ADAPTIVE MULTI-RESOLUTION PROCEDURAL KNOWLEDGE COMPRESSION FOR LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) are widely used to tackle complex tasks with autonomous workflows. Recently, reusable natural language skills have emerged as a popular paradigm to inject procedural knowledge into LLM applications. Since popular skills are often invoked repeatedly, placing their full text in every context significantly increases prefill cost and latency. While text compression techniques have the potential to solve this problem, most existing methods are designed to compress factual knowledge in documents instead of procedural knowledge, making them insufficient for skill compression. In this paper, we argue that an effective skill compression method should: 1) preserve logical dependencies among workflows and tool protocols, 2) enable lightweight, offline compression for frequently updated community skills, and 3) be adaptable to varying complexities across skills. To address this, we present **SKIM** (SKIll coMpression), an adaptive multi-resolution soft token compression framework for procedural skills. Depending on the complexity of each skill, SKIM creates different numbers of soft tokens that not only improve the efficiency of LLM inference, but also preserve the effectiveness of skill usage. Experiments indicate that SKIM's fixed-budget configurations compress skills to as few as 15 to 55 percent of their original token length, while the adaptive variant selects the minimal budget that preserves task fidelity.[1]

1 INTRODUCTION

Skill, a type of data structure that stores reusable procedural knowledge specifying when and how a large language model (LLM) should invoke a capability, follow a workflow, or interact with tools, has been shown to be effective in improving the performance of LLM agents in downstream tasks. Such skills are often written as natural-language files such as `SKILL.md`, making them easy for users to create, edit, and share (Zhou et al., 2026; Su et al., 2026). While this natural-language format has made skills an attractive abstraction for building, distributing, and reusing agent capabilities, deploying them at scale introduces substantial costs in terms of token consumption. Typically, every invocation requires placing the full skill text into the model's context window. As agent skill ecosystems like OpenClaw OpenClaw (2026b) increasingly power daily workflows for many developers, popular skills are downloaded and then invoked across diverse systems (Jiang et al., 2026), causing redundant token overhead to accumulate. Therefore, even reducing a small fraction of prompt tokens for these skills can translate into aggregate computational and latency savings. Furthermore, since users can freely modify these natural-language skills and deploy them across various inference backends, the direct application of traditional key-value (KV) cache reuse (Kwon et al., 2023) becomes unfeasible.

A natural way to reduce this overhead is to compress skill content before deployment. Existing text compression methods can be broadly divided into hard compression, which selects discrete text tokens (Pan et al., 2024; Jiang et al., 2024), and soft compression, which maps contexts into continuous representations (Ge et al., 2024; Li et al., 2025b;a). These methods have been primarily developed and evaluated for generic prompts, long-context understanding, or document-centric tasks, where preserving salient semantic content or query-relevant evidence is often sufficient. In such settings, a compressed context can remain useful as long as the critical facts or evidence are still recoverable.

[1]We have released our code at `https://anonymous.4open.science/r/SKIM`

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.

Table 1: Comparison of prompt compression methods. Lightweight denotes forward compression without gradient optimization.

Work	Form	Offline Compression	Lightweight	Knowledge Type
LLMLingua-2 (Pan et al., 2024)	hard tokens	Yes	Yes	General
LongLLMLingua (Jiang et al., 2024)	hard tokens	No	Yes	Factual
ICAE (Ge et al., 2024)	soft tokens	Yes	Yes	Factual
500xCompressor (Li et al., 2025b)	soft tokens & KV Cache	Yes	Yes	Factual
DRIFT (Xie et al., 2026)	soft tokens	No	Yes	Factual
TokMem (Wu et al., 2026)	soft tokens	Yes	No	Procedural
LatentSkill (Yu et al., 2026)	LoRA parameters	Yes	Yes	Procedural
SKIM	soft tokens	Yes	Yes	Procedural

Unfortunately, unlike descriptive documents, skills encapsulate procedural knowledge (Chen et al., 2026), presenting three unique compression challenges for existing text compression methods. First, skill compression must preserve procedural dependencies. Such information is not stored as simply as some factual keywords. Therefore, rather than merely retaining isolated facts, compression must strictly preserve the dependencies between conditions, tool arguments, and other information in the skill, as severing even one logical link can lead to execution failure. For example, Figure 1 shows a case where we use ICAE (Ge et al., 2024), a representative context compression method for LLMs, to compress a skill. When the compressed skill is used for a factual question, the performance of the system is fine. However, when it is used on a task that actually requires the procedural knowledge in the skill, ICAE breaks the system and produces wrong results.

Second, the sharing and deployment paradigm of skills differs fundamentally from static texts (Hu et al., 2026). Since users actively share, modify, and distribute skills, their compressed representations must be rapidly regenerable and easily transmittable. Consequently, the resulting artifacts must be storage efficient. Furthermore, the process of encoding a new or updated skill must be computationally lightweight, relying only on a simple forward pass rather than expensive per-case gradient backpropagation required by temporary training methods like TokMem (Wu et al., 2026).

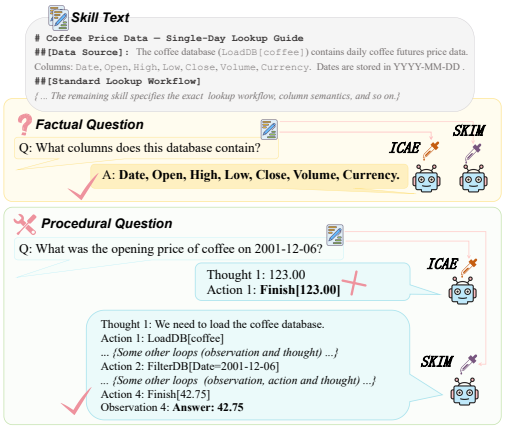

Figure 1: A ToolQA example. Factual information can be recovered from compressed skills, but the procedural question requires preserving workflow specified by skill.

Third, skills differ significantly in complexity and information density. A fixed compression rate cannot simultaneously adapt to all skills, and the optimal rate for one skill might even vary across models. Therefore, an ideal framework should represent a skill under multiple token budgets and automatically select the appropriate compression resolution for specific skill and model pairs.

To address these challenges, we propose **SKIM** (SKIll coMpression), an adaptive multi-resolution soft token compression framework. Following dual model compression architectures such as DRIFT (Xie et al., 2026), SKIM uses a compressor to encode the skill and a projector to map the resulting soft tokens into the target LLM space. SKIM is trained via a progressive three-stage paradigm, including skill reconstruction, procedural warm-up, and skill task alignment, to preserve executable dependencies. During skill task alignment, the target LLM is adapted via a single Low-Rank Adaptation (LoRA) (Hu et al., 2022) module, ensuring parameter efficiency. To accommodate varying skill complexities, SKIM supports representing a skill under multiple token budgets (resolutions) simultaneously. Before deployment, an offline self-judgment mechanism automatically selects the lowest-budget resolution that maintains execution accuracy.

This design perfectly aligns with the nature of skill deployment paradigm. Developers can distribute skills as lightweight, pre-computed soft token bundles for different mainstream target models, functioning similarly to a prebuilt artifact in software distribution. When a skill is updated, its

compressed representation is instantly regenerated via a single forward pass. Furthermore, the multi-resolution design seamlessly adapts to the varying complexities of different skills. During online inference, the serving infrastructure simply loads the pre-selected soft tokens and switches to the shared LoRA adapter, without additional latency. Our contributions are summarized as follows:

- We formulate procedural skill compression as distinct from factual document compression, where preserving executable logic is central.
- We propose SKIM, a skill compression framework that combines progressive training with offline self-judgment for resolution selection.
- We evaluate SKIM on several skill-based datasets, showing that it outperforms existing methods in token reduction while maintaining high skill task accuracy.

2 Related Work

2.1 Skills for Large Language Models

Large Language Models (LLMs) have evolved into agentic problem solvers for complex tasks. A growing paradigm is therefore to package procedural knowledge as *skills* (Chen et al., 2026), which can augment agent frameworks such as OpenClaw (OpenClaw, 2026b). A skill is usually written as a `SKILL.md` file. During inference, after one or more skills are selected, their main body fields are loaded into the LLM context. A recent large scale study reports that online skills average more than 2,000 tokens and that some exceed 10,000 tokens (Cho et al., 2026). Moreover, platforms like ClawHub (OpenClaw, 2026a) provide centralized repositories for skill sharing, accelerating their adoption, which significantly increases prompt token consumption.

2.2 Prompt Compression

Prompt compression reduces computation cost by distilling redundant context. Hard compression prunes discrete tokens: LLMLingua-2 (Pan et al., 2024) provides task-agnostic compression, and LongLLMLingua (Jiang et al., 2024) adds query-aware selection at the cost of online latency. These hard techniques often discard structural elements necessary for executable logic. Soft compression projects text into continuous representations. Early work on gist tokens (Mu et al., 2023) and AutoCompressors (Chevalier et al., 2023) introduced compact continuous context summaries. ICAE (Ge et al., 2024), a direct successor, maps text into continuous embeddings, and 500xCompressor (Li et al., 2025b) retains KV cache tensors, introducing storage pressure unsuitable for skill distribution. TokMem (Wu et al., 2026) distills an instruction sequence into a single token but demands per-procedure gradient optimization, making it impractical for rapidly evolving skill ecosystems. LatentSkill (Yu et al., 2026) converts each skill into LoRA parameters with zero context tokens, but its training is computationally expensive and storage scales linearly with the skill count. DRIFT (Xie et al., 2026) uses a dual model soft token architecture for factual information but requires online compression, introducing runtime overhead. Building upon such architecture, SKIM optimizes a compressor and a target LLM specifically for procedural skills, while ensuring offline compression and lightweight forward pass. We summarize these differences in Table 1.

3 Methodology

We describe SKIM as a framework for replacing skill content with continuous tokens, covering the architecture, progressive training, offline resolution selection, and inference.

