MODULAR SKILL INTERNALIZATION FOR AGENTIC FOUNDATION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Augmenting large language models with reusable procedural skills via in-context prompting suffers from an inherent bottleneck in capability acquisition. Because a model must infer execution procedures entirely from textual descriptions at runtime, the efficacy of injected skills remains strictly bounded by its pre-existing instruction-following and context-comprehension abilities. Furthermore, in-context skill descriptions can only steer model behavior through conditioning on the current input, whereas the model's learned capabilities are instantiated in its parameters. This creates a fundamental mechanistic mismatch: textual skill injection asks the model to reconstruct a new procedure through transient, context-dependent computation, rather than incorporating that procedure into the parameterized transformations that constitute its learned capabilities. To address this limitation, we introduce Modular Skill Internalization (MSI), a framework that compiles declarative skill specifications into modular, lightweight parameter updates. For each skill, MSI trains an independent parameter module from synthetic skill-specific demonstrations, such that attaching the module at inference time equips the base model with the corresponding procedural capability without requiring the original skill description. These modules are learned independently with behavioral anchors to encourage preservation of the backbone's existing behavior, enabling skills to be acquired, stored, and invoked as modular parameterized capabilities. Extensive evaluations across four agentic tasks and four foundation backbones demonstrate that MSI consistently outperforms in-context prompting in task performance[1].

1 INTRODUCTION

Reusable procedural skills provide a lightweight and flexible means of extending large language models (LLMs) with task-specific capabilities, ranging from complex mathematical reasoning to multi-step tool invocation (Wang et al., 2024; 2025; Zhang et al., 2025; Su et al., 2026b). Typically represented as natural-language specifications, these skills offer a convenient interface for describing, inspecting, sharing, and retrieving procedural knowledge (Zhang et al., 2025; Li et al., 2026; Su et al., 2026b). The prevailing paradigm is to inject the relevant skill description into the model's context at inference time, making new capability available without modifying the model's parameters (Brown et al., 2020; Lewis et al., 2020; Su et al., 2026b). While flexible and non-invasive, this paradigm leaves the procedure external to the model: the skill is supplied as textual guidance rather than internalized as a capability encoded in the model's own parameters.

This distinction creates an inherent bottleneck in capability acquisition: *the effectiveness of in-context skill injection is fundamentally limited by the host model's pre-existing instruction-following ability.* A textual skill can influence model behavior only insofar as the model can correctly interpret its specification and translate the prescribed procedure into execution. Consequently, when the base model has limited instruction-following or context-comprehension capability, even a well-specified external skill may fail to translate into reliable task performance. This makes in-context injection a particularly fragile mechanism for capability extension: the acquisition of a new external capability depends on capabilities that the model already possesses (Wei et al., 2022; Ouyang et al., 2022; Zhou et al., 2023).

[1] We have open-sourced all the code and data at: `https://anonymous.4open.science/r/MSI`

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.

More fundamentally, in-context skill injection creates a mismatch between how the model uses external skills and how it uses its internal capabilities. Internal capabilities are encoded in the model's parameters and can be directly exercised during generation. In contrast, an external skill provided in context remains textual information that the model must interpret and apply through context conditioning. As a result, the model cannot use an injected skill in the same way that it uses a capability already encoded in its parameters. This motivates a different approach: can we internalize external skills into lightweight model parameters, so that they can be invoked as model capabilities rather than interpreted as contextual instructions?

We study this problem as *skill internalization*: compiling external skills into modular parameter updates that directly augment the executable capabilities of a frozen foundation model. Such a formulation preserves the modularity of external skill libraries while moving procedural acquisition from online interpretation to offline learning. Once internalized, a skill can be invoked without repeatedly supplying its textual specification, reducing dependence on the LLM's inherent instruction-following capabilities while also avoiding the context and computation overhead introduced by long skill prompts (Jiang et al., 2023; Mu et al., 2023).

Building on this formulation, we introduce **Modular Skill Internalization (MSI)**. Given a declarative skill specification, MSI first uses a teacher model to synthesize diverse application tasks related to the selected skill (Wang et al., 2023) and generate the corresponding solution trajectories, including grounded tool interactions when required (Schick et al., 2023; Yao et al., 2023). After quality filtering, these demonstrations are used to train an independent parameter-efficient adapter (e.g., LoRA) on top of a frozen student backbone (Houlsby et al., 2019; Hu et al., 2022). Importantly, the student learns from the task–solution pairs without access to the original skill specification (Snell et al., 2022; Shin et al., 2025). Successful execution must therefore be acquired through the learned parameter update rather than reconstructed from the skill description at inference time. At deployment time, the resulting adapter can be activated on demand (Sheng et al., 2024; Zhao et al., 2024), allowing the corresponding external capabilities to be injected without repeatedly placing its textual specification in the context.

Beyond improving the current model, this formulation also provides a natural bridge across model generations. For a deployed model, each skill adapter serves as an immediate *capability patch*: a lightweight module that adds a procedural capability without modifying the shared backbone. At the same time, the skill-guided demonstrations used to train that adapter constitute reusable behavioral supervision. When developing a subsequent foundation model, these demonstrations can be incorporated into its post-training mixture (Ouyang et al., 2022; Lambert et al., 2024), allowing capabilities that previously required external adapters to be learned directly by the next-generation model. MSI therefore supports a continuous lifecycle in which textual specifications are converted into deployable capability modules, while the experience accumulated in constructing those modules can contribute to the native capabilities of future models. We evaluate MSI on 408 skills spanning four benchmarks (Su et al., 2026b) across four foundation-model backbones. Experimental results show that MSI consistently outperforms in-context skill injection in task performance. To summarize, our primary contributions are as follows:

- We identify the mismatch between in-context skill injection and the LLM's parametric capabilities, formalizing the paradigm of skill internalization to compile external skills into modular parameter updates.
- We propose Modular Skill Internalization, which compiles external textual skills into independent, plug-and-play adapters by training on synthesized task trajectories while using behavioral anchors to encourage preservation of the backbone's existing behavior.
- Extensive evaluations on 408 skills across four agentic benchmarks and four foundation backbones demonstrate that MSI consistently outperforms in-context skill injection, validating the effectiveness of parameterizing procedural knowledge.

2 Related Work

Skill-Augmented Agents. Skills provide reusable procedures that guide language agents through tasks at inference time. Such procedures can emerge from experience: Voyager stores executable programs (Wang et al., 2024), Agent Workflow Memory induces workflows from past trajecto-

ries (Wang et al., 2025), and SkillWeaver builds reusable APIs through practice (Zheng et al., 2025). Alternatively, people can write skill documents, as in the Agent Skills format, which organizes instructions, scripts, and resources around a `SKILL.md` file (Zhang et al., 2025). To make these resources available at scale, Skill Retrieval Augmentation (Su et al., 2026b) studies how agents retrieve and apply skills from a large corpus. Across these systems, skill instructions, code, or API descriptions enter the context to guide execution. Using these skills depends on the model's instruction-following ability: Su et al. (2026b) show that retrieving the right skill does not ensure correct application. Skill text also takes up context space and can delay the first output token. In contrast, MSI moves skill guidance from the inference context into model parameters.

Parametric Knowledge Injection. Two lines of work reduce the inference cost of long textual guidance. Prompt compression shortens the input by pruning text, as in LLMLingua (Jiang et al., 2023), or encoding it into a few learned tokens (Mu et al., 2023). The model still needs to read and follow the compressed guidance. Parametric knowledge injection instead stores the guidance in model parameters, so answers can be generated without adding the original text to the prompt. For example, context distillation trains a model without the extra guidance to reproduce outputs generated with it (Snell et al., 2022). Generative Prompt Internalization extends this approach to complex agent prompts (Shin et al., 2025), and SIEVE structures the supervision by pairing synthetic queries with relevant parts of the source text (Asawa et al., 2026). For document knowledge, Parametric RAG (PRAG) (Su et al., 2025) augments documents with rewrites and question-answer pairs, then trains a separate LoRA module (Hu et al., 2022) for each document. At inference, it loads and combines the modules of retrieved documents. Building on these document-module pairs, DyPRAG learns to generate parameters from documents (Tan et al., 2025). ReGrad instead stores document gradients and retrieves them for temporary parameter updates (Su et al., 2026a). Adapters can also be generated by hypernetworks from context, as in Doc-to-LoRA (Charakorn et al., 2026), or task descriptions, as in Text-to-LoRA (Charakorn et al., 2025). Like PRAG, MSI converts skill text into parameter modules and loads selected modules on demand. Because skills describe procedures rather than facts or a fixed prompt, it trains each module on new tasks that require the skill, using filtered solution trajectories generated under the skill's guidance.

Skill Internalization. Existing approaches internalize skills either into shared model parameters or into modular parameter components. SKILL0 gradually removes skill context during reinforcement learning (Lu et al., 2026), while Skill-SD exposes trajectory-derived skills only to the teacher during on-policy self-distillation (Wang et al., 2026). Both ultimately consolidate multiple skills into a shared policy that no longer requires skill text at inference time. A separate line of work studies modular adaptation, where independently trained adapters are dynamically selected according to the input, as in LoraRetriever (Zhao et al., 2024) and Arrow (Ostapenko et al., 2024). MSI combines skill internalization with this modular formulation: each skill is compiled into an independent capability module, and a learned router selects the appropriate module directly from the query at inference time.

3 METHODOLOGY

Modular Skill Internalization (MSI) converts a textual skill into a learned procedure that can be invoked through model parameters, as illustrated in Figure 1. The framework has two stages: constructing demonstrations that show how to execute the skill, and using those demonstrations to train an independent adapter. All data synthesis and parameter optimization take place offline.

