Learning to Compress Context into Reusable Experience Memory

Anonymous authors
Paper under double-blind review

Abstract

Historical examples and execution experience provide useful context for language models, but repeatedly processing their full text increases input length, prefill computation, and key/value (KV) cache requirements. We propose a context compression framework that converts retrieved experience into reusable *experience memory* for a frozen language model. The framework first induces common rules from training examples, then jointly encodes each rule and its associated example into a small set of layer-wise KV states. Segmented attention routes information through these memory positions, while compressor-side LoRA, task supervision, and distillation from a full-context view train the compressor to retain information relevant to answer or action prediction. At inference time, the frozen model reads the selected memory alongside the current input, separating experience encoding from online use. Experiments with Qwen2.5-7B-Instruct and LLaMA-3-8B-Instruct on ALFWorld and ContractNLI show improvements over textual retrieval baselines and competitive performance relative to parameter-efficient fine-tuning. In the ContractNLI efficiency evaluation, with caches precomputed and resident on the GPU, 15 memory positions replace approximately 1,347 experience tokens on average, yielding a $4.83\times$ online speedup over processing the original context.

1 Introduction

In-context learning enables large language models to adapt to tasks at inference time using instructions, examples, and external knowledge. Retrieval selects information relevant to the current input, supporting both knowledge access and task-specific prediction (Lewis et al., 2020; Liu et al., 2022). In multi-step agentic tasks, past execution trajectories can also serve as external memory for subsequent decisions (Ferraz et al., 2026). Context thus provides a practical interface through which language models can draw on information and experience beyond their parameters.

Processing this information as text, however, increases prefill computation, key/value (KV) cache storage, and attention costs during generation. Precomputing and reusing experience KV states avoids repeated encoding. For example, Dynamic Block-Sparse Attention (DBSA) pre-encodes example blocks and selects relevant caches for each query (Xiao et al., 2025). Yet generation still attends to token-level states of the selected content, so the cost of accessing experience grows with its length. Reducing this cost further calls for more compact context representations.

Existing approaches reduce context length in several ways. LLMLingua-2 learns to select tokens from the original text (Pan et al., 2024). Gist Tokens and AutoCompressors encode prompts or long contexts into reusable continuous states (Mu et al., 2023; Chevalier et al., 2023). ICAE and 500xCompressor use memory slots and layer-wise KV representations, respectively, to condition frozen models on compressed context (Ge et al., 2023; Li et al., 2025). Latent Context Compilation compiles a given context into buffer memory that can support subsequent queries (Li et al., 2026). These methods replace long text with shorter inputs or representations; their downstream utility depends on which information survives compression.

For context composed of examples and execution trajectories, useful information includes both specific conditions and operational details, as well as decision patterns shared across examples. Such commonalities are often implicit in individual accounts. We therefore summarize coarse-grained regularities before encoding them jointly with fine-grained example content. The summaries provide cues for identifying task-relevant information, while the examples retain the conditions and details

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.

under which those summaries apply. This organization is intended to help the compressor allocate limited representational capacity to information useful for subsequent inputs.

We introduce a latent context compression framework for frozen language models. We first induce shared applicability conditions and procedures from training examples and pair these descriptions with relevant examples. A shared compressor then encodes each pair into a fixed number of memory positions. For a new question or decision step, the model reads the selected memory alongside the current question or interaction history to predict an answer or action.

The compressor and reader share a pretrained backbone, with LoRA enabled only on the compression side and the reader kept frozen. Segmented attention restricts the flow of source information to the output through memory positions. Answer or action supervision trains this bottleneck to retain task-relevant information, supplemented by training signals from a full-context view.

Our contributions are threefold:

1. We introduce a reusable context compression framework that encodes retrieved experience into a fixed memory budget for question answering and interactive decision-making with a frozen language model, supporting precomputation and on-demand access.
2. We combine cross-example abstraction with example encoding, using shared regularities and concrete examples as complementary inputs to compression under a limited memory budget.
3. We train a segmented-attention memory bottleneck using answer or action supervision and auxiliary signals from a full-context view, while keeping the reading model frozen.

2 Method

Our framework converts historical experience into *experience memory* for a frozen language model. It first induces common rules from examples, then jointly encodes each rule and a related example into layer-wise key/value (KV) states. Given a new input, the model reads the retrieved memory to generate an answer or action. Figure 1 summarizes the framework.

2.1 Problem Formulation

Let q denote the current input and $a = (a_1, \ldots, a_{N_a})$ the target output of N_a tokens, including the end token. For question answering, q contains the question and the material to be analyzed. For interactive tasks, q contains the task goal, current observation, and interaction history, and a specifies the next action. A retriever selects K examples from an experience bank based on q. In retrieval order, each example is denoted by $e_i = (x_i, y_i)$, where x_i is a historical input and y_i its reference output.

Given a pretrained language model p_θ, we compress the selected examples and their associated rules into memory M_q with at most B positions. The model predicts $p_\theta(a \mid M_q, q)$ with its parameters θ frozen. Only retrieved experience is compressed; the current input q is provided in its original textual form.

2.2 Experience Abstraction

Training examples are organized into groups $\mathcal{E}_g$ according to problem structure and operational patterns. An induction model summarizes shared applicability conditions, key steps, and exceptions as procedural rules r_g. Let $g(i)$ denote the group associated with example e_i. Each rule and example form a source unit c_i:

$$r_g = \text{Summarize}(\mathcal{E}_g), \qquad c_i = [r_{g(i)}; x_i; y_i]. \tag{1}$$

Semicolons denote sequential concatenation. Rules summarize regularities across examples, while examples retain concrete conditions and execution details. The induction set can be larger than the currently retrieved set, and a rule may be associated with multiple examples. Candidate rules are validated and revised on development data, then fixed as textual inputs to compression. Appendix A.1 provides construction details.