3.1 Overview

A skill often contains name, description, content, and other metadata fields. The content field is the main body and commonly specifies procedural guidance, tool descriptions, and usage constraints. We focus on the content field of a skill, since this field is the skill body loaded into the LLM context after selection. Given skill content s and a user query u, the goal is to replace the full skill content text in the input prompt with a compact continuous representation while preserving the procedural behavior specified by s. For a target LLM M with embedding dimension d_M, SKIM produces a

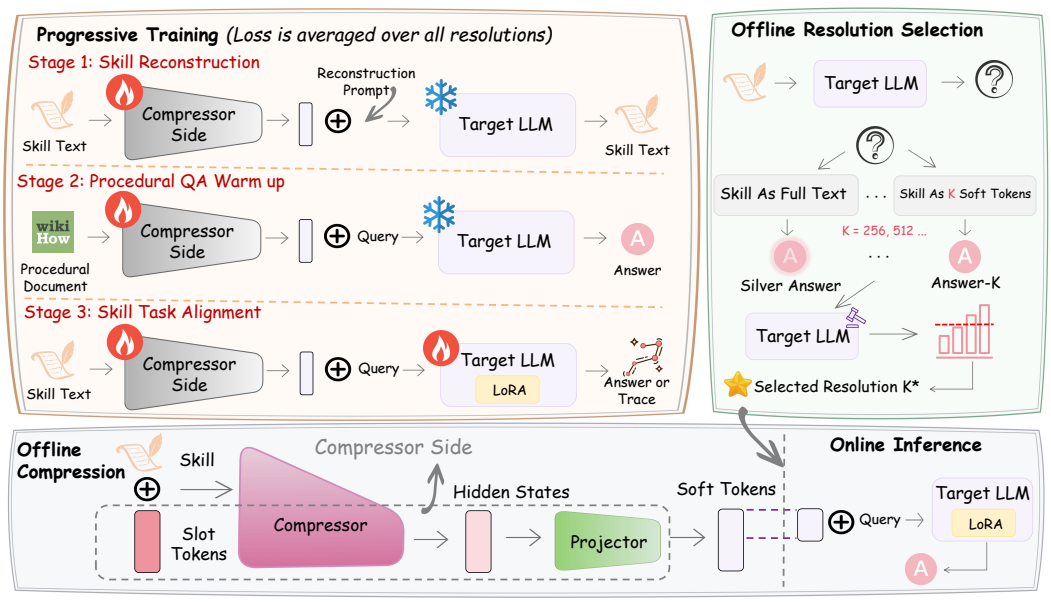

Figure 2: Overview of SKIM. The upper panel illustrates the training process and offline resolution selection, where the compressor (consisting of slot tokens, an LLM-based compressor, and an MLP projector) learns to convert skills into soft-token prefixes aligned with the target LLM's embedding space. The lower panel shows the inference architecture, where SKIM selects the soft-token prefix corresponding to the offline-determined optimal resolution and prepends it to the target LLM.

matrix $E_K(s) \in \mathbb{R}^{K \times d_M}$ under a token budget K. The budget K is selected from an ordered set $\mathcal{K}$, where smaller values correspond to more aggressive compression. SKIM contains a compressor, learnable slot tokens, a projector, and a target LLM adapted with LoRA. The compressor reads the skill and slot tokens, the projector maps slot position hidden states into the target LLM embedding space, and the resulting soft tokens are inserted into the target context together with the user query.

Figure 2 summarizes the pipeline. During training, the same skill is optimized under multiple resolutions so that the compressor learns representations that remain useful when truncated. Before deployment, SKIM compresses each skill for each target model, stores the full soft token artifact, and runs an offline diagnostic evaluation to select the resolution for that skill and target model. At inference, the target LLM receives text tokens for the query and soft tokens for each skill, and switches to the LoRA adapter to generate response.

3.2 Multi-Resolution Compressor

SKIM follows the dual model soft token design used in recent context compression work (Liu & Qiu, 2025; Xie et al., 2026). The compressor C_θ is an autoregressive backbone LLM that receives tokenized skill content followed by $K_{\max}$ learnable slot tokens. These slot tokens are trained parameters and remain fixed after training. After the compressor forward pass, the hidden states at the slot token positions form a latent representation $Z(s) \in \mathbb{R}^{K_{\max} \times d_C}$, where d_C is the compressor hidden dimension. A multilayer perceptron (MLP) projector P_ϕ maps this latent representation into the target model space:

$$E_{K_{\max}}(s) = P_\phi(Z(s)) \in \mathbb{R}^{K_{\max} \times d_M}. \tag{1}$$

For a lower resolution $K < K_{\max}$, SKIM uses the prefix $E_K(s) = E_{K_{\max}}(s)_{1:K}$. Since one high resolution representation can be truncated to obtain lower resolutions, a platform can store a single artifact per skill and target model, then choose a smaller budget without recomputing the representation. When a prompt contains multiple skills, SKIM compresses each independently. The projected soft token sequences are then concatenated in the same order as the selected skills. Then, the target LLM with LoRA adapter is called with the soft token sequence and the user query.

3.3 PROGRESSIVE TRAINING

SKIM uses three training stages. Each stage optimizes the compressor, learnable slot tokens, and projector with full parameter updates, but differs in supervision and whether the target LLM is adapted. For each training example, the loss is averaged over resolutions returned by $\mathcal{K}$. This objective teaches the model to support multiple budgets.

Stage 1: Skill Reconstruction. The first stage learns a general skill representation from a large corpus of skills collected from the web. The target LLM is frozen. Given a skill s, SKIM inserts $E_K(s)$ into a reconstruction prompt and optimizes the negative log likelihood of the original text:

$$\mathcal{L}_{\text{rec}} = -\frac{1}{|\mathcal{K}|} \sum_{K \in \mathcal{K}} \log p_M(s \mid E_K(s), r_{\text{rec}}), \tag{2}$$

where r_{rec} is a short reconstruction instruction. This stage encourages the slot states to retain the core information in the skill content before the model is exposed to downstream tasks.

Stage 2: Procedural Question Answering Warm Up. The second stage aligns the compressed representation with procedural question answering. Unlike factual knowledge Question Answering (QA), where labeled question answer pairs are abundant, labeled QA data for executable skills are much more limited. To address this scarcity, we introduce a procedural warm-up stage that repurposes WikiHow (Koupaee & Wang, 2018) as weakly supervised question answering data. WikiHow is a large scale summarization dataset crawled from "how-to articles" on the *WikiHow* website. For our purpose, the article title serves as the question, the article body serves as procedural text, and the summary serves as a concise answer. Each example therefore contains a procedural document d, a query q, and an answer a. The document is compressed while the query remains plain text:

$$\mathcal{L}_{\text{qa}} = -\frac{1}{|\mathcal{K}|} \sum_{K \in \mathcal{K}} \log p_M(a \mid E_K(d), q). \tag{3}$$

Since the target LLM is frozen, this stage mainly trains the compressor side to expose information in a form that the target model can consume. The warm-up signal moves the compressor beyond reconstruction by aligning procedural text with answer generation before skill tasks are introduced.

Stage 3: Skill Task Alignment. The third stage uses the same task format and loss as the warm-up stage but replaces WikiHow documents with real skills from the first stage. Our goal here is to align the compressed continuous tokens and the target model's behavior with actual skill-conditioned tasks, which often require complex decision making, tool calls, or the joint use of multiple skills. To build this training data, we employ a high-capacity LLM (GPT-5.2 in our implementation (OpenAI, 2025)) as an evaluator. This LLM first evaluates the collected skills, filtering out documents with weak procedural content. For the remaining skills, it generates skill-dependent user questions and extracts relevant metadata (e.g., tool requirements or splittability). Then, supervised answers are generated by the target model. We prompt it to follow skill guidance to prevent it from bypassing the skill text and relying solely on its parametric memory. Furthermore, to effectively train the model on complex real-world behaviors, our data construction specifically addresses two critical properties of skills:

- **Tool Usage Simulation:** Many skills specify external tool usage. To ensure the framework can trigger correct tool workflows, we synthesize ReAct-style traces (Yao et al., 2023) with interleaved Thought, Action, and Observation steps. Since accessing APIs for collected diverse web skills is impractical, when generating answers, we use the target LLM itself as a simulator to generate simulated tool observations conditioned on the tool schema, query, and partial trajectory.
- **Multi-Skill Composition:** Some queries require integrating multiple skills. Since evaluating every pairwise combination of skills is computationally prohibitive, we use a top-down "skill split" strategy. The high-capacity LLM rewrites a complex skill into self-contained subskills while using the original skill text to generate the corresponding question. During training, SKIM compresses these subskills independently and concatenates their soft tokens to model multi-skill scenarios.

Algorithm 2 in Appendix B summarizes this data construction procedure. In this training stage, the target LLM is adapted through a LoRA module, while the compressor, slot tokens, and projector remain trainable. This single, unified LoRA adapter is shared across all skills, aligning the target model's behavior with the continuous skill tokens, without modifying its full parameters.

3.4 Offline Resolution Selection

Since skills differ in complexity, different skills require different compression budgets. A simple skill may be represented with only a few soft tokens, whereas a multi-step procedural skill may require a higher resolution. We therefore select a skill-model specific resolution before deployment. Algorithm 1 shows the related pseudocode.

For each skill s, SKIM selects its compression resolution through an offline calibration step. The target LLM first generates N diagnostic questions from the original skill text. For each question, the target LLM is prompted with the full skill text to produce a reference answer, which serves as a silver standard because no human label is available. We then replace the full skill text with compressed soft tokens at each resolution $K \in \mathcal{K}$, generating one candidate answer per resolution. The model itself compares each candidate answer with the corresponding full-text reference answer and computes a fidelity score $\alpha_K(s)$ for each resolution, defined as the fraction of the N diagnostic questions whose compressed-resolution candidate answer is judged equivalent to the full-text reference. Given a threshold τ, SKIM chooses the smallest resolution whose fidelity reaches the threshold:

$$K^\star(s) = \min\{K \in \mathcal{K} : \alpha_K(s) \geq \tau\}. \tag{4}$$

If no compressed resolution reaches the threshold, the system falls back to the original skill text. This offline calibration adds no latency to user requests.