3.1 SKILL INTERNALIZATION

Let $\mathcal{S}$ be a library of skills, where each skill s has a textual specification w_s describing a reusable procedure. Given a language model with parameters θ, textual skill prompting conditions its response on both a task q and the specification, yielding $p_\theta(y \mid q, w_s)$. Our goal is to learn skill parameters $\phi_{\theta,s}$ that support execution through $p_{\theta,\phi_{\theta,s}}(y \mid q)$, without requiring w_s as an inference-time input. Here, q includes the task prompt and any necessary tool interfaces; only the skill's procedural description is removed.

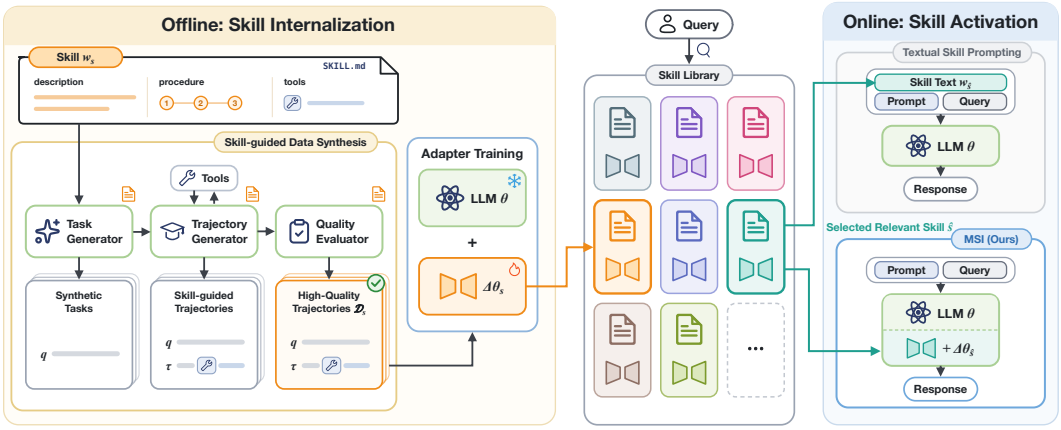


Figure 1: Overview of Modular Skill Internalization (MSI). Offline, MSI generates tasks and execution trajectories from each textual skill, filters the demonstrations, and trains an independent LoRA adapter with the backbone frozen. Online, the selected skill is invoked by activating its adapter, without adding the skill description to the input.

For each skill, we first construct a demonstration set $\mathcal{D}_s^+ = \{(q_i, \tau_i^+)\}_i$, where τ_i^+ records a solution to a task requiring the skill. We then freeze θ and train $\phi_{\theta,s}$ on these demonstrations, supplemented with examples that encourage preservation of the base model's behavior. Adapters are trained independently for each skill and backbone. This preserves an explicit correspondence between a skill specification, its demonstrations, and the parameters used to invoke it.

3.2 From Skill Specifications to Execution Demonstrations

A skill specification describes a general procedure, whereas parameter training requires examples of that procedure in use. We bridge this gap through task generation, teacher execution, and quality filtering.

Generating application tasks. Conditioned on w_s, a task generator produces questions that require applying the skill while varying their contexts, inputs, and difficulty. This variation exposes the student to multiple instantiations of the same procedure. For tasks involving external tools, we provide real data records, tool documentation, and executed examples so that generated questions can be answered from the available environment. Questions undergo format checks and within-skill deduplication. A separate overlap check excludes duplicates and near-duplicates of benchmark test questions, which are never supplied to the generator.

Demonstrating execution. A teacher solves each generated question with access to w_s. The teacher is instructed to make the reasoning self-contained: it must spell out the required steps and intermediate results without referring to the skill document. For interactive tasks, tool calls are executed in the environment and their returned observations inform subsequent decisions. We record the resulting reasoning, tool interactions, and final answer as a trajectory, using the task prompts and interaction protocols of SR-Agents (Su et al., 2026b). The teacher's skill context supplies guidance during data construction; the trajectory makes the resulting behavior available as supervision for a student that will not receive that context.

Filtering demonstrations. A language-model evaluator checks whether each trajectory follows an appropriate method, contains complete and plausible reasoning, and produces a usable answer in the required format. It also checks that the solution can be understood without the skill document. Programmatic checks reject truncated responses, missing answers, and trajectories that cannot be converted into training samples. The accepted tasks and trajectories form $\mathcal{D}_s^+$. Generation difficulty settings and filtering criteria are detailed in Appendix A.1.

3.3 Learning Skills in Parameter Space

Independent skill adapters. We implement $\phi_{\theta,s}$ using LoRA (Hu et al., 2022). For a selected weight matrix $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, the skill-specific transformation is

$$W_s = W + \frac{\alpha}{r} B_s A_s, \tag{1}$$

where $A_s \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B_s \in \mathbb{R}^{d_{\text{out}} \times r}$ are trainable, r is the rank, and α controls the update scale. The collection of these low-rank matrices constitutes $\phi_{\theta,s}$; the original weights remain fixed. We apply the updates to selected attention and FFN projections, allowing supervision to change both how information is combined and how hidden representations are transformed. Each skill has its own adapter, so training a new skill leaves the backbone and previously trained adapters unchanged.

Supervising execution without skill text. We split demonstrations into training and validation sets by question before expanding multi-turn trajectories into response-level samples. Each sample (x_i, y_i) consists of the task and preceding interaction as input x_i, and the current assistant response as target y_i. The skill description w_s and teacher-specific quality instructions are excluded from student inputs. The student is therefore trained to generate the demonstrated reasoning, tool calls, and answers from the task context itself. For multi-turn tasks, previous responses and actual tool observations remain available as interaction history.

Preserving existing behavior. Learning a skill should not require changing behavior indiscriminately on other tasks. We supplement the positive demonstrations with a behavioral anchor set $\mathcal{A}_{\theta,s}$, built from synthetic questions whose recorded source skills exclude s. For each such question, the unadapted student generates a response without skill text; this response becomes the preservation target. We include both questions resembling the target skill's tasks and a broader random sample. These source labels provide a practical proxy for nonmatching tasks, and the responses serve as behavioral references rather than correctness-certified answers. The behavioral anchors act as an auxiliary training constraint; their construction is specified in Appendix A.2.

Training objective. Positive response samples and behavioral anchors are mixed during training. For a microbatch $\mathcal{B}$, we minimize the mean negative log-likelihood over supervised response tokens:

$$\mathcal{L}(\phi_{\theta,s}; \mathcal{B}) = -\frac{1}{\sum_{i \in \mathcal{B}} |y_i|} \sum_{i \in \mathcal{B}} \sum_{t=1}^{|y_i|} \log p_{\theta,\phi_{\theta,s}}(y_{i,t} \mid x_i, y_{i,<t}). \tag{2}$$

Only the current assistant response contributes to the loss. Task prompts, earlier responses, and tool observations are masked. Demonstrations and behavioral anchors use the same objective, with their relative contributions determined by sampling and supervised token counts. We select each adapter using generated answers on held-out positive tasks; behavioral anchors and benchmark test instances do not participate in checkpoint selection. Optimization and selection settings appear in Appendix A.3. Additional training and inference conditions that retain skill text are described in Appendix A.4.

3.4 Invoking and Reusing Internalized Skills

Given a task and an assigned skill identifier s, we load $\phi_{\theta,s}$ and generate with $p_{\theta,\phi_{\theta,s}}$. One adapter remains active throughout the task, including any tool interactions, and no parameters are updated online. The task prompt, tool interfaces, and interaction history are retained, while the skill description is omitted. Skill assignment is an experimental setting separate from internalization; the protocols used in our evaluation are documented in Appendix A.5.

The framework produces two reusable artifacts for each skill: an adapter for the current backbone and a set of self-contained execution demonstrations. The former provides an immediately deployable capability patch. The latter can be reformatted for a successor model and included in its post-training data, providing supervision for learning the procedure directly in that model's parameters. This offers a route from modular capability updates within a model generation to native skill acquisition in the next.

4 EXPERIMENTAL SETUP

Benchmarks and Skill Routing. We evaluate MSI on the four datasets of SRA-Bench (Su et al., 2026b): TheoremQA (Chen et al., 2023), LogicBench (Parmar et al., 2024), ToolQA (Zhuang et al., 2023), and MedCalc-Bench (Khandekar et al., 2024), covering theorem application, logical reasoning, tool use, and medical calculation. Together, they comprise 4,037 instances associated with 408 gold skills. We use these skills as a fixed library and report two routing conditions: *Oracle*, which supplies the annotated gold skill, and *Retrieved*, which selects the top-ranked skill from the joint library. Dataset statistics and implementation details appear in Appendix B.1.

Baselines. We compare against LLM Direct, full-text skill injection (Su et al., 2026b), and Injection with demo, which additionally supplies two synthetic demonstrations (Brown et al., 2020). Baseline implementations are detailed in Appendix B.2.

Implementation Details. We use Qwen3.5-2B (Qwen Team, 2026), Llama-3.2-3B-Instruct (Grattafiori et al., 2024; Meta AI, 2024), Phi-3.5-mini-instruct (Abdin et al., 2024), and Apertus-8B-Instruct-2509 (Project Apertus et al., 2025) as backbones. Qwen3.5-122B-A10B (Qwen Team, 2026) generates and evaluates synthetic demonstrations. The main setting uses 200 positive training instances per skill, nominally 30 near-miss and 30 random behavioral anchors, and 80 held-out positive instances for validation. We train rank-8 LoRA adapters (Hu et al., 2022) for eight epochs with AdamW (Loshchilov & Hutter, 2019), a learning rate of 10^{-4}, and an effective batch size of eight response-level samples, selecting checkpoints using held-out positive accuracy. For retrieval, we fine-tune BGE-M3 (Chen et al., 2024) on synthetic question–skill pairs and freeze the selected checkpoint (detailed in B.3). Further settings are provided in Appendix B.3.

