SR-KI: Scalable and Real-Time Knowledge Integration into LLMs via Supervised Attention

Bohan Yu[1,2,3*], **Wei Huang**[2†], **Kang Liu**[3,4†]

[1]School of Advanced Interdisciplinary Sciences, University of Chinese Academy of Sciences, Beijing, China
[2]MEG, Baidu Inc., Beijing, China
[3]The Key Laboratory of Cognition and Decision Intelligence for Complex Systems, Institute of Automation, CAS, Beijing, China
[4]School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing, China
yubohan2025@ia.ac.cn huangwei16@baidu.com kliu@nlpr.ia.ac.cn

Abstract

This paper proposes SR-KI, a novel approach for integrating real-time and large-scale structured knowledge bases (KBs) into large language models (LLMs). SR-KI begins by encoding KBs into key-value pairs using a pretrained encoder, and injects them into LLMs' KV cache. Building on this representation, we employ a two-stage training paradigm: first locating a dedicated retrieval layer within the LLM, and then applying an attention-based loss at this layer to explicitly supervise attention toward relevant KB entries. Unlike traditional retrieval-augmented generation methods that rely heavily on the performance of external retrievers and multi-stage pipelines, SR-KI supports end-to-end inference by performing retrieval entirely within the model's latent space. This design enables efficient compression of injected knowledge and facilitates dynamic knowledge updates. Comprehensive experiments demonstrate that SR-KI enables the integration of up to 40K KBs into a 7B LLM on a single A100 40GB GPU, and achieves strong retrieval performance, maintaining over 98% Recall@10 on the best-performing task and exceeding 88% on average across all tasks. Task performance on question answering and KB ID generation also demonstrates that SR-KI maintains strong performance while achieving up to 99.75% compression of the injected KBs. Our code will be available at SR-KI.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in understanding, analyzing, and generating texts by leveraging their extensive knowledge and powerful reasoning abilities (Touvron et al. 2023; OpenAI 2024; Zhao et al. 2025a). However, in scenarios where users require external knowledge that is not present in or diverges from the information stored in the model's parameters, efficient and real-time knowledge injection becomes essential. A straightforward approach is to fine-tune the model parameters (Wei et al. 2022; Dubois et al. 2024; Chung et al. 2024). However, this strategy risks catastrophic forgetting and overfitting by disrupting the original parameter distribution, potentially degrading the model's pre-existing knowledge. Moreover, full fine-tuning is resource-intensive and inflexible for frequent knowledge updates. Although parameter-efficient tuning methods (Li and Liang 2021; Han et al. 2024) improve efficiency, they still lack support for continual knowledge updates.

Retrieval-Augmented Generation (Lewis et al. 2021) (RAG) has emerged as a popular alternative, enabling LLMs to access external knowledge by incorporating retrieved content directly into the input prompt. However, RAG follows a pipeline architecture that relies heavily on the performance of external retrievers and is constrained by the limited context window of LLMs (Yu et al. 2024; Zhao et al. 2025b). To address the limitations of retrieval-based methods, recent long-context LLMs (OpenAI 2024; Gemini 2025) extend the context window, enabling direct reasoning over the entire input. However, this comes at the cost of significant computational and memory overhead (Dao 2023), limiting their scalability. KBLaM (Wang et al. 2025) offers an alternative by injecting external knowledge into the KV cache (Pope et al. 2022) through projection adapters, allowing the model to attend to key-value representations rather than store specific facts. Nevertheless, as the scale of injected knowledge increases, KBLaM fails to focus on the most relevant information, resulting in severe performance degradation. Moreover, all of these approaches remain challenging to attribute the model's output to the injected knowledge, raising concerns about controllability and interpretability (Meng et al. 2023a; Ji et al. 2023; Abolghasemi et al. 2025).

In this paper, we propose SR-KI (**S**calable and **R**eal-Time **K**nowledge **I**ntegration into LLMs via Supervised Attention), a novel method for injecting external knowledge into LLMs dynamically via a supervised attention mechanism. Similar to KBLaM, SR-KI transforms unstructured factual knowledge into structured knowledge bases (KBs) of (*subject, relation, object*) triples and injects them into the latent space of LLMs. Specifically, SR-KI converts each knowledge triple into a key-value pair, using the *subject* and *relation* as the key and the *object* as the value. The key and value are independently embedded into a vector pair using a pretrained sentence encoder, and then projected through learnable single-layer adapters to match the embedding dimension of the LLM's KV cache, enabling their injection into the

[*] Work done during an internship at Baidu.
[†] Corresponding authors.

attention mechanism as external key-value pairs. This design allows the model to learn generalizable key-value mappings rather than memorize specific facts, while keeping attention computation linearly scalable with the number of triples.

Prior studies have discovered specific layers in LLMs that are particularly sensitive to knowledge injection (Meng et al. 2023a; Wang et al. 2024). In our study, we further find that this attributes to the model's architecture rather than task-specific factors. Inspired by this observation, a two-stage training paradigm is adopted. The first stage involves identifying the retrieval layer—a critical point where knowledge injection has the greatest influence. In the second stage, an attention-based loss is introduced at this layer to enable multi-objective optimization, explicitly guiding the model to focus on the most relevant KB entries. By directly supervising the attention behavior at this layer, our method equips it with the ability to precisely attend to pertinent knowledge even under large-scale injection. This mechanism allows SR-KI to effectively prune irrelevant knowledge to reduce inference latency and memory usage, while simultaneously enabling the reuse of critical KBs in subsequent layers to promote more efficient and coherent knowledge utilization. All these processes are performed in an end-to-end manner, with knowledge integration and response generation jointly executed in a single forward pass without relying on external modules or multi-stage pipelines.

Beyond efficient knowledge access, SR-KI further enables knowledge reference and traceability by guiding the model to generate both factual content and its corresponding source. This design meets the growing demand for transparent and verifiable outputs in knowledge-intensive tasks (Ji et al. 2023). To support this, the *Reference ID KB* is introduced, where each knowledge triple is assigned a randomly generated uppercase ID. These reference triples are encoded and injected into the LLM following the same key-value format as factual knowledge, allowing the model to jointly predict the knowledge and its associated ID, thereby achieving factual grounding and source attribution.

SR-KI supports real-time and large-scale knowledge injection by aggressively compressing 40K KBs on a single A100 40G GPU down to the top-100 during inference, achieving up to 99.75% compression while maintaining strong reasoning and retrieval performance. Even as the KB size scales to 40K, SR-KI sustains over 95% Recall@100 and 88% Recall@10, demonstrating robust retrieval capabilities. On question answering tasks, it achieves consistently high performance under large-scale settings, with a performance margin of up to 70 points compared to baseline levels observed in same settings.

In conclusion, our key contributions are as follows:

- We propose SR-KI, a two-stage training framework that employs supervised attention for efficient and dynamic knowledge injection into LLMs. By leveraging learnable KV projection adapters and attention-guided supervision, SR-KI enables scalable, precise, and minimally invasive integration of structured external knowledge.
- SR-KI incorporates a dedicated retrieval layer that enables accurate and efficient selection from large-scale KBs, achieving up to 99.75% compression while preserving strong task performance and retrieval effectiveness.
- SR-KI jointly generates the knowledge and its source KB ID, enabling transparent and verifiable outputs.

2 Related Works

Retrieval-Augmented Generation (RAG) Retrieval-Augmented Generation (Lewis et al. 2021) enables LLMs to access external knowledge by retrieving relevant passages and appending them to the input context. While effective, RAG is limited by the context window size and the quadratic attention cost of transformers (Yu et al. 2024; Leng et al. 2024; Zhao et al. 2025b), which constrains scalability and introduces latency. Additionally, the separation of retrieval and generation can lead to retrieval errors and hallucinations (Ru et al. 2024; Xu et al. 2024; Sun et al. 2025), limiting the factual consistency of model outputs. Essentially, while RAG operates as a form of in-context learning relying on retrieval modules, our proposed SR-KI framework integrates external knowledge through supervised attention and internal semantic understanding.

Knowledge Editing in LLMs Existing approaches to knowledge editing in language models primarily follow two paradigms: directly modifying the internal parameters (Mitchell et al. 2022; Meng et al. 2023a; Rozner et al. 2024) or injecting information through adapters (De Cao, Aziz, and Titov 2021; Wang et al. 2024; Zhu et al. 2025). While both methods enable localized updates, they suffer from two key limitations: (1) the edited knowledge is often not dynamically updatable during inference, and (2) the number of editable facts remains limited. To address these issues, KBLaM (Wang et al. 2025) introduces a novel mechanism that maps structured knowledge into key-value pairs and injects them into the model, enabling it to learn generalizable mappings between keys and values. However, KBLaM still struggles to access truly relevant knowledge under large-scale injection, suffers from substantial computational and memory overhead, and experiences significant performance degradation as the injected knowledge scale increases. To overcome these limitations, our proposed SR-KI incorporates a supervised attention mechanism, enabling accurate knowledge access while significantly reducing computational overhead and maintaining strong performance.

3 Preliminaries

Knowledge Base in Triple Representation In real-world scenarios, the knowledge base (KB) is often represented as a triple in the form of (s, r, o), where s, r, and o denote the *subject*, *relation*, and *object*, respectively. For a set of M knowledge triples, we define the KB as $\{(s_m, r_m, o_m)\}_{m=1}^{M}$.

Attention Layer Computation A decoder-based LLM consists of multiple self-attention layers. Each attention layer contains three projection matrices: $W_Q^l \in \mathbb{R}^{D\times D}$, $W_K^l \in \mathbb{R}^{D\times D}$, and $W_V^l \in \mathbb{R}^{D\times D}$, where $l \in \{1, \ldots, L\}$ denotes the layer index and D is the embedding dimension. These projection matrices transform the layer hidden states $X^l = [x_1^l, x_2^l, \ldots, x_N^l]$ of N tokens into their corresponding

query, key, and value matrices: $Q^l = [q_1^l, \ldots, q_N^l], K^l = [k_1^l, \ldots, k_N^l], V^l = [v_1^l, \ldots, v_N^l], \quad Q^l, K^l, V^l \in \mathbb{R}^{N \times D}$. The attention output is then computed as:

$$\text{Att}(Q^l, K^l, V^l) = \text{Softmax}\left(\frac{Q^l (K^l)^\top}{\sqrt{D}}\right) V^l \quad (1)$$

KB Injection and Rectangular Attention Computation Inspired by KBLaM (Wang et al. 2025), we treat attention as a normalized key-value pairing weight, where each knowledge triple (s_m, r_m, o_m) is represented as a key-value pair with (s_m, r_m) as the key and o_m as the value. Using a pretrained sentence encoder, we embed these components and project them from dimension P to the model's embedding space D via learned single-linear adapters $\tilde{W}_K^l, \tilde{W}_V^l \in \mathbb{R}^{P \times D}$, as shown in Equation (2). These adapters are trained over large-scale KBs to generalize the mapping pattern rather than memorize specific facts, enabling generation of appropriate o_m values for unseen (s_m, r_m) pairs.

