru Zou, Xiyuan Yang, Ruizhong Qiu, Gaotang Li, Katherine Tieu, Pan Lu, Ke Shen, Hanghang Tong, Yejin Choi, Jingrui He, James Zou, Mengdi Wang, and Ling Yang. Latent collaboration in multi-agent systems. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=syG9I9ofd8.

A SKILL DISCOVERY

To discover reusable procedural skills for each benchmark, we first collect a small calibration set using the backbone model under evaluation. For each benchmark, we reserve 20 calibration examples that are disjoint from all evaluation examples. Each example is run through the full text-based multi-agent pipeline (Planner $\rightarrow$ Critic $\rightarrow$ Refiner $\rightarrow$ Judger), using temperature 0.6 and top-p 0.95.

For each calibration example x_i, we construct a matched pair of completed multi-agent trajectories

$$P_i = (\tau_i^+, \tau_i^-),$$

where τ_i^+ reaches the correct final answer and τ_i^- reaches an incorrect answer for the same task. Both trajectories preserve the complete sequence of role-specific reasoning stages, including planning, critique, refinement, and final verification. This matched construction allows skill discovery to compare successful and failed reasoning under the same task conditions rather than aggregating unrelated success and failure examples.

We then prompt the same backbone model used to generate the trajectories to compare each matched pair and identify reusable procedures. The comparison can draw on outputs from any agent role, allowing the model to identify procedures relevant to planning, critique, refinement, or verification. Each candidate skill is represented as

$$s_j = (n_j, d_j, p_j),$$

where n_j is a short name, d_j describes when the skill is useful, and p_j specifies the procedure.

To reduce instance-specific artifacts, we remove candidates containing entities, numerical values, or solution details tied to individual calibration examples. We then deduplicate semantically overlapping candidates to obtain the final skill library

$$\mathcal{S} = \{s_1, \ldots, s_N\}.$$

Skill discovery is performed independently for each backbone scale, so the resulting libraries may differ across Qwen3-4B, Qwen3-8B, and Qwen3-14B. Table 4 reports the number of discovered skills for each benchmark and model scale. The complete skill lists are provided in Appendix A.1.

Table 4: Number of discovered skills per benchmark and model scale after filtering instance-specific candidates and deduplicating overlapping skills.

Benchmark	Qwen3-4B	Qwen3-8B	Qwen3-14B
GSM8K	10	10	8
GPQA-D	9	8	8
ARC-E	10	10	10
ARC-C	9	9	9
MedQA	8	10	8
MBPP+	10	10	10

A.1 Discovered Skills

Table 5 lists the skills discovered for each benchmark under the Qwen3-4B backbone (the discovery process is run independently per model scale; skill sets for Qwen3-8B/14B are qualitatively similar but not identical, since discovery reflects each backbone's own error and success patterns).

Table 5: Discovered skills per benchmark (Qwen3-4B backbone).

Dataset	Skill	Description
GSM8K	Rate vs Total Confusion	When dealing with per-unit rates, ensure that the total amount is not confused with the rate itself.
	Quantity Tracking	Track named quantities, updates, and dependencies in a word problem.
	Equation Construction	Translate verbal relations into arithmetic equations before solving.
	Rate and Proportion Reasoning	Handle per-unit rates, totals, fractions, percentages, and repeated events.
	Intermediate Result Verification	Check intermediate arithmetic and whether each number answers the intended subquestion.
	Target Quantity Restatement	Restate the question's target quantity before the final step to avoid answering a different quantity.
	Unit Normalization Before Combining	Convert every rate to a common time unit before combining them.
	Incremental Step Verification	Verify each step of the calculation by explicitly stating the intermediate result.
	Relationship Clarification	Clarify relationships between quantities (e.g., "twice as many") to ensure correct proportional calculation.
	Total Distance Calculation	Calculate total distance by summing individual segments, ensuring all parts are accounted for.
GPQA-D	Rate vs Total Confusion	Avoid conflating per-unit rates with total amounts when a quantity depends on multiple variables.
	Stale Intermediate Value Reuse	Avoid reusing an intermediate value from an earlier step when a different quantity is asked for.
	Approximation as Exact	Avoid treating an approximation as exact when precise calculation is required.
	Dimensional and Scale Analysis	Ensure units and scales are consistent and physical-dimension relationships are correctly interpreted.
	Misapplication of Proportionality	Avoid assuming a linear relationship where a non-linear one is required.
	Target Quantity Restatement	Restate the question's target quantity before the final step.
	Unit Normalization Before Combining	Convert all rates/quantities to a common unit before combining.
	First Principles Modeling	Translate an advanced science question into governing principles and variables.

