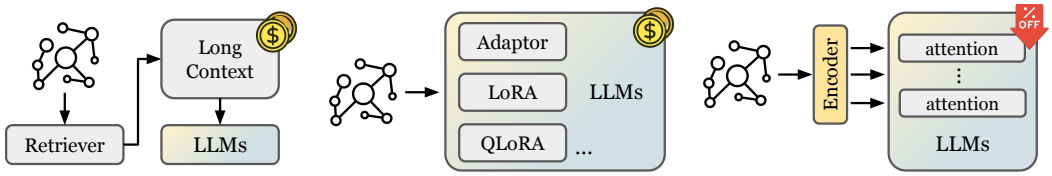
ATLASKV: AUGMENTING LLMS WITH BILLION-SCALE KNOWLEDGE GRAPHS IN 20GB VRAM

Anonymous authors
Paper under double-blind review

ABSTRACT

Retrieval-augmented generation (RAG) has shown some success in augmenting large language models (LLMs) with external knowledge. However, as a non-parametric knowledge integration paradigm for LLMs, RAG methods heavily rely on external retrieval modules and the retrieved textual context prior. Especially for very large scale knowledge augmentation, they would introduce substantial inference latency due to expensive searches and much longer relevant context. In this paper, we propose a parametric knowledge integration method, called **AtlasKV**, a scalable, effective, and general way to augment LLMs with billion-scale knowledge graphs (KGs) (e.g. 1B triples) using very little GPU memory cost (e.g. less than 20GB VRAM). In AtlasKV, we introduce KG2KV and HiKVP to integrate KG triples into LLMs at scale with sub-linear time and memory complexity. It maintains strong knowledge grounding and generalization performance using the LLMs' inherent attention mechanism, and requires no external retrievers, long context priors, or retraining when adapting to new knowledge.



(a) Non-Parametric Methods (b) Traditional Parametric Methods (c) AtlasKV

Figure 1: The simple illustrations of two kinds of popular knowledge augmentation paradigms for LLMs and a new parametric knowledge augmentation paradigm adopted by AtlasKV: (a) Non-parametric methods usually rely on external retrievers and long context prior, which have retriever-limited performance and substantial inference latency. (b) Traditional parametric methods require re-training the model when adapting to new knowledge, which are also expensive. (c) AtlasKV can achieve injecting external knowledge efficiently at scale without need for external retrievers or long context prior with strong generalization ability.

1 INTRODUCTION

Large language models (LLMs) have demonstrated impressive generation abilities in various downstream tasks (Brown et al., 2020; Touvron et al., 2023; Kung et al., 2023; Li et al., 2024; Dagdelen et al., 2024), where their expanding parameter scales enable them to function as comprehensive knowledge stores by encoding factual information directly into their parameters (Petroni et al., 2019; Jiang et al., 2020; Rae et al., 2021; Bubeck et al., 2023; Morris et al., 2025). Retrieval-augmented generation (RAG) (Gao et al., 2023; Fan et al., 2024) is a more cost-efficient solution to enhance the capabilities of LLMs in knowledge-intensive tasks, which may require vast amount of external knowledge, without altering an LLM's parametric representation. RAG methods usually retrieve text chunks (Sarthi et al., 2024; Jimenez Gutierrez et al., 2024) or subgraphs (Edge et al., 2024; Huang et al., 2025) that are relevant to a query from an external textual knowledge base (KB) or knowledge graph (KG), which serve as the context prior for LLMs to generate responses.

Although RAG methods have achieved some success in efficiently augmenting LLMs with external knowledge, they still face some critical limitations. As shown in (a) of Figure 1, these non-parametric

methods heavily rely on external retrieval modules and the textual context prior, which introduce substantial inference latency due to expensive searches (e.g. nearest-neighbor searches) (Khandelwal et al., 2019; He et al., 2021a; Shi et al., 2023) and longer context (Ram et al., 2023; Cao et al., 2025) specially for very large scale knowledge augmentation.

In contrast, parametric approaches (Gururangan et al., 2020; Wang et al., 2024; Cao et al., 2025) can achieve the integration of external knowledge into LLMs without need for external retrievers or long context prior. Traditional knowledge adaptation techniques (Hu et al., 2022; Diao et al., 2023), as shown in (b) of Figure 1, require retraining the model when adapting to a new distribution of knowledge, which significantly limits their application scenarios. A new parametric knowledge augmentation paradigm introduced by KBLaM (Wang et al., 2024) like (c) of Figure 1, well addresses this issue by encoding external knowledge into a series of key-value parametric representations and seamlessly injecting them into the self-attention layers of an LLM, which well preserves the advantages of parametric methods and can also adapt to any new KB in a training-free manner.