$$\begin{aligned} \{(s_m, r_m, o_m)\}_{m=1}^M &\xrightarrow{\text{Encode}} \{(k_m, v_m)\}_{m=1}^M \\ \{(\tilde{k}_m, \tilde{v}_m)\}_{m=1}^M &= \{(k_m \tilde{W}_K, v_m \tilde{W}_V)\}_{m=1}^M \end{aligned} \quad (2)$$

These transformed KB representations can be naturally injected into the model's KV cache. For layer l, the augmented KV cache includes M KBs and N original tokens, resulting in $\tilde{K}^l, \tilde{V}^l \in \mathbb{R}^{(M+N) \times D}$. The structure is defined as:

$$\begin{aligned} \tilde{K}^l &= [\tilde{k}_1^l, \ldots, \tilde{k}_M^l, k_1^l, \ldots, k_N^l] \\ \tilde{V}^l &= [\tilde{v}_1^l, \ldots, \tilde{v}_M^l, v_1^l, \ldots, v_N^l] \end{aligned} \quad (3)$$

We apply a separate projection adapter $\tilde{W}_Q^l$ to map the hidden states X^l into a new query matrix $\tilde{Q}^l = X^l \tilde{W}_Q^l \in \mathbb{R}^{N \times D}$. Attention is computed independently over KB keys $\tilde{K}^l[:M]$ and original keys $\tilde{K}^l[M:]$, producing $A_{\text{KB}}^l \in \mathbb{R}^{N \times M}$ and $A^l \in \mathbb{R}^{N \times N}$. These logits are concatenated and normalized via a unified softmax to form a rectangular attention distribution. As only the KV sequence length changes, the output shape remains unchanged, enabling seamless integration of KB knowledge into hidden states for downstream propagation. The full attention is computed as:

$$\text{RectangleAtt}(Q^l, \tilde{Q}^l, \tilde{K}^l, \tilde{V}^l) = \text{Softmax}\left(\left[A_{\text{KB}}^l \middle| A^l\right]\right) \tilde{V}^l \quad (4)$$

4 Knowledge Injection via Supervised Attention

This section explores how to enhance knowledge injection in LLMs through a supervised attention mechanism, with a focus on the KB attention weights A_{KB}^l during both training and inference processes.

4.1 Training Process

Our training process consists of two sequential stages: (1) first, we freeze the pretrained model parameters and train projection adapters $\tilde{W}_Q^l, \tilde{W}_K^l, \tilde{W}_V^l$ to identify the retrieval layer; then (2) we introduce an attention-based loss on the identified layer for multi-objective training to achieve both accuracy and retrieval efficiency, as shown in Figure 1.

Retrieval Layer Identification Previous studies (Wang et al. 2025, 2024; Meng et al. 2023b,a) have revealed that integrating knowledge into specific layers of the model can produce significant performance gains. Building on prior findings, we identify the critical layer, where knowledge injection is most effective, by selectively injecting correct KBs into each layer individually while introducing randomly sampled negative KBs into all other layers. The layer that achieves the highest retrieval accuracy is designated as the critical layer, as it plays a pivotal role during inference. When a large volume of KBs is injected, the attention distribution becomes dispersed across numerous candidates, weakening the model's ability to focus on the correct entries. As a result, precise retrieval at the critical layer is especially crucial—if the model fails to attend to the correct KBs at this layer, injecting accurate knowledge into other layers becomes significantly less effective. Accordingly, we define this layer as the retrieval layer $\tilde{l}$, which is responsible for identifying and integrating essential information during inference.

Supervised Attention Training Objective The goal of supervised attention training is to guide the identified retrieval layer to focus on the correct KBs, thereby enabling efficient access to large-scale knowledge. For the layer l, we denote the injected KBs as $\text{KB}^l = [kb_1^l, \ldots, kb_M^l]$, among which the correct KBs are defined as $\tilde{\text{KB}}^l = [\tilde{kb}_1^l, \ldots, \tilde{kb}_J^l]$, where J is the number of correct KBs. To quantify the importance of each KB, we compute its aggregated attention scores by averaging the attention matrix $A_{\text{KB}}^l \in \mathbb{R}^{N \times M}$ over the sequence dimension: $\overline{A_{\text{KB}}^l} = \frac{1}{N} \sum_{n=1}^N A_{\text{KB}}^l[n,:] \in \mathbb{R}^M$.

To further enhance the retrieval layer's ability to distinguish hard examples during training, we retain the top k KBs denoted as KB_{top}^l with the highest aggregated attention weights $\overline{A_{\text{KB}}^l}$. The negative KBs among top results serve as hard negative samples, denoted as $\text{KB}_{\text{neg}}^l = \text{KB}_{\text{top}}^l \setminus \tilde{\text{KB}}^l$. If any correct KB is missing from the top k results, we will supplement it to ensure inclusion, and remove an equal number of other KBs to maintain dimensional consistency. For each correct KB $\tilde{kb}_j^l$, we construct a candidate set $\tilde{kb}_j^l \cup \text{KB}_{\text{neg}}^l$ along with corresponding index set $\mathcal{N}_j$. We then compute the cross-entropy loss using the attention scores $\overline{A_{\text{KB}}^l}$ associated with the indices in $\mathcal{N}_j$. This objective encourages the model to assign higher attention to the correct KBs by contrasting it against hard negatives within the candidate set. Due to the relatively small differences in attention logits, we introduce a temperature coefficient $\mathcal{T}$ to amplify the contrast between the correct KBs and negative samples. The attention-based loss for retrieval layer $\tilde{l}$ is defined as:

$$\mathcal{L}_{\text{a}} = -\frac{1}{J} \sum_{j=1}^J \log \left(\frac{\exp\left(\overline{A_{\text{KB}}^{\tilde{l}}}[i_j]/\mathcal{T}\right)}{\sum_{i \in \mathcal{N}_j} \exp\left(\overline{A_{\text{KB}}^{\tilde{l}}}[i]/\mathcal{T}\right)} \right) \quad (5)$$

where i_j denotes the index of the j-th correct KB.

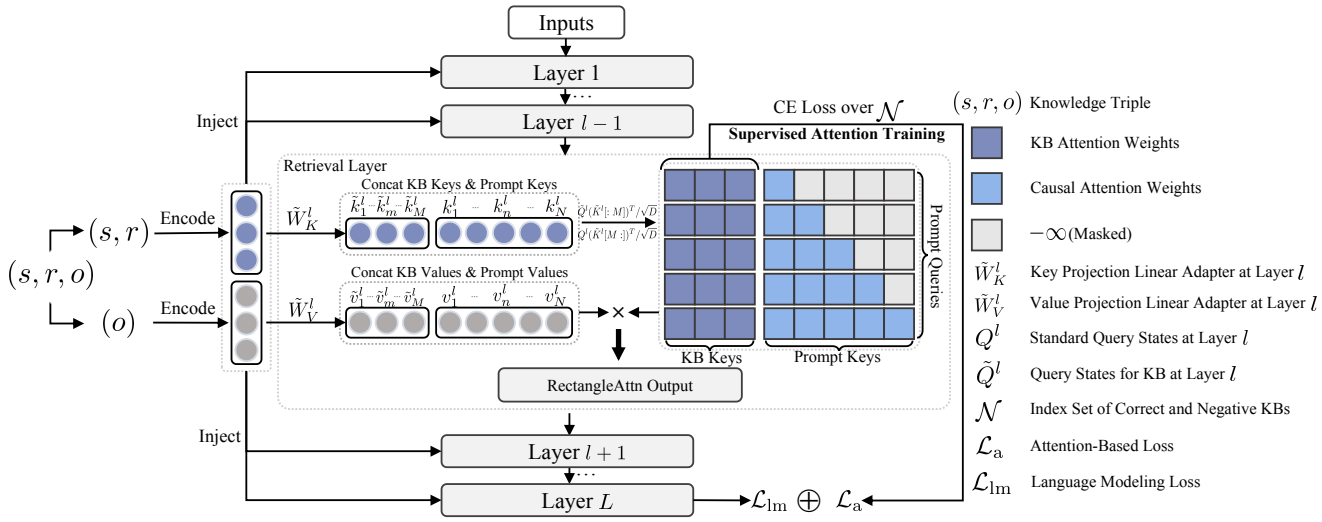

Figure 1: Illustration of the SR-KI training process with supervised attention. We incorporate the attention-based loss, computed using A^l_{KB} from the retrieval layer, into the overall language modeling loss.

The overall multi-objective training loss is defined as:

$$\mathcal{L} = \mathcal{L}_{\text{lm}} + \mathcal{L}_{\text{a}} \tag{6}$$

where $\mathcal{L}_{\text{lm}}$ denotes the autoregressive language modeling loss.

4.2 Inference process

Supervised attention training is applied to the retrieval layer's projection adapters to enable accurate KB selection. Based on this, we adopt a two-stage progressive refinement strategy to further boost reasoning, as shown in Figure 2.

KB Compression via Top-k Attention Weights Injecting a large number of KBs results in computational complexity of $O((M+N)ND)$, which poses challenges under memory constraints when M is large. To address this, we leverage the aggregated KB attention weights $\overline{A^l_{\text{KB}}}$ to retain only the top-k most relevant KB entries in each layer, selecting their indices as $\mathcal{I}^l_{\text{c}} = \text{TopK}\left(\overline{A^l_{\text{KB}}}, k\right)$.

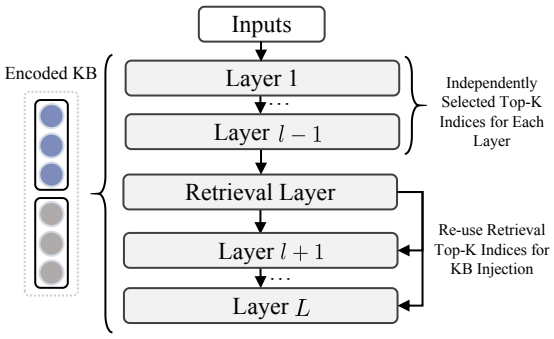

Figure 2: Illustration of the inference process: SR-KI selects top-k KBs individually before the retrieval layer and reuses their indices for injection in later layers.

KB Reuse Across Layers After the retrieval layer selects the relevant KB indices $\mathcal{I}^l_{\text{c}}$, they are reused in all subsequent layers, eliminating redundant compression and reducing inference-time overhead. This design improves efficiency and lowers computational costs, especially under large-scale knowledge injection. Reusing high-recall indices across layers further promotes consistent knowledge utilization and enhances overall performance.

5 Dataset Construction

We construct structured KBs from Wikidata (Vrandečić and Krötzsch 2014) for its broad and comprehensive coverage. To efficiently process large-scale knowledge graphs, we adopt the Knowledge Graph Toolkit (KGTK) (Ilievski et al. 2021), a scalable and flexible framework for knowledge graph construction and manipulation.