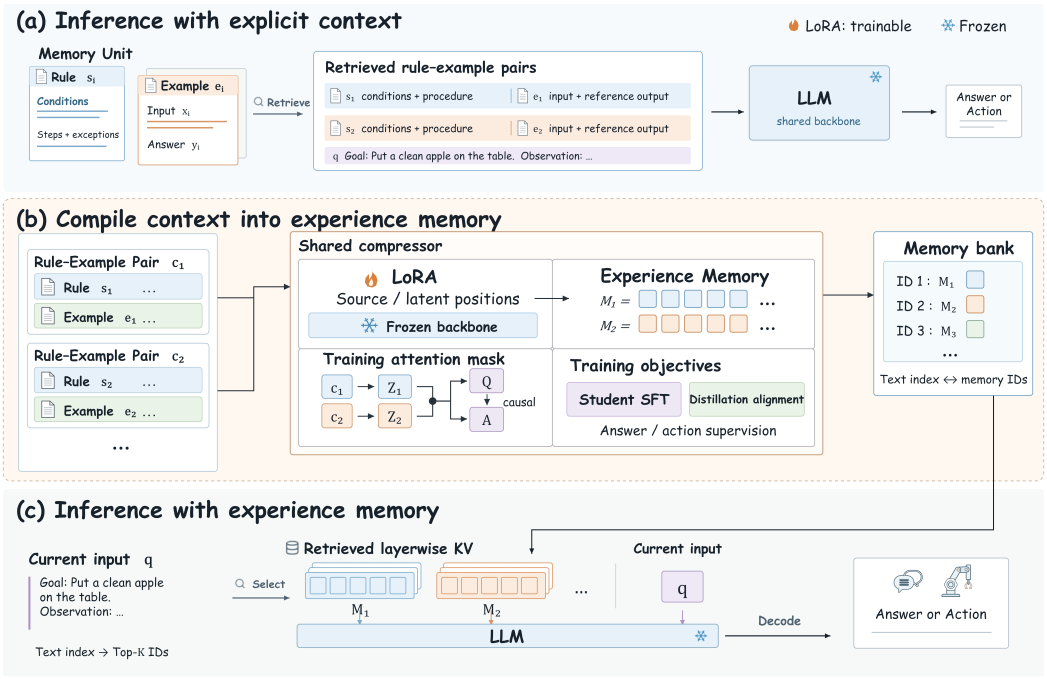


Figure 1: **Overview of experience memory construction and inference.** (a) The model reads retrieved rule–example pairs together with the current input, illustrated by an ALFWorld goal and observation. (b) A shared compressor encodes each pair into experience memory with segmented attention. LoRA is applied to source and memory positions, while the backbone remains frozen. Student supervision and distillation alignment train the compressor, and the resulting memories are stored in a memory bank. (c) The current input retrieves memory IDs through a text index. The corresponding layerwise KV states are loaded alongside the current input for the frozen model to predict an answer or action. Each source unit pairs a rule $s_i = r_{g(i)}$ with an example e_i; Z_i denotes the i-th use of the shared memory-input matrix $\mathbf{Z}$.

2.3 Joint Context Compression

Compressor inputs. Let $E(\cdot)$ be the backbone's frozen token embedding map, which converts tokenized text into a matrix of d-dimensional vectors. The shared compressor $\mathcal{C}_\phi$ appends a memory input matrix $\mathbf{Z} \in \mathbb{R}^{m \times d}$ to $E(c_i)$, producing $[E(c_i); \mathbf{Z}]$. The m rows of $\mathbf{Z}$ are learnable vectors shared across all units. The compressor and reader use the same frozen backbone. LoRA (Hu et al., 2021) is applied only at source and memory positions; all other positions use the original backbone transformations. The trainable parameters ϕ comprise the LoRA parameters and $\mathbf{Z}$.

Segmented attention. Each source unit is encoded causally within its own block. Its memory positions attend to that source and to causally visible positions within the same memory block. Current-input and output positions attend to all memory blocks and their own causal history, but cannot access source units directly. Thus, each rule–example pair is encoded locally, while memories from different units jointly condition generation. Appendix A.2 specifies attention visibility, including access to the shared task prefix.

Experience memory. For a fixed template and position layout, the compressor retains the KV states at memory positions in every layer:

$$M_i = \mathcal{C}_\phi(c_i) = \left(\mathbf{K}_i^{(\ell)}, \mathbf{V}_i^{(\ell)}\right)_{\ell=1}^{L}, \qquad M_q = \mathrm{Concat}_{\mathrm{pos}}(M_1, \ldots, M_K). \tag{2}$$

Here, L is the number of layers, and $\text{Concat}_{\text{pos}}$ concatenates KV states along the position dimension in each layer, with $Km \leq B$. The shared input parameters $\mathbf{Z}$ are distinct from the content-dependent states M_i. Cache dimensions and extraction are specified in Appendix A.3.

2.4 Training Objective

Training conditions on $M_q(\phi)$ constructed from historical examples and updates only the compression parameters ϕ, keeping θ frozen. We present unweighted SFT objectives below; ContractNLI applies fixed class weights to the SFT terms, but not to the KL term.

Student supervision. The student view uses segmented attention to predict the next token from compressed memory and the ground-truth output prefix $a_{<t}$. Its supervised loss is

$$\mathcal{L}_S(\phi; q, a) = -\frac{1}{N_a} \sum_{t=1}^{N_a} \log p_\theta(a_t \mid M_q(\phi), q, a_{<t}). \tag{3}$$

Supervision covers answer or action tokens and the end token. Gradients propagate through the frozen reader to the compressor-side LoRA parameters and shared input matrix $\mathbf{Z}$.

Teacher supervision. The teacher view is a separate forward pass of the same model with access to the original context. It uses the same rules, examples, memory positions, current input, and ground-truth output prefix as the student, but replaces segmented attention with standard causal attention. Predictions can therefore depend directly on both the source text and memory. The two views share parameters and the positional scope of LoRA, while computing their intermediate states separately.