3.5 Inference

At inference time, a single skill uses its selected resolution $K^\star(s)$. For a prompt with multiple skills, SKIM uses the highest selected resolution among the referenced skills, compresses each skill independently under that shared budget, and concatenates their soft token sequences. The system fetches the corresponding soft token prefixes, activates the LoRA adapter, concatenates the soft tokens with the user query, and runs the target LLM forward pass to decode the final response.

Importantly, the LoRA adapter trained in Stage 3 is a single, unified module shared across *all* skills, including unseen ones. The online path merely involves skill lookup (which depends on the skill retrieval system), soft token loading, and standard LLM decoding with the LoRA activated. Modern inference engines like vLLM (Kwon et al., 2023) support pre-loading and switching LoRA adapters with minimal overhead. For queries not requiring skills, the system seamlessly deactivates the adapter to preserve the base model's original capabilities.

When a skill is updated, the shared LoRA or the compressor requires no retraining. The publisher can regenerate the corresponding soft token artifact through a compressor forward pass, and may rerun the offline resolution selection step when the edit is substantial. An empirical study in Appendix I shows that edits below 30% character edit distance tend to preserve the original resolution assignment in our controlled experiments. This preserves the convenient distribution property of natural language skills.

4 Experimental Setup

4.1 Tasks and Data

We evaluate SKIM on five datasets: BigCodeBench (Zhuo et al., 2025), CHAMP (Mao et al., 2024), LogicBench (Parmar et al., 2024), TheoremQA (Chen et al., 2023), and ToolQA (Zhuang et al., 2023). For each task, we use the golden skill annotations provided by SRA-Bench (Su et al., 2026). This setup fixes the relevant skill content for each instance, so the comparison focuses on how the skill is represented in the context. These tasks cover code generation, mathematical reasoning, logical reasoning, theorem question answering, and tool use question answering. The first four datasets use single-turn QA evaluation, whereas ToolQA uses ReAct multi-turn decision making with tool observations. For each instance, the user query remains plain text, while the skill content is either inserted in full or replaced by compressed tokens. Table 5 lists the dataset statistics.

Table 2: Main results on five datasets with golden skills (accuracy %). BigCode, Logic, Theorem abbreviate BigCodeBench, LogicBench, TheoremQA. † = official LatentSkill checkpoint; unmarked = our Stage 3 data. – = not reproduced due to long training time. Gray × ratios show compressed tokens relative to Full Text. SKIM-Adaptive selects among 256, 512, and full text via offline exam; SKIM-Fix uses a fixed budget. Bold = highest accuracy per token group (256, 512).

	Qwen3-8B					Phi-4				
Method	**BigCode**	**CHAMP**	**Logic**	**Theorem**	**ToolQA**	**BigCode**	**CHAMP**	**Logic**	**Theorem**	**ToolQA**
Naive	40.96	48.88	77.11	52.07	28.04	43.77	52.02	48.55	45.38	14.83
LatentSkill†	35.53	39.91	71.71	45.78	6.50	–	–	–	–	–
LatentSkill	40.53	52.91	71.45	41.50	10.35	–	–	–	–	–
256 Token / Skill										
500xCompressor	33.33	19.28	69.61	28.65	0.00	15.19	32.29	47.63	40.03	0.00
ICAE	29.47	27.80	62.11	28.38	7.13	0.88	30.94	60.66	18.61	0.00
LLMLingua-2	40.70	52.47	82.11	58.10	**36.36**	43.37	56.95	52.50	**45.92**	14.76
SKIM-Fix	**44.21**	**54.26**	**83.16**	**60.24**	35.66	**46.58**	**58.74**	**63.16**	43.78	**23.08**
Ratio	×.15	×.23	×.24	×.27	×.25	×.15	×.23	×.23	×.28	×.26
512 Token / Skill										
LLMLingua-2	43.60	54.71	**85.00**	61.98	40.42	45.39	**58.30**	59.87	48.06	10.35
SKIM-Fix	**45.53**	**61.43**	**85.00**	**64.79**	**43.92**	**49.12**	57.85	**74.74**	**48.33**	**26.78**
Ratio	×.30	×.45	×.47	×.54	×.51	×.31	×.45	×.47	×.55	×.51
Variable										
Full Text	49.30	68.16	85.26	68.81	47.97	52.19	61.43	75.66	51.14	9.79
# Tokens	4676	1941	1089	935	1010	4628	1933	1093	921	996
SKIM-Adaptive	46.93	65.92	83.68	67.87	46.92	50.35	61.88	69.61	48.73	17.27
# Tokens	3172	1424	438	727	754	1889	1664	335	682	705
Ratio	×.68	×.73	×.40	×.78	×.75	×.41	×.86	×.31	×.74	×.71

4.2 Models

We evaluate two target LLMs: Qwen3-8B (Yang et al., 2025) (with itself as compressor) and the 14B Phi-4 (Abdin et al., 2024) (with the 3.8B Phi-4-mini-instruct as compressor). Therefore, this setting lets us examine changes across model families, target model sizes, and compressor sizes. For each target model, SKIM runs model specific offline resolution selection for each skill.

4.3 Baselines

We compare against two non compression references: Naive (no skill loaded) and Full Text (golden skills without compression). We include LLMLingua-2 (Pan et al., 2024) as a hard compression baseline, configured to compress each skill to 256 and 512 tokens to match SKIM's fixed budgets. ICAE (Ge et al., 2024) and 500xCompressor (Li et al., 2025b) serve as soft compression baselines, trained on the same data as SKIM; for multi-skill instances, each skill is compressed individually to 256 tokens and then concatenated. We further include LatentSkill (Yu et al., 2026), which converts skills into LoRA parameters with zero added context tokens; since its training is computationally expensive, we evaluate it only on Qwen3-8B with both its official checkpoint and a variant trained on our Stage 3 data. These baselines correspond to those in Table 1 featuring both offline and lightweight compression. Further details are in Appendix C.

4.4 Implementation Details

For SKIM, we report three variants. SKIM-Fix 256 and SKIM-Fix 512 use a fixed soft token budget for every skill. For the few skills shorter than their compressed form, we directly use the full-text form. SKIM (Adaptive) uses the offline resolution selection to choose among 256 tokens, 512 tokens, and the full text for each skill. The offline exam generates 10 diagnostic questions per skill. The selection accuracy threshold is 0.9. For multi-skill instances, we use the highest selected budget among the referenced skills when a shared compression setting is needed. Further hyperparameters and settings, as well as the prompt templates, are provided in Appendix C and D, respectively.

5 Experimental Results

5.1 Main Results

Table 2 reports task accuracy across both target models and five datasets, with methods grouped by their per-skill token budget. Full Text usually outperforms Naive, demonstrating the value of skill annotations, although it also adds hundreds to thousands of tokens per instance. Generic soft compression struggles to retain this benefit. Even when trained on the same data as SKIM, ICAE and 500xCompressor generally underperform Naive, indicating that their compression architectures are insufficient for preserving procedural knowledge (Appendix K). LatentSkill, which encodes skills entirely into LoRA parameters, performs below Naive on most datasets even with aligned training data, since converting procedural logic into parameter updates loses the explicit contextual guidance that the target model requires.

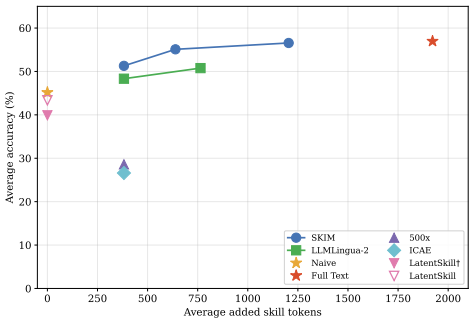

Figure 3: Average accuracy vs. added tokens across both models and five datasets. LatentSkill averages over Qwen3-8B only. Lines connect settings from the same method.

At matched token budgets, SKIM consistently outperforms LLMLingua-2 on most model-dataset pairs at both 256 and 512 tokens per skill. This stems from SKIM aligning continuous tokens with procedural knowledge through supervised skill task training, whereas LLMLingua-2 prunes text without explicit procedural execution supervision. Figure 3 shows the overall trade-off between accuracy and tokens for all methods.

Within SKIM, Adaptive improves over fixed budget variants on most datasets, approaching Full Text accuracy while using fewer tokens, since the offline exam selects lower resolutions when diagnostic answers are sufficient. In a few cases like Phi-4 on LogicBench, a single threshold may heavily favor token reduction by selecting more low-resolution skills, which slightly reduces accuracy relative to Fix-512. Figure 6 in the appendix shows the selected resolution distribution. Furthermore, Adaptive uses Full Text answers as silver references, which is usually reasonable as golden skills generally improve the target model. However, if the skill text is unhelpful or the model fails to follow it (e.g., Phi-4 on ToolQA, where Full Text underperforms Naive), the weak silver reference causes Adaptive to fall below fixed SKIM variants. We also evaluated Phi-4 on ToolQA with the LoRA adapter disabled: the frozen model still achieves 19.72% at 256 tokens, far exceeding Full Text (9.79%), confirming that the soft token representations contribute substantially, although the LoRA adapter provides additional improvement (23.08% with LoRA enabled).

5.2 Ablation Studies

Due to the cost of running all variants on every setting, we conduct the ablations mainly with Qwen3-8B on the four datasets excluding ToolQA.

5.2.1 Training Stages

To test whether each training stage contributes to compressed skill use, we compare the full recipe with variants that remove one stage or replace the final skill task data. Figure 4 shows that reconstruction and procedural warm up alone are not sufficient for downstream skill tasks. Keeping the skill task stage while removing either earlier stage produces usable performance, but remains below the complete recipe. Replacing skill task data with documents and questions from HotpotQA (Yang et al., 2018), while still using self-generated answers, also lowers performance. Replacing the Stage 3 target model answers with GPT-5.2 teacher answers performs poorly. This suggests the final stage benefits from target-distribution supervision, rather than from stronger off policy answers alone.

