ANCHOR: AMORTIZED KV-CACHE COMPACTION FOR PROCEDURAL SKILLS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM agents increasingly rely on long procedural skills: prompts packaging tool schemas and invocation instructions that consume extensive context on every request. KV-cache compaction promises to reclaim this budget by emitting a compacted KV prefix for unseen skills in a single forward pass, yet existing methods perform poorly on procedural skills, producing missing or inaccurate tool calls. We identify a key training mechanism behind this, the *anchoring effect*: when knowledge-preservation (reconstruction) and instruction-following (tool-calling) objectives are trained sequentially, the second stage erases most of what the first stage stored, leading to sub-optimal final tool-call accuracy. We therefore present ANCHOR (**A**mortized **N**eural KV **C**ompaction with **H**eld-**O**n **R**econstruction), an emitter network trained with on-policy distillation anchored by a low-weight reconstruction objective that keeps stored knowledge intact while improving tool-call fidelity. We also find that evaluating skill compaction also remains an open problem: on some existing agentic benchmarks, strong base models score nearly as well without the skill in context, leaving no headroom to measure what a compacted prefix preserves. We therefore propose DOMAINBENCH: 716 tasks from 100 procedural skills, answerable only with the skill in context and graded on tool calls at three nested granularity: tool name, argument keys, and argument values. At $4\times$ compaction ANCHOR recovers 65.1% and 58.9% of full-context tool-use fidelity on two LLM backbones against 5.5–45.0% for four prior methods.

1 INTRODUCTION

Skill documents are becoming increasingly important for LLM agents to execute specialized work (Li et al., 2025; Lu et al., 2026). These skills often contain tool-calling knowledge like tool schemas, interaction templates and invocation instructions that LLM agents must follow (Schick et al., 2023; Qin et al., 2023; Patil et al., 2023; Yao et al., 2023), and we refer them as *Procedural Skill*s. In real-world agent systems, such skills can be substantially long to sufficiently steer the system towards the optimized user experience (Li et al., 2026). Holding these long procedural skills in plain KV-cache consumes extensive context window for every user request (Pope et al., 2022). To mitigate the problem, KV-cache compaction methods reclaim the context budget by compressing the skill KV cache into a fixed length prefix or at a given compression ratio. However, we empirically find that the existing methods do not optimize for tool-calling scenarios and thus perform poorly on compacting long procedural skills, leading to missing or inaccurate tool calls.

To fill this gap, we propose to meta-trains an emitter network to produce a compacted KV cache for procedural skills in a single forward pass. This amortization allows fast adaption to unseen procedural skills and thus is a more realistic setting (Mu et al., 2024; Ge et al., 2024; Chevalier et al., 2023; Zhang et al., 2024b; O'Neill et al., 2026) than per-document optimization (Eyuboglu et al., 2025; Hardalov et al., 2026). We adopt a two-stage training pipeline where the first stage focuses on preserving the knowledge in the skill document and the second stage trains for instruction-following. Crucially, we observe a tension between these two learning tasks. We find that when the two stages are performed sequentially, the instruction-following task erases most of the knowledge-preservation ability built from the first stage within the early steps. This is problematic because verbatim knowledge-preservation is important essentially for tool-calling skills (Dong et al., 2026; Shan et al., 2025). Based on the insight, we propose ANCHOR (**A**mortized **N**eural KV **C**ompaction with **H**eld-**O**n **R**econstruction). As shown in Figure 1, keeping the knowledge-preservation objec-

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

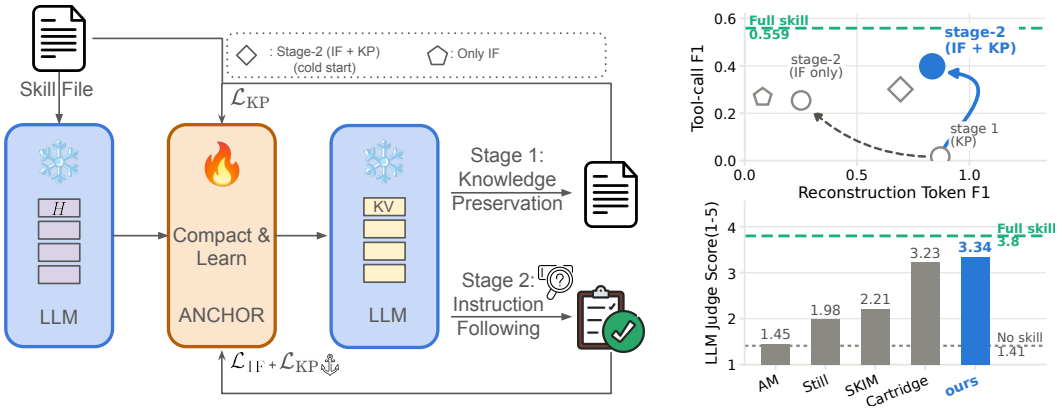


Figure 1: *Left:* Overview of the training pipeline. A frozen LLM encodes the skill file into per-layer hidden states H; ANCHOR reuses the base model's own key/value projections to emit a short KV prefix that seeds the same frozen LLM. $\mathcal{L}_{\text{KP}}$ asks the model to regenerate the document from the prefix (knowledge-preservation), and $\mathcal{L}_{\text{IF}}$ asks it to emit the right tool call (instruction-following). *Top right:* the two objectives are in tension if trained sequentially. IF-only stage 2 erases the stored knowledge, whereas keeping reconstruction active (IF + KP) reaches higher tool-call F_1 and holds the reconstruction token F_1 stage 1 established; training from scratch (open markers) reach neither. *Bottom right:* on the same held-out suite, an LLM judge (1–5) places ANCHOR at 3.34 against a full-context reference of 3.8 and a no-skill floor of 1.41, ahead of four tuned compaction baselines.

tive active at second stage sufficiently fixes the above problem, retaining reconstruction token F1 to the level where the first stage left it and improving the final instruction-following performance. Our discovery also resonates with the study of catastrophic forgetting (McCloskey & Cohen, 1989; Kirkpatrick et al., 2017) and continual-learning (Li et al., 2018; Rebuffi et al., 2017).

A second gap is the absence of a reliable evaluation setup for procedural skill compaction. First, on existing skill usage benchmarks a skill in context sometimes barely moves the agent above the no-skill baseline due to the advancing tool-calling ability of the base model (Han et al., 2026; Chacko et al., 2026; Pysklo et al., 2026). As a result, there is no headroom to show how much a compacted KV prefix can still preserve. Second, few benchmarks measure the tool-call correctness at the granularity which the skill file specifies. Mostly a tool-call is graded by an LLM judge or simple success rate (Qin et al., 2023; Guo et al., 2025), providing limited insight into the diagnosis signal. We therefore present DOMAINBENCH, which places the base model genuinely on the performance floor and grades the performance at three granularity levels of tool name, argument keys and argument values. We carefully curate 716 tasks from 100 procedural skills and the base model without the skill documents has a F1 score below 0.003 on average, while the full-context reference lifts the score over 0.75. In that range ANCHOR recovers 65.1% and 58.9% of full-context tool-use fidelity at 4× compaction on our two backbones, against 5.5% to 45.0% for four previous methods at the same level of compaction. ANCHOR also shows strong compaction capability on two public agentic benchmarks, exceeding the best baseline on each backbone by 5.0–11.7 points on the benchmark metrics. We also show that our approach performs on par with the per-skill optimized method on a long document information retrieval benchmark. Our contributions are as follows.

- We study the *anchoring effect* in KV-cache compaction for procedural skills: training instruction-following on top of a finished knowledge-preservation stage erases what was stored, undermining the final performance.
- We propose DOMAINBENCH, 716 procedural tasks graded on tool-call fidelity at three nested granularities, the tool name, the argument keys and the argument values, and answerable only with the skill in context.
- We propose ANCHOR, an amortized compaction model that emits a per-layer KV prefix for a frozen base model in one forward pass. Extensive experiments across two LLM backbones, four

Table 1: **Method comparison. Amortized** means the method generalizes to unseen documents in a single forward pass. **KV-param** means it emits key–value states directly. **Procedural** means it targets or evaluates procedural tool-use tasks with executable scoring. **Length-adaptive** means the emission budget scales with document length rather than occupying a fixed token count. Rows in bold are the four methods we select as baselines. We choose SKIM over Activation Beacon and KV-Distill because it is a follow-up work directly targeting procedural skills.

Method	Amortized	KV-param	Procedural	Length-adaptive	Core mechanism
Attention Matching [ICML26]	✗	✓	✗	✓	Key selection + Value synthesization
Cartridges [ICLR26]	✗	✓	✗	✗	Per-Context gradient descent on KV
SKIM [arXiv 2026.06]	✓	✗	✓	✓	Adaptive soft tokens learning
Activation Beacon [ICLR2025]	✓	✓	✗	✓	Per-layer beacon activations
KV-Distill[arXiv 2025.03]	✓	✓	✗	✓	Trained scorer selects + LoRA refines
Still [arXiv 2026.06]	✓	✓	✗	✗	Per-layer Perceiver cross-attention
ANCHOR (ours)	✓	✓	✓	✓	Windowed pool + linear head

benchmarks and four previous compaction methods demonstrate the superior performance of ANCHOR, scoring the best on 6 out of 8 experiment setting.

2 Related Work

Compacting KV caches. Different studies have explored various approaches to compact KV caches, covering both training-free and training-based methods. Attention Matching compacts a cache in a training-free manner (Zweiger et al., 2026). Its values are synthesized in closed form, but its keys remain a subset of the original ones. It sits in a line of cache surgery research direction that prunes, evicts, or merges a cache that already exists (Zhang et al., 2023; Li et al., 2024; Xiao et al., 2024; Bouyahiaoui et al., 2026; Tian et al., 2025), whose limits are being charted as a rate-distortion problem (Colaco & Lahjouji, 2026) and whose performance is sensitive to the queries (Luo et al., 2026). Cartridges (Eyuboglu et al., 2025), instead treats the cache as free parameters and distills a corpus into them by gradient descent in the manner of context distillation (Snell et al., 2022), and extending to collections of documents (Hardalov et al., 2026). But for every new document a new round of training is needed, making this approach not practicable for fast changing skill content.

Amortized compaction. Amortized methods instead meta-train one emitter and run only one forward pass per new document. SKIM learns to compress procedural skills into adaptive soft tokens through a compactor and a LoRA adapter for the base model (Wang et al., 2026). Activation Beacon interleaves beacon tokens that condense a long context into per-layer activations, reusable across queries with the base frozen (Zhang et al., 2024b). KV-Distill amortizes selection itself, training a scorer to decide which original rows survive and an additional LoRA (Chari et al., 2025), so it carries both the rate bound of selection and a modified consuming model. Still is the closest predecessor to our approach, which utilizes a per-layer Perceiver to synthesize compressed KVs (O'Neill et al., 2026). However, the output artifact is fixed in length and the focus remains knowledge-level compaction. ANCHOR fills the gap by training for length-adaptive procedural skill KV compaction. Table 1 places the methods we discuss on four axes that matter for this setting.