Table 5 – continued

Benchmark	Skill	Description
	Equation and Assumption Tracking	Track formulas, assumptions, approximations, and variable meanings.
ARC-E	Rate vs Total Confusion	Avoid confusing a per-unit rate with a total amount when interpreting a physical process.
	Stale Intermediate Value Reuse	Avoid reusing an intermediate value instead of the one the question asks for.
	Conceptual Misalignment in Biological Processes	Avoid misinterpreting the biological process described (e.g., confusing adaptation with a specific mechanism).
	Option Evaluation Misalignment	Avoid evaluating options against incorrect assumptions about their relevance.
	Direct Result vs Indirect Consequence Confusion	Distinguish a direct result of a process from an indirect or secondary effect.
	Target Quantity Restatement	Restate the question's target quantity before the final step.
	Unit Normalization Before Combining	Convert every rate to a common unit before combining.
	Specific Function Matching	Match the function of each option to the specific requirement of the question.
	Key Concept Identification	Identify the core concept or principle the question is testing.
	Sequence of Actions in Lab Procedures	Follow a specific sequence of actions in laboratory procedures.
ARC-C	Rate vs Total Confusion	Distinguish per-unit rates (e.g., gravitational acceleration) from total quantities (e.g., potential energy).
	Stale Intermediate Value Reuse	Avoid reusing intermediate values when the question asks for a different quantity or condition.
	Infectious vs Non-Infectious Misclassification	Distinguish infectious from non-infectious diseases by transmission method, not symptoms alone.
	Causal Mechanism Reasoning	Infer direct causes and effects in physical, biological, and earth-science settings.
	Experiment Variable Analysis	Separate manipulated variables, measured variables, controls, and conclusions.
	Target Quantity Restatement	Restate the question's target quantity before the final step.
	Unit Normalization Before Combining	Convert all measurements to a common unit before combining.
	Process Sequence Reasoning	Break a complex process into sequential, interdependent steps.
	Ecological Context Analysis	Use ecological knowledge to interpret species-interaction and environmental-change questions.
MedQA	Stale Intermediate Value Reuse	Avoid reusing an earlier intermediate value without verifying it is still relevant.
	Treats Approximation as Exact	Do not assume an approximate value is exact; account for its precision.
	Clinical Feature Prioritization	Extract high-yield symptoms, exam findings, labs, demographics, and timeline.
	Differential Diagnosis	Compare plausible diagnoses using distinguishing clinical features.
	Pathophysiology to Management	Use disease mechanism to select the appropriate test, treatment, or complication.