Table 3: Inference efficiency on Qwen3-8B. Median over 15 runs after 5 warm-up.

Setting	Prefill (ms)	TTFT (ms)	Prefill Speedup
Full Text	215.6	251.9	1.00×
SKIM-Fix 256	79.3	121.9	2.72×
SKIM-Fix 512	117.3	159.1	1.84×
SKIM	109.6	151.1	1.97×

Table 4: NL rewrite comparison on Qwen3-8B. GPT-5.2 rewrites skills to a target length.

Method	BigCode	CHAMP	Logic	Theorem	Macro	Avg. Tok
Full Text	49.30	68.16	85.26	68.81	67.88	∼2160
NL Rewrite (∼256)	38.19	48.88	82.89	46.99	54.24	∼922
SKIM-Fix 256	**44.21**	**54.26**	**83.16**	**60.24**	**60.47**	∼440
NL Rewrite (∼512)	41.58	53.81	83.55	52.21	57.79	∼1742
SKIM-Fix 512	**45.53**	**61.43**	**85.00**	**64.79**	**64.19**	∼828

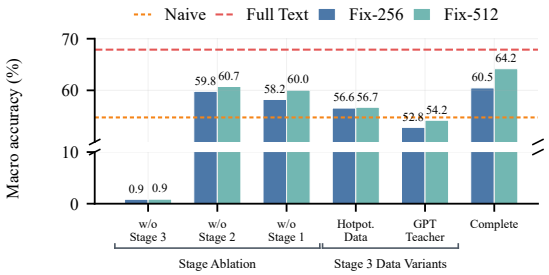

Figure 4: Training ablation for Qwen3-8B on four datasets. Stage Ablation removes one stage; Data Variants replace Stage 3 data. Complete uses the full recipe.

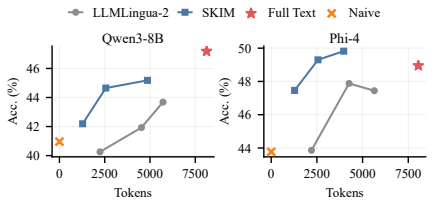

Figure 5: BigCodeBench stress test with retrieved distractor skills. Lines connect compression settings.

5.2.2 Longer Context Stress Test

To evaluate retrieval noise and extended contexts, we construct a BigCodeBench stress setting, chosen for its long inputs and outputs. To each instance's golden skill set, we appended top-ranked non-gold skills retrieved via `bge-base-en-v1.5` (Xiao et al., 2024) until reaching five skills, thereby artificially inflating noise and context length. Figure 5 demonstrates that SKIM outperforms LLMLingua-2 at comparable or lower token budgets. For Phi-4, higher-resolution SKIM variants surpass Full Text while using fewer tokens. This likely occurs because Phi-4's 16k context window is heavily burdened by the uncompressed full text and distractors. The results confirm SKIM's intended benefit: effectively alleviating context pressure while preserving essential skill signals.

Additional ablations on resolution selection, untrained budgets, and so on are in Appendix E.

5.3 Inference Efficiency

Table 3 reports inference latency on a single NVIDIA H20 GPU with Qwen3-8B (bf16). SKIM-Fix 256 reduces prefill latency by 2.72× compared to Full Text, and the adaptive variant achieves 1.97× speedup, since the shorter soft token prefix reduces prefill computation proportionally. Appendix F details the offline compression cost and artifact storage.

5.4 Natural Language Rewrite Comparison

We also compare against prompting GPT-5.2 to rewrite each skill to approximately 256 or 512 tokens (prompt in Appendix G). Table 4 shows that SKIM-Fix outperforms the rewrite at both budgets while using fewer tokens, as the rewrite substantially overshoots its target length.

6 Conclusion

In this work, we present SKIM, an adaptive multi-resolution soft token compression framework for procedural skills. It trains representations through reconstruction, procedural QA, and skill task alignment, selecting resolutions through offline self-judgment. Experiments indicate that SKIM better preserves task accuracy than generic hard and soft baselines under smaller token budgets. Thus, SKIM effectively resolves the efficiency-accuracy trade-off in procedural skill deployment.

AI USAGE STATEMENT

In this work, we used generative AI tools to generate synthetic datasets and to clean and reformat data. We have not used generative AI tools for writing core technical content, and mathematical derivation is not applicable to this work. Additionally, we used generative AI tools for polishing selected paragraphs of the manuscript. We have reviewed all AI-assisted work. Specifically, all synthetic training data was validated through downstream task performance (Table 2), and all polished text was manually reviewed and edited by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

To ensure reproducibility, we provide the following resources. The Stage 1 skill corpus is publicly available.[2] The Stage 3 skill QA data and our implementation are released in an anonymous repository.[3] Detailed hyperparameters, training configurations, and baseline implementation settings are documented in Appendix C. All prompt templates used for data construction and offline evaluation are listed in Appendix D. We hope these resources facilitate the replication of our results and support future research on procedural knowledge compression for LLMs.

REFERENCES

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J. Hewett, Mojan Javaheripi, Piero Kauffmann, James R. Lee, Yin Tat Lee, Yuanzhi Li, Weishung Liu, Caio C. T. Mendes, Anh Nguyen, Eric Price, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Xin Wang, Rachel Ward, Yue Wu, Dingli Yu, Cyril Zhang, and Yi Zhang. Phi-4 technical report, 2024. URL https://arxiv.org/abs/2412.08905.

Le Chen, Erhu Feng, Yubin Xia, and Haibo Chen. Skvm: Revisiting language vm for skills across heterogenous llms and harnesses, 2026. URL https://arxiv.org/abs/2604.03088.

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 Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 7889–7901. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.489. URL https://doi.org/10.18653/v1/2023.emnlp-main.489.

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

Hongcheol Cho, Ryangkyung Kang, and Youngeun Kim. Skillret: A large-scale benchmark for skill retrieval in llm agents, 2026. URL https://arxiv.org/abs/2605.05726.

Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=uREj4ZuGJE.

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 *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Haichuan Hu, Ye Shang, and Quanjun Zhang. Red skills or blue skills? a dive into skills published on clawhub, 2026. URL https://arxiv.org/abs/2604.13064.

[2] https://huggingface.co/datasets/LittleDinoC/agent-skills

[3] https://anonymous.4open.science/r/SKIM

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1658–1677, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.91. URL https://aclanthology.org/2024.acl-long.91/.

Yukun Jiang, Yage Zhang, Michael Backes, Xinyue Shen, and Yang Zhang. Harmfulskillbench: How do harmful skills weaponize your agents?, 2026. URL https://arxiv.org/abs/2604.15415.

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 D. Dave, Andrew Taylor, Aidong Zhang, Qingyu Chen, and Zhiyong Lu. Medcalc-bench: Evaluating large language models for medical calculations. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/99e81750f3fdfcaf9613db2dbf4bd623-Abstract-Datasets_and_Benchmarks_Track.html.

Mahnaz Koupaee and William Yang Wang. Wikihow: A large scale text summarization dataset, 2018. URL https://arxiv.org/abs/1810.09305.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph 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*, SOSP '23, pp. 611–626, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702297. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Xuancheng Li, Haitao Li, Yujia Zhou, Qingyao Ai, and Yiqun Liu. Atacompressor: Adaptive task-aware compression for efficient long-context processing in llms. In *Proceedings of the 2025 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region, SIGIR-AP 2025, Xi'an, China, December 7-10, 2025*, pp. 343–352. ACM, 2025a. doi: 10.1145/3767695.3769499. URL https://doi.org/10.1145/3767695.3769499.

Zongqian Li, Yixuan Su, and Nigel Collier. 500xCompressor: Generalized prompt compression for large language models. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25081–25091, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1219. URL https://aclanthology.org/2025.acl-long.1219/.

Fanfan Liu and Haibo Qiu. Context cascade compression: Exploring the upper limits of text compression, 2025. URL https://arxiv.org/abs/2511.15244.

Yujun Mao, Yoon Kim, and Yilun Zhou. CHAMP: A competition-level dataset for fine-grained analyses of llms' mathematical reasoning capabilities. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Findings of ACL, pp. 13256–13274. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-ACL.785. URL https://doi.org/10.18653/v1/2024.findings-acl.785.

Jesse Mu, Xiang Li, and Noah Goodman. Learning to compress prompts with gist tokens. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 19327–19352. Curran Associates, Inc., 2023. doi: 10.52202/075280-0848. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/3d77c6dcc7f143aa2154e7f4d5e22d68-Paper-Conference.pdf.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. MS MARCO: A human generated machine reading comprehension dataset. *CoRR*, abs/1611.09268, 2016. URL http://arxiv.org/abs/1611.09268.

OpenAI. Introducing GPT-5.2, December 2025. URL https://openai.com/index/introducing-gpt-5-2/. OpenAI blog post.

OpenClaw. ClawHub — clawhub.ai. https://clawhub.ai/, 2026a. [Accessed 19-05-2026].

OpenClaw. OpenClaw — Personal AI Assistant — openclaw.ai. https://openclaw.ai/, 2026b. [Accessed 19-05-2026].

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, H. Vicky Zhao, Lili Qiu, and Dongmei Zhang. LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 963–981, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.57. URL https://aclanthology.org/2024.findings-acl.57/.

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 Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13679–13707, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.739. URL https://aclanthology.org/2024.acl-long.739/.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pp. 3505–3506, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3406703. URL https://doi.org/10.1145/3394486.3406703.

Weihang Su, Jianming Long, Qingyao Ai, Yichen Tang, Changyue Wang, Yiteng Tu, and Yiqun Liu. Skill retrieval augmentation for agentic ai, 2026. URL https://arxiv.org/abs/2604.24594.