Nevertheless, we found that there are two critial challenges of this novel knowledge augmentation paradigm that restrict its real-world applications: (1) **Lack of high quality training data**. It requires query-key-value (Q-K-V) sentences of the external knowledge as the training data. Directly synthesizing Q-K-V training data from unformatted documents with fixed pre-defined schemas suffers from limited query diversity, which could result to poor generalization performance in out-of-distribution (OOD) scenarios. (2) **Poor scalability**. When augmenting LLMs with very large scale external KB or KGs, the computational and memory overhead of this knowledge augmentation paradigm becomes prohibitively high, even with linear time and memory complexity.

To this end, we propose **AtlasKV**, a scalable method that enables end-to-end knowledge augmentation of LLMs with billion-scale KGs (e.g. 1B triples) using very little GPU memory cost (e.g. less than 20GB VRAM) while achieving superior knowledge grounding and generalization performance in OOD scenarios. We achieve this by two innovative designs from both data and algorithm perspectives. Specifically, to address the training data quality issue, we observe that each triple in KGs can be naturally converted into Q-K-V data, which shares very similar structure with the Q-K-V vectors of self-attention networks in LLMs. So we introduce the concept of **KGKV** and propose the **KG2KV** pipeline that naturally converts each KG triple into high-quality Q-K-V data for both training and inference, enabling a better injection of KGs into LLMs. To solve the scalability challenge, we propose a hierarchical key-value pruning (**HiKVP**) algorithm that can dramatically reduce computational and memory overhead while maintaining high knowledge grounding accuracy during inference time.

In summary, we make the following main contributions:

- We propose **AtlasKV**, a scalable method that enables end-to-end augmentation of LLMs with billion-scale KGs (e.g. 1B triples) using very little GPU memory (e.g. less than 20GB VRAM) while achieving superior knowledge grounding performance and strong generalization abilities.
- We introduce **KG2KV** and **HiKVP** as complementary designs to address data and algorithmic challenges respectively: KG2KV naturally transforms KG triples into high-quality Q-K-V data to enhance generalization, while HiKVP enables scalable integration through hierarchical pruning that dramatically reduces computational and memory overhead during inference.
- Extensive experiments and analysis demonstrate the superior effectiveness and scalability of AtlasKV compared to ICL, KBLaM, and RAG methods, with comprehensive ablation studies validating the contribution of each component.

2 Related Work

Non-parametric Knowledge Augmentation Methods For LLMs. The most popular non-parametric knowledge augmentation methods for LLMs is RAG (Lewis et al., 2020; Gao et al., 2023; Fan et al., 2024), which significantly enhances LLMs by incorporating an external retriever that fetches relevant context from external KBs (Zhang et al., 2024a; Sarthi et al., 2024) or KGs (Mavromatis & Karypis, 2024; Edge et al., 2024; Jimenez Gutierrez et al., 2024; Huang et al., 2025). Their retrievers usually heavily rely on either the separately pretrained sentence transformers (Wang et al., 2020a; Izacard et al., 2021; Zhang et al., 2025a), or a well-designed tuning process with the LLM's output as the

feedback signal (Shi et al., 2023; Chang et al., 2025). However, the performance of this knowledge augmentation paradigm could be significantly limited by the capabilities of the retrievers. And the long context retrieved from large KBs or KGs could also introduce substantial inference latency.

Parametric Knowledge Augmentation Methods For LLMs. Parametric knowledge augmentation approaches are much more native to LLMs. Because the inherent memory of LLMs is integrated by pretraining and supervised fine-tuning (Geva et al., 2020; Petroni et al., 2019; Morris et al., 2025), which are also parametic methods. Some early attempts to efficiently integrate new external knowledge into LLMs such as LoRA (Hu et al., 2022) and adapter (He et al., 2021b; Diao et al., 2023) still suffer from retraining the model when integrating new external knowledge into LLMs. MemDec (Cao et al., 2025) provides a domain-specific memory module that enhances various frozen LLMs without parameter modifications. KBLaM (Wang et al., 2024) introduces a new knowledge augmentation paradigm that augments knowledge into the attention layers of LLMs, achieving linear computational complexity while enabling training-free adaptation to new KBs after initial training.

Knowledge Graph Augmentation Methods For LLMs. There are many works augmenting KGs into LLMs in both parametric and non-parametric ways. Except for the graph-based RAG methods mentioned above, some LLM-based KGQA methods like RAR (Shen et al., 2025), KnowGPT (Zhang et al., 2024b), and the work done by Ji et al. (2024) also depend on the training of their retrievers or path aligner to find the most relevant knowledge from KGs as context. KELP (Liu et al., 2024) explores latent semantic matching to improve path-level knowledge selection from KGs. They are still limited by the performance of the retrievers and also face inference latency issues.