Table 5 – continued

Benchmark	Skill	Description
	Target Quantity Restatement	Restate the question's target quantity before the final step.
	Unit Normalization Before Combining	Convert relevant rates/measurements to a common unit before combining.
	Complication Risk Assessment	Evaluate long-term complications or risks associated with a diagnosis.
MBPP+	Rate vs Total Confusion	Check whether a per-unit rate is conflated with a total amount.
	Stale Intermediate Value Reuse	Check whether an intermediate value is reused without being updated.
	Sign Drop in Loss and Gain	Check whether the sign is dropped when combining a loss and a gain.
	Quantity Mismatch in Question	Check whether the answer targets an earlier-computed quantity instead of the one asked for.
	Approximation as Exact	Check whether an approximation is treated as exact without re-checking its scale.
	Identify Correct Parameters	Ensure the function signature matches the required parameters.
	Direct Formula Implementation	Implement the required formula directly without unnecessary intermediate steps.
	Handle Edge Cases	Ensure the function handles edge cases such as zero or negative values.
	Input Output Contract Analysis	Infer the required function behavior, input types, output type, and edge cases.
	Algorithm Decomposition	Break the programming task into simple transformation steps before coding.

B SKILL RETRIEVAL

Each discovered skill is indexed once, offline, by embedding its skill name, description, and skill prompt (concatenated) with the same frozen backbone used for evaluation: we run a forward pass with `output_hidden_states=True`, take the *last* hidden layer, apply attention-mask-weighted mean pooling over token positions, and L_2-normalize the resulting vector. When source questions from the discovery trajectories are available, we additionally embed each source question with the same procedure and aggregate the resulting similarity scores with the description-level score by taking their *mean* (rather than using the skill description alone).

At inference time, the input question is embedded with the identical procedure, and the top-1 skill is retrieved by cosine similarity between the query embedding and each skill's indexed embedding. Retrieval is performed once per evaluation example (not per agent role): all agents in the same multi-agent rollout for a given question consume the same retrieved skill. We do not apply a minimum-similarity threshold for skill selection (threshold $= 0$), so every routed example is assigned a skill; we found that a small positive threshold instead routes a large majority of examples to "no skill" and understates the benefit of skill augmentation. Retrieval is precomputed once per benchmark and cached, and the same cached routing decisions are reused across every skill-based method in Table 1 (KV Skill, S2L, and SkillMem) to ensure a matched comparison.

C EVALUATION PROTOCOL

Table 6 lists the decoding configuration used throughout evaluation. All stochastic generation (calibration rollouts and every agent's turn at evaluation time) samples at temperature 0.6 with top-p 0.95; each agent performs 8 latent reasoning steps before the final (judger) role decodes a textual answer with a maximum of 512 new tokens. Every reported accuracy is the average over 3 evaluation seeds (41, 42, 43), which vary only the decoding RNG on top of a fixed trained checkpoint.

Table 6: Decoding configuration used for evaluation.

Setting	Value
Sampling temperature	0.6
Top-p	0.95
Latent reasoning steps per agent	8
Max. new tokens (judger decode), default	512
Evaluation seeds (averaged)	41, 42, 43

D Experimental Setup: Additional Implementation Details

D.1 SkillMem Hyperparameters

Table 7 lists the hyperparameters used to train SkillMem's latent memories and shared role-conditioned reader, held fixed across all main-table experiments (Table 1) unless otherwise noted in an ablation.

Table 7: SkillMem training hyperparameters (default configuration, $M = 8$).

Hyperparameter	Value
Memory slots per skill (M)	8
Memory dimension (d_{mem})	256
Reader dimension (d_r)	256 (defaults to d_{mem})
Reader insertion layers	$\{L/4,\ L/2,\ 3L/4,\ L-1\}$
Qwen3-4B / 8B (L=36)	layers $\{9, 18, 27, 35\}$
Qwen3-14B (L=40)	layers $\{10, 20, 30, 39\}$
Role embedding dimension	64
Role embedding extraction layer	middle decoder layer ($\lfloor L/2 \rfloor$)
Latent reasoning steps per agent	8
Optimizer	AdamW
Training steps	200
Sigmoid weighting sharpness β	5.0
Loss weights ($\lambda_{\text{token}}, \lambda_{\text{shift}}, \lambda_{\text{safe}}$)	$(1.0,\ 0.5,\ 0.1)$
Backbone	frozen throughout training