Let $p_{T,t}(a_t)$ denote the teacher's probability of target token a_t. Teacher supervision uses the same reference answer or action as student supervision:

$$\mathcal{L}_T = -\frac{1}{N_a} \sum_{t=1}^{N_a} \log p_{T,t}(a_t). \tag{4}$$

Distribution distillation. Distillation aligns predictions with and without direct access to the source context (Hinton et al., 2015). Let $p_{S,t}$ and $p_{T,t}$ denote the student's and teacher's distributions over the full vocabulary, conditioned on the same ground-truth prefix $a_{<t}$. We minimize the forward KL from teacher to student:

$$\mathcal{L}_F = \frac{1}{N_a} \sum_{t=1}^{N_a} D_{\text{KL}}(\text{sg}[p_{T,t}] \| p_{S,t}) \,. \tag{5}$$

The stop-gradient operator sg restricts gradients from this term to the student distribution. Teacher predictions are computed using the current shared parameters and recomputed at subsequent training steps.

Final loss. The overall objective is

$$\mathcal{L} = \mathcal{L}_S + \lambda_T \mathcal{L}_T + \lambda_K \mathcal{L}_F, \qquad \lambda_T, \lambda_K \geq 0. \tag{6}$$

2.5 Inference with Experience Memory

At inference time, we fix ϕ, retrieve experience units relevant to the current input, and obtain their compressed memories. The reader retains the selected memory and the cache required by the fixed template, discards the source-unit KV states, and generates the output conditioned on q. LoRA is disabled at generation positions.

Experience memory can be precomputed and reused across requests with compatible model parameters, templates, and position layouts. When a new retrieval combination changes the layout, the system uses a matching cache or re-encodes affected units. Cache reuse preserves the position conventions used during encoding, as detailed in Appendix A.3.

3 Related Work

3.1 Efficient Context Utilization

Efficient context utilization addresses both input length and repeated computation. LLMLingua-2 shortens text through learned token selection (Pan et al., 2024). Gist Tokens, AutoCompressors, and ICAE map prompts or contexts to a small number of continuous states; AutoCompressors also studies replacing in-context demonstrations with compressed representations (Mu et al., 2023; Chevalier et al., 2023; Ge et al., 2023).

500xCompressor uses encoder-side LoRA to produce layer-wise KV states at a small number of positions for a frozen reader (Li et al., 2025). LCC optimizes a temporary encoding module for a given context, using reconstruction and self-alignment to produce reusable buffer memory (Li et al., 2026). Our framework also freezes the reader, but learns a compressor shared across context units and supervises memory through answers to subsequent inputs.

Dynamic Block-Sparse Attention (DBSA) pre-encodes an example pool with block-sparse attention and selects cached blocks for each query, reducing repeated computation for dynamic contexts (Xiao et al., 2025). Generation still accesses token-level KV states of the selected examples. We additionally encode each source unit into a fixed number of memory positions, using representation compression to reduce reading costs and cache reuse to avoid repeated encoding.

3.2 Memory-Augmented Language Models

Language models retrieve external knowledge or existing examples to supplement the current input. RAG combines pretrained models with a non-parametric document index and conditions generation on retrieved text (Lewis et al., 2020). Liu et al. (2022) selects in-context demonstrations by semantic similarity, providing input–answer examples relevant to the query. Documents and examples thereby serve as information sources outside model parameters.

In multi-step interaction, reusable information extends to execution trajectories, observations, and feedback. Generative Agents stores and retrieves experiences to support planning, while Reflexion uses reflections derived from task feedback to guide subsequent attempts (Park et al., 2023; Shinn et al., 2023). ExpeL extracts insights from successful and failed experiences and retrieves successful trajectories from similar tasks as demonstrations. RAP selects relevant trajectory segments based on the current task and execution state (Zhao et al., 2024; Kagaya et al., 2024).

Experience can also be distilled into transferable task guidance. Prompt-Level Distillation summarizes teacher reasoning patterns as instructions (Badhe & Shah, 2026), whereas LatentSkill maps procedural descriptions to LoRA adapters through a hypernetwork (Yu et al., 2026). These approaches move beyond individual records to more general descriptions or adaptation parameters.

xRAG converts document embeddings into conditioning representations, and One Token encodes evidence into compact latent memory to reduce the text processed by the generator (Cheng et al., 2024; Zheng et al., 2026). Our framework combines experience abstraction with latent compression: it explicitly summarizes cross-example regularities, jointly encodes them with relevant examples, and learns reusable compact memory through task supervision.

4 Experiments

4.1 Experimental Setup

Models. We use Qwen2.5-7B-Instruct (Bai et al., 2023) and LLaMA-3-8B-Instruct (Grattafiori et al., 2024). Each compressor is initialized independently from the original model weights and uses the corresponding tokenizer and chat template. Compression methods train only the LoRA parameters at source and memory positions and the shared memory inputs; the reader remains frozen.

Datasets. We evaluate document-level natural language inference and text-based interaction.

Table 1: Qwen2.5-7B-Instruct results on ALFWorld and ContractNLI. All denotes overall ALFWorld success. ALFWorld results for ReAct, ExpeL, and MemP are taken from SkillRL (Xia et al., 2026). All metrics are percentages; — indicates unavailable results.

	ALFWorld Task						All↑	ContractNLI	
Method	Pick	Look	Clean	Heat	Cool	Pick2		Acc.	M-F1
No parameter updates									
Base	33.40	21.60	19.30	6.90	2.80	3.20	14.80	61.76	50.61
Retrieval examples	70.83	22.22	51.61	73.91	61.90	58.82	57.46	77.03	73.53
Retrieval examples + rules	83.33	50.00	80.65	52.17	23.81	64.71	61.19	75.49	72.27
ReAct (Yao et al., 2023)	48.50	35.40	34.30	13.20	18.20	17.60	31.20	—	—
ExpeL (Zhao et al., 2024)	21.00	67.00	55.00	52.00	71.00	6.00	46.30	—	—
MemP (Fang et al., 2025)	54.30	38.50	48.10	56.20	32.00	16.70	41.40	—	—
Parameter-efficient tuning									
LoRA SFT	70.83	55.56	74.19	73.91	76.19	35.29	66.42	80.11	74.91
Prefix-tuning	66.67	11.11	16.13	43.48	28.57	29.41	32.84	46.50	26.92
Experience memory (LoRA on compressor)									
Experience memory (examples)	83.33	22.22	70.97	95.65	71.43	47.06	67.91	78.29	74.77
Experience memory (rules+examples)	75.00	50.00	83.87	86.96	33.33	76.47	69.40	80.95	77.15

1. **ContractNLI.** Given a contract and a hypothesis, the model predicts one of three labels: Entailment, Contradiction, or NotMentioned (Koreeda & Manning, 2021). We report Accuracy and Macro-F1.
2. **ALFWorld.** ALFWorld is a text-based interactive environment aligned with ALFRED household tasks (Shridhar et al., 2020). The model acts on a natural-language goal and environment observations across six task types: Pick, Look, Clean, Heat, Cool, and Pick2. We report success rates for each type and overall success across evaluation episodes (All), with at most 50 actions per episode.