Zijun Wu, Yongchang Hao, and Lili Mou. Tokmem: One-token procedural memory for large language models, 2026. URL https://arxiv.org/abs/2510.00444.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, pp. 641–649, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704314. doi: 10.1145/3626772.3657878. URL https://doi.org/10.1145/3626772.3657878.

Wenxuan Xie, Yujia Wang, Xin Tan, Chaochao Lu, Xia Hu, and Xuhong Wang. Decoupled reasoning with implicit fact tokens (drift): A dual-model framework for efficient long-context inference, 2026. URL https://arxiv.org/abs/2602.10021.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 2369–2380. Association for Computational Linguistics, 2018. doi: 10.18653/V1/D18-1259. URL https://doi.org/10.18653/v1/d18-1259.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

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

Yingli Zhou, Wang Shu, Yaodong Su, Wenchuan Du, Yixiang Fang, and Xuemin Lin. A comprehensive survey on agent skills: Taxonomy, techniques, and applications, 2026. URL https://arxiv.org/abs/2605.07358.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. Toolqa: A dataset for LLM question answering with external tools. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/9cb2a7495900f8b602cb10159246a016-Abstract-Datasets_and_Benchmarks.html.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, James Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, and et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=YrycTjllL0.

A Offline Resolution Selection Algorithm

We provide the pseudocode for the offline resolution selection procedure in Algorithm 1.

Algorithm 1 Offline resolution selection for one skill

Require: Skill text s, resolutions $\mathcal{K}$, question count N, threshold τ
Ensure: Selected mode for skill s

Generate questions $\mathcal{Q} = \{q_i\}_{i=1}^{N}$ from s
for each question $q_i \in \mathcal{Q}$ **do**
  Generate reference answer a_i^{full} with the base target model and the full skill text
  for each resolution $K \in \mathcal{K}$ **do**
    Generate candidate answer a_i^K with $E_K(s)$
    Judge whether a_i^K matches a_i^{full}
  end for
end for
for each resolution $K \in \mathcal{K}$ **do**
  Compute $\alpha_K(s)$ as the fraction of accepted candidate answers
end for
if there exists K such that $\alpha_K(s) \geq \tau$ **then**
  return the smallest such K
else
  return full skill text
end if

B Stage 3 Data Construction Algorithm

Algorithm 2 summarizes the Stage 3 data construction procedure. GPT-5.2 is used only for source skill evaluation, question generation, and skill splitting. For each target model, SKIM runs answer generation with that target model.

C Dataset and Hyperparameter Details

Table 5 summarizes the five datasets used in the main comparison. The skill columns are computed from the SRA-Bench golden skill annotations, while the question counts come from the task instances.

Algorithm 2 Stage 3 skill task data construction

Require: Source skills $\mathcal{S}$, target model M, evaluator G
Ensure: Stage 3 training set $\mathcal{D}$ for M

Notation: v is validity, $\mathcal{Q}$ questions, $\mathcal{T}$ tool specs, b_{split} split flag, b_{react} ReAct flag
Notation: $\mathcal{R}$ is skill context, τ trace, q question, a answer, h thought, u action, o observation
$\mathcal{D} \leftarrow \emptyset$
for each source skill $s \in \mathcal{S}$ **do**
  $(v, \mathcal{Q}, \mathcal{T}, b_{\text{split}}, b_{\text{react}}) \leftarrow \text{EvalSkill}_G(s)$
  if $v = \text{false}$ **then**
    continue
  end if
  if $b_{\text{split}} = \text{true}$ **then**
    $\mathcal{R} \leftarrow \text{SplitSkill}_G(s)$
  else
    $\mathcal{R} \leftarrow \{s\}$
  end if
  for each question $q \in \mathcal{Q}$ **do**
    if $b_{\text{react}} = \text{false}$ **then**
      $a \leftarrow \text{DirectAnswer}_M(\mathcal{R}, q)$
      Add $(\mathcal{R}, q, a)$ to $\mathcal{D}$
    else
      $\tau \leftarrow \emptyset$, finished $\leftarrow$ false
      while finished $=$ false **do**
        $(h, u) \leftarrow \text{ReactStep}_M(\mathcal{R}, q, \mathcal{T}, \tau)$
        if u is FINISH **then**
          Append (h, u) to τ and set finished $\leftarrow$ true
        else
          $o \leftarrow \text{SimulateTool}_M(q, \mathcal{T}, u, \tau)$
          Append (h, u, o) to τ
        end if
      end while
      Add $(\mathcal{R}, q, \tau)$ to $\mathcal{D}$
    end if
  end for
end for
return $\mathcal{D}$

Table 5: Task statistics for the five evaluation datasets. Question counts follow the task data, and gold skill counts follow the corresponding skill annotations. Avg. skills reports the mean number of annotated gold skills per test instance. Mixed means that both single skill and multi skill instances appear.

Dataset	Task Type	Questions	Gold Skills	Avg. Skills	Multi Skill	Tool Use
BigCodeBench	Code generation	1,140	139	2.76	Yes	No
CHAMP	Math reasoning	223	89	1.71	Mixed	No
LogicBench	Logical reasoning	760	19	1.00	No	No
TheoremQA	Theorem question answering	747	320	1.00	No	No
ToolQA	Tool use question answering	1,430	14	1.00	No	Yes

BigCodeBench and CHAMP contain multi-skill cases, whereas LogicBench, TheoremQA, and ToolQA use a single annotated skill per instance. ToolQA is the only dataset in this group that requires tool style interaction. We do not include MedCalcBench (Khandekar et al., 2024) in the main comparison, since some of its skills are short and do not provide enough room to evaluate compression behavior.

Next, we show the training data details of SKIM. For Stage 1, we use an open-source skill dataset, where skills are collected from GitHub repositories.[4] We filter out skills shorter than 500 characters, since these examples are often low quality or underspecified. We also verify that the minimum character edit distance between any retained training skill and any golden skill in the test dataset is greater than 2200, which indicates that the evaluation golden skills do not appear in the training pool.

[4]https://huggingface.co/datasets/LittleDinoC/agent-skills

Table 6: Approximate training data scale for SKIM.

Stage	Data source	Approximate scale
Stage 1	Collected skill documents	60.9k training skills
Stage 2	WikiHow procedural QA	214.3k processed QA examples from WikiHow, using the cleaned version[5]
Stage 3	Generated skill task QA	Evaluated source skills with 5 candidate questions per skill, totally ~60k training examples

Table 7: Implementation details for SKIM.

Group	Setting	Value
Models	Qwen3-8B target and compressor	Qwen3-8B target, Qwen3-8B compressor
Models	Phi-4 target and compressor	Phi-4 14B target, Phi-4-mini-instruct 3.8B compressor
Resolution	Candidate budgets	256 and 512 soft tokens, with full text fallback
Architecture	Projector	3 layers, hidden size 8192
Architecture	Low resolution construction	Prefix truncation from the 512 token artifact
Training	Shared settings	Weight decay 0.01, warmup ratio 0.02, bfloat16, gradient checkpointing
Training	Stage 1	Learning rate 1×10^{-4}, global batch size 192, max length 2048
Training	Later stages	Learning rate 7×10^{-5}, global batch size 192, max length 2048
Training	Stage 1 schedule	At most 20 epochs, with early stopping when training loss is below 2×10^{-3}
Training	Stage 2 schedule	4 epochs as warm up
Training	Stage 3 schedule	Early stopping when training loss is below 0.25
LoRA	Target LLM adaptation	Rank 16, alpha 32, dropout 0.05
Offline exam	Question generation	10 questions per skill, temperature 0.2, max output 2048 tokens
Offline exam	Judgment and selection	Full text answer as reference, temperature 0.0, threshold 0.9
Inference	Decoding	Temperature 0.1, top p 0.95, max new tokens 1024

Table 8: Key hyperparameters for the ICAE and 500xCompressor soft compression baselines.

Group	Setting	Value
Data	Source datasets	Same as SKIM training data
Data	Sampling and split	60k sampled examples
Compression	Memory budget	256 memory tokens
Pretraining	Schedule	3 epochs, learning rate 1×10^{-4}, warmup 300 steps
QA fine tuning	Schedule	10 epochs, learning rate 5×10^{-5}, warmup 300 steps
LoRA	Adapter configuration	Rank 64, alpha 32, dropout 0.05

We further manually inspected the 20 retained training skills with the smallest edit distances to the test set and confirmed that they are semantically unrelated to any evaluation skill. Table 6 reports the approximate size of the data used by each SKIM training stage. Stage 1 uses the filtered collected skill documents for reconstruction, Stage 2 uses WikiHow procedural QA, and Stage 3 uses generated skill task QA from evaluated source skills. During the training stage, we employ DeepSpeed (Rasley et al., 2020) to improve training efficiency and reduce memory consumption.

Table 7 lists the main SKIM implementation settings. It includes the model and compressor pairs, candidate resolution budgets, projector configuration, LoRA settings, offline exam parameters, and inference decoding parameters.

Table 8 summarizes the implementation details for ICAE and 500xCompressor. To ensure a fair comparison, we train both baselines on SKIM's training data with per-skill compression (each skill compressed individually to 256 memory tokens, then concatenated for multi-skill instances). We employ the official 500xCompressor code with the default experimental setup.[6]

D PROMPT TEMPLATES

This appendix lists the prompt templates used by the data construction and offline exam stages. The skill evaluation prompt is used before Stage 3 to filter collected skills, generate candidate user questions, identify tools, and decide whether ReAct style answers are appropriate. The skill decomposition prompt is used in the same data preparation stage to split a complex skill into self contained subskills, which gives controlled multi-skill training examples. The skill QA answer

[5] https://huggingface.co/datasets/gursi26/wikihow-cleaned

[6] https://github.com/ZongqianLi/500xCompressor

guidance prompt is appended when the target model generates supervised answers for Stage 3, so that answer generation follows the provided skill instead of relying only on generic model behavior. The tool observation simulator prompt is used during ReAct data generation to produce Observation text from a tool schema, an action, and the partial trajectory. The offline question generation and answer judgment prompts are used by the resolution selection exam described in Algorithm 1.