Role representation. For each agent role ρ, we encode its system-instruction text π_ρ with the frozen backbone and pool hidden states from the middle decoder layer ($\lfloor L/2 \rfloor$, i.e. layer 18 for Qwen3-4B/8B and layer 20 for Qwen3-14B) using a *fixed* (untrained) attention query and projection: both are drawn once from a seeded random Gaussian initialization and kept frozen for the lifetime of the model wrapper, computed under `no_grad` and never registered as trainable parameters. The resulting role embedding $e_\rho \in \mathbb{R}^{64}$ is therefore not learned; the four agent roles are distinguished purely by the content of their respective instruction text, not by any trained role-specific parameters. This keeps the shared reader's role-conditioning entirely a function of frozen backbone representations, introducing no additional trainable role-specific parameters.

Reader parameters are not shared across insertion layers. Each of the four insertion layers has its own independent set of reader weights (W_q, W_k, W_v, W_o, W_g, plus a LayerNorm): readers at different depths are separately parameterized modules, not a single reader applied repeatedly. Only the memory bank Z_s is shared as a single per-skill vector across all four readers; each reader learns its own layer-specific mapping from that shared memory into its local residual stream.

D.2 Intermediate-Role Behavioral Distillation

Because LatentMAS agents other than the final (judger) role never emit text during normal inference, obtaining a token-level supervision target for their behavioral-distillation losses (Eqs. 3–5)

requires generating text for those roles *during training only*. For every (skill, role) pair, we generate one teacher continuation per calibration example by running the frozen backbone's ordinary text-generation path – with the retrieved skill's text injected into that role's own prompt and *no* context from other agents (context $= \varnothing$) – and decode greedily. This continuation is cached once and reused for all subsequent training steps; it is never regenerated on the fly. The same context $= \varnothing$ convention is used at inference time (cross-agent information is carried entirely through the shared latent KV cache, not through the prompt text), so the prompt text seen at training time exactly matches the prompt text seen at inference time for every role. The one difference between the two settings is that, at training time, the teacher continuation for a given role is generated from a fresh KV state (no upstream latent context), whereas at inference time the same role's turn is conditioned on the latent KV cache accumulated from preceding agents; the textual content of the prompt itself does not depend on this KV prefix. The generated teacher tokens serve solely as a fixed teacher-forcing target for computing l_{text}, l_{base}, and (during training) l_{mem}: they are never fed back into the model as generation input and never appear anywhere in the deployed (inference-time) pipeline, which loads only the trained memory bank and reader weights.

Because the planner, critic, and refiner roles are, by design, not instructed to produce a final answer, we do not apply an answer-correctness filter to their generated teacher continuations (doing so would be a category error, since these roles have no notion of a "final answer" to check); only the judger's continuation is filtered by answer correctness before being accepted into the teacher cache.

D.3 Skill-to-LoRA (S2L) Baseline Hyperparameters

Table 8 lists the hyperparameters used for the Skill-to-LoRA baseline (Zhang & Qi, 2026), matched to the paper's reported architecture (rank, alpha, target modules) with the training loop otherwise following this work's own conventions (AdamW, fixed step budget) rather than the original paper's QLoRA recipe. One independent LoRA adapter is trained per discovered skill, pooling teacher-forcing supervision from all four agent roles (the direct analogue of the *Role Agnostic* condition in Table 3(a), since the LoRA baseline has no native mechanism for role conditioning).

Table 8: Skill-to-LoRA (S2L) baseline hyperparameters.

Hyperparameter	**Value**
LoRA rank r	16
LoRA α	32
LoRA dropout	0.0
Target modules	W_q, W_v
Learning rate	$1\text{e}{-4}$
Batch size	4
Training steps per skill	200
Optimizer	AdamW
Backbone	frozen throughout training

E Detailed Analysis of Text-to-Latent Skill Transfer

This appendix provides the derivation behind Proposition 1 in Section 5. The analysis separates text-to-latent transfer into two steps: (i) LatentMAS maps a final-layer latent state back toward the model input distribution, and (ii) this input-space alignment propagates through the frozen backbone to the hidden states consumed by the SkillMem reader.