Evaluation Protocol and Metrics. We follow SR-Agents' (Su et al., 2026b) task prompts, tool interfaces, answer extraction, and dataset-specific rule-based scoring, using vLLM (Kwon et al., 2023) with temperature-zero decoding. We report per-subset accuracy and instance-weighted micro accuracy over all 4,037 instances; empty or unparseable answers count as incorrect. Retrieval is additionally evaluated using Recall@1 and Recall@10, with results reported in Appendix C, Table 7. ToolQA follows the benchmark's ReAct protocol (Yao et al., 2023). Output limits and metric definitions appear in B.3 and B.4.

5 EXPERIMENTAL RESULTS

5.1 MAIN RESULTS

Table 1 reports results under oracle skill assignment, where each query is paired with its gold skill. This setting removes skill retrieval as a confounding factor and focuses the comparison on how the assigned skill is represented and applied. Our main findings are as follows.

First, textual skill injection generally improves performance over direct generation, confirming that the provided skill specifications contain useful procedural guidance. However, the gains are not uniform across tasks, and supplementing the skill text with demonstrations yields only inconsistent additional benefits across backbones. These results suggest that providing the correct procedure in context is helpful, but its effectiveness remains constrained by the model's instruction-following ability to interpret and execute that procedure at inference time.

Second, MSI provides a stronger and more consistent alternative by internalizing the assigned skill into model parameters. Without requiring either skill descriptions or demonstrations at inference time, MSI achieves higher overall accuracy than both in-context skill baselines across all backbones, with particularly consistent improvements on LogicBench and ToolQA. This advantage indicates that skill-guided parameter adaptation can translate procedural knowledge into executable behavior more reliably than inference-time textual conditioning alone.

Finally, parametric internalization and in-context skill injection are complementary. Across all configurations in Table 1, MSI + Skill Text achieves the highest accuracy on every backbone–dataset pair. This indicates that explicit skill descriptions provide orthogonal benefits on top of parameter adaptation, while the adapted model itself outperforms purely textual conditioning (with or without demonstrations). Consequently, these findings suggest two practical deployment strategies: deploy-

Table 1: End-task accuracy (%) under oracle skill assignment. MSI + Skill Text uses adapters trained and evaluated with skill text. Micro avg. is instance-weighted over 4,037 instances. The best and second-best results among the methods shown are **bold** and underlined within each backbone and column.

Model	Method	TheoremQA	LogicBench	ToolQA	MedCalc	Micro avg.
Qwen3.5-2B	LLM Direct	53.0	68.0	9.7	41.3	37.3
	Oracle Skill	68.0	80.5	28.3	71.3	57.2
	Injection with demo	62.4	81.4	35.5	72.8	59.3
	MSI	61.4	82.1	45.9	69.9	62.1
	MSI + Skill Text	**69.6**	**83.8**	**49.0**	**79.2**	**67.6**
Llama-3.2-3B	LLM Direct	19.1	61.2	3.1	11.0	19.2
	Oracle Skill	37.6	55.0	6.9	53.0	34.2
	Injection with demo	36.1	55.3	1.6	57.4	33.3
	MSI	43.0	74.6	40.4	57.9	52.1
	MSI + Skill Text	**47.5**	**77.0**	**43.3**	**68.4**	**57.2**
Phi-3.5-mini	LLM Direct	24.6	52.6	15.1	17.0	24.4
	Oracle Skill	40.0	60.0	28.3	28.7	36.5
	Injection with demo	32.0	59.6	34.2	42.1	40.7
	MSI	34.9	63.0	40.1	45.8	45.0
	MSI + Skill Text	**42.0**	**71.8**	**43.8**	**58.0**	**52.6**
Apertus-8B	LLM Direct	21.7	63.6	14.5	15.1	25.2
	Oracle Skill	39.2	59.9	20.9	58.3	41.8
	Injection with demo	33.9	66.3	19.0	55.5	40.6
	MSI	35.6	72.2	41.0	53.8	49.4
	MSI + Skill Text	**42.6**	**75.3**	**45.7**	**62.1**	**55.1**

ing adapters alone to eliminate prompt overhead, or pairing them with skill text to achieve peak performance.

5.2 Skill-Specific Capability Internalization

A central question is whether MSI internalizes the capability described by each skill, or merely induces a generic improvement in behaviors shared across skills, such as structured reasoning, tool use, or trajectory following. To distinguish these explanations, Table 2 compares the matched skill adapter against a randomly selected non-gold adapter from the same dataset and the unadapted backbone. Experimental results demonstrate that the gains depend critically on activating the relevant skill adapter. Across every backbone–dataset pair, the matched adapter substantially outperforms both the random-adapter control and direct generation. In contrast, activating a random adapter provides limited benefit: its effect fluctuates around the unadapted backbone, improving some subsets while degrading others, and its aggregate performance remains far below that of matched MSI.

These results indicate that MSI does not simply teach a generic execution pattern that transfers indiscriminately across adapters. Instead, the learned parameter update is tied to the capability specified by its source skill. This provides evidence that MSI performs *skill-specific capability internalization*, rather than merely learning generic reasoning or tool-use behavior.

5.3 Skill Internalization beyond Oracle Routing

We next examine whether the benefits of skill internalization extend to learned skill selection. Table 3 reports results with a BGE-M3 retriever fine-tuned on synthetic question–skill pairs and frozen at evaluation (Appendix A.5). All backbones and skill-based methods share the same top-1 retrieved skills, without reranking. Full-Skill Injection supplies the retrieved specification in context, while MSI activates the corresponding adapter without skill text. We have the following findings.

First, the advantage of skill internalization persists beyond oracle assignment. Without skill text at inference time, MSI outperforms Full-Skill Injection in micro accuracy for every backbone. More-

Table 2: Adapter-selection ablation: accuracy (%) with no skill text. MSI activates the gold adapter; Random Skill Adapter selects a non-gold adapter from the same dataset, retaining the gold tool interfaces and averaging over three fixed routing seeds. LLM Direct uses no adapter. Micro avg. weights all benchmark instances equally. Bold marks the highest value within each backbone and column, without implying statistical significance.

Model	Method	TheoremQA	LogicBench	ToolQA	MedCalc	Micro avg.
Qwen3.5-2B	LLM Direct	53.0	68.0	9.7	41.3	37.3
	Random Skill Adapter	48.9	71.8	15.9	34.3	37.5
	MSI	**61.4**	**82.1**	**45.9**	**69.9**	**62.1**
Llama-3.2-3B	LLM Direct	19.1	61.2	3.1	11.0	19.2
	Random Skill Adapter	20.1	58.9	12.1	11.8	22.3
	MSI	**43.0**	**74.6**	**40.4**	**57.9**	**52.1**
Phi-3.5-mini	LLM Direct	24.6	52.6	15.1	17.0	24.4
	Random Skill Adapter	26.5	57.1	13.1	20.0	25.8
	MSI	**34.9**	**63.0**	**40.1**	**45.8**	**45.0**
Apertus-8B	LLM Direct	21.7	63.6	14.5	15.1	25.2
	Random Skill Adapter	18.9	59.8	12.1	11.5	22.2
	MSI	**35.6**	**72.2**	**41.0**	**53.8**	**49.4**

Table 3: End-task accuracy (%) under retrieved skill assignment. MSI + Skill Text uses adapters trained and evaluated with skill text. Micro avg. is instance-weighted over 4,037 instances. The best and second-best results among the methods shown are **bold** and underlined within each backbone and column.

Model	Method	TheoremQA	LogicBench	ToolQA	MedCalc	Micro avg.
Qwen3.5-2B	LLM Direct	53.0	68.0	9.7	41.3	37.3
	Full-Skill Injection	63.5	75.1	26.7	70.5	54.5
	MSI	60.4	**80.4**	42.4	65.8	59.3
	MSI + Skill Text	**66.0**	77.0	**45.4**	**75.2**	**63.3**
Llama-3.2-3B	LLM Direct	19.1	61.2	3.1	11.0	19.2
	Full-Skill Injection	34.8	58.6	6.4	51.2	33.7
	MSI	39.2	**69.7**	36.8	55.6	48.6
	MSI + Skill Text	**42.7**	66.1	**39.4**	**59.9**	**50.6**
Phi-3.5-mini	LLM Direct	24.6	52.6	15.1	17.0	24.4
	Full-Skill Injection	37.1	58.9	26.2	27.3	34.7
	MSI	37.1	61.2	37.3	42.9	43.3
	MSI + Skill Text	**40.6**	**69.7**	**41.0**	**54.6**	**50.0**
Apertus-8B	LLM Direct	21.7	63.6	14.5	15.1	25.2
	Full-Skill Injection	35.3	60.3	20.3	**55.8**	40.3
	MSI	32.8	**67.5**	37.4	52.6	46.4
	MSI + Skill Text	**39.1**	65.1	**42.0**	54.5	**49.2**

over, retrieved MSI exceeds oracle skill injection in micro accuracy across all backbones (Table 1). This comparison highlights the value of learning to apply skills beyond access to the correct specification alone.

Second, the complementarity of skill text becomes less uniform under retrieval. MSI + Skill Text, trained and evaluated with skill text, achieves the highest micro accuracy among the compared methods for every backbone. However, unlike under oracle assignment, this configuration does not consistently improve over adapter-only MSI: it performs worse on LogicBench for Qwen, Llama, and Apertus. Relative to oracle assignment, both MSI variants lose micro accuracy, with a larger decline for the text-conditioned variant on every backbone, narrowing its aggregate advantage over adapter-only invocation. These results qualify the complementarity observed in the main experiment: textual guidance remains useful overall, but its benefits depend on the task and skill-selection setting.