Baselines. We compare three groups of methods. **No parameter updates:** Base uses only the current input; Retrieval examples provides the full text of three retrieved examples; Retrieval examples + rules adds the associated rules to the same examples. For ALFWorld, we also include ReAct (Yao et al., 2023), ExpeL (Zhao et al., 2024), and MemP (Fang et al., 2025), with results reported by SkillRL (Xia et al., 2026). **Parameter-efficient tuning:** LoRA SFT trains low-rank adapters in the generation model; Prefix-tuning (Li & Liang, 2021) learns 15 prefix positions. Both keep the backbone weights frozen. **Experience memory:** Experience memory (examples) compresses retrieved examples alone, whereas Experience memory (rules+examples) jointly compresses rules and examples. Both use Student SFT and forward KL to train the compressor, with the resulting layer-wise KV memory read by the frozen model.

Implementation details. Experiments use $4\times$ NVIDIA A100 GPUs. Rules are induced from training examples, and retrieval excludes the current example and examples from the same source. ContractNLI is grouped by contract document, while ALFWorld training experience and evaluation tasks are separated by scene. Within each task, the baselines we train share the training data, effective batch size, update budget, task instructions, and output interface. Hyperparameters and checkpoints are selected using development data only. Appendix B provides training and evaluation details.

4.2 Main Results on Qwen and LLaMA

Tables 1 and 2 compare Qwen and LLaMA on ALFWorld and ContractNLI.

Comparison with textual context. Experience memory (rules+examples) outperforms both textual retrieval variants on the aggregate metrics for both models and tasks. Compared with Retrieval examples, ALFWorld overall success increases from 57.46% to 69.40% for Qwen and from 26.12% to 55.97% for LLaMA; ContractNLI Accuracy and Macro-F1 also improve. The Qwen ALFWorld result exceeds the reported ReAct, ExpeL, and MemP results in Table 1. These findings support

Table 2: LLaMA-3-8B-Instruct results on ALFWorld and ContractNLI. Metrics follow Table 1.

	ALFWorld Task						All↑	ContractNLI	
Method	Pick	Look	Clean	Heat	Cool	Pick2		Acc.	M-F1
No parameter updates									
Base	20.83	5.56	6.45	8.70	4.76	5.88	8.96	50.98	36.65
Retrieval examples	41.67	0.00	22.58	60.87	9.52	11.76	26.12	67.79	65.34
Retrieval examples + rules	50.00	11.11	16.13	43.48	14.28	5.88	24.62	71.27	65.93
Parameter-efficient tuning									
LoRA SFT	91.67	33.33	67.74	82.61	52.38	11.76	60.45	72.97	68.92
Prefix-tuning	54.17	0.00	32.26	39.13	19.05	5.88	27.61	49.72	33.89
Experience memory (LoRA on compressor)									
Experience memory (examples)	75.00	27.78	54.84	65.22	52.38	58.82	56.72	74.09	70.47
Experience memory (rules+examples)	91.67	11.11	74.19	73.91	23.81	35.29	55.97	75.49	72.57

learning task-conditioned compact representations as an effective way to use retrieved experience beyond supplying its original text.

Comparison with parameter-efficient tuning. Joint compression exceeds LoRA SFT in ContractNLI Macro-F1 by 2.24 points for Qwen and 3.65 points for LLaMA, and achieves higher ALFWorld overall success for Qwen. However, it falls below LoRA SFT on LLaMA's ALFWorld evaluation, indicating that the advantage depends on the model and task. Experience memory remains competitive while keeping the reader frozen. Experience is stored as KV memory that can be precomputed, retrieved, and replaced, allowing relevant memories to be loaded for each input. Table 4 reports the online efficiency of precomputed memory.

Joint compression of rules and examples. Adding rules to example compression improves ContractNLI Macro-F1 by 2.38 points for Qwen and 2.10 points for LLaMA. On ALFWorld, overall success increases by 1.49 points for Qwen but decreases by 0.75 points for LLaMA. One possible explanation is that inducing rules before joint encoding provides complementary information about shared patterns and example-specific conditions. This interpretation is consistent with the motivation for multi-stage compression, although the benefit of rules remains task- and model-dependent.

4.3 Memory Capacity

Figure 2 (left) examines memory capacity for Qwen2.5-7B-Instruct on ContractNLI, using joint rules+examples compression with Student SFT and forward KL. We fix three retrieved examples and vary the memory length per example, $m \in \{1, 3, 5, 10\}$. Increasing m from 1 to 5 improves Accuracy by 2.24 points and Macro-F1 by 4.32 points, suggesting that a single position may constrain retention of task-relevant information. Doubling the total memory from 15 to 30 positions ($m = 5$ to $m = 10$), however, yields only a 0.14-point Accuracy gain, with essentially unchanged Macro-F1 (77.15% vs. 77.13%). The diminishing gains make $m = 5$ a favorable trade-off among the tested settings: it approaches the performance of $m = 10$ with half as many memory positions.

4.4 Number of Retrieved Examples

Figure 2 (right) varies the number of retrieved examples under the same joint compression objective, fixing $m = 5$ memory positions per example. Increasing K from 1 to 3 improves Accuracy by 1.26 points and Macro-F1 by 3.66 points. With five examples, Macro-F1 rises further from 77.15% to 78.29%, while Accuracy remains close to the three-example result (80.81% vs. 80.95%). Additional examples therefore mainly benefit Macro-F1 in this comparison. Three examples achieve the highest Accuracy with 15 memory positions, whereas five achieve the highest Macro-F1 with 25 positions. Because total memory grows with K, this experiment measures the combined effect of greater example coverage and increased memory capacity.