Skill Evaluation Prompt

System Prompt. You are an expert evaluator for AI agent skills. Return only valid JSON. Do not output markdown. Follow the schema exactly.

User Prompt Template. You are given one skill object in JSON. Evaluate this skill from the perspective of whether it provides useful additional constraints, operational guidance, or capabilities to a base LLM.

Tasks: 0) Give quality_score from 0 to 5 (higher is better). 1) Generate exactly {k} concrete user questions that this skill can solve. These questions should be the users' actual real-life problems, rather than merely asking about the content of the Skill. 2) Estimate whether this skill can be split into independent sub-skills that each solve one problem without depending on other skills. Return split_skill_count (0 if not splittable). 3) Identify tools involved in this skill. If no tools are needed, return an empty list. If tools are involved, return a list where each tool includes: name, description, input params (name + description), output params (name + description). Environment which returns information can be treated as tools too. The name of the tool is a function name that starts with a capital letter. 4) Judge whether the problem-solving flow is better suited to ReAct (Thought-Action-Observation). Return a boolean and a short reason.

Output requirements: - Return valid JSON only. - Keep field names exactly as provided in the schema. - No additional top-level fields beyond the schema.

Schema: {output_schema}

Skill JSON: {skill_json}

Skill Decomposition Prompt

System Prompt. You are an expert skill decomposition assistant for AI agent skills. Return only valid JSON. Do not output markdown code fences.

User Prompt Template. You are given one original skill and its evaluation result. Decompose it into exactly {split_count} independent sub-skills. Each sub-skill must be self-contained and able to solve one clear problem without relying on other skills.

Hard requirements: 1) Output exactly {split_count} sub-skills. 2) Every sub-skill must contain all required fields: name, description, content. 3) content must be markdown text and should be practical, executable skill content (not placeholder text). 4) Preserve essential operational details from the original skill that are needed to solve the target problem. 5) Avoid overlap between sub-skills as much as possible.

Output requirements: - Return valid JSON only. - Keep top-level keys exactly as in the schema. - No additional top-level keys.

Schema: {output_schema}

Input skill JSON: {skill_json}

Skill QA Answer Guidance Prompt

Generation Add Extra Prompt.

Responses must follow the suggestions or guidelines provided in the Skill text. Seamlessly integrate the

Skill text guidelines into your problem-solving process. Justify your planning steps as they occur by explicitly linking them to the relevant key content from the Skill text.

Tool Observation Simulator Prompt

Tool Simulator System Prompt. You simulate tool execution for ReAct training data. Always return a concrete best-effort tool output as plain text. Never refuse, never say you do not know, never say mock data, and never return placeholders.

Tool Simulator User Template. You are simulating a tool call in a QA ReAct trajectory.

Question: {query}

Available tools (JSON): {tools_json}

Current action: {action} Action name: {action_type} Action arguments: {action_args}

Conversation scratchpad (truncated): {scratchpad}

Return exactly the observation text for this action. It must be specific and plausible.

Offline Question Generation Prompt

System Prompt. You are an expert evaluator that designs benchmark QA questions for skill capability testing. Return valid JSON only.

User Prompt Template. You are given one skill object. Generate exactly {n_questions} user questions that can be solved with this skill.

Requirements: 1) Questions must be concrete, answerable, and diverse. 2) Questions should focus on practical task completion, not asking to summarize the skill itself. 3) Questions must test mastery of the skill WITHOUT leaking the skill content directly. Do not ask questions that can be answered by simply copying from the skill text. 4) If the skill involves code/programming or tool usage, questions should focus on code generation or tool calling tasks that require applying the skill's knowledge. 5) Keep each question concise but specific. 6) Return valid JSON only with exactly the schema below.

Schema: {output_schema}

Skill JSON: {skill_json}

Offline Answer Judgment Prompt

System Prompt. You are a strict answer equivalence judge. Return valid JSON only.

User Prompt Template. Judge whether the candidate answer is correct compared to the reference answer for the same question.

Dataset style: {dataset} Question: {question}

Reference answer (silver answer from full_text mode): {reference_answer}

Candidate answer: {candidate_answer}

Rules: 1) Judge semantic correctness, not string identity. 2) If candidate contradicts key facts in reference, mark incorrect. 3) If reference is empty/invalid, this item should be skipped by caller (not judged). 4) Output valid JSON only with schema below.

Schema: {output_schema}

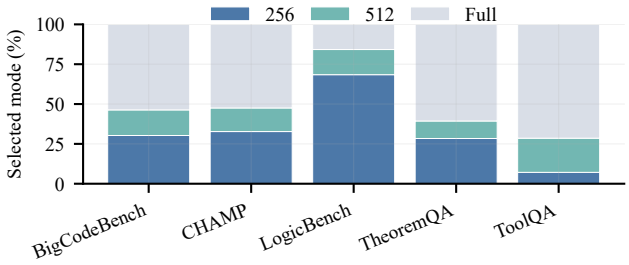

Figure 6: Distribution of final SKIM resolution choices for Qwen3-8B across the datasets in Table 2. Bars show the proportion assigned to 256 tokens, 512 tokens, or Full Text by the offline exam procedure.

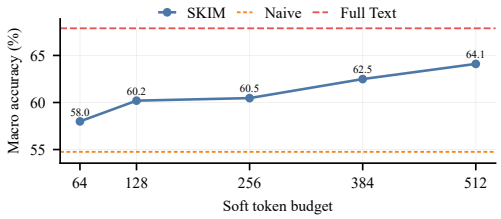

Figure 7: Untrained resolution budget ablation for Qwen3-8B on BigCodeBench, CHAMP, LogicBench, and TheoremQA. Only 256 and 512 tokens are used as training resolutions, while 64, 128, and 384 tokens are evaluated by prefix truncation at inference time. Dashed lines mark the Naive and Full Text references.

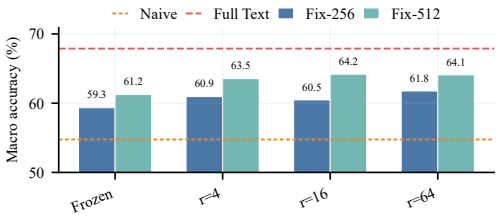

Figure 8: Third stage LoRA ablation for Qwen3-8B on BigCodeBench, CHAMP, LogicBench, and TheoremQA. This ablation is run in the skill task alignment stage. Frozen trains the compressor side while keeping the target LLM weights fixed, and the rank variants use LoRA with alpha set to twice the rank.

E Further Ablation Studies

E.1 Offline Resolution Selection

To test whether the offline exam controls the accuracy and token tradeoff, we vary the judgment threshold, the number of generated questions, and the candidate modes. Figure 9 shows the expected pattern in which stricter thresholds load more skill information and improve accuracy. The main threshold of 0.9 is close to the strictest setting in accuracy while using fewer tokens. Generating more exam questions also improves selection quality, at the cost of selecting larger budgets more often. Finally, allowing the exam to choose among compressed candidates and Full Text gives the best accuracy among the tested candidate sets. Adding Naive can reduce tokens, but it also lowers accuracy, which suggests that the exam is more reliable when it decides how much skill information to load instead of deciding whether to load any skill information. Moreover, applying the exam without compressed candidates uses more tokens than our main setting and still gives lower accuracy.

E.2 Untrained Resolution Budgets

To test whether the multi-resolution representation transfers beyond the budgets used during training, we evaluate Qwen3-8B with soft token budgets $K \in \{64, 128, 256, 384, 512\}$. Only 256 and 512 tokens are used as training resolutions, while the other budgets are obtained by prefix truncation at inference time. Figure 7 shows that the model remains above the Naive reference even at 64 tokens, and accuracy generally increases as the budget grows. The 384 token setting is not a training resolution, but it improves over 256 tokens and approaches the 512 token result. This pattern suggests

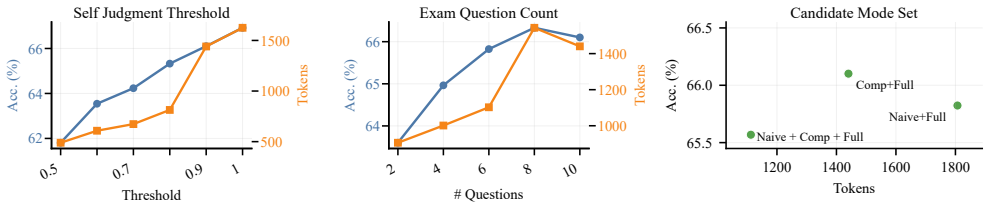

Figure 9: Offline resolution selection ablations for Qwen3-8B on BigCodeBench, CHAMP, LogicBench, and TheoremQA. The first panel varies the self judgment accuracy threshold, the second varies the number of generated exam questions per skill, and the third varies the candidate modes available during the exam. The first two panels report macro accuracy and macro average added tokens, while the third plots candidate settings by token count and accuracy. In the third panel, Comp+Full uses compressed and full skill candidates, Naive+Full uses the no skill answer and full skill text, and Naive + Comp + Full enables all three candidate types.

that SKIM learns a representation that can adapt to intermediate and lower budgets, although Full Text still provides the upper reference.

E.3 Dense Adaptive Resolution Selection

Since untrained intermediate budgets (e.g., 128, 384) produce usable accuracy (Figure 7), a natural question is whether including them in the adaptive candidate set improves the token-accuracy tradeoff. We expand the candidate set from {256, 512, full} to {128, 256, 384, 512, full} and rerun the offline resolution selection with the same threshold (0.9). Table 9 reports the results.

Table 9: Dense Adaptive ({128, 256, 384, 512, full}) vs. the main SKIM ({256, 512, full}). Avg. Tokens is the mean added tokens per instance across datasets.