5.1 KB Construction

Our constructed KB consists of two primary types. Detailed examples can be found in Appendix A.1:

- **Factual Knowledge KB** Each KB is represented as a triple (s_m, r_m, o_m), where the key is composed of (s_m, r_m), expressed in natural language as "the r_m of s_m", and the value is the entity o_m. All triples are encoded into embedding vectors using a pretrained sentence encoder.
- **Reference ID KB** Inspired by prior generative retrieval work (Qian et al. 2023; Nakano et al. 2022), we construct reference ID KBs where each triple is assigned a key like "The ID of the knowledge 'the r_m of s_m is o_m'" and a randomly generated uppercase letter as its value. During training, the same triple may be assigned different IDs across training steps and depending on its role (correct or negative), encouraging the model to learn robust KV projection patterns and verify whether referenced knowledge is from the injected KBs.

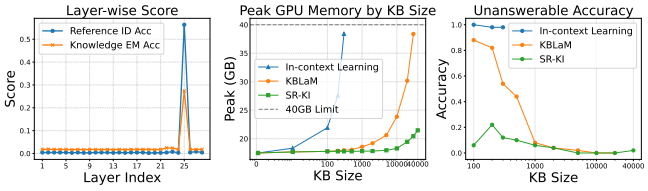

Figure 3: **Left**: reference ID accuracy and exact match (EM) accuracy for *object* without supervised attention training, using correct KB injected at a single layer. **Middle**: peak GPU memory usage of In-context Learning, KBLaM, and SR-KI under varying KB sizes (40GB limit shown). **Right**: results of unanswerable QA accuracy.

5.2 QA Dataset Construction

Our training set of 150K question-answer pairs covers over 140K knowledge triples. To rigorously test projection learning, (s_m, r_m) pairs in training are excluded from the test set.

- **Single-entity QA** Questions that query the relation r_m for a given entity s_m. The correct answer is o_m with its reference ID.
- **Multi-entity QA** This category includes two subtypes: (i) questions requiring two relations for the *same entity*, and (ii) questions involving one relation for *each of two distinct entities*. Each answer is linked to its corresponding reference ID. The dataset contains an equal number of examples for both subtypes.
- **Unanswerable QA** Questions for which the relevant knowledge is not included in the injected KB. The model is expected to respond with a refusal.

Our QA task is more complex as the model must generate both the knowledge answer and its reference ID by jointly retrieving from the factual and reference ID KBs. See Appendix A.2 and A.3 for dataset and template details.

6 Experiments

6.1 Experiment Setting

Model Selection We use Qwen2.5-7B-Instruct (Qwen 2025) as the base LLM, bge-large-zh-v1.5 (Xiao et al. 2023) for KB embedding, and bert-base-chinese (Devlin et al. 2019) for BERTScore evaluation.

Training Setting We freeze the original model and initialize $\tilde{W}_K$, $\tilde{W}_V$ randomly while copying W_Q to $\tilde{W}_Q$. Training is conducted on a single A100 (40GB) using DeepSpeed ZeRO Stage-2 (Rajbhandari et al. 2020) with CPU offloading and bf16 precision. We inject up to 100 KBs before supervised attention training, using a per-device batch size of 5, gradient accumulation of 20 (effective batch size 100), and a cosine scheduler with learning rate 1×10^{-4}, warm-up ratio 1×10^{-2}, and weight decay 1×10^{-4}. Each batch includes 40% Single-entity, 40% Multi-entity, and 20% Unanswerable QA, with one-to-one pairing of factual knowledge and reference ID KBs.

Retrieval Layer Identification and Subsequent Training We identify the retrieval layer by individually injecting correct KBs into each layer over 100 samples; the 25th layer achieves the best performance (Figure 3, left) on both ID generation and knowledge reasoning, demonstrating that this capability is attributed to the model architecture rather than task-specific factors. We fix this layer for supervised attention training, injecting 1000 KBs and compressing them to top-100 using the method in Section 4.2. The training is conducted with temperature $\mathcal{T} = 0.05$, and as shown in Figure 4, it leads to sharper attention on relevant KBs.

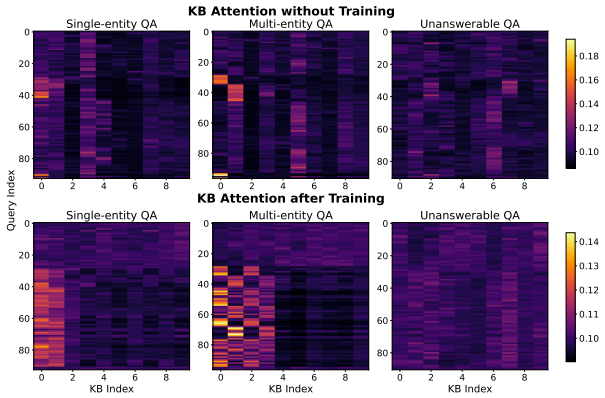

Figure 4: Task-specific KB attention weights at the retrieval layer: for Single-entity QA, correct KBs are placed at indices 0–1; for Multi-entity QA, at indices 0–3 for clarity. In Unanswerable QA, attention is spread across all entries.

Baseline We consider the following methods as baselines:

- **In-context Learning** All KBs are flattened and prepended to the prompt. Due to quadratic memory growth, we limit the KB size to 300 triples.
- **KBLaM** (Wang et al. 2025) A state-of-the-art KV projection method without supervised attention training. It supports up to 30K KBs before hitting memory limits.

Evaluation Setting All evaluations are conducted with 5 random seeds, each on 100 samples, and results are averaged over 500 questions. We inject all KBs when the size is equal to 100, and apply top-k=100 selection when the size exceeds 100. In contrast, KBLaM injects all KBs without any compression.

6.2 Experiment Results

We report results across KB sizes (100–40K), evaluating reference ID via accuracy and factual knowledge via BERTScore (F1) on the generated *object*. Additional experiments and results across more KB sizes and comparison settings are presented in Appendix B.

Experiments on Factual Knowledge and Reference ID Reasoning As shown in Table 1, we conduct comprehensive evaluations under varying KB sizes to assess the robustness and scalability of SR-KI. When the KB size is small

Method	ID-Acc				K-BERT			
	Single	Multi-S	Multi-D	Avg.	Single	Multi-S	Multi-D	Avg.
KB Size = 100								
ICL	0.8640	0.4300	0.7250	0.6730	**0.9796**	**0.9926**	**0.9832**	**0.9851**
KBLaM	0.9840	**0.9800**	0.9550	0.9730	0.8909	0.8817	0.8450	0.8725
SR-KI	**0.9960**	0.9750	**0.9800**	**0.9837**	0.8760	0.8779	0.8103	0.8547
KB Size = 1000								
KBLaM	0.8400	0.7550	0.7500	0.7817	0.7003	0.7105	0.6449	0.6852
SR-KI	**0.9800**	**0.9700**	**0.8900**	**0.9467**	**0.7944**	**0.8526**	**0.6982**	**0.7817**
KB Size = 10000								
KBLaM	0.0160	0.0100	0.0000	0.0087	-1.2723	-1.2678	-1.2723	-1.2708
SR-KI	**0.8000**	**0.7950**	**0.7450**	**0.7800**	**0.6764**	**0.7066**	**0.6201**	**0.6677**
KB Size = 40000								
KBLaM	*OOM*				*OOM*			
SR-KI	**0.6720**	**0.7650**	**0.6450**	**0.6940**	**0.6108**	**0.6508**	**0.5500**	**0.6039**

Table 1: Reference ID Accuracy (ID-Acc) and Knowledge BERTScore (F1) (K-BERT) across different QA sub-types and KB sizes. **Single**: Single-entity; **Multi-S**: Multi-entity (single entity with two relations); **Multi-D**: Multi-entity (different entities, each with one relation), **OOM**: out-of-memory.

Method	R@100				R@10				R@Top			
	Single	Multi-S	Multi-D	Avg.	Single	Multi-S	Multi-D	Avg.	Single	Multi-S	Multi-D	Avg.
KB Size = 100												
KBLaM	-	-	-	-	0.4800	0.4850	0.4500	0.4717	0.1740	0.2375	0.2500	0.2205
SR-KI	-	-	-	-	**1.0000**	**1.0000**	**0.9900**	**0.9967**	**1.0000**	**0.9975**	**0.9550**	**0.9842**
KB Size = 1000												
KBLaM	0.4500	0.4450	0.4175	0.4375	0.1080	0.0775	0.1000	0.0952	0.0420	0.0400	0.0575	0.0465
SR-KI	**1.0000**	**1.0000**	**0.9925**	**0.9975**	**1.0000**	**1.0000**	**0.9425**	**0.9808**	**0.9920**	**0.9875**	**0.8450**	**0.9415**
KB Size = 10000												
KBLaM	0.0960	0.0375	0.0875	0.0737	0.0180	0.0025	0.0150	0.0118	0.0060	0.0000	0.0100	0.0053
SR-KI	**1.0000**	**1.0000**	**0.9425**	**0.9808**	**0.9980**	**0.9950**	**0.8025**	**0.9318**	**0.9680**	**0.9350**	**0.7075**	**0.8702**
KB Size = 40000												
KBLaM	*OOM*				*OOM*				*OOM*			
SR-KI	**0.9980**	**1.0000**	**0.8800**	**0.9593**	**0.9860**	**0.9750**	**0.7050**	**0.8887**	**0.9180**	**0.9000**	**0.5900**	**0.8027**

Table 2: Recall at Top-K retrieved KBs (R@100, R@10, R@Top) across different QA sub-types and KB sizes. **Single**: Single-entity; **Multi-S**: Multi-entity (single entity with two relations); **Multi-D**: Multi-entity (different entities, each with one relation). **OOM**: out-of-memory. Missing entries (-) indicate results not reported.

(i.e., 100), both KBLaM and SR-KI achieve strong performance on reference ID accuracy. However, at KB size of 100, KBLaM attains a slightly higher average BERTScore (F1) than SR-KI (0.8725 vs. 0.8547), indicating that the fine-grained knowledge alignment of KBLaM is still comparable but not superior under modest KB sizes. Compared with In-context Learning (ICL) at 100 KBs, although it exhibits a very high BERTScore, we observe that it performs poorly on alphabetic ID prediction and cannot support large-scale KB injection due to memory constraints; hence, we only report results at 100 KBs. As the KB size increases, KBLaM experiences significant degradation, with BERTScore dropping below zero in extreme cases (e.g., KB size=10K), reflecting the method's difficulty in handling large-scale noisy KBs. In contrast, SR-KI maintains strong robustness, achieving 0.78 accuracy and 0.67 F1 at 10K KBs, and still retaining 0.69 accuracy and 0.60 F1 at 40K KBs. These results demonstrate that SR-KI scales effectively to large knowledge corpora while preserving task performance; the high ID accuracy further verifies its ability to perform accurate reasoning and source attribution over the injected KBs. For the unanswerable QA, as shown in Figure 3 (right), supervised attention training introduces a decline in refusal accuracy. However, this trade-off is marginal relative to the consistent gains SR-KI delivers in knowledge reasoning and ID accuracy across all KB sizes. Notably, while both SR-KI and KBLaM exhibit comparable refusal capabilities at 1K KBs, they struggle to reject unanswerable queries as the KB size increases. These results indicate that the slight compromise in refusal ability is outweighed by SR-KI's overall advantage in handling large-scale knowledge while preserving task effectiveness.