E.1 Input-Output Alignment in Latent Reasoning

In standard autoregressive generation, a discrete token is mapped through the input embedding matrix $\boldsymbol{W}_{\text{in}}$ before being processed by the Transformer. In latent reasoning, LatentMAS instead continues from the final-layer hidden state $\boldsymbol{h}_t^{(L)}$. Because final-layer hidden states and learned input embeddings generally follow different activation distributions, directly feeding $\boldsymbol{h}_t^{(L)}$ back as the next input may introduce distribution mismatch.

LatentMAS addresses this by applying an input-output alignment operator (Zou et al., 2026):

$$\tilde{\boldsymbol{e}}_{t+1} = \boldsymbol{h}_t^{(L)} \boldsymbol{W}_a, \qquad \boldsymbol{W}_a \approx \boldsymbol{W}_{\text{out}}^{\dagger} \boldsymbol{W}_{\text{in}}, \tag{6}$$

where $\boldsymbol{W}_{\text{out}}^{\dagger}$ denotes the pseudoinverse of the output embedding matrix. The aligned vector $\tilde{\boldsymbol{e}}_{t+1}$ is then used as the next latent input.

For the transfer analysis, let $\boldsymbol{e}$ denote a text-derived input embedding and $\tilde{\boldsymbol{e}}$ an aligned latent input representing a nearby reasoning state. We characterize the residual input mismatch by

$$\|\tilde{\boldsymbol{e}} - \boldsymbol{e}\| \le \epsilon_{\text{align}}. \tag{7}$$

We do not assume that the two inputs are identical. The quantity ϵ_{align} simply measures the remaining discrepancy after the LatentMAS alignment step.

E.2 From Input Alignment to Reader-State Alignment

SkillMem does not directly read the final-layer latent state $\boldsymbol{h}_t^{(L)}$. Instead, the aligned input is processed by the same frozen backbone, and the role-conditioned reader operates on intermediate hidden states at the selected insertion layers.

Let Φ_ℓ denote the frozen mapping from the model input to insertion layer ℓ, conditioned on the preceding context. We define

$$\boldsymbol{u}_{\text{text}} = \Phi_\ell(\boldsymbol{e}), \qquad \boldsymbol{u}_{\text{lat}} = \Phi_\ell(\tilde{\boldsymbol{e}}), \tag{8}$$

where $\boldsymbol{u}_{\text{text}}$ and $\boldsymbol{u}_{\text{lat}}$ are the reader inputs induced by text-derived and latent reasoning, respectively.

Assume that Φ_ℓ is locally Lipschitz over the region visited by these states, with constant L_Φ:

$$\|\Phi_\ell(\boldsymbol{e}_1) - \Phi_\ell(\boldsymbol{e}_2)\| \le L_\Phi \|\boldsymbol{e}_1 - \boldsymbol{e}_2\|. \tag{9}$$

Applying this condition to $\boldsymbol{e}$ and $\tilde{\boldsymbol{e}}$ gives

$$\|\boldsymbol{u}_{\text{lat}} - \boldsymbol{u}_{\text{text}}\| \le L_\Phi \epsilon_{\text{align}}. \tag{10}$$

Equation 10 connects the alignment mechanism of LatentMAS to SkillMem. If the aligned latent input remains close to the text-derived input space, then the hidden states presented to the reader also remain close, up to the local sensitivity of the frozen backbone.