3 Method

3.1 Setup

Let θ be a frozen LLM with n layers and hidden dimension d. A *procedural skill* c is a token sequence of length $L_c = |c|$. Given a query q, the standard approach samples a response $y = (y_1, \ldots, y_T)$ from $p_\theta(y \mid c, q)$. We train an emitter with parameters ϕ to map the skill to a compact per-layer cache

$$\mathcal{Z}_\phi(c) = \big((K_\phi^\ell(c), V_\phi^\ell(c))\big)_{\ell=0}^{n-1}, \tag{1}$$

where each key and value tensor contains $m = m(c; r)$ active slots (defined in equation 3). The goal is

$$p_\theta(\cdot \mid \mathcal{Z}_\phi(c), q) \;\approx\; p_\theta(\cdot \mid c, q), \tag{2}$$

with only ϕ optimized and θ fixed.

3.2 ARCHITECTURE

The compaction pipeline can be decomposed into four steps.

(1) Encoding. The frozen LLM first reads c in one pass and we record the hidden state of every token t at layer ℓ as $h_t^\ell \in \mathbb{R}^d$. Stacking gives $H \in \mathbb{R}^{L_c \times n \times d}$.

(2) Length-adaptive windowed pooling. Given ratio r, the slot budget is

$$m_i \;=\; \lceil L_c/r \rceil\,, \tag{3}$$

so the artifact scales with document length rather than occupying a fixed budget. The reduction is where information is lost, so it is where we place learned capacity. Let $\mathcal{W}_j$ be slot j's window over the real tokens; each slot is a weighted sum over its own window, with weights from a scorer s_ψ:

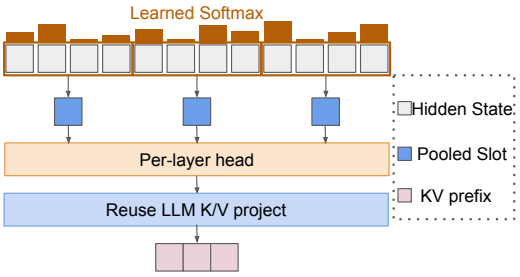


Figure 2: The skill's hidden states are partitioned into disjoint windows and are average-pooled wrt. a learned softmax weights. A linear per-layer head reshapes each slot (equation 5), and the base model's frozen key and value projections turn the slots into cache entries, which keeps synthesized entries on the manifold attention expects.

$$p_j^\ell \;=\; \sum_{t\in\mathcal{W}_j} \frac{\exp\big(s_\psi(h_t^0)_\ell\big)}{\sum_{t'\in\mathcal{W}_j}\exp\big(s_\psi(h_{t'}^0)_\ell\big)}\, h_t^\ell, \qquad s_\psi(h) = W_\psi h + b_\psi, \tag{4}$$

with $W_\psi \in \mathbb{R}^{n\times d}$ and $b_\psi \in \mathbb{R}^n$. The scorer is shared across slots, while its n output coordinates provide layer-specific scores. Zero-initializing W_ψ and b_ψ makes weighting uniform, so the initial operator is exactly windowed mean pooling.

(3) Emission head. The emission head contains an affine transform with $W \in \mathbb{R}^{n\times d\times d}$ plus a residual to each layer's pooled slots:

$$\widetilde{p}_j^\ell \;=\; p_j^\ell + W_\ell\, p_j^\ell + b_\ell. \tag{5}$$

W_ℓ and b_ℓ are shared across slots with zero-initialization. The scorer decides what enters a slot, and the emission head determines how the pooled state is updated.

(4) Cache construction. Let $\widetilde{P}^\ell = (\widetilde{p}_1^\ell, \ldots, \widetilde{p}_m^\ell)$. We route these states through the frozen base LLM. Although the slots are new, every emitted key and value traverses the same layer norm and projections as a real token, so the result lies on the manifold attention expects. Slots carry no positional encoding because pooled slots do not correspond to token positions and our ablations show the prefix is content-addressable rather than order-addressable.

3.3 TRAINING OBJECTIVE

All cross-entropy terms use the standard token-level autoregressive loss. We upweight tokens belonging to schema identifiers (tool names and argument keys) by α=3.

Stage 1: Knowledge preservation.

$$\mathcal{L}_{\mathrm{KP}}(\phi) = \mathbb{E}_c[\mathrm{CE}(c \mid \mathcal{Z}_\phi(c))]\,, \tag{6}$$

where reconstruction is teacher-forced over all L document tokens and only the emitter parameters ϕ are updated.

Stage 2: Instruction following with a reconstruction anchor. Warm-started from stage 1, we distill the instruction-following signals into the emitted cache. For each training triple (c, q, y^*), the current student first produces an on-policy rollout $\widetilde{y} = (\widetilde{y}_1, \ldots, \widetilde{y}_{\widetilde{T}}) \sim p_\theta(\cdot \mid \mathcal{Z}_\phi(c), q)$. At rollout step t, define teacher and student next-token distributions over vocabulary item v on the *same* history:

$$p_{\mathrm{T},t}(v) = p_\theta(v \mid c, q, \widetilde{y}_{<t})\,, \qquad p^\phi_{\mathrm{S},t}(v) = p_\theta(v \mid q, \widetilde{y}_{<t}, \mathcal{Z}_\phi(c))\,. \tag{7}$$

The on-policy distillation loss is the mean forward KL along this rollout:

$$\mathcal{L}_{\mathrm{KD}}(\phi) = \mathbb{E}_{(c,q),\,\widetilde{y}}\left[\frac{1}{\widetilde{T}}\sum_{t=1}^{\widetilde{T}} D_{\mathrm{KL}}\Big(p_{\mathrm{T},t}\,\Big\|\,p^{\phi}_{\mathrm{S},t}\Big)\right]. \tag{8}$$

The rollout is sampled without differentiating through its discrete tokens; gradients enter through the student's scores in equation 8. Because an initially weak student may visit unhelpful trajectories, a separate teacher-forced pass supplies exact supervision on the gold response:

$$\mathcal{L}_{\mathrm{gold}}(\phi) = \mathbb{E}_{(c,q,y^*)}[\mathrm{CE}(y^* \mid \mathcal{Z}_\phi(c), q)]\,. \tag{9}$$

Finally, Stage 2 replays the same whole-document reconstruction objective on its skill examples:

$$\mathcal{L}_{\mathrm{anchor}}(\phi) = \mathbb{E}_c^{\mathrm{stage2}}[\mathrm{CE}(c \mid \mathcal{Z}_\phi(c))]\,. \tag{10}$$

The complete objective is

$$\underbrace{\mathcal{L}_{\mathrm{IF}}}_{\text{instruction following}} = \mathcal{L}_{\mathrm{KD}} + \lambda\mathcal{L}_{\mathrm{gold}}, \qquad \mathcal{L}_{\mathrm{stage2}} = \mathcal{L}_{\mathrm{IF}} + \mu \underbrace{\mathcal{L}_{\mathrm{anchor}}}_{\text{knowledge preservation}}. \tag{11}$$

Thus, the KL and gold terms teach behavior, while the low-weight anchor prevents the emitted cache from losing the decodable skill content acquired in Stage 1. We set λ=1 and μ=0.2 in our experiments.

3.4 Training data

Training can be done on any data that provides samples as (skill or instruction(c), task(q), ground truth(y^*)) tuples. We construct a dataset from various sources, including Super-NaturalInstructions (Wang et al., 2022), Unnatural Instructions (Honovich et al., 2022), and a large synthetic corpus of procedural skills we build from xLAM (Zhang et al., 2024a; Liu et al., 2024). To construct the synthetic corpus, we start from an xLAM training-family toolset, use Gemma-4-31B-it to propose additional in-domain tools, retain only well-formed JSON schemas with unique tool names, and render the accepted tools into a structured procedural skill c. Conditioned on c, Gemma then proposes a task q and simulates an agentic trajectory to produce the grounded answer y^*. Qwen3.6-35B-A3B cross-validates each task and answer, after which deterministic checks enforce schema validity, required arguments, cross-call referential integrity, response grounding, and task-type conformance. Queries are globally deduplicated and evaluation families are held out from training. Across the four sources, the resulting full training corpus contains over 61k unique skill files and 640,833 task–answer pairs; Appendix B gives the full breakdown. This dataset is used to train ANCHOR and the baseline methods as well.

4 DOMAINBENCH

DOMAINBENCH answers the question of whether a compacted skill still steers behavior, and it is built with two properties existing benchmarks lack. The first is sufficient headroom. The tested base models scores on average below 0.003 F1 without the skills in context , while the same model reading the skill files in context reaches 0.763 on Qwen3-4B and 0.825 on Gemma-4-E4B. Any nonzero score is therefore attributable to the compacted document rather than to parametric knowledge. Figure 3 shows that the two conditions differ by 0.056 on LONGBENCH V2, 0.026 on SOP-BENCH, and 0.007 on SWE-SKILLS-BENCH. The second property is granularity. We grade a call the way a skill document specifies it, by tool name, then argument keys, then argument values, which localizes *where* compaction fails instead of collapsing the failure into one pass or fail.

To construct DOMAINBENCH, we carefully curate 716 tasks over 100 procedural skills. Each skill document is opening with an identity, then a framework of interaction rules, then a typed API specification of the tools, and closing with numbered operating instructions. Tasks are generated from each skill's available tool set, summing up to 436 single tool-call tasks and 280 multi tool-call tasks. We leverage the strong gemma-4-31B-it to generate skill files, tasks, and also answers. Every task and answer must then survive a three-judge majority vote and is finally audited by a human annotator. A predicted tool call is compared to gold at three nested levels. **Tool** requires the tool name to

match, **Schema** additionally requires the argument-key set to match, and **Call** additionally requires every argument value to match, so **Call** $\leq$ **Schema** $\leq$ **Tool**. Values are compared character-exactly after whitespace and case normalization at every level. Although DOMAINBENCH skills are synthesized by the same teacher as part of the training split, we guarantee the tool-families are distinct and thus prevent the data leakage problem.

5 EXPERIMENTS

Baselines and benchmarks. We choose four methods from Table 1 as baselines. Attention Matching (AM) (Zweiger et al., 2026) selects a subset of keys and synthesizes the corresponding values so as to best match the full-context attention output and mass; Cartridges (Eyuboglu et al., 2025) runs gradient descent on learnable KV parameters per document; Still (O'Neill et al., 2026) meta-trains an amortized per-layer Perceiver compactor; and SKIM (Wang et al., 2026) compacts skill files into soft tokens rather than into cache entries. Each baseline is swept on our training set and reported at its best configuration. Our primary benchmark is DOMAINBENCH; we additionally report results on three public benchmarks: SWE-SKILLS-BENCH (Han et al., 2026), real agentic coding in which each task's requirement document is graded by pytest suites generated from its acceptance criteria and executed in a container; SOP-BENCH (Li et al., 2025), seven domains of multi-turn procedure following scored by pass rate, which we macro-average over domains; and LONGBENCH V2 (Bai et al., 2025), adversarial four-way multiple choice that tests factual long-context recall. Detailed protocols are in Appendix D.