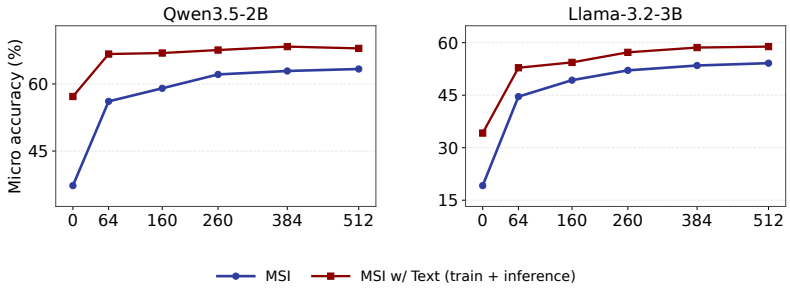


Figure 2: Effect of the nominal per-skill training quota on micro accuracy under oracle routing. The quota includes both positive demonstrations and behavioral anchors. MSI omits skill text during training and inference; the text-conditioned curve supplies it during both, corresponding to MSI + Skill Text in Table 1. At quota 0, the two curves show the unadapted backbone without skill text (LLM Direct) and with oracle skill text (Oracle Skill), respectively.

5.4 Data Requirements for Skill Internalization

Figure 2 examines how performance changes with the nominal per-skill training quota, including both skill demonstrations and behavior-preservation anchors. All evaluations use oracle routing, separating this analysis from skill-retrieval errors. **Additional training data yields diminishing returns for the default MSI**: performance improves most rapidly at smaller budgets, while subsequent increases produce progressively smaller gains on both backbones. This pattern suggests that expanding a small practice set is particularly useful for skill acquisition, whereas increasing an already substantial budget is less productive under the evaluated synthesis and training pipeline.

The text-conditioned variant shows that this trend depends on how the skill is supplied. Training and evaluating with skill text gives the strongest performance throughout the tested budgets and also shows diminishing gains toward the upper end. For Qwen, the text-conditioned curve largely plateaus, with a slight decline at the largest quota; for Llama, this condition shows a small further gain at the largest quota. The curves thus support diminishing returns for the default internalization setting while indicating that the useful training budget varies with the backbone and text condition.

6 Conclusion

In this work, we study *skill internalization* as an alternative to injecting procedural skills through in-context prompting. We argue that textual skill injection is fundamentally constrained by the host model's ability to interpret and execute contextual instructions, while also creating a mismatch between external skills represented as text and internal capabilities encoded in model parameters. To bridge this gap, we introduce **Modular Skill Internalization (MSI)**, which compiles declarative skill specifications into independent, lightweight parameter adapters through synthesized skill-specific demonstrations, while using behavioral anchors to encourage preservation of the backbone's existing behavior. Experiments across diverse agentic tasks and foundation models show that MSI outperforms in-context skill injection. These results demonstrate the effectiveness of internalizing external procedures as modular parameterized capabilities, offering a practical alternative to repeatedly supplying skills as contextual instructions.

AI Use Statement

Generative AI tools were used to assist with editing and improving the clarity and readability of the manuscript, as well as with analyzing and interpreting experimental results during the research process. All AI-assisted outputs, including textual revisions and research-related analyses, were carefully reviewed and verified by the authors. The authors take full responsibility for the final content of this work, including all claims, analyses, and conclusions.

Reproducibility Statement

We ensure the reproducibility of this work by releasing all relevant code, models, and data in our anonymous repository[2]. This anonymous repository also provides detailed instructions for reproducing every result reported in the paper.

References

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. *arXiv preprint arXiv:2404.14219*, 2024. URL https://arxiv.org/abs/2404.14219.

Parth Asawa, Alexandros G. Dimakis, and Matei Zaharia. SIEVE: Sample-Efficient Parametric Learning from Natural Language. *arXiv preprint arXiv:2604.02339*, 2026. URL https://arxiv.org/abs/2604.02339.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

Rujikorn Charakorn, Edoardo Cetin, Yujin Tang, and Robert Tjarko Lange. Text-to-LoRA: Instant Transformer Adaption. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 7485–7514. PMLR, 2025. URL https://proceedings.mlr.press/v267/charakorn25a.html.

Rujikorn Charakorn, Edoardo Cetin, Shinnosuke Uesaka, and Robert Tjarko Lange. Doc-to-LoRA: Learning to Instantly Internalize Contexts. *arXiv preprint arXiv:2602.15902*, 2026. URL https://arxiv.org/abs/2602.15902.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-Embedding: Multi-Linguality, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 2318–2335. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.137.

Wenhu Chen, Ming Yin, Max Ku, Pan Lu, Yixin Wan, Xueguang Ma, Jianyu Xu, Xinyi Wang, and Tony Xia. TheoremQA: A Theorem-driven Question Answering Dataset. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7889–7901. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.489.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-Efficient Transfer Learning for NLP. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2790–2799. PMLR, 2019.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. In *Proceedings of the*

[2] https://anonymous.4open.science/r/MSI

2023 Conference on Empirical Methods in Natural Language Processing, pp. 13358–13376. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.825. URL https://aclanthology.org/2023.emnlp-main.825/.

Nikhil Khandekar, Qiao Jin, Guangzhi Xiong, Soren Dunn, Serina S. Applebaum, Zain Anwar, Maame Sarfo-Gyamfi, Conrad W. Safranek, Abid A. Anwar, Andrew Zhang, Aidan Gilson, Maxwell B. Singer, Amisha Dave, Andrew Taylor, Aidong Zhang, Qingyu Chen, and Zhiyong Lu. MedCalc-Bench: Evaluating Large Language Models for Medical Calculations. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 37, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing Frontiers in Open Language Model Post-Training. *arXiv preprint arXiv:2411.15124*, 2024. URL https://arxiv.org/abs/2411.15124.

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

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, et al. SkillsBench: Benchmarking How Well Agent Skills Work Across Diverse Tasks. *arXiv preprint arXiv:2602.12670*, 2026. URL https://arxiv.org/abs/2602.12670.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

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. URL https://arxiv.org/abs/2604.02268.

Meta AI. Llama-3.2-3B-Instruct Model Card. Hugging Face, 2024. URL https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to Compress Prompts with Gist Tokens. In *Advances in Neural Information Processing Systems*, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/3d77c6dcc7f143aa2154e7f4d5e22d68-Abstract.html.

Oleksiy Ostapenko, Zhan Su, Edoardo Ponti, Laurent Charlin, Nicolas Le Roux, Lucas Caccia, and Alessandro Sordoni. Towards Modular LLMs by Building and Reusing a Library of LoRAs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 38885–38904. PMLR, 2024. URL https://proceedings.mlr.press/v235/ostapenko24a.html.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training Language Models to Follow Instructions with Human Feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744, 2022.

Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, Santosh Mashetty, Arindam Mitra, and Chitta Baral. LogicBench: Towards Systematic Evaluation of Logical Reasoning Ability of Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13679–13707. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.739.

Project Apertus, Alejandro Hernández-Cano, Alexander Hägele, Allen Hao Huang, Angelika Romanou, et al. Apertus: Democratizing Open and Compliant LLMs for Global Language Environments. *arXiv preprint arXiv:2509.14233*, 2025. URL https://arxiv.org/abs/2509.14233.

Qwen Team. Qwen3.5: Towards Native Multimodal Agents. Qwen Blog, 2026. URL https://qwen.ai/blog?id=qwen3.5.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. S-LoRA: Serving Thousands of Concurrent LoRA Adapters. In *Proceedings of Machine Learning and Systems*, volume 6, 2024.

Haebin Shin, Lei Ji, Yeyun Gong, Sungdong Kim, Eunbi Choi, and Minjoon Seo. Generative Prompt Internalization. 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. 7338–7363. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-long.376. URL https://aclanthology.org/2025.naacl-long.376/.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by Distilling Context. *arXiv preprint arXiv:2209.15189*, 2022. URL https://arxiv.org/abs/2209.15189.

Weihang Su, Yichen Tang, Qingyao Ai, Junxi Yan, Changyue Wang, Hongning Wang, Ziyi Ye, Yujia Zhou, and Yiqun Liu. Parametric Retrieval Augmented Generation. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2025. doi: 10.1145/3726302.3729957. URL https://doi.org/10.1145/3726302.3729957.

Weihang Su, Jiacheng Kang, Jingyan Xu, Qingyao Ai, Jianming Long, Hanwen Zhang, Bangde Du, Xinyuan Cao, Min Zhang, and Yiqun Liu. Retrievable Gradients: Continual Post-Training Without Cumulative Weight Drift. *arXiv preprint arXiv:2606.15734*, 2026a. URL https://arxiv.org/abs/2606.15734.

Weihang Su, Jianming Long, Qingyao Ai, Qiaozhi He, Yichen Tang, Changyue Wang, Yiteng Tu, Yingbo Wang, and Yiqun Liu. Skill Retrieval Augmentation for Agentic AI. *arXiv preprint arXiv:2604.24594*, 2026b. URL https://arxiv.org/abs/2604.24594.

Yuqiao Tan, Shizhu He, Huanxuan Liao, Jun Zhao, and Kang Liu. Dynamic Parametric Retrieval Augmented Generation for Test-time Knowledge Enhancement. *arXiv preprint arXiv:2503.23895*, 2025. URL https://arxiv.org/abs/2503.23895.

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.

Hao Wang, Guozhi Wang, Han Xiao, Yufeng Zhou, Yue Pan, Jichao Wang, Ke Xu, Yafei Wen, Xiaohu Ruan, Xiaoxin Chen, and Honggang Qi. Skill-SD: Skill-Conditioned Self-Distillation for Multi-turn LLM Agents. *arXiv preprint arXiv:2604.10674*, 2026. URL https://arxiv.org/abs/2604.10674.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-Instruct: Aligning Language Models with Self-Generated Instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–13508. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.acl-long.754.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent Workflow Memory. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 2025. URL https://proceedings.mlr.press/v267/wang25bx.html.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned Language Models are Zero-Shot Learners. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=gEZrGCozdqR.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping Agents for the Real World with Agent Skills. Anthropic Engineering Blog, 2025. URL https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills.