E.3 Behavioral Effect Matching

SkillMem is trained to reproduce the behavioral change induced by a textual skill rather than its textual or hidden representation. For prediction context $\boldsymbol{c}_t^\rho = (x, \rho, y_{<t})$, the text-induced and memory-induced effects are

$$\Delta_t^{\text{text},\rho} = \log \frac{p_\theta(y_t \mid \boldsymbol{c}_t^\rho, s)}{p_\theta(y_t \mid \boldsymbol{c}_t^\rho)}, \qquad \Delta_t^{\text{mem},\rho} = \log \frac{p_{\theta,\psi}(y_t \mid \boldsymbol{c}_t^\rho, \boldsymbol{Z}_s)}{p_\theta(y_t \mid \boldsymbol{c}_t^\rho)}. \tag{11}$$

These are exactly the changes relative to the shared no-skill condition used in the behavioral distillation objective.

Let $F_s^\rho(\boldsymbol{u})$ denote the behavioral effect induced by the textual skill when the relevant hidden state is $\boldsymbol{u}$, and let $G_s^\rho(\boldsymbol{u})$ denote the corresponding effect induced by SkillMem. Behavioral distillation aims to make these two effects close on the text-derived training states. We write this as

$$|G_s^\rho(\boldsymbol{u}_{\text{text}}) - F_s^\rho(\boldsymbol{u}_{\text{text}})| \le \delta, \tag{12}$$

where δ denotes the residual distillation error.

We additionally assume that both behavioral-effect functions are locally Lipschitz in the reader-state region:

$$|F_s^\rho(\boldsymbol{u}_1) - F_s^\rho(\boldsymbol{u}_2)| \le L_F \|\boldsymbol{u}_1 - \boldsymbol{u}_2\|, \tag{13}$$

$$|G_s^\rho(\boldsymbol{u}_1) - G_s^\rho(\boldsymbol{u}_2)| \le L_G \|\boldsymbol{u}_1 - \boldsymbol{u}_2\|. \tag{14}$$

E.4 Proof of Text-to-Latent Behavioral Transfer

We now bound the difference between the memory-induced and text-induced effects at a latent reasoning state. By adding and subtracting the corresponding effects at the text-derived state,

$$\begin{aligned} & |G_s^\rho(\boldsymbol{u}_{\text{lat}}) - F_s^\rho(\boldsymbol{u}_{\text{lat}})| \\ \leq\ & |G_s^\rho(\boldsymbol{u}_{\text{lat}}) - G_s^\rho(\boldsymbol{u}_{\text{text}})| \\ & + |G_s^\rho(\boldsymbol{u}_{\text{text}}) - F_s^\rho(\boldsymbol{u}_{\text{text}})| \\ & + |F_s^\rho(\boldsymbol{u}_{\text{text}}) - F_s^\rho(\boldsymbol{u}_{\text{lat}})|\,. \end{aligned} \tag{15}$$

Using Equations 12, 13, and 14, we obtain

$$|G_s^\rho(\boldsymbol{u}_{\text{lat}}) - F_s^\rho(\boldsymbol{u}_{\text{lat}})| \leq \delta + (L_F + L_G)\|\boldsymbol{u}_{\text{lat}} - \boldsymbol{u}_{\text{text}}\|. \tag{16}$$

Finally, substituting the hidden-state alignment bound from Equation 10 yields

$$\boxed{|G_s^\rho(\boldsymbol{u}_{\text{lat}}) - F_s^\rho(\boldsymbol{u}_{\text{lat}})| \leq \delta + (L_F + L_G)L_\Phi \epsilon_{\text{align}}}. \tag{17}$$

This establishes Proposition 1.

Interpretation. The bound contains two terms with different meanings. The first, δ, measures how accurately behavioral distillation reproduces the textual skill effect on the states observed during training. The second, $(L_F + L_G)L_\Phi \epsilon_{\text{align}}$, captures the additional error introduced when the learned skill is applied to latent reasoning states. LatentMAS directly targets the input component of this term through its input-output alignment operator, while the frozen backbone maps both text-derived and aligned latent inputs into the same intermediate hidden-state interface.

The analysis therefore does not require textual and latent reasoning states to be identical. It only states that behavioral transfer remains controlled when the aligned latent trajectory stays within a local region where the frozen backbone and the two skill-effect functions vary smoothly.