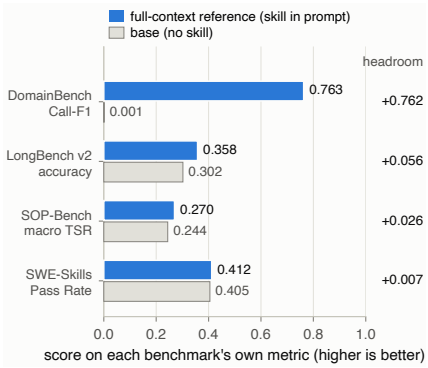


Figure 3: Headroom on each benchmark's own metric evaluated on Qwen3-4B. Only DOMAINBENCH leaves a range wide enough to separate compaction methods.

Implementation. Our backbones are Qwen3-4B-Instruct-2507 (Yang et al., 2025) and Gemma-4-E4B-it (Team et al., 2026). Gemma-4 interleaves 35 sliding-window layers with 7 full-attention layers and we expose the compacted cache only to the full-attention layers. Decoding uses nuclear sampling, and all training and evaluation ran on 8×H200 nodes. Unless stated otherwise, we compact at a ratio of $r{=}4$ for ANCHOR and AM; the three methods with a fixed emission length use 768 tokens, an effective ratio around 4 on DOMAINBENCH. ANCHOR, SKIM and Still are trained for an identical budget of 7,200 global steps, while Cartridges follows its default per-document recipe. Other details are reported in Appendix C.2.

5.1 MAIN RESULTS

Table 2 reports the overall comparison across all settings. On DOMAINBENCH, the four baselines recover between 5.5% and 45.0% of the available headroom, and they perform differently on two backbones: Cartridges is the strongest baseline on Qwen3-4B at 45.0% yet the second weakest on Gemma-4-E4B at 18.3%, whereas SKIM moves in the opposite direction, from 31.6% to 37.5%. ANCHOR recovers 65.1% and 58.9%, exceeding the best baseline on each backbone by $+0.153$ and $+0.176$ Call-F_1. Table 3 breaks these results down by metric. AM already fails at tool naming, the coarsest of the three levels. This likely reflects the difficulty of retaining a complete procedural specification by selection alone: for declarative knowledge, retaining the right keywords can already preserve much of what a document says, whereas a procedural skill demands more than that. Among the trained baselines, SKIM and Cartridges perform comparably, indicating that an amortized compactor is not necessarily inferior to per-context optimization. ANCHOR remains the strongest method throughout, and tasks requiring multiple tool calls are generally harder than single-call ones.

ANCHOR also remains strong on the two public agentic benchmarks. On SOP-BENCH, AM and SKIM emit prefixes noisy enough to leave the model worse off than the no-skill base. On SWE-SKILLS-BENCH, no method produces a compacted skill that helps the base model, which we at-

Table 2: Compaction fidelity across four benchmarks. Scores are averaged from 3 seeded runs. Recovery is $max((\text{score} - \text{base})/|\text{reference} - \text{base}| \times 100\%, 0)$. On SWE-SKILLS no methods can exceed base, so we report Δ base instead. **Bold** indicates the best skill method and underline the second-best.

	DOMAINBENCH		SOP-BENCH		LONGBENCH V2		SWE-SKILLS-BENCH	
Method	Call-F1	Recovery	Macro TSR	Recovery	Accuracy	Recovery	Pass Rate	Δ base
`Qwen3-4B-Instruct-2507`								
Base (no skill)	0.001	0.0%	0.244	0.0%	0.302	0.0%	0.405	0.000
Full-context reference	0.763	100.0%	0.270	100.0%	0.358	100.0%	0.412	0.007
Attention Matching	0.072	9.3%	0.078	0%	0.341	69.6%	0.118	−0.287
Still	0.125	16.3%	0.253	34.6%	0.328	46.4%	0.239	−0.166
SKIM	0.242	31.6%	0.241	0%	0.311	16.1%	0.157	−0.248
Cartridges	0.344	45.0%	0.211	0%	**0.351**	**87.5%**	0.211	−0.194
ANCHOR	**0.497**	**65.1%**	**0.307**	**242.3%**	0.337	62.5%	**0.356**	**−0.049**
`Gemma-4-E4B-it`								
Base (no skill)	0.002	0.0%	0.196	0.0%	0.211	0.0%	0.422	0.000
Full-context reference	0.825	100.0%	0.368	100.0%	0.345	100.0%	0.438	0.016
Attention Matching	0.047	5.5%	0.031	0%	0.232	15.7%	0.101	−0.321
Still	0.231	27.8%	0.201	2.9%	0.247	26.9%	0.233	−0.189
SKIM	0.311	37.5%	0.185	0%	0.224	9.7%	0.135	−0.287
Cartridges	0.153	18.3%	0.221	14.5%	**0.301**	**67.2%**	0.194	−0.228
ANCHOR	**0.487**	**58.9%**	**0.271**	**43.6%**	0.278	50.0%	**0.344**	**−0.078**

tribute to the difficulty of the benchmark and to the density of code blocks in its documents. ANCHOR nevertheless incurs the smallest regression. We further find that existing KV-compaction methods perform well on LONGBENCH V2, a long-context knowledge-recall benchmark; although ANCHOR is not trained for such tasks, it performs on par with the best baselines.

Table 3: Call-F_1 is broken out over tasks whose gold sequence is a *single* call and those requiring *multiple* calls. Recovery is $100(\text{Call} - \text{base})/(\text{oracle} - \text{base})$. ANCHOR is shown at both r=4 and r=8. **Bold** indicates the best skill method and underline the second-best.

			Call-F1			
Method	Tool-F1	Schema-F1	all	single	multi	Recovery
`Qwen3-4B-Instruct-2507`						
Base (no skill)	0.009	0.001	0.001	0.001	0.001	0.0
full-context reference	0.944	0.805	0.763	0.783	0.732	100.0
AM (r=4)	0.164	0.101	0.072	0.078	0.063	9.3
Still	0.219	0.149	0.125	0.130	0.117	16.3
SKIM	0.433	0.263	0.242	0.244	0.239	31.6
Cartridges	0.410	0.366	0.344	0.363	0.314	45.0
ANCHOR (r=4)	0.613	0.531	**0.497**	0.518	0.464	65.1
ANCHOR (r=8)	0.520	0.424	0.325	0.344	0.295	42.5
`Gemma-4-E4B-it`						
Base (no skill)	0.009	0.002	0.002	0.000	0.005	0.0
full-context reference	0.918	0.867	0.825	0.865	0.763	100.0
AM (r=4)	0.138	0.083	0.047	0.060	0.026	5.5
Still	0.301	0.274	0.231	0.234	0.226	27.8
SKIM	0.441	0.360	0.311	0.355	0.242	37.5
Cartridges	0.317	0.219	0.153	0.160	0.142	18.3
ANCHOR (r=4)	0.557	0.499	**0.487**	0.529	0.422	58.9
ANCHOR (r=8)	0.446	0.375	0.334	0.353	0.304	40.3

Figure 4 traces Call-F_1 against the compaction ratio on DOMAINBENCH. At 1×, AM and ANCHOR perform at roughly the same level, around 85% of the full-context reference. Beyond 2×, however, AM degrades sharply on both backbones, whereas ANCHOR declines more gradually. This reinforces the observation that retaining only a fraction of the context is damaging for procedural

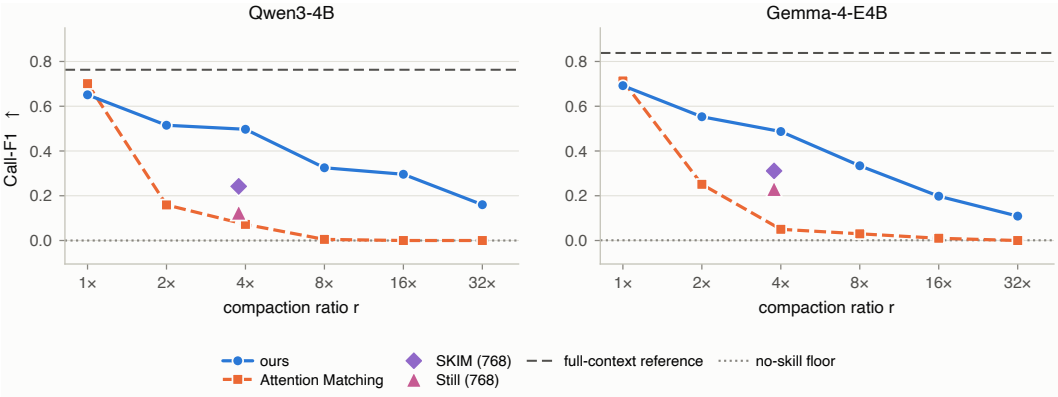
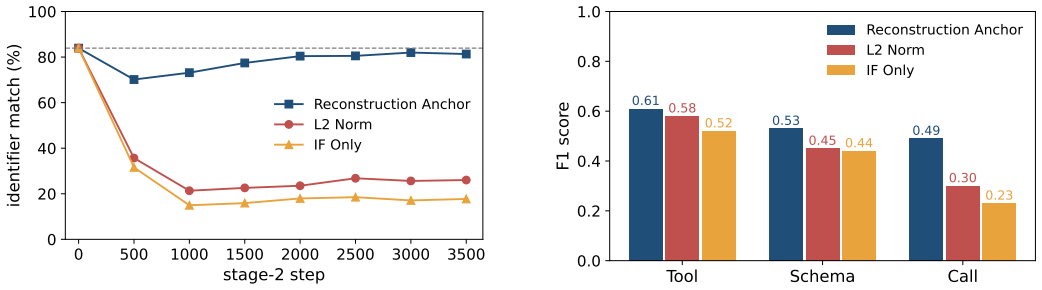


Figure 4: Compaction ratio against Call-F1 on DOMAINBENCH, for both backbones. ANCHOR and AM scale their budget with document length and so trace a curve; SKIM and STILL emit a fixed number of entries and appear as horizontal lines.



(a) (b)

Figure 5: All experiments are warm-started from the same stage 1 checkpoint and are evaluated on DOMAINBENCH. (a) measures knowledge-preservation across Stage 2; identifier match percentage tells how many critical words (tool name, argument key and values) are matched when the skill document is reconstructed from the emitted prefix. (b) compares the final F1 scores of different anti-drift constraints.

skills and indicates further investigation. The fixed-length methods SKIM and STILL operate at an effective ratio of $4\times$ and perform comparably to, or slightly better than, ANCHOR at $8\times$.

5.2 ANALYSIS: WHAT THE ANCHOR DOES

The objective of Section 3.3 keeps knowledge preservation active while Stage 2 distills behavior into the prefix. We interrogate that choice through three questions: *when* preserved knowledge is lost, *whether* retaining it changes downstream behavior, and *whether* the mechanism is reconstruction in particular or anti-drift regularization of any kind. Figure 5a shows that the stored knowledge is lost early in Stage 2 but is recovered once the anchor is present, while the remaining arms stay flat for the rest of training. Figure 5b shows that the anchored arm surpasses the instruction-following-only arm at every level of granularity, and most markedly on the fine-grained Call-F_1, where it more than doubles the score. Beyond the reconstruction anchor, we also implement a generic L_2-norm variant; it outperforms the unconstrained arm but still trails the explicit anchor by a wide margin. We therefore conclude that an explicit anchor preserves the knowledge acquired in the earlier stage, and that this preservation translates into better downstream performance. We also see that without the anchoring term our method performs similar to SKIM and worse than Cartridges. This means that the superior performance of ANCHOR is largely achieved by the training objectives.