To further demonstrate scalability, Figure 3 (middle) shows peak GPU memory usage across KB sizes. SR-KI maintains low and stable memory usage, remaining nearly

Method	ID-Gen	K-Gen	R@100	R@10	R@Top
		KB Size = 100			
ICL	0.5727	0.5801	-	-	-
KBLaM	0.8907	0.6346	-	0.4300	0.2182
SR-KI	**0.9357**	**0.7124**	-	**0.9790**	**0.9407**
		KB Size = 1000			
KBLaM	0.6810	0.4561	0.4063	0.1003	0.0458
SR-KI	**0.8089**	**0.5876**	**0.9817**	**0.9306**	**0.8775**
		KB Size = 10000			
KBLaM	0.0100	-1.2709	0.0825	0.0077	0.0040
SR-KI	**0.6677**	**0.5102**	**0.9237**	**0.8507**	**0.7683**
		KB Size = 40000			
KBLaM	*OOM*	*OOM*	*OOM*	*OOM*	*OOM*
SR-KI	**0.5227**	**0.4227**	**0.8893**	**0.7798**	**0.6917**

Table 3: Generalization experimental results by QA type. **ID-Gen**: reference ID generalization accuracy; **K-Gen**: knowledge generalization BERTScore (F1) on *object*; **R@K**: recall at top-K retrieved KBs. **OOM**: out of memory. Scores are averaged over three QA subtypes. Full KBs used at size=100; Top-k=100 selection for larger KBs.

flat up to 5K KBs and growing modestly at 40K and staying well under the 40GB A100 limit. In contrast, KBLaM exceeds 30GB at 30K and overflows at 40K, while In-context Learning also quickly breaches memory limits. By incorporating a lightweight key-value projection and retrieval mechanism, SR-KI enables efficient large-scale inference while maintaining strong task performance.

Experiments on KB Retrieval We evaluate the retrieval performance of SR-KI and KBLaM at the retrieval layer under varying KB sizes to assess their ability to identify the most relevant KBs, as shown in Table 2. When the KB size is small (i.e., 100), where all relevant KBs are injected and thus retrieval is relatively easy, SR-KI already demonstrates strong retrieval ability, with Recall@10 reaching 0.99 and Recall@Top exceeding 0.98, indicating its precision under this base condition. Here, Recall@Top refers to whether the correct KBs are retrieved within the number of KBs required by the task type — for example, top-2 for Single-entity QA and top-4 for Multi-entity QA. As the KB size scales from 1K to 40K, KBLaM's retrieval performance rapidly deteriorates, with Recall@100 and Recall@Top already falling below 0.45 and 0.05 at 1K. At 10K KBs, the degradation becomes drastic — Recall@100 drops below 0.08, while both Recall@10 and Recall@Top approach almost zero. In contrast, SR-KI consistently maintains strong retrieval ability, with Recall@100 and Recall@Top remaining above 0.95 and 0.80 respectively at 40K KBs. This indicates that SR-KI is able to identify all required KBs within the top few ranks, even in large-scale noisy KBs. These results collectively demonstrate the scalability and effectiveness of our retrieval framework under extreme KB conditions.

Experiments on Generalization We construct the generalization dataset using Wikidata alias information, where the *subject* and *relation* in each question are randomly replaced to assess the generalization capability of SR-KI. Experimental results in Table 3 show that SR-KI consistently outperforms the baselines in both task performance and retrieval effectiveness. Specifically, at KB size of 1000, SR-KI yields 0.80 accuracy and 0.58 F1, surpassing KBLaM by large margins (0.68 and 0.45, respectively). In terms of retrieval, SR-KI maintains robust recall across scales, with Recall@100 and Recall@Top exceeding 0.98 and 0.87 at 1000 KBs, while KBLaM falls below 0.41 and 0.05. Notably, in-context learning already suffers from significant performance degradation at 100 KBs, being more vulnerable to alias-perturbed queries, which further highlights the robustness of our SR-KI framework. Remarkably, SR-KI remains effective even under extreme KB injection conditions. At 40K KBs, where KBLaM fails due to out-of-memory errors, SR-KI achieves 0.52 accuracy, 0.88 Recall@100, and a near-0.70 Recall@Top, demonstrating strong scalability and robustness. These results confirm that SR-KI generalizes well to alias-perturbed queries while preserving high retrieval and QA performance under large-scale knowledge injection.

Method	ID-Acc	K-BERT	ID-Gen	K-Gen
		KB Size = 1000		
SR-KI$_{\text{w/o re-use}}$	0.8167	0.5677	0.6872	0.4051
SR-KI	**0.9467**	**0.7817**	**0.8089**	**0.5876**
		KB Size = 10000		
SR-KI$_{\text{w/o re-use}}$	0.5000	0.4219	0.3957	0.2644
SR-KI	**0.7800**	**0.6677**	**0.6677**	**0.5102**
		KB Size = 40000		
SR-KI$_{\text{w/o re-use}}$	0.3600	0.3636	0.2693	0.2213
SR-KI	**0.6940**	**0.6039**	**0.5227**	**0.4227**

Table 4: Evaluation results of the ablation study by QA type, including generalization performance. Scores are averaged over three QA subtypes. **ID-Acc**: reference ID accuracy; **K-BERT**: knowledge BERTScore (F1); **ID-Gen**: reference ID generalization accuracy; **K-Gen**: knowledge generalization BERTScore. **w/o re-use**: without reusing the top-k indices selected by the retrieval layer in subsequent layers.

Ablation Study As shown in Table 4, reusing the top-100 indices selected by the retrieval layer in subsequent layers significantly enhances performance across all settings. This strategy consistently boosts both the main task performance (ID-Acc and K-BERT) and generalization ability (ID-Gen and K-Gen), with particularly substantial improvements observed at larger KB sizes, demonstrating its effectiveness in enhancing reasoning with large-scale knowledge.

Limitations and Future Work SR-KI excels at large-scale KB injection, but its refusal ability is limited. Future work may enhance this via retrieval-level losses (e.g., KL regularization) or by freezing projection layers during full fine-tuning. More complex tasks like multi-hop reasoning and multimodal retrieval also merit further exploration.

7 Conclusion

We propose SR-KI, a framework for efficient and real-time knowledge injection into LLMs via supervised attention. Unlike traditional RAG methods that depend on external retrievers, SR-KI introduces a dedicated retrieval layer trained with an attention-based loss, enabling efficient knowledge access entirely within the model's latent space. Extensive experiments demonstrate that SR-KI supports injection of up to 40K KBs while maintaining strong task performance and retrieval effectiveness, whereas other methods encounter out-of-memory errors and suffer severe performance degradation. SR-KI thus offers a practical and scalable solution for large-scale knowledge injection and supports real-time knowledge updates.

References

Abolghasemi, A.; Azzopardi, L.; Hashemi, S. H.; de Rijke, M.; and Verberne, S. 2025. Evaluation of Attribution Bias in Generator-Aware Retrieval-Augmented Large Language Models. In Che, W.; Nabende, J.; Shutova, E.; and Pilehvar, M. T., eds., *Findings of the Association for Computational Linguistics: ACL 2025*, 21105–21124. Vienna, Austria: Association for Computational Linguistics. ISBN 979-8-89176-256-5.

Chen, J.; Xiao, S.; Zhang, P.; Luo, K.; Lian, D.; and Liu, Z. 2024. BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation. arXiv:2402.03216.

Chung, H. W.; Hou, L.; Longpre, S.; Zoph, B.; Tai, Y.; Fedus, W.; Li, Y.; Wang, X.; Dehghani, M.; Brahma, S.; Webson, A.; Gu, S. S.; Dai, Z.; Suzgun, M.; Chen, X.; Chowdhery, A.; Castro-Ros, A.; Pellat, M.; Robinson, K.; Valter, D.; Narang, S.; Mishra, G.; Yu, A.; Zhao, V.; Huang, Y.; Dai, A.; Yu, H.; Petrov, S.; Chi, E. H.; Dean, J.; Devlin, J.; Roberts, A.; Zhou, D.; Le, Q. V.; and Wei, J. 2024. Scaling instruction-finetuned language models. *J. Mach. Learn. Res.*, 25(1).

Dao, T. 2023. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. arXiv:2307.08691.

De Cao, N.; Aziz, W.; and Titov, I. 2021. Editing Factual Knowledge in Language Models. In Moens, M.-F.; Huang, X.; Specia, L.; and Yih, S. W.-t., eds., *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 6491–6506. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805.

Dubois, Y.; Li, X.; Taori, R.; Zhang, T.; Gulrajani, I.; Ba, J.; Guestrin, C.; Liang, P.; and Hashimoto, T. B. 2024. AlpacaFarm: A Simulation Framework for Methods that Learn from Human Feedback. arXiv:2305.14387.

Gemini. 2025. Gemini: A Family of Highly Capable Multimodal Models. arXiv:2312.11805.

Grattafiori, A.; Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Vaughan, A.; and Amy Yang, e. a. 2024. The Llama 3 Herd of Models. arXiv:2407.21783.

Han, Z.; Gao, C.; Liu, J.; Zhang, J.; and Zhang, S. Q. 2024. Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey. arXiv:2403.14608.

Ilievski, F.; Garijo, D.; Chalupsky, H.; Divvala, N. T.; Yao, Y.; Rogers, C.; Li, R.; Liu, J.; Singh, A.; Schwabe, D.; and Szekely, P. 2021. KGTK: A Toolkit for Large Knowledge Graph Manipulation and Analysis. arXiv:2006.00088.

Ji, Z.; Lee, N.; Frieske, R.; Yu, T.; Su, D.; Xu, Y.; Ishii, E.; Bang, Y. J.; Madotto, A.; and Fung, P. 2023. Survey of Hallucination in Natural Language Generation. *ACM Comput. Surv.*, 55(12).

Leng, Q.; Portes, J.; Havens, S.; Zaharia, M.; and Carbin, M. 2024. Long Context RAG Performance of Large Language Models. arXiv:2411.03538.

Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; tau Yih, W.; Rocktäschel, T.; Riedel, S.; and Kiela, D. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401.

Li, X. L.; and Liang, P. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. arXiv:2101.00190.

Li, Y.; Huang, Y.; Yang, B.; Venkitesh, B.; Locatelli, A.; Ye, H.; Cai, T.; Lewis, P.; and Chen, D. 2024. SnapKV: LLM Knows What You are Looking for Before Generation. arXiv:2404.14469.

Meng, K.; Bau, D.; Andonian, A.; and Belinkov, Y. 2023a. Locating and Editing Factual Associations in GPT. arXiv:2202.05262.

Meng, K.; Sharma, A. S.; Andonian, A.; Belinkov, Y.; and Bau, D. 2023b. Mass-Editing Memory in a Transformer. arXiv:2210.07229.

Mitchell, E.; Lin, C.; Bosselut, A.; Finn, C.; and Manning, C. D. 2022. Fast Model Editing at Scale. In *International Conference on Learning Representations*.