Ziyu Zhao, Leilei Gan, Guoyin Wang, Wangchunshu Zhou, Hongxia Yang, Kun Kuang, and Fei Wu. LoraRetriever: Input-Aware LoRA Retrieval and Composition for Mixed Tasks in the Wild. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 4447–4462. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.263. URL https://aclanthology.org/2024.findings-acl.263/.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. SkillWeaver: Web Agents can Self-Improve by Discovering and Honing Skills. *arXiv preprint arXiv:2504.07079*, 2025. URL https://arxiv.org/abs/2504.07079.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-Following Evaluation for Large Language Models. *arXiv preprint arXiv:2311.07911*, 2023. URL https://arxiv.org/abs/2311.07911.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. ToolQA: A Dataset for LLM Question Answering with External Tools. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, volume 36, 2023.

A Implementation Details

A.1 Demonstration Synthesis

Task-generation requests allocate approximately 30%, 40%, and 30% of the questions to easy, medium, and hard prompts, respectively. Easy prompts request a direct application with clearly available inputs. Medium prompts request two or three dependent steps, potentially with irrelevant context or atypical conditions. Hard prompts request longer dependencies, boundary conditions, implicit constraints, or combinations of subprocedures. These proportions govern generation requests rather than the realized difficulty distribution after filtering.

For tool tasks, generation is grounded in records from the available databases, graphs, or text collections, together with tool documentation and actual execution examples. Question prompts should identify the query target without disclosing the answer or explicitly naming the intended skill. The teacher executes tools through the SR-Agents task protocol (Su et al., 2026b) and uses the observations to continue its solution.

The evaluator receives the question, skill specification, and candidate trajectory. It checks methodological alignment, independence from the skill document, reasoning and calculation completeness, answer plausibility, and compliance with task-specific output and tool-use requirements. Acceptance requires satisfying all criteria under the evaluator's judgment. Programmatic checks additionally reject truncated or overlong trajectories, empty extracted answers, and examples without usable response samples. These checks provide quality control, not a formal correctness guarantee.

A.2 Behavior-Preservation Anchors

Candidate questions come from the synthetic task pool. Duplicate questions retain the union of their recorded source skills, and a question is eligible for skill s only if that union excludes s. A fixed auxiliary retriever R_0 is used to construct two groups. Near-miss questions rank s first; if this pool is insufficient, candidates ranking s in positions 2–10 provide additional examples. Random questions are sampled from candidates for which s is outside the top ten. The documented setting uses 30 near-miss and 30 random anchors per skill. This source-based selection approximates nonmatching inputs and does not establish that s is inapplicable to every selected question.

For each selected question q, the unadapted student generates $y^0_\theta(q)$ without a skill adapter or skill text. The pair $(q, y^0_\theta(q))$ provides a reference for the student's original behavior. In tool settings, anchor generation records a single model response rather than a complete environment rollout, so its supervision concerns that response. Anchors are specific to the student backbone and are added only to training. The auxiliary retriever used for anchor construction is distinct from the separately trained encoder used in the skill-selection setting below.

A.3 Adapter Training and Checkpoint Selection

For each skill, the documented data split reserves 200 accepted questions for positive training and 80 for validation. The split precedes response-level expansion, keeping all turns from one question in the same partition. The 60 anchors are then added to training. The default LoRA configuration uses rank $r = 8$, scaling $\alpha = 16$, and dropout 0.05; embeddings and the output head are not adapted. Standard attention query, key, value, and output projections and FFN gate, up, and down projections are adapted where present, with fused equivalents used for architectures that combine these matrices. The effective-weight expression in Equation 1 omits training-time dropout for clarity.

Training runs for eight epochs, with generation-based validation at epochs $\mathcal{E} = \{2, 4, 6, 8\}$. The model answers the held-out questions using the same text condition as in training, and the task's answer-extraction and scoring interfaces determine validation accuracy. Let a_e be accuracy at epoch e, $a_{\max} = \max_{e \in \mathcal{E}} a_e$, and $n_v = 80$. We use a one-standard-error rule to choose the earliest sufficiently accurate checkpoint:

$$e^* = \min \left\{ e \in \mathcal{E} : a_e \geq a_{\max} - \sqrt{\frac{a_{\max}(1 - a_{\max})}{n_v}} \right\}. \tag{3}$$

Neither anchors nor benchmark test questions enter this selection criterion. Supervised response tokens from both training sources enter the same microbatch mean loss, and optimizer updates accumulate these microbatch losses.

A.4 Training and Inference Text Conditions

The main internalization condition removes w_s from both positive and behavioral anchor inputs during training and omits it during inference. For comparison, we train a separate adapter with w_s included in both kinds of training input. Targets remain identical across these training conditions. In particular, behavioral anchors in the text-present condition still use the unadapted student's no-text responses as targets.

At inference time, MSI uses the no-text adapter without the skill description. MSI + Skill Text uses the separately trained text-present adapter with the description included. All conditions preserve the task prompt, required tool interfaces, and interaction history, and use one adapter for the full task.

A.5 Skill-Selection Settings

Skill selection is held separate from adapter learning. Our evaluation includes oracle assignment and a learned-retriever setting. Within each setting, comparisons between textual prompting and adapter invocation use the same assigned skill identifiers. All student backbones and text conditions share the frozen retrieval outputs in the learned-retriever setting.

Query and skill encoding. A shared BGE-M3 encoder g_η maps questions and skill descriptions to embeddings. Following SR-Agents (Su et al., 2026b), let $\kappa(q)$ and d_s denote the rendered query and skill text, and let $\bar{g}_\eta$ denote L_2-normalized embeddings. Selection uses

$$\hat{s} = \arg\max_{s \in \mathcal{S}} z_\eta(q, s), \qquad z_\eta(q, s) = \bar{g}_\eta(\kappa(q))^\top \bar{g}_\eta(d_s). \tag{4}$$

The query excludes reference answers and gold skill specifications. The returned identifier determines both the adapter and, where applicable, the skill text supplied to the generator.

Training examples. Synthetic questions provide question–skill supervision. Normalized duplicates merge their source skills into a positive set P_i. Questions are split approximately 90%/10% within source-skill groups for training and validation, with benchmark-overlap and cross-split duplicate filtering. For questions with multiple source labels, the implementation groups by the smallest sorted source identifier. This is a question-level split within a fixed skill library; it does not hold out entire skills. The split is independent of the adapter-data partition.

Each question contributes one source positive and one fixed hard negative sampled from R_0's top ten after excluding every known positive. For questions with multiple positives, the contributed positive rotates across epochs. Candidates from the global batch are deduplicated by skill identifier into a set C, and all known positives present in C are treated as positives for that question. The remaining batch candidates supply additional negatives.

Objective and selection. For a question batch $\mathcal{B}_r$, the retriever minimizes

$$\mathcal{L}_{\text{ret}} = -\frac{1}{|\mathcal{B}_r|} \sum_{i \in \mathcal{B}_r} \log \frac{\sum_{s \in P_i \cap C} \exp(z_\eta(q_i, s)/T)}{\sum_{s \in C} \exp(z_\eta(q_i, s)/T)}, \tag{5}$$

where T is the temperature. The encoder is optimized independently of the student task loss. Checkpoint selection maximizes mean per-dataset validation Recall@1 over the four datasets, with ties resolved in favor of the earlier epoch. The selected encoder is frozen for evaluation and uses no additional reranker.

B Experimental Setup

This appendix complements Section 4 with benchmark statistics, baseline configurations, optimization settings, and evaluation protocols. The data-synthesis pipeline, behavioral anchor construction, checkpoint-selection rule, and retriever objective are described in Appendix A.

B.1 Benchmarks and Skill Library

We use the four single-skill subsets of SRA-Bench (Su et al., 2026b), in which each evaluation instance has a reference answer and one annotated gold skill. TheoremQA evaluates theorem application, LogicBench evaluates logical reasoning, ToolQA evaluates tool-use workflows, and MedCalc-Bench evaluates medical calculation. Table 4 reports the instance and skill counts for the SRA-Bench subsets used here.

Table 4: SRA-Bench subsets used in our experiments. Counts refer to the evaluation instances and associated gold skills in SRA-Bench.

Subset	Capability	Instances	Skills
TheoremQA	Theorem application	747	320
LogicBench	Logical reasoning	760	19
ToolQA	Tool-use workflows	1,430	14
MedCalc-Bench	Medical calculation	1,100	55
Total		4,037	408

Library and routing conditions. The fixed library consists of all 408 gold skills associated with these subsets. Retrieval searches the joint library across all four datasets, and MSI activates one adapter per query. Under *Oracle* routing, the annotated gold skill determines the supplied text or adapter. Under *Retrieved* routing, the frozen retriever's top-1 prediction determines both. The retrieval query excludes reference answers and gold skill specifications. All backbones and text conditions share the same frozen routing outputs. LLM Direct require no skill selection; the random-adapter control follows the separate protocol below. This is a closed-world evaluation over skills already represented in the adapter library, rather than an evaluation on unseen skills.

B.2 Compared Methods

Unless otherwise specified, the selected skill denotes the gold skill under Oracle routing or the top-1 retrieved skill under Retrieved routing. Methods use the SR-Agents task prompts and interfaces, with skill text and demonstrations added as specified below.