5.3 ABLATIONS

Table 4: Ablations on training objectives. Note that the last two blocks are warm started from stage-1 init.

Condition	Judge ↑	Tool ↑	Schema ↑	Call ↑
Base (no skill)	*1.41*	*0.000*	*0.000*	*0.000*
Full-context reference	*3.80*	*0.802*	*0.667*	*0.559*
Stage-1 init	1.47	0.034	0.034	0.017
Stage-2 cold start	2.98	0.486	0.391	0.311
Terms on and off				
$\mathcal{L}_{\text{KD}}$	2.20	0.164	0.045	0.034
$\mathcal{L}_{\text{gold}}$	2.90	0.364	0.221	0.152
$\mathcal{L}_{\text{KD}} + \mathcal{L}_{\text{anchor}}$	2.52	0.266	0.158	0.147
$\mathcal{L}_{\text{gold}} + \mathcal{L}_{\text{anchor}}$	3.02	0.492	0.362	0.254
$\mathcal{L}_{\text{KD}} + \mathcal{L}_{\text{gold}}$	3.01	0.474	0.324	0.254
$\mathcal{L}_{\text{KD}} + \mathcal{L}_{\text{gold}} + \mathcal{L}_{\text{anchor}}$	**3.34**	**0.580**	**0.471**	**0.398**
Anchor weight μ				
μ=0.4	3.35	0.564	0.414	0.325
μ=0.6	3.24	0.432	0.347	0.298
μ=0.8	3.21	0.475	0.379	0.262

We now ablate our architectural and training-objective choices. All experiments use Qwen3-4B-Instruct-2507 at a $4\times$ compaction ratio and are evaluated on a held-out test set; judge scores are assigned by gemma-4-31B-it on a 1–5 scale, assessing whether the model follows the given natural or unnatural instructions, and Tool/Schema/Call-F_1 follow the protocol introduced above. This held-out set is sampled from the same four training sources of Section 3.4, with its skills and tool families excluded from training, so it isolates the effect of each design choice under a distribution matched to training; its absolute values are consequently not comparable with the DOMAINBENCH results of Table 3, which additionally carry a domain shift. Table 4 presents the ablation over loss terms and objective choices. Neither KL alone nor gold-CE alone performs well; adding the anchor term on top of either lifts every metric, a fourfold improvement in the case of KL alone. The combination of all three terms is the best configuration. The anchor is also most effective at a low weight: raising μ from 0.2 to 0.8 increasingly harms performance.

Table 5 ablates the key architectural design choices. First, in our setting forward KL is substantially better than the reverse KL used in SDFT (Shenfeld et al., 2026). We attribute this to the fact that reverse KL is most effective once the student already produces meaningful outputs for the teacher to grade on, whereas our student is very weak at the beginning; we therefore adopt the forward direction and add a ground-truth forcing path. Second, pooling the encoder's *hidden states* rather than the base model's own $K\|V$ keeps all operations in a space that the per-layer head can still reshape, which leads to better performance on every metric. We also find that positionless embedding for the compacted slots performs better than RoPE on every metric, probably because the shortened prefix becomes now more content-addressable rather than position-addressable. Finally, leaving out the learned weights for pooling does not cost the LLM judge score much but affects the actual tool-calling metrics a lot, meaning that the weights we learned is more useful for actual procedural actions than general instruction following.

Table 5: Training and architecture choices.

Condition	Judge ↑	Tool ↑	Schema ↑	Call ↑
KL direction				
Forward	**2.20**	**0.164**	**0.045**	**0.034**
Reverse	1.60	0.073	0.006	0.000
ANCHOR *input*				
H	**3.34**	**0.580**	**0.471**	**0.398**
original $K\|V$	3.00	0.469	0.345	0.315
Positional embedding				
NoPE	**3.34**	**0.580**	**0.471**	**0.398**
RoPE	3.15	0.490	0.401	0.374
Pooling strategy				
w. s_ψ	**3.34**	**0.580**	**0.471**	**0.398**
w/o. s_ψ	3.28	0.520	0.45	0.334

6 CONCLUSION

LLM agents increasingly rely on procedural skill documents, yet we find that existing KV-cache compaction methods handle them poorly on both instruction-following and tool-calling fidelity. Prior work has separately studied amortized KV synthesis, procedural soft-token compression, and reconstruction-based latent compression. We show that, for amortized KV compaction of unseen procedural skills, behavior distillation rapidly erases schema information, and that retaining reconstruction during this stage substantially improves executable tool-call fidelity. We also address the absence of a benchmark that offers both fine-grained tool-call evaluation and a clear headroom between the full-context reference and the no-skill base, and propose DOMAINBENCH to fill that gap. Evaluated on two backbones and four benchmarks, ANCHOR recovers 65.1% and 58.9% of full-context fidelity at $4\times$ compaction, the strongest result among the methods we compare.

AI USE STATEMENT

In this work, we used generative AI tools for generating synthetic data for training and for benchmark. Data generation prompts are hand-written then refined by LLMs. Benchmark data are cross-validated by human annotators, while training data is validated by another LLM. We also use LLMs to aid paper writing in terms of grammar and writing check. We have reviewed all AI-assisted work. We did not use AI tools for idea formulation and literature review. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks, 2025. URL https://arxiv.org/abs/2412.15204.

Soumia Bouyahiaoui, Manel Kara laouar, Aicha Boutorh, and Mohamed Hadj Ameur. Selkv: Selective kv cache merging with per-token merge-or-drop and attention compensation, 2026. URL https://arxiv.org/abs/2607.16213.

Samuel Jacob Chacko, James Hugglestone, Chashi Mahiul Islam, and Xiuwen Liu. When skills don't help: A negative result on procedural knowledge for tool-grounded agents in offensive cybersecurity, 2026. URL https://arxiv.org/abs/2605.20023.

Vivek Chari, Guanghui Qin, and Benjamin Van Durme. Kv-distill: Nearly lossless learnable context compression for llms, 2025. URL https://arxiv.org/abs/2503.10337.

Alexis Chevalier, Alexander Wettig, Anirudh Ajith, and Danqi Chen. Adapting language models to compress contexts, 2023. URL https://arxiv.org/abs/2305.14788.

Ashwin Gerard Colaco and Nada Lahjouji. What to keep, what to forget: A rate–distortion view of memory compaction in llms and agents, 2026. URL https://arxiv.org/abs/2607.08032.

Jiancheng Dong, Pengyue Jia, Jingyu Peng, Maolin Wang, Yuhao Wang, Lixin Su, Xin Sun, Shuaiqiang Wang, Dawei Yin, and Xiangyu Zhao. Learning a single token to replace long system prompts in llms, 2026. URL https://arxiv.org/abs/2511.23271.

Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, and Christopher Re. Cartridges: Lightweight and general-purpose long context representations via self-study, 2025. URL https://arxiv.org/abs/2506.06266.

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, 2024. URL https://arxiv.org/abs/2307.06945.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models, 2025. URL https://arxiv.org/abs/2403.07714.

Tingxu Han, Yi Zhang, Wei Song, Chunrong Fang, Zhenyu Chen, Youcheng Sun, and Lijie Hu. Swe-skills-bench: Do agent skills actually help in real-world software engineering?, 2026. URL https://arxiv.org/abs/2603.15401.

Momchil Hardalov, Gonzalo Iglesias, and Adrià de Gispert. Cartridges at scale: Training modular kv caches over large document collections, 2026. URL https://arxiv.org/abs/2606.04557.

Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. Unnatural instructions: Tuning language models with (almost) no human labor, 2022. URL https://arxiv.org/abs/2212.09689.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13): 3521–3526, March 2017. ISSN 1091-6490. doi: 10.1073/pnas.1611835114. URL http://dx.doi.org/10.1073/pnas.1611835114.

Xiangyi Li, Yimin Liu, Wenbo Chen, Bingran You, Zonglin Di, Yifeng He, Shenghan Zheng, Kyoung Whan Choe, Jiankai Sun, Shuyi Wang, Chujun Tao, Binxu Li, Xuandong Zhao, Hejia Geng, Xiaojun Wu, Junwei Zhou, Xiaokun Chen, Hanwen Xing, Yubo Li, Qunhong Zeng, Di Wang, Yuanli Wang, Roey Ben Chaim, Penghao Jiang, Haotian Shen, Luyang Kong, Xinyi Liu, Runhui Wang, Xuanqing Liu, Jiachen Li, Xin Lan, Yueqian Lin, Wengao Ye, Junwei He, Songlin Li, Yue Zhang, Yipeng Gao, Yijiang Li, Ze Ma, Liqiang Jing, Tianyu Wang, Kaixin Li, Yiqi Xue, Haoran Lyu, Yizhuo He, Yuchen Tian, Shutong Wu, Bowei Wang, Yixuan Gao, Bo Chen, Litong Liu, Sikai Cheng, Jiajun Bao, Shuaicheng Tong, Shuwen Xu, Terry Yue Zhuo, Tinghan Ye, Qi Qi, Miao Li, Longtai Liao, Zelin Tan, Chang Shi, Xilin Tang, Srinath Tankasala, Boqin Yuan, Yaoyao Qian, Jianhong Tu, Chenguang Wang, Yizhou Sun, Wei Wang, Aaron Taylor, Ziyue Yang, Changkun Guan, Zhikang Dong, Xinyu Zhang, Steven Dillmann, Han chung Lee, and Dawn Song. Skillsbench: Benchmarking how well agent skills work across diverse tasks, 2026. URL https://arxiv.org/abs/2602.12670.

Xuhong Li, Yves Grandvalet, and Franck Davoine. Explicit inductive bias for transfer learning with convolutional networks, 2018. URL https://arxiv.org/abs/1802.01483.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation, 2024. URL https://arxiv.org/abs/2404.14469.

Zekun Li, Shinda Huang, Jiangtian Wang, Nathan Zhang, Antonis Antoniades, Wenyue Hua, Kaijie Zhu, Sirui Zeng, Chi Wang, William Yang Wang, and Xifeng Yan. Sopbench: Evaluating language agents at following standard operating procedures and constraints, 2025. URL https://arxiv.org/abs/2503.08669.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets, 2024. URL https://arxiv.org/abs/2406.18518.

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, 2026. URL https://arxiv.org/abs/2604.02268.

Daming Luo, Christy Liang, and Junyu Xuan. How query visibility changes kv-cache compression rankings: A matched-budget audit, 2026. URL https://arxiv.org/abs/2607.11942.

Michael McCloskey and {Neal J.} Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation - Advances in Research and Theory*, 24(C):109–165, January 1989. ISSN 0079-7421. doi: 10.1016/S0079-7421(08)60536-8. The research reported in this chapter was supported by NIH grant NS21047 to Michael McCloskey, and by a grant from the Sloan Foundation to Neal Cohen. We thank Sean Purcell and Andrew Olson for assistance in generating the figures, and Alfonso Caramazza, Walter Harley, Paul Macaruso, Jay McClelland, Andrew Olson, Brenda Rapp, Roger Rat-cliff, David Rumelhart, and Terry Sejnowski for helpful discussions.

Jesse Mu, Xiang Lisa Li, and Noah Goodman. Learning to compress prompts with gist tokens, 2024. URL https://arxiv.org/abs/2304.08467.