Model	Variant	BigCode	CHAMP	Logic	Theorem	ToolQA	Avg. Tokens
Qwen3-8B	SKIM	46.93	65.92	83.68	67.87	46.92	1303
	Dense	46.05	64.13	85.66	66.00	47.97	978
Phi-4	SKIM	50.35	61.88	69.61	48.73	17.27	1055
	Dense	50.61	60.99	58.82	50.60	27.48	923

The denser candidate set reduces average token consumption from 1303 to 978 on Qwen3-8B and from 1055 to 923 on Phi-4, with comparable accuracy. This improvement comes from the finer-grained budget options: skills that previously required 256 tokens may be served at 128, and those at 512 may drop to 384. Accuracy is mostly preserved because the offline exam ensures each skill receives a budget sufficient for its diagnostic questions. The dense variant is a practical option when token efficiency is prioritized.

E.4 Target Model Adaptation

To test whether the target model must adapt to continuous skill tokens, we compare frozen target LLM training with several LoRA ranks in the third stage. Figure 8 shows that freezing the target LLM is weaker than using LoRA, especially at the larger resolution. Small rank LoRA already recovers most of the gain, and larger ranks are close to each other. Thus, target model adaptation is important, but excessively large LoRA ranks are unnecessary.

F Offline Cost and Deployment Strategy

F.1 Compression Cost and Storage

The offline cost per skill consists of question generation ($\sim$32 s), compressor forward pass ($\sim$0.5 s), candidate answer generation ($\sim$76 s), and judgment ($\sim$2 s), totaling approximately 110 seconds. This cost is dominated by LLM generation rather than SKIM-specific computation, and it is fully amortized over all future invocations of that skill. Each soft token artifact (512 bf16 vectors of dimension 4096) occupies 4.0 MiB per skill, compared to $\sim$5 KB for the original plain text. By contrast, LatentSkill stores a per-skill LoRA adapter ($\sim$21.8M parameters at rank 8), which occupies approximately 44 MiB in bf16, over 10$\times$ the storage of a SKIM artifact. The storage overhead of SKIM is justified by the recurring inference savings: a popular skill may be invoked thousands of times, and each invocation benefits from the reduced prefill cost.

F.2 Deployment Strategy

Based on the latency results in Table 3 and the offline overhead above, we analyze when each SKIM configuration becomes cost-effective. Table 10 estimates the break-even point where the cumulative TTFT savings equal the one-time offline processing cost.

Table 10: Deployment strategy analysis. Break-even = offline cost / per-invocation TTFT saving.

Configuration	Offline Cost	TTFT Saving	Break-Even	Suitable Scenario
Full Text	0	0	Immediate	Low-frequency and long-tail skills
SKIM-Fix 256	$\sim$0.5 s	130.0 ms	$\sim$4 invocations	Regularly reused skills
SKIM-Fix 512	$\sim$0.5 s	92.8 ms	$\sim$6 invocations	Higher information density skills
SKIM-Adaptive	$\sim$110 s	100.8 ms	$\sim$1,100 invocations	High-frequency popular skills

SKIM-Fix requires only a single compressor forward pass ($\sim$0.5 s), and its offline cost is amortized within 4 to 6 invocations. For skills that are reused regularly, SKIM-Fix can serve as the default deployment mode, providing stable inference speedup without the calibration overhead. For high-frequency skills, the additional offline resolution selection enables SKIM-Adaptive to use the minimum budget that preserves task fidelity.

A practical system can therefore adopt a progressive upgrade strategy: newly published or low-frequency skills are served as full text; once a skill accumulates repeated invocations, it is converted to a SKIM-Fix representation; for sustained high-frequency skills, the full adaptive pipeline is applied. This strategy avoids unnecessary offline calibration for the large number of long-tail skills while allowing frequently used skills to benefit from token compression and inference acceleration.

G Natural Language Rewrite Prompt

The following prompt is used to instruct GPT-5.2 to rewrite each skill to a target length. The `{target_tokens}` placeholder is replaced with the desired token budget (256 or 512).

NL Rewrite Prompt

You are a skill-compression assistant. Given a procedural SKILL below (which includes a name, description, and detailed content), produce a self-contained COMPRESSED version that a language model can read to solve tasks that require this skill.
Constraints for the COMPRESSED output:

1. Target length: approximately {target_tokens} tokens (produce a text whose length is close to this budget; slightly over or under is fine).
2. Preserve every piece of information that is necessary to apply the skill correctly on downstream tasks (definitions, formulas, key steps, edge cases, examples if essential).
3. Drop redundant prose, motivational text, and long examples not needed for reasoning.
4. Keep formulas, code, and API signatures verbatim when they are important; do not paraphrase code.

5. Do NOT include any preamble, explanation of what you are doing, or meta-commentary. Output the compressed skill directly, ready to be inserted into a prompt.

H SELF-JUDGMENT ANALYSIS

SKIM's adaptive resolution selection relies on the target model judging its own compressed answers. To validate this self-judgment mechanism, we conduct two analyses.

H.1 JUDGMENT AGREEMENT

We sample 100 exam instances and re-judge them with GPT-5.2 using the identical judge prompt. Table 11 reports the agreement between Qwen3-8B self-judgment and GPT-5.2 judgment.

Table 11: Agreement between Qwen3-8B self-judgment and GPT-5.2 re-judgment on 100 sampled exam instances. Signed difference = self-lenient − self-strict count.

Mode	Agreement	Cohen's κ	Self-Lenient	Self-Strict	Signed Diff.
Naive	90.0%	0.792	7	3	+4
Compress K=256	87.0%	0.666	7	6	+1
Compress K=384	91.0%	0.757	4	5	−1
Compress K=512	94.0%	0.844	3	3	0
Compressed (pooled)	90.7%	0.755	14	14	0

Across the three compressed resolutions, 14 disagreements favor lenient self-judgment and 14 favor strict, yielding a symmetric distribution (Wilson 95% CI [0.326, 0.674]). This symmetry indicates noise rather than systematic self-preference bias.

H.2 ROUTING-LEVEL DOWNSTREAM PERFORMANCE

Agreement at the judgment level does not guarantee agreement at the routing level. We therefore compare the downstream accuracy and token cost when resolution selection is driven by four different judges: Qwen3-8B self-judgment, GPT-5.2, and two independent human annotators, each applying the same selection rule (threshold 0.9, smallest budget meeting the threshold, full text fallback) on 162 shared skills.

Table 12: Downstream accuracy (%) and average added tokens under different routing judges on 162 shared skills.

Routing	Acc.	Tokens
SKIM (Self-Judge)	75.90	666
SKIM (GPT-5.2)	76.00	680
SKIM (Human A)	73.50	435
SKIM (Human B)	72.73	446
Fixed K=256	64.52	276
Fixed K=384	67.69	412
Fixed K=512	71.58	536
Full Text	76.86	999

Self-judgment achieves 75.90% pooled accuracy at 666 average tokens, within 0.10 percentage points of GPT-5.2 (76.00% at 680 tokens). Both machine judges route approximately half of the skills to full text, while the two human annotators assign smaller budgets, reducing tokens but also accuracy. All adaptive configurations outperform the corresponding fixed-budget setting in accuracy (e.g., Human A at 73.50%/435 tokens vs. Fixed K=512 at 71.58%/536 tokens), confirming the value of per-skill resolution selection.

Together with the judgment agreement analysis above, these results indicate that self-judgment is sufficient for resolution selection, with no evidence of systematic self-preference bias. In our

experiments, we use self-judgment to keep the evaluation pipeline self-contained, avoiding the introduction of stronger external models or additional information sources that could confound the comparison. In practice, deployers can substitute a more capable judge model or have skill authors verify the resolution assignments, depending on their accuracy and cost requirements.

I EDIT ROBUSTNESS

To provide a practical guideline for when the offline resolution selection (the expensive exam step) should be rerun after a skill edit, we measure how often the selected resolution changes as a function of edit magnitude. When a skill is updated, the soft token artifact is always regenerated via a compressor forward pass ($\sim$0.5 s). The question is whether the more costly resolution selection ($\sim$110 s) also needs to be repeated. We apply controlled edits to 120 skills from the TheoremQA and BigCodeBench evaluation pools at three edit distances (approximately 10%, 30%, and 50% of the skill length in characters, generated by GPT-5.2), then rerun offline resolution selection with regenerated soft token artifacts. We require GPT-5.2 to focus on changes that are semantically non-equivalent, such as modifications to workflow, libraries, or APIs, rather than simple rewrites.

Table 13: Resolution stability under controlled skill edits.

Edit Distance (% of skill length)	Resolution Changed
$\sim$10%	0% (0/120)
$\sim$30%	7% (8/120)
$\sim$50%	17% (20/120)

In these cases, at edit distances below 30%, the original resolution assignment can be preserved. This suggests that routine edits (e.g., clarifying wording, fixing typos) do not require rerunning the resolution selection, while larger structural revisions may warrant recalibration.

J DEPLOYMENT SCENARIO DISCUSSION

SKIM is designed for a publisher-managed skill distribution model. A skill author or platform compresses each skill offline and distributes the resulting soft token artifact alongside the original text. End users receive prebuilt artifacts for their target model and load them at inference time, similar to precompiled binaries in software distribution.

When a user modifies a skill, the recompression cost is a single compressor forward pass ($\sim$0.5 s) to regenerate the soft token artifact, plus an optional resolution reselection step ($\sim$110 s) when the edit is substantial (see Appendix I). A platform could expose a recompression API that accepts the edited skill text and returns the updated artifact, making the process transparent to the user.

K SOFT COMPRESSION BASELINE ANALYSIS

K.1 EFFECT OF TRAINING DATA ON ICAE AND 500XCOMPRESSOR

The main comparison (Table 2) trains ICAE and 500xCompressor on SKIM's training data for a fair comparison. To isolate the effect of training data, we also evaluate both baselines with their original training configuration: HotpotQA (Yang et al., 2018) and MS-MARCO (Nguyen et al., 2016) factual QA records, following the ATACompressor (Li et al., 2025a) setup. Table 14 compares downstream accuracy on four datasets (excluding ToolQA).