- **LLM Direct.** The unadapted backbone answers the query without skill text or an adapter, providing a reference for its existing capabilities.
- **Oracle Skill / Full-Skill Injection.** The selected skill's full text is included in the task prompt, while model weights remain unchanged. The two names refer to Oracle and Retrieved routing, respectively.
- **Injection with demo.** Two synthetic training examples are appended to the selected skill text without modifying model weights. Each example includes a question and its solution; ToolQA examples retain the full interaction trajectory. Examples are selected using a length-based rule, fixed per skill, and shared across backbones. This baseline tests whether the synthetic demonstrations can improve performance through in-context learning.
- **MSI.** An independent LoRA adapter is trained for each skill using positive demonstrations and behavior-preservation anchors. The default setting omits skill text during both training and inference. The selected adapter is loaded at inference, with no online parameter updates. The MSI + Skill Text variant uses a separate adapter trained and evaluated with skill text. For this variant, skill text is included in both positive and behavioral anchor inputs during training, while anchor targets remain the unadapted backbone's responses generated without skill text.
- **Random skill adapter.** To probe skill specificity, we replace the gold adapter with a uniformly sampled non-gold adapter from the same dataset, using the existing adapter library trained without text. Skill text is omitted, while gold tool access is retained to focus the comparison on the procedure encoded in the adapter. We average results over three fixed routing seeds, 101, 202, and 303, shared across backbones. These seeds change adapter assignments, not adapter training.

B.3 Implementation Details

Models and synthetic data. The four backbones are Qwen3.5-2B, Llama-3.2-3B-Instruct, Phi-3.5-mini-instruct, and Apertus-8B-Instruct-2509. We use Qwen3.5-122B-A10B for task generation, trajectory generation, and quality evaluation, with temperatures of 1.0, 0.7, and 0, respectively. Generation requests use approximate difficulty quotas of 30% easy, 40% medium, and 30% hard; these are requested proportions, not measured proportions of accepted examples. The synthesis and filtering procedures are detailed in Appendix A.1. Benchmark test questions are excluded from generation prompts, and separate overlap checks filter exact and near matches.

Training and validation instances. The main per-skill setting uses 200 positive training instances, nominally 30 near-miss anchors and 30 random nonmatching anchors, and 80 additional positive validation instances. The nominal training quota of 260 excludes validation instances and counts instances before response-level expansion. Positive data are split by question before multi-turn trajectories are expanded, so responses from the same trajectory remain in the same split. Anchors are used only for training. Their responses are generated by the corresponding unadapted backbone without skill text and serve as behavior-preservation targets, not correctness-certified labels. Appendix A.2 specifies candidate selection using the fixed auxiliary retriever R_0, which is distinct from the trained online retriever.

Adapter optimization. We freeze the backbone and optimize LoRA parameters on attention and MLP projections. Only tokens in the current assistant response contribute to the loss; task prompts, questions, preceding responses, and tool observations are masked. Supervised tokens from positives and anchors enter the same mean cross-entropy loss within each microbatch, and gradient accumulation combines microbatch losses. Table 5 summarizes the optimization settings. Microbatch sizes vary with backbone and sequence length, with gradient accumulation maintaining an effective batch size of eight response-level samples. The maximum training length includes both the context and supervised response.

Table 5: Optimization settings for the main per-skill adapters and the shared retriever.

Setting	Per-skill adapter	Retriever
Optimizer	AdamW	AdamW
Learning rate	10^{-4}	2×10^{-5}
Weight decay	0.01	0.01
Warmup fraction	0.10	0.10
Epochs	8	3
Effective batch size	8 responses	16 queries
Precision	bfloat16	bfloat16
Maximum sequence length	20,480	8,192
Split/training seed	42	42
LoRA rank / scaling / dropout	8 / 16 / 0.05	Not applicable
Contrastive temperature	Not applicable	0.05

Target projections follow each backbone's architecture. Qwen and Llama adapt the query, key, value, attention-output, gate, up, and down projections. Phi adapts the fused query–key–value and gate–up projections, together with attention-output and down projections. Apertus adapts the query, key, value, attention-output, up, and down projections.

Checkpoint selection. For each skill and training-text condition, we generate answers to the 80 held-out positive questions at epochs 2, 4, 6, and 8. Validation uses the same text condition as training and task-specific scoring against reference answers extracted from the synthetic trajectories. We select the earliest checkpoint within one standard error of the highest validation accuracy, as defined in Equation 3. Neither anchors nor benchmark test instances participate in checkpoint selection.

Retriever training and selection. We fine-tune BGE-M3 on synthetic questions paired with their source skills, using 187,763 training questions and 20,858 validation questions. The approximately 90/10 split is constructed by grouping questions by source skill, independently of the adapter-data split. Normalized duplicate questions merge their source skills into a positive set, and separate checks exclude benchmark overlaps and duplicates across the retriever splits. The retriever's question pool is not limited by the per-adapter training quota. Training uses the multi-positive contrastive objective and fixed hard negatives described in Appendix A.5. We optimize for three epochs using the settings in Table 5, with an 8,192-token encoding limit for both queries and skill documents. Checkpoint selection maximizes the arithmetic mean of per-dataset validation Recall@1 over the full skill library, breaking ties in favor of earlier epochs. The selected epoch-3 checkpoint, BGE-M3-FT, is frozen for all backbones and text conditions. Online retrieval ranks normalized query and skill embeddings by dot product and selects the top-1 skill without reranking.

Inference and tool execution. We serve the backbones and adapters with vLLM using temperature-zero decoding. Methods share the task-specific prompts and output-token limits in Table 6, with additional context supplied according to Appendix B.2. ToolQA uses the benchmark's ReAct protocol, with at most 20 steps and 6,144 generated tokens per model call. Its 16,384-token limit applies to cumulative model output across the interaction trajectory. This inference budget is distinct from the 6,144-token admission limit for a synthetic ToolQA training trajectory.

The Llama evaluations reported under the "512" data-size setting use a 30-second SQL execution budget. Timeouts are returned as tool-error observations, allowing subsequent ReAct steps. Earlier completed evaluations were not rerun under this budget.

Table 6: Benchmark output-token limits, shared across backbones. ToolQA's budget is cumulative over the trajectory, with a separate per-call limit of 6,144 tokens.

SRA-Bench subset	Output-token limit
TheoremQA	10,240
LogicBench	8,192
ToolQA	16,384
MedCalc-Bench	8,192

B.4 Evaluation Protocols and Metrics

We evaluate all instances in the four SRA-Bench subsets, using SR-Agents' dataset-specific answer-extraction and rule-based scoring routines. An instance is correct only when its final answer passes the corresponding scorer; empty or unparseable answers count as incorrect. Oracle and Retrieved results are reported separately.

End-task accuracy. For subset d with N_d instances and C_d correct answers, accuracy is $100C_d/N_d$. The aggregate is instance-weighted micro accuracy,

$$\mathrm{Acc}_{\mathrm{micro}} = 100\,\frac{\sum_d C_d}{\sum_d N_d}. \tag{6}$$

Every benchmark instance receives equal weight, and the denominator is 4,037. We report this aggregate only when all four subsets have complete results. Unavailable results are shown as dashes; evaluations from a different retriever are not substituted for missing cells.

Retrieval recall. Recall@k is the fraction of benchmark queries whose annotated gold skill appears among the top-k candidates from the 408-skill library. Table 7 in Appendix C reports Recall@1 and Recall@10 for each subset. Retriever checkpoint selection instead uses synthetic validation queries and the unweighted mean of the four per-dataset Recall@1 values.

Random-adapter comparison. For the gold-versus-random comparison, random-adapter accuracy is averaged over the three fixed routing seeds.

Result highlighting. In Tables 1 and 3, bold and underlined values identify the best and second-best results among the methods shown, using unrounded counts. In Table 2, bold identifies the highest accuracy within each backbone and column. These formatting conventions do not indicate statistical significance.

C Skill Retrieval Performance

Table 7 reports retriever performance on the four single-skill subsets of SRA-Bench (Su et al., 2026b), using its query–skill relevance annotations to identify the gold skill for each query. All retrievers are evaluated on the complete benchmark instance sets against the same library of 408 skills. We report Recall@1 and Recall@10 (R@1 and R@10), measuring the percentage of queries whose gold skill appears among the top-1 and top-10 retrieved candidates, respectively. BGE-M3-FT is our retriever fine-tuned on synthetic query–skill pairs and evaluated without reranking.

D Prompt Templates

This appendix provides the prompt templates used for synthetic task generation, trajectory generation, quality evaluation, and demonstration-based skill injection. Task prompts and tool interfaces follow SR-Agents (Su et al., 2026b). Unless otherwise stated, `{skill_content}` denotes the full skill specification and `{question}` denotes the task question.

D.1 Synthetic Task Generation

Table 7: Skill retrieval recall (%) on SRA-Bench. Bold and underlined values indicate the best and second-best results per column, ranked using unrounded counts.