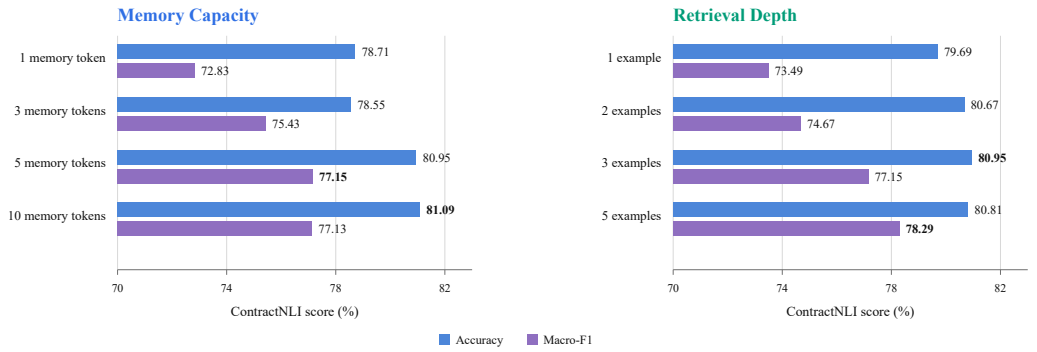


Figure 2: **Effects of memory capacity and retrieval depth on ContractNLI.** Results use Qwen2.5-7B-Instruct. Left: memory tokens per example, with three examples. Right: retrieved examples, with five memory tokens per example. Blue and purple denote Accuracy and Macro-F1; bold values mark each metric's maximum within a panel. Horizontal axes start at 70%.

4.5 Auxiliary Supervision

We further examine whether supervision from a full-context view improves experience memory under joint rules+examples compression. Table 3 adds Teacher SFT, forward KL, or reverse KL separately to Student SFT. All runs are initialized independently from the original model and use the same inputs, memory budget, compressor architecture, and training budget. Both KL variants stop gradients through the teacher distribution and do not additionally include Teacher SFT.

Forward KL achieves the highest ALFWorld overall success, ContractNLI Accuracy, and ContractNLI Macro-F1 in this comparison, improving over Student SFT by 2.98, 1.26, and 0.83 points, respectively. Teacher SFT also improves all three metrics, indicating that the full-context view provides useful supplementary supervision. Reverse KL improves ContractNLI Accuracy but not Macro-F1, and reduces ALFWorld overall success. Auxiliary supervision thus depends on the specific objective, with forward KL producing more consistent gains across the two tasks.

Table 3: Effect of auxiliary supervision. All configurations retain Student SFT; the last three add Teacher SFT, forward KL, or reverse KL, respectively. All metrics are percentages; bold indicates the best value in each column.

	ALFWorld Task						All↑	ContractNLI	
Training objective	Pick	Look	Clean	Heat	Cool	Pick2		Acc.	M-F1
Student SFT loss	87.50	27.78	77.42	73.91	**52.38**	64.71	66.42	79.69	76.32
+ Teacher SFT loss	**95.83**	27.78	77.42	78.26	47.62	70.59	68.66	80.53	76.66
+ Forward KL loss	75.00	**50.00**	**83.87**	**86.96**	33.33	**76.47**	**69.40**	**80.95**	**77.15**
+ Reverse KL loss	**95.83**	5.56	80.65	82.61	47.62	58.82	65.67	80.81	76.29

4.6 Inference Efficiency

Table 4 compares online inference with original text, raw KV caching, and experience memory. Let C be the original experience length, $R = Km$ the memory length, n the current-input length, and L and d the number of layers and hidden width. We assume the feed-forward width is proportional to d and omit the fixed shared prefix and output-head computation common to all methods. Both caching methods reuse precomputed, GPU-resident KV states; the compressed path retains only experience memory and the shared prefix. Let $\mathcal{T}_{\text{raw}}$, $\mathcal{T}_{\text{rawKV}}$, and $\mathcal{T}_{\text{memory}}$ denote the prefill computation for Raw context, Raw KV cache, and Experience memory KV, respectively:

$$\begin{aligned} \mathcal{T}_{\text{raw}} &= \mathcal{O}\big(L[(C+n)d^2 + (C+n)^2 d]\big)\,, \\ \mathcal{T}_{\text{rawKV}} &= \mathcal{O}\big(L[nd^2 + n(C+n)d]\big)\,, \\ \mathcal{T}_{\text{memory}} &= \mathcal{O}\big(L[nd^2 + n(R+n)d]\big)\,. \end{aligned} \tag{7}$$

Raw KV caching avoids re-encoding experience, but the current input still attends to all original KV states. Experience memory further reduces the experience-dependent attention term per layer from nCd to nRd. Since memory is precomputed, its hidden states need not be recomputed. For a fixed model and precision, experience KV storage decreases to a fraction R/C of the original.

ContractNLI predicts labels from logits at the answer position, so the measurements in Table 4 correspond to this prefill computation, without subsequent long-form generation. Raw KV cache reduces mean online tokens from 1884.00 to 467.63 and provides a $2.98\times$ speedup. At the same online encoding length, Experience memory KV reduces experience positions from 1347.38 to 15 and experience KV storage from 73.685 MiB to 0.820 MiB. It achieves a $4.83\times$ speedup over Raw context, or approximately $1.62\times$ over raw KV caching. These measurements show the additional efficiency benefit of shortening the experience representation beyond cache reuse alone.

Table 4: Online inference efficiency of Qwen2.5-7B-Instruct on ContractNLI. Caches are precomputed and GPU-resident. Speedup is relative to Raw context and excludes cache construction and loading. Experience KV storage excludes the shared prefix.

Inference method	Mean experience positions↓	Mean online tokens↓	Experience KV (MiB)↓	Speedup vs. Raw context↑
Raw context	1347.38	1884.00	73.685	$1.00\times$
Raw KV cache	1347.38	467.63	73.685	$2.98\times$
Experience memory KV	**15.00**	**467.63**	**0.820**	**4.83**$\times$

Note: Raw context constructs experience KV states within each request rather than reusing a precomputed cache. Online tokens count positions re-encoded for the current request.