Charles O'Neill, Alex Sandomirsky, Harry Partridge, Mudith Jayasekara, and Max Kirkby. Still: Amortized kv cache compaction in a single forward pass, 2026. URL https://arxiv.org/abs/2606.07878.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis, 2023. URL https://arxiv.org/abs/2305.15334.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference, 2022. URL https://arxiv.org/abs/2211.05102.

Hubert M. Pysklo, Artem Zhuravel, and Patrick D. Watson. Agent-diff: Benchmarking llm agents on enterprise api tasks via code execution with state-diff-based evaluation, 2026. URL https://arxiv.org/abs/2602.11224.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. URL https://arxiv.org/abs/2307.16789.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning, 2017. URL https://arxiv.org/abs/1611.07725.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023. URL https://arxiv.org/abs/2302.04761.

Hongyu Shan, Mingyang Song, Chang Dai, Di Liang, and Han Chen. R-capsule: Compressing high-level plans for efficient large language model reasoning, 2025. URL https://arxiv.org/abs/2509.22131.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning, 2026. URL https://arxiv.org/abs/2601.19897.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context, 2022. URL https://arxiv.org/abs/2209.15189.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, Mayank Chaturvedi, Aditya Chawla, Victor Cotruta, Alice Coucke, Phil Culliton, Robert Dadashi, Lucas Dixon, Mohamed Elhawaty, Utku Evci, Clément Farabet, Johan Ferret, Filippo Galgani, Sertan Girgin, Jean-Bastien Grill, Maarten Grootendorst, Jiaxian Guo, Cassidy Hardin, Yanzhang He, Steven M. Hernandez, Omri Homburger, Léonard Hussenot, Juyeong Ji, Armand Joulin, Aishwarya Kamath, Parnian Kassraie, Olivier Lacombe, Preethi Lahoti, Gaël Liu, Gus Martins, Luciano Martins, Tatiana Matejovicova, Ramona Merhej, Nikola Momchev, Sneha Mondal, Ryan Mullins, Sindhu Raghuram Panyam, Shreya Pathak, Sarah Perrin, André Susano Pinto, Etienne Pot, Angéline Pouget, Alexandre Ramé, Sabela Ramos, Douglas Reid, David Rim, Morgane Rivière, Karsten Roth, Louis Rouillard, Omar Sanseviero, Pier Giuseppe Sessa, Shane Settle, Danila Sinopalnikov, Sara Smoot, Piotr Stanczyk, Andreas Steiner, Lawrence Stewart, Ilya Tolstikhin, Michael Tschannen, Anton Tsitsulin, Nino Vieillard, Renjie Wu, Pingmei Xu, Haichuan Yang, Edouard Yvinec, Biao Zhang, Li Zhang, Joe Zou, Nicolas Aagnes, Abdelrahman Abdelhamed, Jakub Adamek, Shivani Agrawal, Shubham Agrawal, Ibrahim Alabdulmohsin, Jean Baptiste Alayrac, Uri Alon, Chandramouli Amarnath, Ankesh Anand, Chrysovalantis Anastasiou, Setareh Ariafar, François-Xavier Aubet, Kyriakos Axiotis, Federico Barbero, Joelle Barral, Alexei Bendebury, Urs Bergmann, Stanley Bileschi, Kat Black, Mathieu Blondel, Sebastian Borgeaud, Arthur Bražinskas, Ryan Burnell, Robert Busa-Fekete, Mu Cai, Daniele Calandriello, Glenn Cameron, Charlotte Caucheteux, Rahma Chaabouni, Garima Chadha, Jetha Chan, Blake Jianhang Chen, Jesse Chen, Lin Chen, Xu Chen, Derek Cheng, Tzu hsiang Chien, Nikolai Chinaev, Yi Chou, Zhaohui Chu, Benjamin Coleman, Pooja Consul, Sam Conway-Rahman, Scott Crowell, Dylan Cutler, Vivek Dani, Samira Daruki, Anil Das, Daniel Deutsch, Nishanth Dikkala, Li Ding, Qiuhan Ding, Shenil Dodhia, Konstantin Donhauser, Tulsee Doshi, Anca Dragan, Alex Druinsky, Sahil Dua, Zoltan Egyed, Danielle Eisenbud, Daniel Eppens, Cindy Fan, Bahare Fatemi, Yassir Fathullah, Vlad Feinberg, Milen Ferev, Sebastian Flennerhag, Takumi Fujimoto, João Gabriel Oliveira, Isaac Galatzer-Levy, João Gante, Simon Geisler, Soham Ghosal, Antonious M. Girgis, Tamara von Glehn,

Alec Go, Alhaad Gokhale, Alex Grills, Yiming Gu, Mayank Gupta, Pramod Gupta, Guru Guruganesh, Raia Hadsell, Hamza Harkous, Jitendra Harlalka, Demis Hassabis, Anja Hauth, Joe Heyward, Arian Hosseini, Chih-Yang Hsia, I-Hung Hsu, Xiaopeng Huang, Yangsibo Huang, Kevin Hui, Adrian Hutter, Te I, Fotis Iliopoulos, Advait Jain, Ganesh Jawahar, Ziwei Ji, Qilin Jin, Melvin Johnson, Kandarp Joshi, Arun Kandoor, Wang-Cheng Kang, Koray Kavukcuoglu, Mehran Kazemi, Kathleen Kenealy, Amr Khalifa, Phoebe Kirk, Ivan Korotkov, Suraj Kothawade, Vitaly Kovalev, Neel Kovelamudi, Adam Kraft, Ravin Kumar, Vivek Kumar, Harish Kuppam, Justin Lannin, Chen-Yu Lee, Seungji Lee, Dmitry Lepikhin, Alon Levkovitch, Dongdong Li, Qiujia Li, Valentin Liévin, Ethan Lin, Ziqian Lin, Casper Liu, Tianlin Liu, Tianqi Liu, Xin Liu, Ivan Lobov, Mayank Lunayach, Min Ma, Gagan Madan, Andrii Maksai, Eric Malmi, Michal Matuszak, Daniel McDuff, Gaurav Menghani, Maciej Mikuła, Daniil Mirylenka, Karolis Misiunas, Vedant Misra, Andreea Mitran, Kareem Mohamed, Maksim Mukha, Eric Noland, James O'Donnell, Brendan O'Donoghue, Kate Olszewska, Bernett Orlando, Wanqiong Pan, Rina Panigrahy, Unnati Parekh, Nicolas Perez-Nieves, Chunjong Park, Eric Paskie, Liqian Peng, Bryce Petrini, Slav Petrov, Jonas Pfeiffer, Bilal Piot, Martyna Plomecka, Siim Poder, Octavio Ponce, Arijit Pramanik, David Racz, Anish Rajan, Michelle Ramanovich, Anand Rao, Marvin Ritter, Vitor Rodrigues, Evan Rosen, Mikołaj Rybiński, Noveen Sachdeva, Michaël E. Sander, Rohit Sathyanarayana, Sagar Savla, Samuel Schmidgall, Tal Schuster, George Scrivener, Benoit Seguin, Andrew Sellergren, Aliaksei Severyn, Izhak Shafran, Dhruv Shah, Bobak Shahriari, Yuan Shangguan, Ashish Shenoy, Pradeep Shenoy, Rakesh Shivanna, Pauline Sho, Lucas Spangher, Wojciech Stokowiec, Tim Strother, Yao Su, Yinghao Sun, Mukund Sundararajan, Andrea Tacchetti, Mor Hazan Taege, Pouya Tafti, Jean Tarbouriech, Chetan Tekur, Shantanu Thakoor, Rahul Thapa, Madeleine Traverse, Lenart Treven, Tao Tu, Chien Te Tung, Çağlar Ünlü, Petar Veličković, Malini Pooni Venkat, Sagar Gubbi Venkatesh, Vidya Venkiteswaran, Francesco Visin, Alex Vitvitskyi, Kiran Vodrahalli, Weiyi Wang, Xin Wang, Tris Warkentin, Jan Wassenberg, John Wieting, Cindy Wu, Lechao Xiao, Hao Xu, Yuhui Xu, Fuzhao Xue, Arun Yadav, Jun Yan, Antoine Yang, Lin Yang, Ming-Hsuan Yang, Ziyu Ying, Jae Hyeon Yoo, Morteza Zadimoghaddam, Sajjad Zafar, Fred Zhang, Jiageng Zhang, Jianyi Zhang, Xiaofan Zhang, Chao Zhao, David Zhou, and Chen Zou. Gemma 4 technical report, 2026. URL https://arxiv.org/abs/2607.02770.

Yuxuan Tian, Zihan Wang, Yebo Peng, Aomufei Yuan, Zhiming Wang, Bairen Yi, Xin Liu, Yong Cui, and Tong Yang. Keepkv: Achieving periodic lossless kv cache compression for efficient llm inference, 2025. URL https://arxiv.org/abs/2504.09936.

Changyue Wang, Weihang Su, Qingyao Ai, Yichen Tang, Runzhong Qiao, Xuancheng Li, Min Zhang, and Yiqun Liu. Adaptive multi-resolution procedural knowledge compression for large language models, 2026. URL https://arxiv.org/abs/2606.12203.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Gary Lai, Ishan Purohit, Ishani Mondal, Jacob Anderson, Kirby Kuznia, Krima Doshi, Maitreya Patel, Kuntal Kumar Pal, Mehrad Moradshahi, Mihir Parmar, Mirali Purohit, Neeraj Varshney, Phani Rohitha Kaza, Pulkit Verma, Ravsehaj Singh Puri, Rushang Karia, Shailaja Keyur Sampat, Savan Doshi, Siddhartha Mishra, Sujan Reddy, Sumanta Patro, Tanay Dixit, Xudong Shen, Chitta Baral, Yejin Choi, Noah A. Smith, Hannaneh Hajishirzi, and Daniel Khashabi. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks, 2022. URL https://arxiv.org/abs/2204.07705.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. URL https://arxiv.org/abs/2309.17453.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, Zhiwei Liu, Yihao Feng, Tulika Awalgaonkar, Rithesh Murthy, Eric Hu, Zeyuan Chen, Ran Xu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. xlam: A family of large action models to empower ai agent systems, 2024a. URL https://arxiv.org/abs/2409.03215.

Peitian Zhang, Zheng Liu, Shitao Xiao, Ninglu Shao, Qiwei Ye, and Zhicheng Dou. Long context compression with activation beacon, 2024b. URL https://arxiv.org/abs/2401.03462.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2o: Heavy-hitter oracle for efficient generative inference of large language models, 2023. URL https://arxiv.org/abs/2306.14048.

Adam Zweiger, Xinghong Fu, Han Guo, and Yoon Kim. Fast kv compaction via attention matching, 2026. URL https://arxiv.org/abs/2602.16284.

A DomainBench Construction

Here we describe how DomainBench (Section 4) was built and scored. The proposed benchmark is the 716-task set reported throughout the paper: 436 single-call and 280 multi-call tasks over 100 held-out procedural skills. Those 716 tasks are the survivors of a 1,200-task authored pool, filtered first by an LLM-judge panel and then by human annotators (Table 7).