Retriever	TheoremQA		LogicBench		ToolQA		MedCalc-Bench	
	R@1	R@10	R@1	R@10	R@1	R@10	R@1	R@10
BM25	68.7	87.6	<u>20.8</u>	<u>79.7</u>	7.0	72.9	45.0	78.1
BGE-base	70.0	90.2	10.3	55.7	33.0	<u>99.7</u>	42.8	73.9
BGE-M3	67.3	<u>91.3</u>	12.2	55.7	30.0	86.8	<u>71.6</u>	<u>91.9</u>
BGE-base + reranker	69.9	90.2	17.4	55.7	<u>49.3</u>	<u>99.7</u>	27.4	73.9
BGE-M3 + reranker	<u>71.2</u>	<u>91.3</u>	15.8	55.7	43.9	86.8	24.5	<u>91.9</u>
BGE-M3-FT (Ours)	**73.9**	**95.6**	**48.8**	**99.5**	**77.0**	**100.0**	**93.3**	**98.9**

For each skill, the generator is asked to produce questions that require its methods. The system prompt combines a target difficulty with the dataset format in Section D.2. For ToolQA, `answerability` adds the grounding requirements in Section D.3; it is empty for other datasets:

Task generator: system

```
You are a training data generator for an AI skill system. Generate
questions that require applying the given skill's methods.

## Quality Requirements
- The problem must genuinely require the skill -- no shortcuts or
bypasses.
- The question must not exceed the skill's demonstrated scope --
require only methods the skill actually shows, and do not extrapolate
capabilities beyond its examples. Universal primitives any solver has
(arithmetic, comparison, formatting) may be composed in freely, but
cannot substitute for the skill.
- Vary parameters, contexts, and sub-methods across tasks.
{answerability}
- NO skill/tool names in questions. NO copied phrases from skill text.
- Do NOT compute or include the answer. Only generate the question.
{difficulty_block}
{format_instruction}

Output ONLY the question text, nothing else. No JSON, no labels, no
explanation.
```

The user prompt contains the skill specification, tool descriptions for ToolQA (`tool_env`), and previously generated questions (`prev_block`). We generate one question per request, setting `count_instruction` to `a question`:

Task generator: user

```
Generate {count_instruction} based on this skill.

## Skill
{skill_content}
{tool_env}
{prev_block}
```

For easy tasks, `difficulty_block` requests a direct application of one concept, with explicit inputs and no distracting context:

Difficulty block: Easy

```
## Difficulty Target: EASY
Generate a task requiring straightforward, single-concept application
of the skill:
- Provide all necessary information directly and explicitly. Zero
irrelevant, redundant, or misleading details.
- The problem should require one clear reasoning step. No chaining,
multi-stage synthesis, or sequential operations.
- Use standard, typical scenarios and conventional parameter values.
No edge cases, boundary values, or unusual inputs.
- Keep the problem statement concise and focused -- strip narrative
fluff. The core task should be immediately recognizable.
- The correct approach should be obvious to anyone familiar with the
skill. No tricks, no hidden assumptions, no double meanings.
```

For medium tasks, the generator is asked to require dependent reasoning steps while varying the conditions and context:

Difficulty block: Medium

```
## Difficulty Target: MEDIUM
Generate a task requiring competent, multi-step application of the
skill:
- Include some extraneous context that is plausible but ultimately
irrelevant to the solution -- the solver must distinguish signal from
noise.
- Require 2-3 distinct reasoning steps that depend on each other
(output of step 1 feeds step 2, etc.).
- Vary at least one parameter or condition away from the most standard
formulation. Use realistic but non-trivial values that require careful
handling.
- For applied domains (clinical, engineering), frame the problem in a
concrete scenario. For abstract domains (math, logic), a direct but
non-trivial formulation is fine.
- The solution path should require genuine thought but should be
discoverable by a competent practitioner without hints or external
references.
```

For hard tasks, the generator is asked to introduce information complexity, conceptual depth, or both. Each request uses one of the three difficulty instructions:

Difficulty block: Hard

```
## Difficulty Target: HARD
Generate a task that challenges even skilled practitioners.
Difficulty may come from either or both of:
  (a) Information complexity -- verbose context with
  irrelevant/distracting/misleading details that the solver must
  filter through.
  (b) Conceptual depth -- a compact but deeply challenging problem
  requiring insight, non-obvious connections, or creative composition
  of ideas.
Apply whichever mode fits the skill's domain:
- Require 4+ reasoning steps with complex dependencies between
intermediate results, OR a few steps that each require deep,
non-routine thinking.
- Use non-standard framing: unusual parameter combinations, edge cases
that break naive assumptions, boundary values, or implicit constraints
that must be inferred.
```

Difficulty block: Hard (continued)

```
- Mix multiple sub-concepts or sub-techniques in ways not explicitly
illustrated in the skill documentation. The solver must adapt and
compose, not follow a recipe.
- Add subtle constraints that change the solution logic: exceptions to
the typical rule, double negatives, or cases where the standard
approach does not directly apply.
- Multiple plausible but incorrect paths should exist -- identifying
the right one requires deep understanding of the skill's principles.
```

To reduce repetition, `prev_block` provides the last five generated questions, each truncated to 200 characters. The questions are listed as bullet points; this context is omitted when no previous questions are available:

Previously generated questions: user suffix

```
## Previously Generated Questions (avoid duplication):
{previous_question_lines}
```

D.2 Dataset Formats for the Generator

For TheoremQA, `format_instruction` constrains the answer type and provides two reference questions to guide generation:

Question format and fixed examples: theoremqa

```
## Question Format for theoremqa

The answer to your question must be exactly one of: a plain decimal
number, an integer, True or False, an option like (a)-(f), or a list
like [1, 2, 3].

The question must be self-contained with all necessary parameters.

## Reference Examples

Example 1:
A bag contains 6 red marbles and 4 blue marbles. Two marbles are drawn
without replacement. What is the probability that both marbles are the
same color?

Example 2:
Find the absolute maximum value of f(x) = x^3 - 3x^2 - 9x + 5 on the
interval [-2, 4].
```

For LogicBench, the generator is given format requirements and reference examples for binary and multiple-choice reasoning questions:

Question format and fixed examples: logicbench

```
## Question Format for logicbench

Your question should present a logical reasoning problem, either as a
binary question (BQA) or multiple choice (MCQA). For yes/no reasoning
questions: end with a determinate yes/no question.
For multiple choice: end with a question offer at most 5 distinct
options embedded at the end (e.g., 'choice_1: ...', 'choice_2: ...')

The question must be self-contained with all necessary parameters.
```

Question format and fixed examples: logicbench (continued)

```
## Reference Examples

Example 1 (BQA):
If a city improves its public transit system, more residents will use
it for daily commuting. However, if ticket prices are raised
significantly, ridership may decrease even with better service. We
know that either Metroville improved its transit system or ridership
did not increase. Based on this information, can we conclude that
Metroville raised ticket prices significantly?

Example 2 (MCQA):
A study finds that regular exercise reduces the risk of heart disease.
However, excessive exercise without proper recovery can lead to
injuries that discourage further physical activity. We know that at
least one of the following holds: (1) John exercises regularly, or (2)
John's risk of heart disease is not reduced. Given this, which of the
following must be true?
choice_1: John exercises excessively
choice_2: John has a reduced risk of heart disease
choice_3: If John exercises regularly, his risk of heart disease is
reduced
choice_4: John does not exercise at all
choice_5: John will definitely develop heart disease
```

For MedCalc-Bench, the generator is asked to describe a patient scenario and specify a calculation target. `calculator_scope` identifies the calculators associated with the supplied skill:

Question format and fixed examples: medcalcbench

```
## Question Format for medcalcbench

Describe a patient scenario specifying which score to calculate. The
answer will be a number or date.
{calculator_scope}

The question must be self-contained with all necessary parameters.

## Reference Examples

Example 1:
A 72-year-old female with a history of hypertension and type 2
diabetes presents to the clinic for a routine cardiovascular risk
assessment. She is a non-smoker. Her blood pressure is 138/85 mmHg,
total cholesterol is 210 mg/dL, HDL cholesterol is 45 mg/dL, and she
is currently on antihypertensive medication. Calculate her ASCVD
10-year risk score. All necessary clinical parameters are provided
above.

Example 2:
A 65-year-old man with a history of atrial fibrillation presents to
the emergency department with palpitations. He has a history of
congestive heart failure and hypertension. His blood pressure is
145/90 mmHg. Calculate his CHA2DS2-VASc score. All necessary clinical
parameters are provided above.
```

Here, `calculator_names` is a comma-separated list of clinical calculator names, such as *Body Mass Index*, obtained by matching the skill's tools to entries in the calculator catalog. The following instruction forms `calculator_scope` and tells the generator which calculations to target when constructing patient scenarios. It is omitted when no matching calculator is found:

Optional calculator scope

```
This skill covers: {calculator_names}.
```

For ToolQA, the generator is asked to produce questions whose answers must be obtained through tool calls. The format requirements and reference questions are:

Question format and fixed examples: toolqa

```
## Question Format for toolqa

The question must require the solver to retrieve information by
calling tools (LoadDB, FilterDB, GetValue, LoadGraph, etc.). The
solver starts with NO knowledge of database contents -- every piece of
data needed to answer must be obtained through tool calls. Do NOT
embed database facts as given information in the question; only
provide identifiers or criteria the solver can use as query
parameters. The answer will be a plain factual result discovered via
tools.

Write DIRECT questions starting with
What/Who/When/Where/How/Which/Can. Do NOT use narrative framing (no "A
researcher is...", "A traveler wants...", "You are planning...",
etc.). State the question concisely, under 200 characters.

The question must be self-contained with all necessary parameters.

## Reference Examples

Example 1:
What was the average price of coffee products across all stores in the
Downtown district during the month of March 2022?

Example 2:
Which airline had the highest number of delayed flights departing from
Terminal B between January and March 2023?
```

D.3 TOOLQA GROUNDING FOR TASK GENERATION

For ToolQA, the system prompt includes the following `answerability` instruction to require questions grounded in actual tool outputs:

ToolQA answerability: system insertion

```
- The question must have a concrete factual answer obtainable from
actual tool output. Verify the answer against the supplied real data
before writing the question.
```

When the skill uses `RetrieveScirex` or `RetrieveAgenda`, `answerability` additionally requires the answer to be explicitly stated in a supplied passage:

Retrieval answerability: conditional system insertion

```
- For retrieval questions, ground the question in one of the supplied
real passages and ask only for a fact explicitly stated in that
passage. Include enough distinctive context for that passage to be
recovered by retrieval.
```

The ToolQA generator is given the agent's tool descriptions through `tool_env`. Here, `REACT_INSTRUCTION` denotes the SR-Agents ToolQA instruction specifying the 13 available actions and the ReAct output format:

Tool environment: user insertion

```
## Available Tools (the agent's complete toolset)
{REACT_INSTRUCTION}
```

When database records, passages, or graph entities are available, they are appended to the generator's user prompt as `sampled_records`. The generator is instructed to use these records to construct answerable questions without revealing the answers:

Real-data grounding: user suffix

```
## Real Data from Underlying Databases
Below are sampled records from the databases this skill queries. Use
these concrete entities (names, dates, IDs, values) to GROUND your
question in real, queryable data.
CRITICAL: The person answering the question does NOT have direct
access to this data. They must call tools (LoadDB, FilterDB, GetValue,
etc.) to retrieve any information from the databases. Therefore:
- Do NOT embed data values from this sample as given facts in the
question.
- Design the question so that the answer must be DISCOVERED through
tool calls.
- You may reference entity names or IDs that exist in the data as
search targets, but never reveal the actual answer values that come
from the database.
{sampled_records}
```

For DBLP, the generator is instructed to use entities from the supplied graph snapshot. The following instruction is appended after the sampled entities:

DBLP grounding restriction

```
IMPORTANT: Only use author names, paper titles, and other entities
listed above (or clearly derivable from them, e.g. other collaborators
of a listed author) in your questions. Do NOT use well-known
researcher names (e.g. Yann LeCun, Fei-Fei Li, Geoffrey Hinton) or
famous paper titles (e.g. 'Attention Is All You Need', 'BERT') from
your training data -- they are NOT in this graph snapshot.
```

The generator also receives examples of tool calls and their actual outputs when available. We include at most six calls, with one example per action type, and truncate outputs longer than 400 characters:

Grounded tool workflow: user suffix

```
### Tool Workflow Demonstration (real tool calls with actual outputs)
Below are actual tool calls extracted from the skill examples,
executed against the real database. Use these to understand each
tool's capabilities, return format, and limitations.

{action_type}[{argument}]
  → {tool_output}
```

D.4 TRAJECTORY TEACHER

The teacher solves each generated question with the source skill and the corresponding task prompt. The following instruction is prepended to its system prompt to request a complete solution that can be understood without the skill document:

Trajectory teacher: system prefix

```
You are solving the problem. The provided skill describes a method
that may help; use it where it applies. Produce a complete,
fully-worked solution.

- When the skill's method can solve the problem, use it strictly -- do
not switch to a different method, and do not extrapolate to tools or
procedures the skill does not teach. Universal primitives (direct
reasoning over small sets, basic arithmetic, comparison) execute the
method; they do not replace it. Reserve a different method only for
problems the skill genuinely cannot solve. Apply the method as your
own reasoning, without quoting or naming the skill. In your response,
never mention or refer back to the skill, its text, provided reference
material, or these instructions.
- Reason naturally and coherently through the problem, step by step,
writing out each step of reasoning and its intermediate result. Do not
skip or combine steps, in computation or in reasoning, and carry the
method through to completion: every step that derives the answer must
be present and actually performed -- do not stop at an intermediate
result or declare the answer absent before the method's steps are
exhausted.
- Verify error-prone steps as you go: when a computation, an
assignment, or a case choice could be wrong, briefly confirm it
(substitute back, recount, or check the magnitude) before moving on.
- Do not use "obviously" or "clearly" in place of a step.
- Ensure the final result is unambiguous and directly extractable: for
code tasks, output the complete function body; for other tasks, state
the answer clearly on its own line.
```

The teacher-specific instruction is omitted from student inputs. ToolQA trajectory generation retains the standard ReAct examples and interaction history. Anchor targets are generated by the unadapted model with the base dataset prompt in a single chat call, without skill text, the teacher-specific instruction, or the ToolQA ReAct examples.

D.5 Trajectory Quality Evaluation

The evaluator is instructed to accept a trajectory only when it satisfies the method, reasoning, answer, and format criteria below. `tool_context` specifies the applicable tool-use requirements, and `format_context` specifies the dataset's answer format:

Trajectory quality evaluator: system

```
You are a trajectory quality evaluator for an AI skill training
system. Judge whether the reasoning trace is acceptable as training
data.

## Evaluation Criteria (ALL must pass for acceptable=true)

1. Method Appropriateness (symmetric): The trajectory's method must
match the skill-task relationship in BOTH directions.
   - If the skill's method applies, the trajectory uses it. Reject if
   it solves without the skill when the skill does apply (lost
   benefit: a fitting skill must actually be used).
   - If the skill's method does NOT apply, the trajectory solves by an
   appropriate method. Reject if it forces the skill onto a problem it
   does not fit (abandonment harm).
```

Trajectory quality evaluator: system (continued)

```
   No mismatch declaration is required or desired (implicit
   abandonment).
{tool_context}

2. Skill Independence: The trajectory reasons as self-contained
problem-solving and does not lean on the skill text. Reject if it
contains dangling references that would be incoherent without the
skill (e.g. "the skill states", "according to the provided method",
"as given"). Applies uniformly whether the skill is used or implicitly
abandoned (no declaration, hence no exception).

3. Step Completeness: Every step of reasoning and computation is
shown. Reject on a skipped or combined step, an unstated intermediate
value, or an unjustified leap - EVEN IF the final answer is correct. A
non-trivial computation must be evaluated one operation per line.
Example to REJECT (answer correct, but collapsed): "0.75*9.2103 +
0.25*9.1050 = 9.184". Example to ACCEPT: "0.75 x 9.2103 = 6.9077; 0.25
x 9.1050 = 2.2762; sum = 6.9077 + 2.2762 = 9.184". Applies to
arithmetic AND the setup/reasoning that precedes it. Computation
delegated to a provided tool (TOOL_CALL) is exempt.

4. Soundness: No calculation errors, circular reasoning, or
"obviously"/"clearly" gaps.

5. Answer: A final answer is clearly stated and extractable; magnitude
plausible.

6. Format: The answer follows the dataset's expected format.
{format_context}

Output JSON only:
{"acceptable": true/false, "reason": "brief explanation",
"extracted_answer": "value"}
```

The evaluator is given the dataset, generated question, source skill, and completed trajectory in the following user prompt. It returns a JSON object containing its acceptance decision, rationale, and extracted answer:

Trajectory quality evaluator: user

```
## Dataset
{dataset}

## Question
{question}

## Skill Content
{skill_content}

## Trajectory to Evaluate
{trajectory_output}
```

For TheoremQA, `format_context` specifies the final-answer phrase and supported answer types:

Evaluator format context: theoremqa

```
Dataset: theoremqa (math/science).
```

Evaluator format context: theoremqa (continued)

```
Expected answer format: 'Therefore, the answer is ...' in the final
sentence. Answer must be one of: plain decimal number, integer,
True/False, option like (a)-(f), or list like [1, 2, 3]. No symbolic
expressions.
```

For MedCalc-Bench, `format_context` requires a step-by-step calculation and an explicit numeric or date answer:

Evaluator format context: medcalcbench

```
Dataset: medcalcbench (medical calculations).
Expected answer format: ANSWER: <numeric value> on its own line.
Step-by-step calculation must be shown. Value can be integer, decimal,
or date (MM/DD/YYYY).
```

For LogicBench, `format_context` distinguishes binary answers from multiple-choice labels:

Evaluator format context: logicbench

```
Dataset: logicbench (logical reasoning).
Expected answer format: BQA → yes or no (lowercase). MCQA → choice_1
through choice_5.
```

For ToolQA, `format_context` specifies the ReAct trajectory structure and the final `Finish` action:

Evaluator format context: toolqa

```
Dataset: toolqa (tool-augmented QA).
The trajectory uses the ReAct protocol. Each step is structured as:
  Thought N: <single paragraph of reasoning>
  Action N: <ActionType[args]>
  Observation N: <tool output>
The final step ends with: Action N: Finish[<answer>], followed by
  Observation N: Answer: <answer>
The answer is plain factual text.
```

For MedCalc-Bench, `tool_context` permits manual computation when it follows the skill's method, shows correct intermediate calculations, and agrees with the tool's result:

Evaluator tool context: medcalcbench

```
Tool usage: This skill provides a calculation tool (TOOL_CALL). Using
the tool is preferred, but a trajectory that computes manually is also
acceptable IF it satisfies ALL of the following:
- Follows the exact formula/logic described in the skill content.
- Each intermediate calculation step is shown and arithmetically
correct (verify the math -- reject on any arithmetic error).
- The final answer matches what the tool would produce.
Reject if the trajectory uses a fundamentally different method, has
calculation errors, or skips intermediate verification steps.
```

For ToolQA, `tool_context` requires the trajectory to use the available tools and rejects unlisted tools. TheoremQA and LogicBench use an empty `tool_context`:

Evaluator tool context: toolqa

```
Tool usage: This skill's core method requires tool calls via the ReAct
action protocol. The available tools are:

{REACT_INSTRUCTION}

Reject if the trajectory uses tools not listed above or attempts to
answer without tools.
```

D.6 Synthetic Demonstrations for Skill Injection

For the demonstration baseline, two fixed synthetic training examples are appended to the selected skill. The augmented skill content is then prepended to the task prompt using the Full-Skill Injection format:

Injection with demo: augmented skill content

```
{skill_content}

Worked examples of this skill:

{demo_1}

---

{demo_2}
```

For TheoremQA, LogicBench, and MedCalc-Bench, each demonstration contains a synthetic question and its solution. The `demo_1` and `demo_2` fields use the following format:

Direct-task demonstration

```
Problem:
{demo_question}
Solution:
{demo_solution}
```

For ToolQA, each demonstration contains the synthetic question and its complete Thought–Action–Observation trajectory. These demonstrations are provided in addition to the standard ToolQA ReAct examples:

ToolQA demonstration

```
Problem:
{demo_question}
Trajectory:
{demo_trajectory}
```