Nakano, R.; Hilton, J.; Balaji, S.; Wu, J.; Ouyang, L.; Kim, C.; Hesse, C.; Jain, S.; Kosaraju, V.; Saunders, W.; Jiang, X.; Cobbe, K.; Eloundou, T.; Krueger, G.; Button, K.; Knight, M.; Chess, B.; and Schulman, J. 2022. WebGPT: Browser-assisted question-answering with human feedback. arXiv:2112.09332.

OpenAI. 2024. GPT-4 Technical Report. arXiv:2303.08774.

Pope, R.; Douglas, S.; Chowdhery, A.; Devlin, J.; Bradbury, J.; Levskaya, A.; Heek, J.; Xiao, K.; Agrawal, S.; and Dean, J. 2022. Efficiently Scaling Transformer Inference. arXiv:2211.05102.

Qian, H.; Zhu, Y.; Dou, Z.; Gu, H.; Zhang, X.; Liu, Z.; Lai, R.; Cao, Z.; Nie, J.-Y.; and Wen, J.-R. 2023. WebBrain: Learning to Generate Factually Correct Articles for Queries by Grounding on Large Web Corpus. arXiv:2304.04358.

Qwen. 2025. Qwen2.5 Technical Report. arXiv:2412.15115.

Rajbhandari, S.; Rasley, J.; Ruwase, O.; and He, Y. 2020. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. arXiv:1910.02054.

Rozner, A.; Battash, B.; Wolf, L.; and Lindenbaum, O. 2024. Knowledge Editing in Language Models via Adapted Direct Preference Optimization. In Al-Onaizan, Y.; Bansal, M.; and Chen, Y.-N., eds., *Findings of the Association for Computational Linguistics: EMNLP 2024*, 4761–4774. Miami, Florida, USA: Association for Computational Linguistics.

Ru, D.; Qiu, L.; Hu, X.; Zhang, T.; Shi, P.; Chang, S.; Jiayang, C.; Wang, C.; Sun, S.; Li, H.; Zhang, Z.; Wang, B.; Jiang, J.; He, T.; Wang, Z.; Liu, P.; Zhang, Y.; and Zhang, Z. 2024. RAGChecker: A Fine-grained Framework for Diagnosing Retrieval-Augmented Generation. arXiv:2408.08067.

Sun, Z.; Zang, X.; Zheng, K.; Song, Y.; Xu, J.; Zhang, X.; Yu, W.; Song, Y.; and Li, H. 2025. ReDeEP: Detecting Hallucination in Retrieval-Augmented Generation via Mechanistic Interpretability. arXiv:2410.11414.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; Rodriguez, A.; Joulin, A.; Grave, E.; and Lample, G. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971.

Vrandečić, D.; and Krötzsch, M. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10): 78–85.

Wang, P.; Li, Z.; Zhang, N.; Xu, Z.; Yao, Y.; Jiang, Y.; Xie, P.; Huang, F.; and Chen, H. 2024. WISE: Rethinking the Knowledge Memory for Lifelong Model Editing of Large Language Models. arXiv:2405.14768.

Wang, X.; Isazawa, T.; Mikaelyan, L.; and Hensman, J. 2025. KBLaM: Knowledge Base augmented Language Model. arXiv:2410.10450.

Wei, J.; Bosma, M.; Zhao, V. Y.; Guu, K.; Yu, A. W.; Lester, B.; Du, N.; Dai, A. M.; and Le, Q. V. 2022. Finetuned Language Models Are Zero-Shot Learners. arXiv:2109.01652.

Xiao, S.; Liu, Z.; Zhang, P.; and Muennighoff, N. 2023. C-Pack: Packaged Resources To Advance General Chinese Embedding. arXiv:2309.07597.

Xu, Y.; Cai, T.; Jiang, J.; and Song, X. 2024. Face4Rag: Factual Consistency Evaluation for Retrieval Augmented Generation in Chinese. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, 6083–6094. ACM.

Yu, B.; and Chai, Y. 2025. EvolKV: Evolutionary KV Cache Compression for LLM Inference. arXiv:2509.08315.

Yu, Y.; Ping, W.; Liu, Z.; Wang, B.; You, J.; Zhang, C.; Shoeybi, M.; and Catanzaro, B. 2024. RankRAG: Unifying Context Ranking with Retrieval-Augmented Generation in LLMs. arXiv:2407.02485.

Zhang, Y.; Li, M.; Long, D.; Zhang, X.; Lin, H.; Yang, B.; Xie, P.; Yang, A.; Liu, D.; Lin, J.; Huang, F.; and Zhou, J. 2025. Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models. arXiv:2506.05176.

Zhao, W. X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; Du, Y.; Yang, C.; Chen, Y.; Chen, Z.; Jiang, J.; Ren, R.; Li, Y.; Tang, X.; Liu, Z.; Liu, P.; Nie, J.-Y.; and Wen, J.-R. 2025a. A Survey of Large Language Models. arXiv:2303.18223.

Zhao, X.; Zhong, Y.; Sun, Z.; Hu, X.; Liu, Z.; Li, D.; Hu, B.; and Zhang, M. 2025b. FunnelRAG: A Coarse-to-Fine Progressive Retrieval Paradigm for RAG. arXiv:2410.10293.

Zhu, H.; Lan, Y.; Li, X.; and Qian, W. 2025. Initializing and Retrofitting Key-Value Adaptors for Traceable Model Editing. In Che, W.; Nabende, J.; Shutova, E.; and Pilehvar, M. T., eds., *Findings of the Association for Computational Linguistics: ACL 2025*, 2958–2971. Vienna, Austria: Association for Computational Linguistics. ISBN 979-8-89176-256-5.

A Details of Dataset

A.1 Examples of KB

We construct our dataset based on Wikidata (Vrandečić and Krötzsch 2014). All KBs are constructed in Chinese, as shown in Figure 5. Each knowledge triple in the form of (*subject*, *relation*, *object*) is transformed into a key-value pair, using (*subject*, *relation*) as key and *object* as value. In the reference ID KB, each knowledge triple is encoded as a key, with its corresponding randomly assigned uppercase ID encoded as the value. Then we encode these key-value pairs via a pretrained sentence encoder. The resulting key-value embeddings are injected into the LLM's KV cache through projection adapters, which align their dimensions with that of the cache.

Factual Knowledge KB

Knowledge Triple	Key	Value
(白湛, 表字, 季清)	白湛的表字	季清
(尼古拉·克里连科, 党籍, 苏联共产党)	尼古拉·克里连科的党籍	苏联共产党
(布莱恩·奥斯汀·格林, 职业, 电视演员)	布莱恩·奥斯汀·格林的职业	电视演员

Reference ID KB

Knowledge Triple	Key	Value
(白湛, 表字, 季清)	"白湛的表字是季清"这条知识对应的物料库编号	A
(尼古拉·克里连科, 党籍, 苏联共产党)	"尼古拉·克里连科的党籍是苏联共产党"这条知识对应的物料库编号	B
(布莱恩·奥斯汀·格林, 职业, 电视演员)	"布莱恩·奥斯汀·格林的职业是电视演员"这条知识对应的物料库编号	C

Figure 5: Key-value representations of factual knowledge and reference ID KBs.

A.2 Data Statistics

As summarized in Table 5, the training and test sets include diverse *relation* types and *(subject, relation, object)* pairs, encouraging the projection adapters of SR-KI to learn generalizable mapping patterns of key-value pairs rather than memorizing specific facts. Since each factual knowledge KB is paired with a corresponding reference ID KB, the total number of KBs is doubled.

A.3 QA Templates

As shown in Figure 6 (top), our QA examples follow a consistent template. During both training and inference, the model is prompted to generate the knowledge fact along with its corresponding ID, where all IDs are randomly sampled from uppercase letters. Notably, *Multi-entity-S* refers to

Category	Train	Validation	Test
Relation Type Num	229	99	139
(Subject, Relation, object) Num	141783	2819	21253
QA Num	150000	2000	20000

Table 5: Statistics of the training, validation, and test sets.

questions involving two relations associated with the *same entity*, whereas *Multi-entity-D* pertains to questions involving one relation for *each of two distinct entities*. For the in-context learning QA template, we incorporate the full KB representation into the prompt (Figure 7, top) and provide 4-shot examples that do not exist in the test set (bottom) to help the model understand the task.

Question Template

根据物料库回答："{question}"，若物料库包含相关信息，则给出答案，并在答案末尾标注对应物料编号，如 [A]、[B]、[C]；否则请直接回答"根据物料库信息无法回答该问题"。

Single-entity QA

Q: 白湛的表字是什么?

A: 根据提供的物料库信息，白湛的表字是季清[A]

Muti-entity-S QA

Q: 尼古拉·克里连科的党籍和所获奖项分别是什么?

A: 根据提供的物料库信息，尼古拉·克里连科的党籍是苏联共产党[B]，所获奖项是列宁勋章[H]

Muti-entity-D QA

Q: 耶稣圣心堂的所在行政领土实体和德川家庆的职业分别是什么?

A: 根据提供的物料库信息，耶稣圣心堂的所在行政领土实体是爱丁堡政治[A]，德川家庆的职业是征夷大将军[U]

Unanswerable QA

Q: 杰姬·科林斯的所获奖项是什么?

A: 根据物料库信息无法回答该问题

Figure 6: QA examples. We adopt a uniform question template for all QA types.

KB Samples

廖胜文的出生地是大甲区
"廖胜文的出生地是大甲区"这条知识对应的物料库编号是A
国家档案馆大楼的所在街道是宾夕法尼亚大道
"国家档案馆大楼的所在街道是宾夕法尼亚大道"这条知识对应的物料库编号是X

ICL QA Template

{KB_samples}

根据物料库回答问题，若物料库包含相关信息，则给出答案，并在答案末尾标注对应物料编号，如 [A]、[B]、[C]；否则请直接回答"根据物料库信息无法回答该问题"。

示例：
问题：白湛的表字是什么?
答案：根据提供的物料库信息，白湛的表字是季清[A]

问题：尼古拉·克里连科的党籍和所获奖项分别是什么
答案：根据提供的物料库信息，尼古拉·克里连科的党籍是苏联共产党[B]，所获奖项是列宁勋章[H]

问题：耶稣圣心堂的所在行政领土实体和德川家庆的职业分别是什么?
答案：根据提供的物料库信息，耶稣圣心堂的所在行政领土实体是爱丁堡政治[A]，德川家庆的职业是征夷大将军[U]

问题：无法回答的问题
答案：根据物料库信息无法回答该问题

问题：{question}

Figure 7: QA template for in-context learning.