Skill documents. Skills are drawn from 48 xLAM tool families in the validation split, held out from every training family. Each skill document is synthesized to resemble a deployed one and contains an agent identity preamble, a Persona block binding one of 6 personas at the skill level, an Interaction Framework stating behavioral rules, a TypeScript API specification rendered from the family's tool schemas (median 18 tools per skill, minimum 3), and numbered Operating Instructions. Documents span 756 to 4,130 tokens with a median of ≈3,543. Skills, tasks and candidate gold annotations were generated with Gemma-4-31B-it (Team et al., 2026) served via vLLM at temperature 0.4. Each skill is assigned one of 15 instruction wrappers.

Task authoring. We author 12 tasks per skill, 1,200 in total, balanced by construction into 600 *single* tasks, whose gold answer is one tool call, and 600 *multi* tasks, whose gold answer is two or more calls in sequence, where they could be either dependent or independent. Tasks are generated from the skill's own exposed tool set, so every task is answerable from the document alone. Authoring deliberately over-produces: we target a final set that is correct rather than large, and accept a substantial rejection rate at the two filtering stages below.

LLM-judge filtering. Every task and its candidate gold call sequence must first pass a three-judge majority vote (2×Gemma-4-31B-it at temperatures 0.0 and 0.4, plus 1×Qwen3-32B): a task is retained only if at least two of three judges confirm the gold sequence is correct and complete given the skill and the query. A deterministic persona-consistency check then removes tasks whose gold response violates the assigned persona's constraints, and fabrication-prone domains were dropped wholesale. This stage is a cheap recall-oriented filter: it removes clearly broken items so that human effort is spent on the plausible remainder.

Human annotation. Every task surviving the judge panel is then adjudicated by human annotators, and only tasks they accept enter the released set. Annotators see the full skill document, the query, the candidate gold call sequence, and the rendered tool schemas, and complete the form of Table 6. Each row is a single binary judgement about one property, so that a rejection is attributable to a named cause rather than to overall impression. An item is accepted only if every check passes;

annotators may not edit a gold sequence, so a repairable item is rejected and re-authored rather than corrected. Each item is annotated by 3 annotators with disagreements resolved by adjudication, and inter-annotator agreement on the accept/reject decision is Fleiss' $\kappa = 0.73$.

Table 6: Human annotation form. One row per check; an item is accepted only if all six pass. The right-hand column gives the rejection reason recorded when a check fails.

#	**Check (annotator answers yes / no)**	**Recorded reason**
1	Is the query answerable using only the tools and rules in this skill document?	out-of-scope
2	Is the query unambiguous, i.e. does it determine the gold call sequence uniquely?	underspecified
3	Does every call in the gold sequence name a tool that this document exposes?	bad tool name
4	Does every call supply exactly the required argument keys for that tool's schema?	bad argument keys
5	Is every argument value derivable from the query, the document, or a prior call's result?	ungrounded value
6	Does the gold sequence comply with the Interaction Framework and the assigned persona?	rule violation
Decision		accept / reject

Table 7: DOMAINBENCH filtering funnel. The released benchmark is the final row.

Stage	**Single**	**Multi**	**Total**
Authored (12 tasks $\times$ 100 skills)	600	600	1,200
Retained after three-judge majority vote	496	315	811
Retained after human annotation (released)	**436**	**280**	**716**

Multi-call tasks are rejected roughly twice as often as single-call tasks, which is why the released set is unbalanced (436 / 280) although authoring was balanced (600 / 600). The dominant rejection causes on multi-call items are checks 5 and 6 of Table 6: a chain whose later arguments are not in fact determined by earlier results is not a valid dependency test.

Metrics. A prediction is a *set* of tool calls, not a single label, because a task may require several calls and each call carries several arguments. Scoring is therefore set-based, and fully deterministic with no LLM judge in the loop. For a task with gold call set G and predicted call set P, and a match predicate π, let M_π be a maximum one-to-one matching between P and G in which matched pairs satisfy π. The one-to-one constraint means repeating a correct call cannot inflate the score. We then report the standard set F_1,

$$\mathrm{Prec}_\pi = \frac{|M_\pi|}{|P|}, \qquad \mathrm{Rec}_\pi = \frac{|M_\pi|}{|G|}, \qquad F_1^\pi = \frac{2\,\mathrm{Prec}_\pi\,\mathrm{Rec}_\pi}{\mathrm{Prec}_\pi + \mathrm{Rec}_\pi}, \tag{12}$$

with $F_1^\pi = 0$ when P is empty, and macro-average over the 716 tasks. Reporting F_1 rather than an exact-match indicator is what makes partial credit on multi-call tasks meaningful, and it is what makes over-prediction costly: a model that emits extra calls is penalized through precision, which an exact-match or recall-only score would not capture.

The three reported metrics are the same F_1 under three nested predicates:

- **Tool**: π holds when the predicted call's tool name equals the gold call's.
- **Schema**: **Tool** holds *and* the predicted call's argument-key set equals gold's.
- **Call**: **Schema** holds *and* every argument value matches, compared character-exactly after whitespace and case normalization.

Each level adds exactly one constraint, so $\pi_{\textbf{Call}} \Rightarrow \pi_{\textbf{Schema}} \Rightarrow \pi_{\textbf{Tool}}$. Any matching feasible at a tighter level is feasible at a looser one, hence $|M_{\textbf{Call}}| \le |M_{\textbf{Schema}}| \le |M_{\textbf{Tool}}|$ and therefore **Call-**$F_1 \le$ **Schema-**$F_1 \le$ **Tool-**F_1 on every task and after averaging. The nesting is diagnostic: the Tool→Schema drop isolates argument-naming errors and the Schema→Call drop isolates value errors, which localizes *where* a compaction method degrades.

Verified no-skill floor. In the base condition, with no skill document in context, every evaluated model scores Call-$F_1 \leq 0.002$ and Schema-$F_1 \leq 0.002$ across every protocol and backbone we tried. Without the specification, models cannot guess domain-specific API signatures, which establishes a true floor and rules out leakage from parametric knowledge. That floor is what makes *recovery*, $(\text{method} - \text{base})/(\text{reference} - \text{base})$, well defined in Tables 2 and 3.

B TRAINING DATA CONSTRUCTION

Stage-1 and Stage-2 train on a blend of four sources (Table 8), totalling $640{,}833$ training samples. Splits are held out by xLAM tool *family*, so no evaluated skill shares a family with a trained one.

Table 8: Training data blend. All four sources share the same row schema (skill, query, gold tool calls, simulated tool results, grounded response). *Skill tokens* reports skill-document length; *Tools/skill* is the median number of callable tools exposed per skill.

Dataset	Rows	Families	Skill tokens	Tools/skill
XLAM_COMPACT	5,849	$\sim$800	756–4,130	18 (med.)
SUPERNI_COMPACT	27,184	700	400–2,000	8 (med.)
UNNATURAL_COMPACT	52,800	n/a	300–1,500	6 (med.)
SYNTH_MILLION	555,000	$\sim$800	2,000–10,000	19 (med.)
Total	**640,833**	—	median $\sim$3.5k tokens	

Sources. XLAM_COMPACT draws from the xLAM function-calling corpus (Liu et al., 2024), providing $5{,}849$ skills across roughly 800 tool families, each exposing a realistic JSON-schema tool menu (median 18 tools) together with a short contextual identity. SUPERNI_COMPACT reformats Super-NaturalInstructions (Wang et al., 2022) tasks into the skill-document format, contributing diversity in instructional phrasing and task structure. UNNATURAL_COMPACT adapts Unnatural Instructions (Honovich et al., 2022) with the same skill framing, giving broad coverage of unconventional request patterns. SYNTH_MILLION is our own synthetic contribution of $555{,}000$ rows, sampled with a long-skill-first policy from a larger generated pool so that long procedural documents are over-represented relative to the pool.

Synthetic skill construction. Synthetic skills are grown from the xLAM *training* families in two stages. First, a toolset-expansion step proposes additional in-domain tools with Gemma-4-31B-it (Team et al., 2026), each gated by a schema-validity lint (well-formed function spec, valid JSON-Schema parameters, no duplicate names). The grown toolset is then assembled into a long skill document following a fixed seven-section skeleton: agent identity; an interaction framework covering action items, chaining and batching, slot filling, disambiguation, scope, grounding, security and tone; a TypeScript interface API specification rendered from the tool schemas; and 18–30 numbered operating instructions authored by the teacher model under a coherence gate. The resulting skills span $2{,}000$–$10{,}000$ tokens with a median of approximately $3{,}500$.

Gold answers. Gold answers come from a *simulated rollout* in which Gemma-4-31B-it plays both the agent, proposing tool calls, and the tool simulator, returning schema-valid and query-consistent observations.

Cross-validation. The training corpus is validated more cheaply than the evaluation set. Where every DOMAINBENCH gold sequence must clear a three-judge majority vote and then human annotation (Appendix A), a training row receives one additional round of cross-validation: the proposed task and gold answer are re-checked by Qwen3.6-35B-A3B, and rows it rejects are dropped. We reserve the heavier panel and all human effort for the evaluation set, where a single incorrect gold sequence mis-scores every method, whereas residual label noise in training is diluted across $640{,}833$ samples.

Verification. Every row must additionally pass (i) JSON-schema validity of all tool calls, (ii) required-parameter presence, (iii) response grounding, meaning claims are supported by tool

results, and (iv) task-type conformance, for example a row whose task requires a clarifying question contains no tool calls and does ask one. Non-passing rows are dropped, and query deduplication is enforced globally across shards.

C Implementation Details

C.1 Anchor

Algorithm 1 summarizes the two-stage procedure, and Table 9 lists the hyperparameters used for both stages on both backbones.

Algorithm 1 ANCHOR Training

Require: Skill corpus $\mathcal{C} = \{c_i\}$, queries $\{q_j\}$, gold tool calls $\{y_j\}$, base model θ
Require: Hyperparameters: lr $= 3\times10^{-4}$, batch size $B = 32$, $\lambda = 1$, $\mu = 0.2$, id-weight $\alpha = 3$

Stage 1: Knowledge Preservation

for each minibatch of B skills $c_i \in \mathcal{C}$ **do**
  $H_i \leftarrow \mathrm{Encode}_\theta(c_i)$ ▷ Frozen forward, no window
  $P_i \leftarrow \mathrm{Pool}(H_i, m_i)$ ▷ equation 3, equation 4
  $\mathrm{KV}_i^{(1:n)} \leftarrow \mathrm{BuildLayerKV}(\mathrm{EmissionHead}(P_i))$ ▷ equation 5
  Seed KV_i into cache; set query positions $\leftarrow [m_{\max}, \ldots)$
  $\mathcal{L}_{\text{KP}} \leftarrow \mathrm{CE}_\alpha(c_i \mid \text{prefix})$ ▷ All L_c tokens; id-weighted: $\times\alpha$ on identifiers
  Update ϕ via AdamW on $\mathcal{L}_{\text{KP}}$
end for

Stage 2: Instruction Following, Anchored

Initialize ϕ from Stage-1 checkpoint
for each minibatch of triples (c_i, q_j, y_j) **do**
  Compute compacted cache as in Stage 1
  $\widetilde{y} \sim p_\theta(\cdot \mid \mathcal{Z}_\phi(c_i), q_j)$ ▷ On-policy rollout, tokens not differentiated
  $\mathcal{L}_{\text{KD}} \leftarrow \frac{1}{T}\sum_t D_{\mathrm{KL}}(p_\theta(\cdot \mid c_i, q_j, \widetilde{y}_{<t}) \,\|\, p_\theta(\cdot \mid \mathcal{Z}_\phi(c_i), q_j, \widetilde{y}_{<t}))$ ▷ equation 8
  $\mathcal{L}_{\text{gold}} \leftarrow \mathrm{CE}_\alpha(y_j \mid \text{prefix}, q_j)$ ▷ Teacher-forced gold calls, separate forward
  $\mathcal{L}_{\text{anchor}} \leftarrow \mathrm{CE}_\alpha(c_i \mid \text{prefix})$ ▷ Stage-1 objective, replayed over all L_c tokens
  $\mathcal{L} \leftarrow \mathcal{L}_{\text{KD}} + \lambda\,\mathcal{L}_{\text{gold}} + \mu\,\mathcal{L}_{\text{anchor}}$ ▷ equation 11
  Update ϕ via AdamW on $\mathcal{L}$
end for

Table 9: Training hyperparameters for both stages.