5 Conclusion

We presented a framework for compressing historical experience into reusable experience memory. It induces rules from training examples and jointly encodes rules and related examples into a small set of layer-wise KV states. Segmented attention and task supervision train the compressor to retain information useful for answer or action prediction while keeping the reader frozen. Across Qwen and LLaMA, experiments on ALFWorld and ContractNLI show improvements over textual retrieval and performance matching or exceeding LoRA SFT on several metrics. Inference with precomputed memory further reduces experience KV storage and online prediction time beyond the gains from cache reuse alone. By separating experience encoding from online reading, the framework supports on-demand access to historical experience in a compact form.

Limitations

Information retention under a fixed budget. A finite number of memory positions requires the compressor to select which information to retain. Task supervision favors information relevant to the training objective, but rare conditions, exceptions, or fine-grained details may be weakened during compression. A fixed memory budget may therefore be insufficient when subsequent inputs require different evidence or reasoning details. Memory capacity remains a trade-off against the information demands of the downstream task.

Dependence on retrieval and rule induction. Experience memory depends on the quality of retrieved examples and their associated rules. Joint encoding does not automatically correct irrelevant examples or inaccurate generalizations. Our current pipeline fixes induced rules before training the compressor, rather than jointly optimizing retrieval, induction, and encoding. The benefits of rules may therefore vary with retrieval relevance and induction quality, and need not extend uniformly across tasks and models.

REFERENCES

Sanket Badhe and Deep Shah. Prompt-Level Distillation: A Non-Parametric Alternative to Model Fine-Tuning for Efficient Reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 2131–2147. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-industry.142. URL https://aclanthology.org/2026.acl-industry.142/.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen Technical Report. *arXiv preprint arXiv:2309.16609*, 2023. doi: 10.48550/arXiv.2309.16609. URL https://arxiv.org/abs/2309.16609.

Xin Cheng, Xun Wang, Xingxing Zhang, Tao Ge, Si-Qing Chen, Furu Wei, Huishuai Zhang, and Dongyan Zhao. xRAG: Extreme Context Compression for Retrieval-augmented Generation with One Token. In *Advances in Neural Information Processing Systems*, volume 37, pp. 109487–109516. Curran Associates, Inc., 2024. doi: 10.52202/079017-3476. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/c5cf13bfd3762821ef7607e63ee90075-Abstract.html.

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

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring Agent Procedural Memory. *arXiv preprint arXiv:2508.06433*, 2025. doi: 10.48550/arXiv.2508.06433. URL https://arxiv.org/abs/2508.06433.

Thomas Palmeira Ferraz, Romain Deffayet, Vassilina Nikoulina, Hervé Déjean, and Stéphane Clinchant. Retrieval-augmented LLM agents: Learning to learn from experience. *arXiv preprint arXiv:2603.18272*, 2026. doi: 10.48550/arXiv.2603.18272. URL https://arxiv.org/abs/2603.18272.

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. *arXiv preprint arXiv:2307.06945*, 2023. doi: 10.48550/arXiv.2307.06945. URL https://arxiv.org/abs/2307.06945.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024. doi: 10.48550/arXiv.2407.21783. URL https://arxiv.org/abs/2407.21783.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. doi: 10.48550/arXiv.1503.02531. URL https://arxiv.org/abs/1503.02531.

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. *arXiv preprint arXiv:2106.09685*, 2021. doi: 10.48550/arXiv.2106.09685. URL https://arxiv.org/abs/2106.09685.

Tomoyuki Kagaya, Thong Jing Yuan, Yuxuan Lou, Jayashree Karlekar, Sugiri Pranata, Akira Kinose, Koki Oguri, Felix Wick, and Yang You. RAP: Retrieval-augmented planning with contextual memory for multimodal LLM agents. *arXiv preprint arXiv:2402.03610*, 2024. doi: 10.48550/arXiv.2402.03610. URL https://arxiv.org/abs/2402.03610.

Yuta Koreeda and Christopher D. Manning. ContractNLI: A Dataset for Document-level Natural Language Inference for Contracts. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 1907–1919. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.findings-emnlp.164. URL https://aclanthology.org/2021.findings-emnlp.164/.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

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

Zeju Li, Yizhou Zhou, and Qiang Xu. Latent Context Compilation: Distilling long context into compact portable memory. *arXiv preprint arXiv:2602.21221*, 2026. doi: 10.48550/arXiv.2602.21221. URL https://arxiv.org/abs/2602.21221.

Zongqian Li, Yixuan Su, and Nigel Collier. 500xCompressor: Generalized Prompt Compression for Large Language Models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25081–25091. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1219. URL https://aclanthology.org/2025.acl-long.1219/.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, William B. Dolan, Lawrence Carin, and Weizhu Chen. What Makes Good In-Context Examples for GPT-3? In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pp. 100–114. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.deelio-1.10. URL https://aclanthology.org/2022.deelio-1.10/.

Jesse Mu, Xiang Li, and Noah Goodman. Learning to Compress Prompts with Gist Tokens. In *Advances in Neural Information Processing Systems*, volume 36, pp. 19327–19352. Curran Associates, Inc., 2023. doi: 10.52202/075280-0848. URL https://proceedings.nips.cc/paper_files/paper/2023/hash/3d77c6dcc7f143aa2154e7f4d5e22d68-Abstract-Conference.html.

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 *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 963–981. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.57. URL https://aclanthology.org/2024.findings-acl.57/.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652. Curran Associates, Inc., 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*, 2020. doi: 10.48550/arXiv.2010.03768. URL https://arxiv.org/abs/2010.03768.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, Zeyu Zheng, Cihang Xie, and Huaxiu Yao. SkillRL: Evolving Agents via Recursive Skill-Augmented Reinforcement Learning. *arXiv preprint*

arXiv:2602.08234, 2026. doi: 10.48550/arXiv.2602.08234. URL https://arxiv.org/abs/2602.08234.

Emily Xiao, Chin-Jou Li, Yilin Zhang, Graham Neubig, and Amanda Bertsch. Efficient Many-Shot In-Context Learning with Dynamic Block-Sparse Attention. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 31946–31958. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1542. URL https://aclanthology.org/2025.acl-long.1542/.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://arxiv.org/abs/2210.03629v3.