F Directly Discover Latent Vector as Skills

The pipeline described in the main text discovers skills in two stages: the backbone model compares calibration trajectories and produces a natural-language skill description, after which behavioral distillation compiles that description into the memory vector $\mathbf{Z}_s$ used at inference. A natural question is whether both stages can be skipped, discovering Z directly from the base MAS model's own hidden states, with zero gradient updates. We explored two variants of this idea.

Contrastive extraction. For each calibration question we sample 8 attempts under temperature decoding with the TextMAS carrier and keep any pair where at least one attempt reached the correct final answer and at least one did not (MedQA and GPQA-D yielded 13 such pairs out of 20 calibration questions each; GSM8K yielded only 5, since its baseline accuracy is high enough that few sampled attempts fail). For each pair we render the two trajectories to text and extract a representation from *each* side using the same mechanism already used to initialize Z from skill text: hidden states from a fixed decoder layer are pooled by attention against $M{=}8$ fixed random queries and projected to $d_{\text{mem}}{=}256$ by a fixed random matrix. The skill vector for a pair is the difference $Z_{\text{pair}} = Z_{\text{success}} - Z_{\text{failure}}$, and the final Z_{skill} is the mean of Z_{pair} over all valid pairs. The raw extracted norm is roughly two orders of magnitude smaller than a gradient-trained Z (e.g. $\sim$1–11 vs. $\sim$335–360), so we rescale Z_{skill} to match the mean norm of an already-trained checkpoint's skills before evaluating it through that checkpoint's (already-trained, skill-agnostic) cross-attention reader.

Latent generation. Rather than hand-extracting a direction, this variant lets the base model generate one. We concatenate several (success, failure) pairs into a single reflection prompt instructing the model to identify what separates the correct attempts from the incorrect ones, then run $M{=}8$ steps of the model's own continuous latent reasoning – the identical mechanism the Planner/Critic/Refiner agents already use for their internal thinking, with no text ever decoded. The resulting M "thought" embeddings are projected to d_{mem} with the same fixed random map and rescaled identically to the extraction variant.

Results. Both variants require zero gradient training and reuse an already-trained, shared cross-attention reader; only Z itself is produced without training. Table 9 reports the 3-seed accuracy delta against the no-skill LatentMAS baseline on Qwen3-8B, forcing the discovered skill on every held-out example (no routing).

Benchmark	Contrastive extraction Δ	Latent generation Δ
MedQA	$+0.00$ pp	-1.57 pp
GSM8K	$+0.27$ pp	-0.13 pp
GPQA-D	-32.53 pp	-4.96 pp

Table 9: 3-seed accuracy delta vs. no-skill LatentMAS for the two zero-training, latent-space skill discovery variants (Qwen3-8B).

Contrastive extraction matches or marginally beats the no-skill baseline on MedQA and GSM8K, but on GPQA-D the extracted direction destabilizes generation outright: outputs collapse into degenerate repetition (e.g. an unbounded repeated fragment) rather than producing a final answer, consistently across all three seeds. Inspecting the pairs' internal consistency does not explain this in advance – GSM8K's pairs are the most mutually consistent of the three benchmarks yet produce only a noise-level effect, while MedQA and GPQA-D have similar consistency but opposite outcomes. Latent generation never collapses on any benchmark (verified by inspecting raw predictions before committing to a full run) but also never exceeds the no-skill baseline.

Neither variant is competitive with the text-skill-plus-distillation pipeline used elsewhere in this work. The comparison is nonetheless informative: it indicates that the discovery and distillation stages are not easily bypassed by directly manipulating hidden states with an off-the-shelf, untrained reader – the reader's cross-attention weights are shaped by gradient training on a particular distribution of Z, and an out-of-distribution direction can be safe on some benchmarks and destructive on others, with no obvious way to predict which in advance.