B Details of Experiments

B.1 Max-Pooling-Based KB Compression for Inference

We find a method to further enhance the reasoning ability of SR-KI called **Max-Pooling-Based Compression** which is demonstrated in (Li et al. 2024; Yu and Chai 2025). We apply 1D max-pooling over the KB attention weights in every layer. First, we normalize the attention weights via softmax at dimension of D: $\tilde{A}^l_{\text{KB}} = \text{softmax}(A^l_{\text{KB}})$. Then we sum over the query dimension to obtain $\tilde{\mathbf{A}}^l_{\text{KB}} = \sum_{n=1}^{N} \tilde{A}^l_{\text{KB}}[n,:] \in \mathbb{R}^M$. Next, 1D max-pooling is applied over the KB dimension: $\hat{\mathbf{A}}^l_{\text{KB}} = \text{MaxPool1D}(\tilde{\mathbf{A}}^l_{\text{KB}})$. The pooled scores are used for the top-k KB indices selection: $\mathcal{I}^l_{\text{Pool}} = \text{TopK}\left(\hat{\mathbf{A}}^l_{\text{KB}}, k\right)$. This allows KBs with high attention weights to dominate nearby entries, effectively compressing the KB through a pooling mechanism. We incorporate the results of the pooling-based method into our overall results, with the convolutional kernel size set to 7.

Method	Single	Multi-S	Multi-D	Avg.
		KB Size = 100		
BM25	0.7700	0.7375	0.8000	0.7692
Dense Retrieval	0.9900	0.9750	**0.9625**	0.9758
SR-KI	**1.0000**	**0.9975**	0.9550	**0.9842**
		KB Size = 1000		
BM25	0.7200	0.6500	0.6750	0.6817
Dense Retrieval	0.9900	0.9375	**0.8625**	0.9300
SR-KI	**0.9920**	**0.9875**	0.8450	**0.9415**
		KB Size = 10000		
BM25	0.5600	0.4375	0.5000	0.4992
Dense Retrieval	0.9500	0.9000	0.6750	0.8417
SR-KI	**0.9680**	**0.9350**	**0.7075**	**0.8702**
		KB Size = 40000		
BM25	0.4900	0.3500	0.2500	0.3633
Dense Retrieval	0.8700	0.8625	0.4000	0.7108
SR-KI	**0.9180**	**0.9000**	**0.5900**	**0.8027**

Table 6: Comparison results with BM25 and dense retrieval on Recall@Top across different QA sub-types and KB sizes. **Single**: Single-entity; **Multi-S**: Multi-entity (single entity with two relations); **Multi-D**: Multi-entity (different entities, each with one relation).

B.2 Experiment Results across Varying KB Sizes

Main Experiment Results We report comprehensive evaluation results across KB sizes from 100 to 40K. As shown in Table 7, SR-KI consistently outperforms all baselines under different KB size settings. Notably, SR-KI$_{+\text{pool}}$ with max pooling achieves up to 10-point improvements over base SR-KI on ID generation and knowledge reasoning at larger scales (10K–40K), highlighting its strong scalability and effectiveness. Interestingly, SR-KI$_{+\text{pool}}$ struggles to recall the correct KBs among the top-ranked candidates (e.g., top-10), as shown in Table 8. This may be attributed to the max pooling, which retains the KBs with high attention weights in early layers and leads to a sparser distribution of attention weights. During training, the model is optimized to adapt to a dense attention distribution over the KB entries. However, during inference, max pooling significantly sparsifies this distribution. Despite this sparsification, the model maintains a high Recall@100, indicating that the essential KBs are still retained within the receptive scope of

the reasoning process, thereby preserving retrieval effectiveness while enhancing reasoning focus.

Generalization Experiment Results We provide a thorough evaluation covering KB sizes ranging from 100 to 40K. As demonstrated in Table 9, SR-KI consistently outperforms all baselines, with max pooling providing an additional boost—yielding up to a 7-point improvement over base SR-KI under large-scale injection settings. In contrast, KBLaM exhibits substantial performance degradation at a KB size of 2K, with scores even falling below zero at 5K. Moreover, as shown in Table 10, its retrieval ability begins to deteriorate at much smaller KB sizes, whereas SR-KI consistently maintains strong performance across all scales. In comparison, ICL also suffers from a notable performance drop, with significantly worse results than those reported in the main experiments, highlighting its instability. Taken together, these results demonstrate the strong robustness and effectiveness of SR-KI across a wide range of KB sizes.

Analysis Among the three QA types, the model exhibits notably poor performance on multi-entity QA, where each of the two entities is associated with an independent relation. This underperformance is significantly more pronounced compared to other QA types across ID generation, knowledge reasoning, and KB retrieval. We attribute this to the increased complexity of the task, which requires the model to simultaneously identify and reason over KBs associated with multiple entities.

B.3 Extended Experiment Results

Retrieval Layer Identification across Different Model Sizes, Series and Encoders We further perform retrieval layer identification across different model sizes, series and encoders, using 100 samples with the maximum KB set to 100. As shown in the left and middle of Figure 8, similar to the results on Qwen2.5-7B-Instruct, the retrieval layer of Qwen2.5-3B-Instruct is the 32nd, while that of Qwen2.5-14B-Instruct is the 37th. At the retrieval layer, both models achieve a significant improvement in ID accuracy and knowledge reasoning, surpassing the performance observed when the correct KBs are injected into other layers. This suggests that the retrieval layer consistently emerges across models of different sizes, underscoring the universality of SR-KI. Furthermore, we conduct retrieval layer identification on Llama-3-8B-Instruct (Grattafiori et al. 2024), with results shown in the right of Figure 8. We observe that a retrieval layer also emerges at the 30th layer, which demonstrates that SR-KI consistently identifies retrieval layers across different model series and underscores its broad applicability. Additionally, we employ bge-m3 (Chen et al. 2024) and Qwen3-Embedding-8B (Zhang et al. 2025) as KB encoders to further identify the retrieval layer on Qwen2.5-7B-Instruct. As illustrated in Figure 9, the 25th layer consistently emerges as the retrieval layer, aligning with our main experiments. This consistency highlights that the retrieval layer is strongly tied to the model's architecture, demonstrating the generalizability of our method.

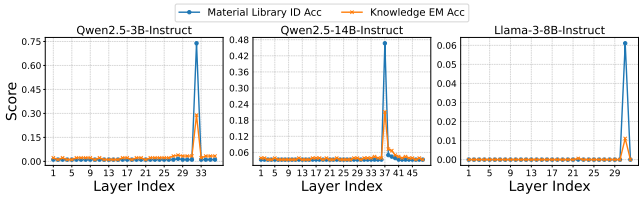

Figure 8: Experimental results of retrieval layer identification on Qwen2.5-3B-Instruct (**left**), Qwen2.5-14B-Instruct (**middle**) and Llama-3-8B-Instruct (**right**).

Retrieval Ability Comparison with BM25 and Dense Retrieval To ensure fairness, we adopt the same QA template when comparing the retrieval ability of SR-KI, BM25, and dense retrieval, with Recall@Top as the evaluation metric. We apply bge-large-zh-v1.5 as the encoder model to generate embeddings and compute the cosine similarity between queries and KB entries. Table 6 shows that SR-KI consistently outperforms BM25 and dense retrieval across a wide KB size range (100 to 40K). Notably, at a KB size of 40K, SR-KI delivers substantial gains of 9.19 and 43.94 points over dense retrieval and BM25, respectively, highlighting its deep semantic understanding and clear superiority over both dense and sparse retrieval methods. This suggests that traditional sparse or dense retrieval methods struggle to match the truly relevant information when the query involves complex instructions or multifaceted semantic content. In contrast, SR-KI adopts an end-to-end retrieval–reasoning paradigm that eliminates the need for external pipelines, performing retrieval entirely within the model and enabling more efficient knowledge injection.

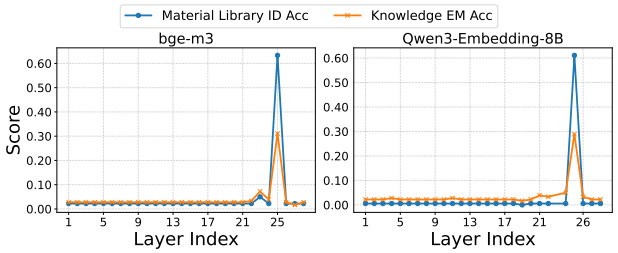

Figure 9: Experimental results of retrieval layer identification on Qwen2.5-7B-Instruct, using bge-m3 (**left**) and Qwen3-Embedding-8B (**right**) as KB encoders.

Accurate KB Retrieval before Retrieval Layer is not Necessary To examine the impact of hidden representations influenced by KB injection on the retrieval layer, we randomly inject negative KBs into the layers preceding the retrieval layer. As shown in Table 11, under both the standard and generalization settings, SR-KI$_{+\text{random}}$ achieves comparable or even superior performance to the base SR-KI across ID generation, knowledge reasoning, and KB retrieval. This indicates that accurate KB injection prior to the retrieval layer may not be essential, potentially serving instead as a trigger signal for subsequent layers.

Method	Reference ID Acc				Knowledge BERTScore (F1)			
	Single	Multi-S	Multi-D	Avg.	Single	Multi-S	Multi-D	Avg.
KB Size = 100								
ICL	0.8640	0.4300	0.7250	0.6730	**0.9796**	**0.9926**	**0.9832**	**0.9851**
KBLaM	0.9840	**0.9800**	0.9550	0.9730	0.8909	0.8817	0.8450	0.8725
SR-KI	**0.9960**	0.9750	**0.9800**	**0.9837**	0.8760	0.8779	0.8103	0.8547
KB Size = 200								
ICL	0.8120	0.4450	0.8000	0.6857	**0.9790**	**0.9923**	**0.9766**	**0.9826**
KBLaM	0.9720	0.9650	0.9400	0.9590	0.8789	0.8743	0.8251	0.8594
SR-KI	0.9880	**0.9950**	**0.9850**	**0.9893**	0.8407	0.8807	0.7861	0.8358
SR-KI$_{+\text{pool}}$	**0.9960**	0.9900	0.9650	0.9837	0.8577	0.8746	0.7964	0.8429
KB Size = 300								
ICL	0.7880	0.3500	0.7200	0.6193	**0.9799**	**0.9812**	**0.9679**	**0.9763**
KBLaM	0.9360	0.9300	0.9200	0.9287	0.8499	0.8472	0.8047	0.8339
SR-KI	**0.9840**	**0.9950**	**0.9550**	**0.9780**	0.8360	0.8797	0.7699	0.8285
SR-KI$_{+\text{pool}}$	0.9800	**0.9950**	**0.9550**	0.9767	0.8522	0.8911	0.7834	0.8422
KB Size = 500								
KBLaM	0.9240	0.8800	0.8650	0.8897	0.8095	0.8292	**0.7587**	0.7991
SR-KI	0.9800	**0.9800**	0.9300	0.9633	0.8220	**0.8732**	0.7341	0.8098
SR-KI$_{+\text{pool}}$	**0.9960**	0.9650	**0.9350**	**0.9653**	**0.8542**	0.8544	0.7330	**0.8139**
KB Size = 1000								
KBLaM	0.8400	0.7550	0.7500	0.7817	0.7003	0.7105	0.6449	0.6852
SR-KI	0.9800	0.9700	0.8900	0.9467	0.7944	0.8526	0.6982	0.7817
SR-KI$_{+\text{pool}}$	**0.9880**	**0.9850**	**0.9100**	**0.9610**	**0.8507**	**0.8725**	**0.7450**	**0.8227**
KB Size = 2000								
KBLaM	0.5040	0.4800	0.4850	0.4897	0.3454	0.4651	0.3349	0.3818
SR-KI	0.9120	0.9650	0.8750	0.9173	0.7885	0.8023	0.6845	0.7584
SR-KI$_{+\text{pool}}$	**0.9520**	**0.9700**	**0.8800**	**0.9340**	**0.8341**	**0.8517**	**0.7262**	**0.8040**
KB Size = 5000								
KBLaM	0.0240	0.0100	0.0250	0.0197	-1.1210	-1.1627	-1.1625	-1.1487
SR-KI	0.8640	0.9150	0.8250	0.8680	0.7235	0.7683	0.6384	0.7101
SR-KI$_{+\text{pool}}$	**0.9560**	**0.9500**	**0.8500**	**0.9187**	**0.8100**	**0.8154**	**0.6401**	**0.7552**
KB Size = 10000								
KBLaM	0.0160	0.0100	0.0000	0.0087	-1.2723	-1.2678	-1.2723	-1.2708
SR-KI	0.8000	0.7950	0.7450	0.7800	0.6764	0.7066	0.6201	0.6677
SR-KI$_{+\text{pool}}$	**0.9120**	**0.9050**	**0.7900**	**0.8690**	**0.7884**	**0.7837**	**0.6638**	**0.7453**
KB Size = 20000								
KBLaM	0.0100	0.0062	0.0000	0.0054	-1.2723	-1.2723	-1.2723	-1.2723
SR-KI	0.7200	0.7800	0.7000	0.7333	0.6449	0.6468	0.5775	0.6231
SR-KI$_{+\text{pool}}$	**0.8800**	**0.8250**	**0.7400**	**0.8150**	**0.7296**	**0.7810**	**0.6444**	**0.7183**
KB Size = 40000								
SR-KI	0.6720	0.7650	0.6450	0.6940	0.6108	0.6508	0.5500	0.6039
SR-KI$_{+\text{pool}}$	**0.8040**	**0.7800**	**0.6550**	**0.7463**	**0.7163**	**0.7570**	**0.6278**	**0.7004**