Aofan Yu, Chenyu Zhou, Tianyi Xu, Zihan Guo, Rong Shan, Zhihui Fu, Jun Wang, Weiwen Liu, Yong Yu, Weinan Zhang, and Jianghao Lin. LatentSkill: From in-context textual skills to in-weight latent skills for LLM agents. *arXiv preprint arXiv:2606.06087*, 2026. doi: 10.48550/arXiv.2606.06087. URL https://arxiv.org/abs/2606.06087.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19632–19642, 2024. doi: 10.1609/aaai.v38i17.29936. URL https://ojs.aaai.org/index.php/AAAI/article/view/29936.

Zhi Zheng, Ziqiao Meng, Hao Luan, Wei Liu, and Wee Sun Lee. One Token per multimodal evidence: Latent memory for resource-constrained QA. *arXiv preprint arXiv:2606.10572*, 2026. doi: 10.48550/arXiv.2606.10572. URL https://arxiv.org/abs/2606.10572.

A Method Details

A.1 Rule Construction

Rules are constructed before training the continuous compressor. Training examples are grouped by task structure and operational pattern, and representative examples are selected to cover different conditions and execution branches. Using reference outputs and available feedback, the induction model summarizes applicability conditions, key steps, and exceptions, omitting example identifiers, example-specific entities, and answers tied to individual examples. When task-level rules are used, the training experience forms a single group.

Candidate rules are validated on separate development data. Counterexamples guide revisions to overgeneralizations, conflicting conditions, and missing steps. Once selected, rule text and rule–example associations are fixed; gradient-based training updates only continuous parameters. Final test data are not used to generate or select rules.

A.2 Input Representation and Attention

Omitting the fixed shared prefix, the input for predicting a_t is arranged as

$$\mathbf{X}_t = [E(c_1); \mathbf{Z}; \ldots; E(c_K); \mathbf{Z}; E(q); E(a_{<t})]. \tag{8}$$

The embedding map E converts a sequence of n tokens into an $n \times d$ matrix, and the components are concatenated along the sequence dimension. Each row of $\mathbf{Z}$ is a trainable memory input vector reused across source units. Shifted next-token supervision and a causal mask allow losses at all output positions to be computed in one forward pass; the state predicting a_t cannot access a_t.

Attention visibility. Table 5 specifies attention visibility between parts of the input. Fixed task instructions form a shared prefix accessible to all subsequent valid positions under causal attention; LoRA is disabled at prefix positions. The listed connections also obey causality, and padding is excluded. In the full-context view, standard causal attention allows later source units, memory positions, and query positions to access earlier source content. The two views therefore compute separate intermediate states.

Table 5: Attention visibility and LoRA scope in the compressed view. Each row can also attend to the causally visible fixed task prefix.

Position	Visible content	LoRA
Source unit c_i	Causally visible source content within the same unit	Y
Memory block i	Corresponding c_i and causally visible positions in the same memory block	Y
Current input q	All memory blocks and the causal history of the current input	N
Output a	All memory blocks, the current input, and the causal output history	N

Trainable parameters. The original token embeddings, output head, and Transformer backbone remain frozen. LoRA produces low-rank updates only at source and memory positions; ϕ consists of LoRA parameters in the target modules and $\mathbf{Z}$. Gradients to shared parameters accumulate across units, while the encoded KV states depend on unit content. LoRA target modules, rank, scaling, and dropout are fixed within each experimental configuration.

A.3 Experience Memory and Cache Reuse

Let $\mathcal{J}_i$ be the ordered set of row indices for memory block i in the complete model input, with $|\mathcal{J}_i| = m$. The matrix $\mathbf{Z}$ contains the input vectors at these positions, whereas $\mathcal{J}_i$ selects their encoded cache rows. After merging the feature dimensions of the KV heads, the layer-ℓ cache matrices are $\mathbf{K}^{(\ell)}, \mathbf{V}^{(\ell)} \in \mathbb{R}^{N \times d_{\text{KV}}}$, where N is the valid input length including the fixed template and $d_{\text{KV}} = n_{\text{KV}} d_{\text{head}}$. Here, n_{KV} is the number of KV heads and d_{head} their width.

Extracting rows in increasing order of $\mathcal{J}_i$ gives

$$\mathbf{K}_i^{(\ell)} = \mathbf{K}^{(\ell)}[\mathcal{J}_i, :], \qquad \mathbf{V}_i^{(\ell)} = \mathbf{V}^{(\ell)}[\mathcal{J}_i, :]. \tag{9}$$

Both matrices have shape $m \times d_{\text{KV}}$. Unit memory $M_i = (\mathbf{K}_i^{(\ell)}, \mathbf{V}_i^{(\ell)})_{\ell=1}^{L}$ stores a KV pair per layer, and the combined memory is

$$M_q = \left([\mathbf{K}_1^{(\ell)}; \ldots; \mathbf{K}_K^{(\ell)}], [\mathbf{V}_1^{(\ell)}; \ldots; \mathbf{V}_K^{(\ell)}] \right)_{\ell=1}^{L}. \tag{10}$$

Each layer's key and value matrices have shape $Km \times d_{\text{KV}}$; the fixed task-prefix cache is retained separately. Under grouped-query attention, d_{KV} can differ from the embedding width d. When $K = 0$, experience memory is empty, and the model uses only the task template and current input.

Position-compatible reuse. A cache is tied to the model parameters, template, and position layout. After source-unit KV states are removed, retained keys preserve their original positional encodings, and current-input and generated tokens use position IDs from the original layout. Cache storage indices and model position IDs are maintained separately. Position-compatible caches can be reused directly; units affected by position changes under a new retrieval combination are re-encoded. Comparisons between cached and corresponding uncached paths use the same inputs and position configuration.

B Implementation Details

The default configuration for both tasks retrieves $K = 3$ examples and allocates $m = 5$ memory tokens to each, giving $Km = 15$ experience memory positions. This count excludes the shared prefix, current input, and generation history. Count ablations use the corresponding values of K and m. Tables 6 and 7 list the optimization settings.