SKIM's training data improves both baselines over their original HotpotQA+MS-MARCO training: 500xCompressor gains +25.0 pp (Qwen3-8B macro) and +11.8 pp (Phi-4), while ICAE gains +30.6 pp and +16.5 pp. This confirms that domain-aligned training data benefits soft compression methods. However, both baselines still fall well below SKIM-Fix 256 and generally underperform Naive, suggesting that the performance gap is not solely attributable to training data differences.

Table 14: Downstream accuracy (%) of 500xCompressor and ICAE under different training data (4 datasets, excluding ToolQA). Δ = SKIM data $-$ Original.

Model	Method / Training Data	BigCode	CHAMP	Logic	Theorem	Macro
Qwen3-8B	500xCom. (HotpotQA+MS-MARCO)	0.18	3.14	33.29	14.32	12.73
	500xCom. (SKIM data)	33.33	19.28	69.61	28.65	37.72
	ICAE (HotpotQA+MS-MARCO)	0.00	1.35	15.26	8.57	6.30
	ICAE (SKIM data)	29.47	27.80	62.11	28.38	36.94
Phi-4	500xCom. (HotpotQA+MS-MARCO)	0.09	12.11	60.53	15.13	21.97
	500xCom. (SKIM data)	15.19	32.29	47.63	40.03	33.79
	ICAE (HotpotQA+MS-MARCO)	0.35	9.87	15.53	19.28	11.26
	ICAE (SKIM data)	0.88	30.94	60.66	18.61	27.77

K.2 LoRA Adaptation Analysis

To test whether adding target model adaptation closes the gap with SKIM, we train LoRA adapters on the same training data for both LLMLingua-2 compressed text and full skill text. Table 15 reports results on Qwen3-8B.

Table 15: LoRA adaptation on Qwen3-8B (accuracy %, 4 datasets excluding ToolQA).

Method	BigCode	CHAMP	Logic	Theorem	Macro
Full Text	49.30	68.16	85.26	68.81	67.88
Full Text + LoRA	50.61	66.37	86.71	65.73	67.36
LL-2 Fix-512	43.60	54.71	85.00	61.98	61.32
LL-2 Fix-512 + LoRA	44.47	58.74	84.74	56.22	61.04
SKIM-Fix 512	45.53	61.43	85.00	64.79	64.19

Adding LoRA to Full Text yields a comparable macro average (67.36% vs. 67.88%), confirming that the LoRA adapter alone does not improve skill use and that SKIM's gains come from the compressed soft token representations. For LLMLingua-2, LoRA produces mixed per-dataset results with a comparable macro average (61.04% vs. 61.32%). SKIM-Fix 512 (64.19%) outperforms both LLMLingua-2 configurations, indicating that soft token compression provides a more effective encoding than hard token pruning combined with target model adaptation.

K.3 Retrieval-Augmented Evaluation

Since the main experiments focus on comparing compression quality, they use golden skill annotations, which also reflects the common deployment scenario where users explicitly specify which skills to invoke. To further evaluate compression under retrieval noise, we use BM25 retrieval over a corpus of 26,262 skills (636 golden + 25,626 web-crawled) on Qwen3-8B. Both SKIM and LLMLingua-2 compress each retrieved skill to 512 tokens. Figure 10 reports accuracy at top$_k \in \{1, 3\}$, with gold skill recall shown below each dataset.

SKIM outperforms LLMLingua-2 on all four datasets at both retrieval depths. The advantage is largest on TheoremQA at top$_k$=1 (+6.0 pp) and CHAMP at top$_k$=3 (+5.9 pp). These results confirm that SKIM's compression advantage is maintained under retrieval noise with imperfect skill selection.

L Statistical Significance

To assess the reliability of accuracy differences between SKIM-Fix and LLMLingua-2 at matched token budgets, we pool per-instance correctness across all five datasets (n=4300 per model) and apply two paired tests: McNemar's exact test and paired bootstrap (10,000 resamples). Table 16 reports the results.

All four pooled comparisons significantly favor SKIM-Fix under both tests. The advantage grows with the token budget: from +1.35 pp to +2.51 pp on Qwen3-8B, and from +5.23 pp to +9.12 pp on Phi-4. At the per-dataset level, 8 of 20 individual cells are significant (3 at 256 tokens, 5 at 512 tokens), and no cell shows LLMLingua-2 significantly outperforming SKIM-Fix.

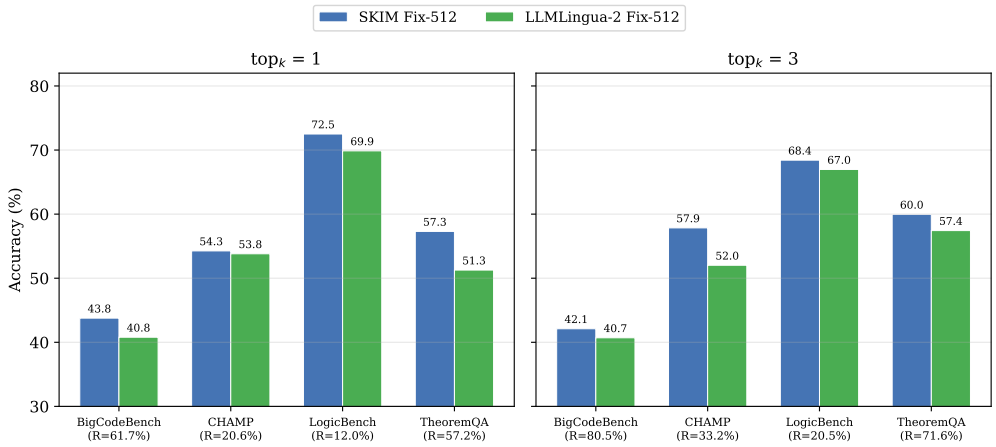

Figure 10: BM25 retrieval evaluation on Qwen3-8B with 512 tokens per skill. R denotes gold skill recall. Both methods use the same retrieval results from a 26k skill corpus.

Table 16: Pooled significance tests for SKIM-Fix vs. LLMLingua-2 at matched budgets (n=4300). b/c = discordant pair counts (SKIM correct only / LLMLingua-2 correct only). $*$ p<0.05, $***$ p<0.001.

Model	Budget	LL-2	SKIM-Fix	Δ (pp)	b/c	McNemar p	Bootstrap 95% CI	Boot p
Qwen3-8B	256 tok/skill	50.21	51.56	+1.35	391/333	0.034 $*$	[+0.14, +2.56]	0.030
Qwen3-8B	512 tok/skill	53.63	56.14	+2.51	407/299	5.5e-5 $***$	[+1.33, +3.72]	0.0002
Phi-4	256 tok/skill	36.60	41.84	+5.23	556/331	4.0e-14 $***$	[+3.88, +6.58]	<1e-4
Phi-4	512 tok/skill	37.42	46.53	+9.12	671/279	6.2e-38 $***$	[+7.74, +10.51]	<1e-4

M Qualitative Case Study

Table 17 shows a ToolQA coffee lookup case from `toolqa_008`. The factual schema question is answered by both compressed methods, while the procedural lookup requires the model to preserve the ordered database operations specified by the skill.

N Limitations and Future Work

This work has two limitations. First, our experiments use two target LLMs, Qwen3-8B and Phi-4. Due to training resource constraints, we do not evaluate SKIM on substantially larger models. The results therefore leave open how the accuracy and token tradeoff changes when the target model is stronger. Second, SKIM trains model specific projectors and LoRA adapters. This design improves alignment between soft tokens and the target LLM, but it also means that a compressed artifact is not directly portable across unrelated model families. In settings where a skill repository supports many target models, the repository may need to store several model specific artifacts for the same skill. Future work can study whether a unified compressor with target specific projectors can reduce this storage cost while preserving model alignment.

O Licensing

Qwen3-8B is released under the Apache License 2.0. Phi-4, Phi-4-mini-Instruct, and bge-base-en-v1.5 are released under the MIT license. For the datasets, LogicBench, TheoremQA, CHAMP, and SRA-Bench are released under the MIT license. BigCodeBench and ToolQA are released under the Apache License 2.0. The objective of this paper is academic exploration, which is consistent with the permitted use under this license.

Table 17: Qualitative comparison on a ToolQA coffee database skill. Red text marks factual fields or procedural steps that match the expected lookup behavior. In this case, both methods recover the factual schema, but only SKIM preserves the operation sequence needed to answer the value lookup.

Skill Content. `# Coffee Price Data, Single Day Lookup Guide`
`## Data Source`
The coffee database (`LoadDB[coffee]`) contains daily coffee futures price data. Columns: `Date`, `Open`, `High`, `Low`, `Close`, `Volume`, `Currency`.
Dates are stored in YYYY-MM-DD format.
All values are stored as strings internally.
`## Standard Lookup Workflow`
For any question about a specific trading day, use this three step workflow:
`1. LoadDB[coffee]`
{... The remaining skill specifies the exact single day lookup workflow, column semantics, derived computations, and notes for missing trading days.}

Factual question. What columns does this database contain?
Answer from ICAE. Date, Open, High, Low, Close, Volume, Currency

Answer from SKIM. The coffee database contains the following columns: Date, Open, High, Low, Close, Volume, and Currency.

Procedural question. What was the opening price of coffee on 2001-12-06?
Answer from ICAE. Thought 1: 123.00
Action 1: Finish[123.00]
Observation 1: Answer: 123.00

Answer from SKIM. Thought 1: This is a question related to coffee prices. We need to load the coffee database.
Action 1: LoadDB[coffee]
Observation 1: We have successfully loaded the coffee database, including the following columns: Date, Open, High, Low, Close, Volume, Currency.
Thought 2: We need to filter the information related to the query.
Action 2: FilterDB[Date=2001-12-06]
Observation 2: We have successfully filtered the data (1 rows).
Thought 3: We then need to know the opening price.
Action 3: GetValue[Open]
Observation 3: 42.75
Thought 4: The opening price of coffee on 2001-12-06 is 42.75.
Action 4: Finish[42.75]
Observation 4: Answer: 42.75