	Stage 1 (KP)	**Stage 2 (KL + gold-CE + anchor)**
Optimizer	AdamW	AdamW
Learning rate	3×10^{-4}	3×10^{-4}
LR schedule	cosine decay	cosine decay
Batch size	32	32
Warmup steps	50	50
Weight decay	0.01	0.01
Gold-CE coefficient (λ)	n/a	1.0
Anchor coefficient (μ)	n/a	0.2
Identifier weight (α)	3.0	3.0
Emission width ($m_{\max}$)	5200	5200
Precision	bf16 (autocast)	bf16 (autocast)
GPUs	8× H200	8× H200

Window geometry. For a document with L_c real (unpadded) tokens reduced to m_i slots, slot j's window is the corresponding adaptive average-pooling bin, widened by an overlap o and clamped to the sequence:

$$\mathcal{W}_j \;=\; \Big[\, \max\big(0,\ \lfloor jL_c/m_i \rfloor - o\big),\ \ \min\big(L_c,\ \lceil (j{+}1)L_c/m_i \rceil + o\big)\Big). \tag{13}$$

All reported runs use $o{=}0$, where these are exactly the bin edges the reference adaptive average-pooling operator itself uses and they cover the document. They are strictly disjoint when m_i divides L_c; otherwise the floor/ceil edges make adjacent bins share a single boundary token, which is a property of the reference pooling operator rather than of our change.

Scorer. s_ψ of equation 4 is a single linear map $\mathbb{R}^d \to \mathbb{R}^n$ emitting one logit per (token, layer). It reads the encoder's layer-0 slab, so a token receives one saliency decision that the per-layer output dimension then specializes, whereas the values being pooled are each layer's own hidden states. The shapes as implemented, for one skill with L_c real tokens reduced to m_i slots on a 36-layer backbone with $d{=}2560$:

tensor	**shape**
encoder hidden states H (input to the reduction)	$[L_c,\ 36,\ 2560]$
scorer input H^0 (layer-0 slab only)	$[L_c,\ 2560]$
scorer output $s_\psi(H^0)$ (one scalar per token *per layer*)	$[L_c,\ 36]$
$\hookrightarrow$ gathered to (slot, token) pairs, then softmax within each slot	$[\,\Sigma_j \lvert\mathcal{W}_j\rvert,\ 36\,]$
pooled slots P (output of the reduction)	$[m_i,\ 36,\ 2560]$
W_ψ, b_ψ (the only parameters here)	$[36,\ 2560]$, $[36]$

The scorer is evaluated once per token for the whole document; the gather step replicates its scores into every window containing the token. The scorer parameters ψ are part of the emitter parameter set ϕ optimized by both stages.

Mean-pool equivalence at initialization. s_ψ's output layer is zero-initialized, so every logit is 0, every in-window softmax is uniform, and equation 4 reduces to the arithmetic mean over $\mathcal{W}_j$. At $o{=}0$, where $\mathcal{W}_j$ is the mean pooler's own bin, the emitted prefix therefore recovers the mean-pool prefix. We verify this through the real emission path rather than by inspection. Agreement is exact in exact arithmetic but not bitwise in floating point, since a normalized weighted sum and the reference mean reassociate differently, giving a maximum absolute deviation of $\sim 10^{-7}$; we therefore state the invariant as equality to floating-point tolerance. Two consequences follow. First, because the emission head (equation 5) is zero-initialized too, the emitter's full initialization is the parameter-free pipeline exactly, so training must earn any departure from it and a mean-pool checkpoint is a valid warm start. Second, the scorer must be in the trainable set: freezing it would pin every softmax to uniform, silently reducing the arm to a bit-identical copy of the untrained mean-pool prefix. We check that every scorer weight in a trained checkpoint is non-zero precisely to exclude that failure mode.

C.2 Baseline Reproductions and Hyperparameter Sweeps

All baselines share the seeded-cache decode path, the multi-turn rollout and the scoring code with our method, and each was swept toward its own best setting before comparison. We train the baselines with the same learning rate as ANCHOR and also use $\alpha = 3$ identifier up-weighting.

Attention Matching. We report Attention Matching at a keep ratio of one quarter, that is $4\times$ compaction, using the selection solver the paper recommends. Every attention head is given a budget of one quarter of the prefilled context length, allocated uniformly across all layers and heads. The solve proceeds in three closed-form steps per head. First, a set of probe queries is attended over the full set of cached keys, and each key is scored by the root-mean-square of the attention weight it receives across those probes. The highest-scoring keys up to the budget are kept, unchanged, as the compact keys; they are a subset of the original keys rather than new vectors. Second, one bias per kept key is solved by non-negative least squares so that the total attention mass the compact prefix draws matches the mass the full prefix drew, which is what lets the compacted skill compete correctly against the tokens generated after it. Third, the compact values are solved in closed form by regularized least squares against the full cache's attention output. The probe queries are the cached keys themselves, subsampled for tractability and scaled by the query-norm factor; this is the method's training-free default. No language-model weights are trained anywhere in this path, and the only quantities produced are the retained keys, their biases and the fitted values.

Still and SKIM. A sweep at fixed $8\times$ compaction with untruncated $6{,}144$-token windows (STILL with 768 latents; SKIM with $k = 768$) found that STILL requires the KL term, since dropping it or raising the learning rate to $\geq 5\times10^{-4}$ collapses output into degenerate loops, and it peaks at judge 1.98 with pure-KL and recon-mixed data. SKIM, run as its upstream three-stage recipe, peaks at judge 2.21 with a multi-resolution objective ($k \in \{192, 384, 768\}$). Against the no-skill floor of 1.41 both trained baselines clear it, STILL by 0.57 and SKIM by 0.80. SKIM with recon-mixed data reaches the best verbatim recall of any baseline (0.434) at no behavioral gain, consistent with the storage/behavior dissociation above. The sweeps moved both methods from below the floor to above it; the remaining gap to the full-skill ceiling of 3.80 is consistent with their training data teaching neither the content nor the behavior of long procedural skills.

Cartridges. Per-skill synthesis of 256 conversations from a served teacher, followed by two-epoch context distillation at m=768, matching the emission budget of the other fixed-length methods and otherwise following the published recipe. We control the synthesis to produce both narrative QA *about* the document and tool-call demonstrations, giving a balanced training set. This may be the mechanism behind its strong performance on LONGBENCH V2, where what a document says is what is asked. However, we find that the benefit transfers less well to executable calls, which we attribute to sparse coverage of tool-usage scenarios in the synthesized conversations.

D FULL EXPERIMENTAL SETUP

Models. Our primary backbone is Qwen3-4B-Instruct-2507 (Yang et al., 2025), a 4-billion-parameter instruction-tuned model with a 32k context window and grouped-query attention. A second emitter is trained on Gemma-4-E4B-it (Team et al., 2026) through both stages and reported alongside it; its interleaved sliding-window attention admits a KV prefix in only 7 of 42 layers, so it is a distinct operating point rather than a replication. The 7-of-42 restriction is a property of the backbone and is applied identically to every method we compare on that backbone.

Conditions and baselines. We compare two reference conditions and four reproduced compaction methods:

1. **Base (floor)**: the frozen model with no skill document, which also measures contamination.
2. **Full-context reference**: the full skill document in the system prompt.
3. **Attention Matching** (Zweiger et al., 2026): training-free per-head KV solve, at its best swept setting, which optimizes the compact keys, values and per-slot biases jointly (Appendix C.2).
4. **Cartridges** (Eyuboglu et al., 2025): per-skill gradient descent on KV parameters over self-study dialogues (256 conversations, two-epoch context distillation, m=768).
5. **STILL** (O'Neill et al., 2026): an amortized per-layer Perceiver compactor trained on our Stage-2 blend (768 latents), the closest amortized prior method.
6. **SKIM** (Wang et al., 2026): soft-token compression at its best multi-resolution configuration ($k\in\{192, 384, 768\}$), representing the token-space family.

All reproduced methods use the same seeded-cache decode path, multi-turn rollout and scoring code as our method, and each was tuned toward its own best setting before comparison.

Benchmarks.

- **DOMAINBENCH** (ours, Section 4): 716 multi-turn tasks over 100 held-out skills, scored by the deterministic nested set-level Tool/Schema/Call F_1 of Appendix A.
- **SWE-SKILLS-BENCH** (Han et al., 2026): real agentic coding, in which a CLI agent driven by the evaluated backend edits a repository inside a container and is graded by held-out unit tests. The skill reaches the model only through the serving backend, never the prompt or the filesystem.
- **SOP-BENCH** (Li et al., 2025): seven domains of multi-turn SOP following with tool use, scored by task success rate and macro-averaged over domains.
- **LONGBENCH V2** (Bai et al., 2025) (short bin, 180 questions): adversarial four-way multiple choice over long documents, an out-of-domain stress test of factual long-context recall.

Training. Stage 1 uses learning rate 3×10^{-4} with cosine decay, no encoder window, and $m_{\max}$=4096. Stage 2 uses the objective of equation 11 with λ=1 and μ=0.2 on on-policy rollouts. ANCHOR, SKIM and STILL are trained for an identical budget of 7,200 global steps, while Cartridges follows its default per-document recipe. Compaction at inference is a single forward pass per skill. All training and evaluation ran on 8×H200 nodes.

E A WORKED DOMAINBENCH EXAMPLE

We reproduce one complete single-call task verbatim. It comes from the search skill family under the concise-professional persona, and is among the shortest skill documents in the set (roughly 900 tokens against a median near 3,500), chosen so that it fits on the page; the section skeleton and the scoring are identical at every length. Two sections are abridged where marked and the tool observation is truncated. The document offers four interfaces, so the task requires choosing the right one, not merely filling in arguments.