Table 7: Reference ID accuracy (Reference ID Acc), knowledge BERTScore (F1) on *object*. We inject all KBs when the KB size is equal to 100, and apply top-k=100 selection when the KB size exceeds 100. Avg. denotes the average of the three subtypes. Single denotes Single-entity QA; Multi-S denotes Multi-entity QA with two relations for one entity; Multi-D denotes Multi-entity QA with one relation for each of two distinct entities; $_{+\text{pool}}$ denotes the Max-Pooling-Based Compression.

Method	Recall@100				Recall@10				Recall@Top			
	Single	Multi-S	Multi-D	Avg.	Single	Multi-S	Multi-D	Avg.	Single	Multi-S	Multi-D	Avg.
KB Size = 100												
KBLaM	-	-	-	-	0.4800	0.4850	0.4500	0.4717	0.1740	0.2375	0.2500	0.2205
SR-KI	-	-	-	-	**1.0000**	**1.0000**	**0.9900**	**0.9967**	**1.0000**	**0.9975**	**0.9550**	**0.9842**
KB Size = 200												
KBLaM	0.8920	0.9075	0.8550	0.8848	0.3100	0.3150	0.2975	0.3075	0.1040	0.1575	0.1700	0.1438
SR-KI	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**0.9825**	**0.9942**	**0.9980**	**0.9975**	**0.9300**	**0.9752**
SR-KI$_{+\text{pool}}$	**1.0000**	**1.0000**	0.9975	0.9992	0.6660	0.3400	0.3675	0.4578	0.0820	0.2050	0.2075	0.1648
KB Size = 300												
KBLaM	0.7660	0.8100	0.7350	0.7703	0.2580	0.2300	0.2225	0.2368	0.0640	0.1125	0.1300	0.1022
SR-KI	**1.0000**	**1.0000**	**0.9975**	**0.9992**	**1.0000**	**1.0000**	**0.9725**	**0.9908**	**0.9960**	**1.0000**	**0.9075**	**0.9678**
SR-KI$_{+\text{pool}}$	**1.0000**	**1.0000**	0.9950	0.9983	0.6500	0.3450	0.3550	0.4500	0.0640	0.2025	0.2150	0.1605
KB Size = 500												
KBLaM	0.6100	0.6550	0.5925	0.6192	0.1660	0.1425	0.1725	0.1603	0.0540	0.0600	0.0950	0.0697
SR-KI	**1.0000**	**1.0000**	**0.9950**	**0.9983**	**1.0000**	**1.0000**	**0.9675**	**0.9892**	**0.9900**	**0.9975**	**0.8900**	**0.9592**
SR-KI$_{+\text{pool}}$	**1.0000**	0.9975	0.9900	0.9958	0.6560	0.3350	0.3600	0.4503	0.0460	0.2250	0.1800	0.1503
KB Size = 1000												
KBLaM	0.4500	0.4450	0.4175	0.4375	0.1080	0.0775	0.1000	0.0952	0.0420	0.0400	0.0575	0.0465
SR-KI	**1.0000**	**1.0000**	**0.9925**	**0.9975**	**1.0000**	**1.0000**	**0.9425**	**0.9808**	**0.9920**	**0.9875**	**0.8450**	**0.9415**
SR-KI$_{+\text{pool}}$	0.9980	0.9975	0.9725	0.9893	0.6320	0.2900	0.3400	0.4207	0.0640	0.2200	0.2075	0.1638
KB Size = 2000												
KBLaM	0.3220	0.2475	0.2875	0.2857	0.0680	0.0375	0.0650	0.0568	0.0160	0.0100	0.0350	0.0203
SR-KI	**1.0000**	**1.0000**	**0.9850**	**0.9950**	**0.9980**	**1.0000**	**0.9075**	**0.9685**	**0.9820**	**0.9900**	**0.8000**	**0.9240**
SR-KI$_{+\text{pool}}$	0.9960	0.9975	0.9675	0.9870	0.6240	0.2900	0.3000	0.4047	0.0340	0.2200	0.2150	0.1563
KB Size = 5000												
KBLaM	0.1480	0.1025	0.1525	0.1343	0.0340	0.0075	0.0225	0.0213	0.0060	0.0025	0.0150	0.0078
SR-KI	**1.0000**	**1.0000**	**0.9650**	**0.9883**	**1.0000**	**0.9975**	**0.8550**	**0.9508**	**0.9800**	**0.9750**	**0.7350**	**0.8967**
SR-KI$_{+\text{pool}}$	0.9960	0.9900	0.9450	0.9770	0.5780	0.2850	0.3000	0.3877	0.0400	0.2225	0.2100	0.1575
KB Size = 10000												
KBLaM	0.0960	0.0375	0.0875	0.0737	0.0180	0.0025	0.0150	0.0118	0.0060	0.0000	0.0100	0.0053
SR-KI	**1.0000**	**1.0000**	**0.9425**	**0.9808**	**0.9980**	**0.9950**	**0.8025**	**0.9318**	**0.9680**	**0.9350**	**0.7075**	**0.8702**
SR-KI$_{+\text{pool}}$	0.9880	0.9750	0.9150	0.9593	0.5480	0.2700	0.3050	0.3743	0.0220	0.2325	0.2375	0.1640
KB Size = 20000												
KBLaM	0.0400	0.0031	0.0281	0.0237	0.0075	0.0000	0.0062	0.0046	0.0025	0.0000	0.0000	0.0008
SR-KI	**1.0000**	**1.0000**	**0.9250**	**0.9750**	**0.9920**	**0.9875**	**0.7575**	**0.9123**	**0.9480**	**0.9250**	**0.6425**	**0.8385**
SR-KI$_{+\text{pool}}$	0.9760	0.9400	0.8750	0.9303	0.5300	0.2575	0.2675	0.3517	0.0200	0.2300	0.2425	0.1642
KB Size = 40000												
SR-KI	**0.9980**	**1.0000**	**0.8800**	**0.9593**	**0.9860**	**0.9750**	**0.7050**	**0.8887**	**0.9180**	**0.9000**	**0.5900**	**0.8027**
SR-KI$_{+\text{pool}}$	0.9560	0.9000	0.8325	0.8962	0.5060	0.2600	0.2575	0.3412	0.0040	0.2400	0.2375	0.1605

Table 8: Retrieval performance across KB sizes on *retrieval layer*. We inject all KBs when the KB size is equal to 100, and apply top-k=100 selection when the KB size exceeds 100. Each group includes recall on Single, Multi-S, and Multi-D, along with their average. Note: This is a multi-target retrieval task. For Single-entity QA, two KB are expected to be retrieved; for Multi-entity QA, four are required. Therefore, @Top denotes whether all required KB appear within the top 2 or top 4 positions, respectively.

Method	Reference ID Acc				Knowledge BERTScore (F1)			
	Single	Multi-S	Multi-D	Avg.	Single	Multi-S	Multi-D	Avg.
KB Size = 100								
ICL	0.6480	0.3950	0.6750	0.5727	0.3302	0.6005	0.8096	0.5801
KBLaM	0.9120	0.8950	0.8650	0.8907	0.5895	0.7229	0.5914	0.6346
SR-KI	**0.9720**	**0.9150**	**0.9200**	**0.9357**	**0.7589**	**0.7266**	**0.6518**	**0.7124**
KB Size = 200								
ICL	0.6080	0.3150	0.6750	0.5327	0.3426	0.5657	0.8508	0.5864
KBLaM	0.9000	0.8750	0.7950	0.8567	0.5725	0.6959	0.4657	0.5780
SR-KI	**0.9720**	0.8750	0.8600	0.9023	0.7426	0.7263	0.6059	0.6916
SR-KI$_{+\text{pool}}$	0.9680	**0.8900**	**0.9000**	**0.9193**	**0.7603**	**0.7244**	**0.6331**	**0.7059**
KB Size = 300								
ICL	0.6400	0.3600	0.5200	0.5067	0.3507	0.5758	0.6835	0.5367
KBLaM	0.8800	0.8650	0.7850	0.8433	0.5480	0.6588	0.5112	0.5727
SR-KI	**0.9600**	**0.8900**	0.8350	0.8950	0.7166	0.6975	0.5990	0.6710
SR-KI$_{+\text{pool}}$	0.9560	0.8700	**0.8850**	**0.9037**	**0.7449**	**0.7131**	**0.6298**	**0.6959**
KB Size = 500								
KBLaM	0.8400	0.8300	0.7650	0.8117	0.5083	0.6361	0.4818	0.5421
SR-KI	0.9440	0.8700	0.7900	0.8680	0.7211	0.6757	0.5556	0.6508
SR-KI$_{+\text{pool}}$	**0.9480**	**0.8850**	**0.8550**	**0.8960**	**0.7218**	**0.6950**	**0.5950**	**0.6706**
KB Size = 1000								
KBLaM	0.7280	0.7200	0.5950	0.6810	0.4188	0.5414	0.4081	0.4561
SR-KI	0.8933	**0.8250**	0.7083	0.8089	**0.7364**	0.6138	0.4126	0.5876
SR-KI$_{+\text{pool}}$	**0.9300**	0.8000	**0.7875**	**0.8392**	0.6584	**0.6497**	**0.5212**	**0.6098**
KB Size = 2000								
KBLaM	0.4560	0.4100	0.4200	0.4287	0.1894	0.1777	0.1166	0.1612
SR-KI	0.8760	0.7900	0.7400	0.8020	**0.7190**	0.6142	0.4700	0.6011
SR-KI$_{+\text{pool}}$	**0.9040**	**0.8050**	**0.8300**	**0.8463**	0.7147	**0.6435**	**0.5117**	**0.6233**
KB Size = 5000								
KBLaM	0.0200	0.0250	0.0100	0.0183	-1.1423	-1.1846	-1.1996	-1.1755
SR-KI	0.8160	0.7200	0.6750	0.7370	0.6318	0.5756	0.4246	0.5440
SR-KI$_{+\text{pool}}$	**0.8960**	**0.7750**	**0.7200**	**0.7970**	**0.6992**	**0.6003**	**0.4632**	**0.5876**
KB Size = 10000								
KBLaM	0.0200	0.0050	0.0050	0.0100	-1.2723	-1.2681	-1.2723	-1.2709
SR-KI	0.6880	0.7200	0.5950	0.6677	0.6141	0.5133	0.4032	0.5102
SR-KI$_{+\text{pool}}$	**0.8600**	**0.7400**	**0.6250**	**0.7417**	**0.6188**	**0.5620**	**0.4220**	**0.5343**
KB Size = 20000								
KBLaM	0.0000	0.0000	0.0000	0.0000	-1.2723	-1.2723	-1.2723	-1.2723
SR-KI	0.6360	0.6400	0.5250	0.6003	0.5810	0.4512	0.3565	0.4629
SR-KI$_{+\text{pool}}$	**0.8200**	**0.6550**	**0.5800**	**0.6850**	**0.6341**	**0.5413**	**0.4230**	**0.5328**
KB Size = 40000								
SR-KI	0.5680	0.5500	0.4500	0.5227	0.5335	0.4158	0.3187	0.4227
SR-KI$_{+\text{pool}}$	**0.7480**	**0.5950**	**0.5000**	**0.6143**	**0.5967**	**0.5031**	**0.3890**	**0.4963**