B.1 ContractNLI Configuration

The ContractNLI compressor is trained on 640 supervised examples. Student SFT supervises the answer label and the model's native end-of-turn or EOS token over the full vocabulary, with fixed class weights to account for the label distribution. The main configuration also includes forward KL. Compression parameters are initialized independently from the corresponding pretrained model for each run. Table 6 specifies 160 updates with an effective batch size of 4, corresponding to $160 \times 4 = 640$ example presentations, or one epoch. The separate 800-step checkpoint used for efficiency measurement is described in Appendix B.3.

Table 6: ContractNLI training configuration.

Hyperparameter	Setting
Backbone	Qwen2.5-7B-Instruct / LLaMA-3-8B-Instruct
Supervised training examples	640
LoRA target modules	`q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj`
LoRA rank r	8
LoRA alpha	16
Learning rate	2×10^{-5}
Learning-rate schedule	Constant
Optimizer	AdamW
Weight decay	0
LoRA dropout	0.05
Effective batch size	4
Micro-batch size	1
Gradient accumulation steps	4
Checkpoint updates / epochs	160 updates / 1 epoch
Retrieved examples K	3
Memory tokens per example m	5
Total memory tokens	15
Training objective	Class-weighted Student SFT over the answer label and native end-of-turn/EOS token, with forward KL
Random seed	0

B.2 ALFWorld Configuration

The joint rule–example compressor for ALFWorld is trained on 1,600 action-level examples. Each target covers the fixed `ACTION:` prefix, action tokens, and the model's native end-of-turn or EOS token. Rule construction and pairing are described in Appendix A.1. The learning-rate schedule spans 500 updates, while the reported checkpoint is at update 200; these values specify the schedule horizon and checkpoint position, respectively.

Table 7: ALFWorld configuration for joint compression of group-level rules and examples.

Hyperparameter	Setting
Backbone	Qwen2.5-7B-Instruct / LLaMA-3-8B-Instruct
Supervised training examples	1,600 action examples
LoRA target modules	`q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj`
LoRA rank r	8
LoRA alpha	16
Peak learning rate	1×10^{-4}
Learning-rate schedule	Cosine with a 500-update schedule horizon
Warmup updates	15
Optimizer	AdamW
Weight decay	0.01
LoRA dropout	0.05
Effective batch size	8
Micro-batch size	1
Gradient accumulation steps	8
Checkpoint updates	200
Retrieved examples K	3
Memory tokens per example m	5
Total memory tokens	15
Training objective	Student SFT over `ACTION:`, action tokens, and the native end-of-turn/EOS token, with forward KL
Trainable components	Compressor-side partial LoRA and memory-input embeddings; QA-side LoRA disabled
Random seed	0

B.3 Dataset Statistics and Evaluation Protocol

Table 8 summarizes the subsets used in this study, rather than the full original datasets. ALFWorld uses 1,600 single-action training examples, 60 development episodes, and 134 unseen evaluation episodes. One episode can yield multiple action-level training examples. ContractNLI contains 640 training and 170 development document–hypothesis pairs; evaluation uses an existing diagnostic set of 714 pairs.

Table 8: Dataset statistics. ALFWorld training size counts action examples, whereas development and evaluation sizes count episodes. All ContractNLI counts refer to document–hypothesis pairs.

Dataset	Train	Dev	Test / Evaluation	Training unit	Evaluation metric
ALFWorld	1,600 action examples	60 episodes	134 Unseen episodes	Single-action supervision	Task success rate / All
ContractNLI	640 pairs	170 pairs	714 pairs, existing diagnostic set	Document–hypothesis pair	Accuracy / Macro-F1

ALFWorld training and evaluation are separated by scene, with at most 50 actions per episode. Per-task success rates are computed within each task type. All is the overall success rate across the 134 episodes, not the arithmetic mean of the six task rates. ContractNLI is grouped by document; predictions select the highest-scoring valid label. Accuracy is the fraction of correct predictions, and Macro-F1 averages F1 over the three classes. Retrieved examples and rule sources are restricted to training material for both tasks. Examples without valid rule mappings are excluded from paired rule–example comparisons.

Efficiency measurement. Efficiency is measured using Qwen2.5-7B-Instruct and an example-only compression checkpoint at 800 updates, on an A100-SXM4-80GB with BF16, batch size 1, and SDPA MATH. Each of 32 diagnostic inputs is timed for three rounds, yielding 96 measurements per method. Caches are precomputed and resident on the GPU. Online timing includes input transfer to the GPU, the forward pass, and label transfer to the CPU, with CUDA synchronization before and after the timed interval. It excludes retrieval, tokenization, model loading, offline encoding, and cache transfer. Labels are selected from A/B/C logits at the first answer position.

Mean latency is 480.22, 161.04, and 99.34 ms for Raw context, Raw KV cache, and Experience memory KV, respectively; speedups are ratios of mean latencies. Experience KV storage excludes the 69-token shared prefix. The current input and trailing template contain 467.625 tokens on average. The GPU was shared during measurement. The BF16 cache paths still exhibit numerical and some prediction differences and have not passed the full-set equivalence check; Table 4 reports the runtime of the current implementation.

C Memory Budget Allocation

Experience memory has length Km, subject to $Km \leq B$. With a fully allocated budget of $B = 15$ positions, $(K, m) = (1, 15), (3, 5), (5, 3)$ correspond to one example with 15 positions, three examples with five positions each, or five examples with three positions each. These allocations have the same total memory length but differ in example coverage and per-example capacity. This relation describes budget allocation only. In the main-text ablations, varying m at fixed K or varying K at fixed m also changes total memory length.

AI Use Statement

We used generative AI tools to assist with literature discovery, summarizing related work, drafting and revising portions of the manuscript, and improving its clarity and organization. The authors reviewed all AI-assisted content and individually verified each cited reference and the claims attributed to it against the original publication. All final decisions regarding the selection and inclusion of citations were made by the authors. Generative AI was not used to design the proposed method or experiments, implement the method, generate experimental data, or analyze results. The authors take full responsibility for the accuracy, originality, and integrity of this work.