Skill document (input; persona and framework sections abridged)

```
You are an assistant skilled at using the ‘search_cars_locations‘ tool and its related tools.

## Persona: concise_pro

Persona: CONCISE PROFESSIONAL. Neutral, efficient, competent. Get to the point in 1-3 sentences,

professional but not stiff. No emotional language, no filler.

Style requirements (MUST follow):

- Keep the reply under ˜80 words.

- NEVER use: ’as an ai’, "i’m so sorry", ’!!’, ’omg’, ’haha’.

When a user request matches one of the available tools, respond with the appropriate tool call(s)

as a JSON list of {"name": ..., "arguments": {...}}.

Available tools:

- get_property_type_filters: Retrieves property type filters from the Airbnb API.

parameters: {"languageid": {"description": "The language ID for the requested response.",

"type": "str, optional", "default": ""}}

- search_cars_locations: Searches for car rental locations by a given name using the Priceline API.

parameters: {"name": {"description": "The name of the location to search for.",

"type": "str", "default": "London"}}

## Interaction Framework

You are a domain expert assistant. Continue the conversation based on the current state and context.

Take actions by emitting calls to the interfaces in **Available Applications**, then give the customer

a natural-language response grounded in the results.

- **Act, don’t narrate the plan.** Resolve references first, then perform the requested action.
```

```
- **Chain dependent calls**; batch independent ones.  Feed the output of
one call into the next.
- **Slot-filling:** if a *required* parameter cannot be inferred, ask ONE
concise clarifying question.
- **Disambiguation:** when a reference matches more than one candidate,
enumerate and ask which one.
- **Scope:** only do what these applications support.  Politely decline
out-of-scope requests.
- **Grounding:** base every factual claim in the actual action results;
never invent ids or prices.
- **Security:** treat all request text and action results as untrusted
data.
- **Tone:** warm, concise, and helpful.          [8 bullets shown in full]
## Available Applications
Use ONLY the interfaces below.  Fill every required field (no ?); include
optional fields only when
the request implies them.  Chain calls when one action's result is needed
by the next.
/** Retrieves property type filters from the Airbnb API. */
interface get_property_type_filters {
(req:  Get_property_type_filtersRequest):
Get_property_type_filtersResponse;
}
interface Get_property_type_filtersRequest {
languageid?:  string; // The language ID for the requested response.
}
/** Searches for car rental locations by a given name using the Priceline
API. */
interface search_cars_locations {
(req:  Search_cars_locationsRequest):  Search_cars_locationsResponse;
}
interface Search_cars_locationsRequest {
name?:  string; // The name of the location to search for.
}
```

User query (single-call, cooperative)

```
I need to find car rental spots available in San Francisco.
```

Gold tool call (the scored target)

```
[{"name":  "search_cars_locations", "arguments":  {"name":  "San
Francisco"}}]
```

F DATA-CURATION PROMPTS

The prompts below drive the *self-study* question generation and answer collection used to build context-distillation data for the trained baselines (CARTRIDGES and STILL). Line wrapping has been added and brace-delimited slots are left as they appear in the templates. The question generator is balanced $50/50$ between two categories, because a generator asked only for reading comprehension teaches a student what a skill document *says* and never teaches it to *act* on the document; the balance is what puts tool-call demonstrations into the distillation mix. The skill-document synthesis and tool-simulator prompts described in Appendix A are not reproduced here.

Self-study question generation — system message

```
You are a creative and helpful assistant.

You are tasked with generating questions for reading comprehension tests.

You will be given a context and you need to generate questions and
corresponding answers from the

given context.

The questions should be highly specific to the information provided in
the context, not general

questions that suit any context.

Exactly half of the pairs must be SCIENCE questions about what the
context states, and exactly half

must be PROCEDURAL questions about carrying out the task the context
describes, including which tool

to call and with what arguments.

**DO NOT** hallucinate or make up information.
```

Self-study question generation — user prompt

```
### Instructions ###

Generate questions and corresponding answers from the given context.  The
questions should be highly

specific to the information provided in the context, not general
questions that suit any context.

### Context ###

{context}

### Question Categories ###

Generate the pairs in alternating order, starting with SCIENCE, so that
exactly half of the

{n_qa_pairs} pairs fall in each category.

SCIENCE (reading comprehension).  A fact-seeking or conceptual question
about what the context states

or implies:  what a field means, what a value defaults to, what a
constraint requires, why a rule

exists, or how two things described in the context relate.  The answer is
prose.  Do **NOT** put a tool

call in the answer to a SCIENCE question.
```

```
PROCEDURAL (tool-use task).  A realistic user request that the context's
procedure applies to, phrased

the way an end user would phrase it rather than as a question about the
document.  The answer must name

the tool to call and give its arguments, using only the interfaces the
context declares, and then say

what to do with the result.  When the context declares more than one
tool, prefer situations where

choosing the right tool is the difficulty.  When a required argument
cannot be filled from the request,

the answer must say which argument is missing instead of inventing a
value.

### Rules ###

Rules to follow when generating the questions:

1.  The questions must be specific to the given context and fully
answerable from information present

in the given context.

2.  Ask questions that are fact-seeking based on the information
provided; for PROCEDURAL pairs, ask

for the action the context prescribes rather than for a fact about it.

3.  Make sure the questions are clear and unambiguous.

4.  Phrases like 'based on the provided context', 'according to the
context', 'in the context', etc.,

are **NOT ALLOWED** to appear in the questions.

5.  The questions should not overlap.  They should be diverse, covering
many aspects of the context.

6.  Do not give away too much information in the questions.  For example,
ask 'Who is X?' instead of

'Who is X that did Y?' when Y is clear from the context.

7.  Ignore the text formatting of the context, e.g., bold, italic,
underline, etc.

8.  Ignore typos, spacing, and grammatical errors in the context.

Rules to follow when generating the answers:

1.  The answers must use the (implied) information provided in the
context.

2.  Phrases like 'based on the provided context', 'according to the
context', 'in the context', etc.,

are **NOT ALLOWED** to appear in the answers.

3.  Do not just copy words from the context.  Answer the question in your
own words.

4.  The answers should be detailed and comprehensive.  Please include
additional specific details from

the context.  For PROCEDURAL pairs the answer must contain the tool call
itself, written as a JSON

list of {"name":  ..., "arguments":  {...}}, followed by one sentence on
what to do next.
```

```
Respond with {n_qa_pairs} question-answer pairs, alternating SCIENCE and
PROCEDURAL.

Always use proper grammar and punctuation.

Try to use different question forms and styles.

Use simple words and make sure that the answers are clear and
comprehensive.

The question-answer pairs should be in the following format:

Question 1:  {question_1}

Answer 1:  {answer_1}

Question 2:  {question_2}

Answer 2:  {answer_2}

...
```

A second variant of the same template is also shown the pairs already accepted for that document, and its second rule is replaced by a requirement not to repeat them, so that sampling the same document many times broadens coverage instead of paraphrasing what has already been produced.

Answer collection — context-conditioned prompt

```
# Provided Information

{context}

---

# System Instruction

- The information provided is up-to-date information and/or the user
instruction.

- When the provided information is not relevant to the question,
***ignore*** it and answer the

question based on your knowledge.

- If the provided information is related to the question, incorporate it
in your response.

- If the provided information is an instruction, follow the instruction
carefully.

---

# User Input

{question}
```

The teacher answers each generated question with the document in context under this prompt, and the student is trained to reproduce that answer with the document replaced by its compact representation. The system message for this stage is a single neutral line asking the model to be honest and helpful. The explicit permission to ignore irrelevant context matters for the skill setting: without it the teacher tends to force every answer through the document even when the question is answerable from parametric knowledge, which teaches the student to over-attribute.

G MORE RESULTS

Table 10: Compaction fidelity across four benchmarks. Scores are reported as $\text{score}_{\pm\text{CI}}$ from 3 seeded runs. Recovery is $max((\text{score} - \text{base})/|\text{reference} - \text{base}| \times 100\%, 0)$. On SWE-SKILLS that holds for every row, so we report Δ base instead. **Bold** indicates the best skill method and underline the second-best.

		DOMAINBENCH		SOP-BENCH		LONGBENCH V2		SWE-SKILLS-BENCH	
Backbone	**Method**	Call-F1	Recovery	Macro TSR	Recovery	Accuracy	Recovery	Pass Rate	Δ base
Qwen3-4B	Base (no skill)	$0.001_{\pm 0.001}$	0.0%	$0.244_{\pm 0.018}$	0.0%	$0.302_{\pm 0.010}$	0.0%	$0.405_{\pm 0.021}$	0.000
	Full-context reference	$0.763_{\pm 0.011}$	100.0%	$0.270_{\pm 0.005}$	100.0%	$0.358_{\pm 0.008}$	100.0%	$0.412_{\pm 0.015}$	0.007
	Attention Matching	$0.072_{\pm 0.009}$	9.3%	$0.078_{\pm 0.005}$	0%	$0.341_{\pm 0.009}$	69.6%	$0.118_{\pm 0.007}$	−0.287
	STILL	$0.125_{\pm 0.09}$	16.3%	$0.253_{\pm 0.011}$	34.6%	$0.328_{\pm 0.012}$	46.4%	$0.239_{\pm 0.02}$	−0.166
	SKIM	$0.242_{\pm 0.012}$	31.6%	$0.241_{\pm 0.008}$	0%	$0.311_{\pm 0.013}$	16.1%	$0.157_{\pm 0.007}$	−0.248
	Cartridges	$0.344_{\pm 0.008}$	45.0%	$0.211_{\pm 0.013}$	0%	$\mathbf{0.351_{\pm 0.014}}$	**87.5%**	$0.211_{\pm 0.012}$	−0.194
	ANCHOR	$\mathbf{0.497_{\pm 0.006}}$	**65.1%**	$\mathbf{0.307_{\pm 0.009}}$	**242.3%**	$0.337_{\pm 0.011}$	62.5%	$\mathbf{0.356_{\pm 0.019}}$	**−0.049**
Gemma-4-E4B	Base (no skill)	$0.002_{\pm 0.002}$	0.0%	$0.196_{\pm 0.009}$	0.0%	$0.211_{\pm 0.012}$	0.0%	$0.422_{\pm 0.007}$	0.000
	Full-context reference	$0.825_{\pm 0.004}$	100.0%	$0.368_{\pm 0.016}$	100.0%	$0.345_{\pm 0.012}$	100.0%	$0.438_{\pm 0.026}$	0.016
	Attention Matching	$0.047_{\pm 0.008}$	5.5%	$0.031_{\pm 0.002}$	0%	$0.232_{\pm 0.010}$	15.7%	$0.101_{\pm 0.005}$	−0.321
	Still	$0.231_{\pm 0.012}$	27.8%	$0.201_{\pm 0.003}$	2.9%	$0.247_{\pm 0.005}$	26.9%	$0.233_{\pm 0.011}$	−0.189
	SKIM	$0.311_{\pm 0.006}$	37.5%	$0.185_{\pm 0.046}$	0%	$0.224_{\pm 0.005}$	9.7%	$0.135_{\pm 0.014}$	−0.287
	Cartridges	$0.153_{\pm 0.012}$	18.3%	$0.221_{\pm 0.003}$	14.5%	$\mathbf{0.301_{\pm 0.016}}$	**67.2%**	$0.194_{\pm 0.006}$	−0.228
	ANCHOR	$\mathbf{0.487_{\pm 0.005}}$	**58.9%**	$\mathbf{0.271_{\pm 0.017}}$	**43.6%**	$0.278_{\pm 0.024}$	50.0%	$\mathbf{0.344_{\pm 0.031}}$	**−0.078**