Table 9: Generalization evaluation results of Reference ID accuracy (Reference ID Acc), knowledge BERTScore (F1) on *object*. We inject all KBs when the KB size is equal to 100, and apply top-k=100 selection when the KB size exceeds 100. Avg. denotes the average of the three sub-types. Single denotes Single-entity QA; Multi-S denotes Multi-entity QA with two relations for one entity; Multi-D denotes Multi-entity QA with one relation for each of two distinct entities; $_{+\text{pool}}$ denotes the Max-Pooling-Based Compression.

Method	Recall@100				Recall@10				Recall@Top			
	Single	Multi-S	Multi-D	Avg.	Single	Multi-S	Multi-D	Avg.	Single	Multi-S	Multi-D	Avg.
KB Size = 100												
KBLaM	-	-	-	-	0.4500	0.4700	0.3700	0.4300	0.1820	0.2550	0.2175	0.2182
SR-KI	-	-	-	-	**0.9920**	**0.9775**	**0.9675**	**0.9790**	**0.9720**	**0.9525**	**0.8975**	**0.9407**
KB Size = 200												
KBLaM	0.8320	0.9250	0.8175	0.8582	0.3300	0.3250	0.2625	0.3058	0.1340	0.1750	0.1400	0.1497
SR-KI	**1.0000**	**0.9950**	**0.9950**	**0.9967**	**0.9820**	**0.9675**	**0.9500**	**0.9665**	**0.9600**	**0.9325**	**0.8525**	**0.9150**
SR-KI$_{+\text{pool}}$	0.9920	0.9825	**0.9950**	0.9898	0.6640	0.3400	0.3650	0.4563	0.0840	0.1900	0.1750	0.1497
KB Size = 300												
KBLaM	0.7340	0.7950	0.7200	0.7497	0.2660	0.2325	0.2025	0.2337	0.1120	0.1100	0.0975	0.1065
SR-KI	**0.9980**	**0.9875**	**0.9925**	**0.9927**	**0.9760**	**0.9550**	**0.9200**	**0.9503**	**0.9560**	**0.9350**	**0.8275**	**0.9062**
SR-KI$_{+\text{pool}}$	0.9940	0.9750	0.9900	0.9863	0.6740	0.3475	0.3675	0.4630	0.0900	0.2175	0.2075	0.1472
KB Size = 500												
KBLaM	0.5860	0.6325	0.5425	0.5870	0.2000	0.1500	0.1450	0.1650	0.0540	0.0725	0.0700	0.0655
SR-KI	**0.9980**	**0.9850**	**0.9850**	**0.9893**	**0.9680**	**0.9450**	**0.8975**	**0.9368**	**0.9520**	**0.9050**	**0.8000**	**0.8857**
SR-KI$_{+\text{pool}}$	0.9840	0.9700	0.9825	0.9788	0.6080	0.3275	0.3350	0.4235	0.0540	0.1925	0.1950	0.1472
KB Size = 1000												
KBLaM	0.4340	0.4225	0.3625	0.4063	0.1260	0.0825	0.0925	0.1003	0.0400	0.0450	0.0525	0.0458
SR-KI	**0.9867**	**0.9917**	**0.9667**	**0.9817**	**0.9667**	**0.9500**	**0.8750**	**0.9306**	**0.9533**	**0.9250**	**0.7542**	**0.8775**
SR-KI$_{+\text{pool}}$	0.9750	0.9500	0.9500	0.9583	0.5750	0.2750	0.3125	0.3875	0.0600	0.2125	0.1938	0.1554
KB Size = 2000												
KBLaM	0.3180	0.2700	0.2300	0.2727	0.0640	0.0550	0.0600	0.0597	0.0160	0.0175	0.0225	0.0187
SR-KI	**0.9820**	**0.9650**	**0.9500**	**0.9657**	**0.9600**	**0.9125**	**0.8375**	**0.9033**	**0.9400**	**0.8725**	**0.7100**	**0.8408**
SR-KI$_{+\text{pool}}$	0.9720	0.9275	0.9450	0.9482	0.5720	0.2900	0.3125	0.3915	0.0500	0.1925	0.2075	0.1500
KB Size = 5000												
KBLaM	0.1700	0.1000	0.1525	0.1408	0.0260	0.0075	0.0300	0.0212	0.0060	0.0000	0.0150	0.0070
SR-KI	**0.9720**	**0.9500**	**0.9075**	**0.9432**	**0.9580**	**0.8975**	**0.7775**	**0.8777**	**0.9320**	**0.8550**	**0.6200**	**0.8023**
SR-KI$_{+\text{pool}}$	0.9600	0.8975	0.8900	0.9158	0.5600	0.2650	0.3150	0.3800	0.0380	0.2025	0.2050	0.1485
KB Size = 10000												
KBLaM	0.0900	0.0675	0.0900	0.0825	0.0080	0.0000	0.0150	0.0077	0.0020	0.0000	0.0100	0.0040
SR-KI	**0.9660**	**0.9375**	**0.8675**	**0.9237**	**0.9520**	**0.8825**	**0.7175**	**0.8507**	**0.9100**	**0.8200**	**0.5750**	**0.7683**
SR-KI$_{+\text{pool}}$	0.9420	0.8500	0.8525	0.8815	0.5240	0.2600	0.2850	0.3563	0.0220	0.1975	0.2200	0.1465
KB Size = 20000												
KBLaM	0.0250	0.0188	0.0438	0.0292	0.0000	0.0000	0.0062	0.0021	0.0000	0.0000	0.0000	0.0000
SR-KI	**0.9600**	**0.9175**	**0.8400**	**0.9058**	**0.9440**	**0.8625**	**0.6425**	**0.8163**	**0.8800**	**0.7850**	**0.5225**	**0.7292**
SR-KI$_{+\text{pool}}$	0.9280	0.8275	0.8025	0.8527	0.5040	0.2425	0.2525	0.3330	0.0260	0.1925	0.2125	0.1437
KB Size = 40000												
SR-KI	**0.9580**	**0.9075**	**0.8025**	**0.8893**	**0.9320**	**0.8275**	**0.5800**	**0.7798**	**0.8500**	**0.7525**	**0.4725**	**0.6917**
SR-KI$_{+\text{pool}}$	0.8900	0.7800	0.7275	0.7992	0.4620	0.2150	0.2475	0.3082	0.0180	0.2000	0.2150	0.1443

Table 10: Generalization evaluation on retrieval performance across KB sizes at *retrieval layer*. We inject all KBs when the KB size is equal to 100, and apply top-k=100 selection when the KB size exceeds 100. Each group includes recall on Single, Multi-S, and Multi-D, along with their average. Note: This is a multi-target retrieval task. For Single-entity QA, two KB are expected to be retrieved; for Multi-entity QA, four are required. Therefore, @Top denotes whether all required KB appear within the top 2 or top 4 positions, respectively.

Method	ID-Acc	K-BERT	R@100	R@10	R@Top	ID-Gen	K-Gen	R@100-Gen	R@10-Gen	R@Top-Gen
					KB Size = 1000					
SR-KI	**0.9467**	**0.7817**	**0.9975**	0.9808	0.9415	0.8089	**0.5876**	**0.9817**	0.9306	0.8775
SR-KI$_{+\text{random}}$	0.9420	0.7430	**0.9975**	**0.9825**	**0.9480**	**0.8144**	0.5531	0.9814	**0.9336**	**0.8792**
					KB Size = 10000					
SR-KI	0.7800	0.6677	0.9808	0.9318	**0.8702**	0.6677	**0.5102**	0.9237	0.8507	0.7683
SR-KI$_{+\text{random}}$	**0.8257**	**0.6818**	**0.9858**	**0.9410**	0.8693	**0.6813**	0.5070	**0.9312**	**0.8563**	**0.7805**
					KB Size = 40000					
SR-KI	**0.7463**	**0.7004**	0.9593	0.8887	0.8027	0.5227	0.4227	0.8893	0.7798	0.6917
SR-KI$_{+\text{random}}$	0.7130	0.6125	**0.9635**	**0.8985**	**0.8165**	**0.5590**	**0.4578**	**0.8978**	**0.7940**	**0.6993**

Table 11: Comparison of base SR-KI with injection of random negative KBs before the retrieval layer. **ID-Acc**: reference ID accuracy; **K-BERT**: knowledge BERTScore (F1); **ID-Gen**: reference ID generalization accuracy; **K-Gen**: knowledge generalization BERTScore. **ID-Gen**: reference ID generalization accuracy; **K-Gen**: knowledge generalization BERTScore (F1) on *object*; **R@K**: recall at top-K retrieved KBs. **R@K-Gen**: recall at top-K retrieved knowledge bases during generalization evaluation. Scores are averaged over three QA subtypes. +**random**: the injection of randomly sampled negative KBs before retrieval layer. In both settings, the top-100 KB indices selected at the retrieval layer are reused in subsequent layers.