3 Backgrounds and Definitions

3.1 Knowledge Graphs

As most of the existing works on graph-based RAG systems did, we use the textual triples (h, r, t) as the basic knowledge unit of KGs in our work, which could be extracted from unstructured text with any existing KG extraction method (Angeli et al., 2015; Huang et al., 2024; Mo et al., 2025). Then the KGs can be defined as $\mathcal{G} = \{(h, r, t) | h, t \in \mathcal{E}, r \in \mathcal{R}\}$, where $\mathcal{E}$ is the set of entities and $\mathcal{R}$ is the set of relations. Note that h, t could be either named entities, or other types of entities, such as concepts, events, etc (Zhang et al., 2020; Tan et al., 2024; Bai et al., 2025). In our work, $\mathcal{G}$ will be integrated as external factual knowledge for LLMs to ground facts and answer questions. And the process of extracting KGs from documents is not the focus of our work.

3.2 Attention Networks

In this section, we will first give the definitions of self-attention layers, which are the key components of the transformer (Vaswani et al., 2017) backbone. Then we will describe the definitions of the rectangular attention in KBLaM (Wang et al., 2024), which is a new knowledge augmentation paradigm adopted by AtlasKV.

Self-Attention Layers. In each attention layer, we input a query $x \in \mathbb{R}^{N \times 1}$ with N token length, which embedding vector can be denoted as $\boldsymbol{x}^{(l)} \in \mathbb{R}^{N \times D}$. D is the embedding dimension of attention layers and $l \in \{1, .., L\}$, where L is the number of attention layers. There are also three attention heads $\mathbf{W}_Q^{(l)}, \mathbf{W}_K^{(l)}, \mathbf{W}_V^{(l)} \in \mathbb{R}^{D \times D}$, which are designed to project each input token into Q-K-V embeddings $\mathbf{q}^{(l)}, \mathbf{k}^{(l)}, \mathbf{v}^{(l)} \in \mathbb{R}^{N \times D}$. Then the output at the l-th layer and n-th token is computed as

$$\mathbf{y}_n^{(l)} = \frac{\sum_{i=1}^{n} \exp(\langle \mathbf{q}_n^{(l)}, \mathbf{k}_i^{(l)} \rangle / \sqrt{D}) \mathbf{v}_i^{(l)}}{\sum_{i=1}^{n} \exp(\langle \mathbf{q}_n^{(l)}, \mathbf{k}_i^{(l)} \rangle / \sqrt{D})}, \tag{1}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product of two vectors. This standard implementation of self-attention can have a time complexity of $\mathcal{O}(N^2 \cdot D)$ and memory complexity of $\mathcal{O}(N \cdot (N + D))$, which could lead to significant computational overheads and time delay as the input length N gets larger.

Rectangular Attention in KBLaM. The KB in KBLaM is a set of key-value pairs, which is denoted as $\mathcal{M} = \{(\boldsymbol{k}^{(l)m}, \boldsymbol{v}^{(l)m})\}_{m=1}^{M}$ at the l-th attention layer, where $\boldsymbol{k}^{(l)m}, \boldsymbol{v}^{(l)m} \in \mathbb{R}^{M \times D_E}$ are the base embedding vectors of the m-th key-value pair. M is the size of the KB and D_E is the output

dimension of the sentence encoder. Then the knowledge augmented output at the l-th attention layer and n-th token is computed as

$$\tilde{\mathbf{y}}_n^{(l)} = \frac{\sum_{m=1}^{M} \exp(\langle \tilde{\mathbf{q}}_n^{(l)}, \tilde{\mathbf{k}}^{(l)m} \rangle / \sqrt{D}) \tilde{\mathbf{v}}^{(l)m} + \sum_{i=1}^{n} \exp(\langle \mathbf{q}_n^{(l)}, \mathbf{k}^{(l)i} \rangle / \sqrt{D}) \mathbf{v}^{(l)i}}{\sum_{m=1}^{M} \exp(\langle \tilde{\mathbf{q}}_n^{(l)}, \tilde{\mathbf{k}}^{(l)m} \rangle / \sqrt{D}) + \sum_{i=1}^{n} \exp(\langle \mathbf{q}_n^{(l)}, \mathbf{k}^{(l)i} \rangle / \sqrt{D})}, \tag{2}$$

where $\tilde{\mathbf{q}}_n^{(l)} = \tilde{\mathbf{W}}_Q^{(l)} \boldsymbol{x}_n^{(l)}$, $\tilde{\mathbf{k}}^{(l)m} = \tilde{\mathbf{W}}_K^{(l)} \boldsymbol{k}^{(l)m}$, $\tilde{\mathbf{v}}^{(l)m} = \tilde{\mathbf{W}}_V^{(l)} \boldsymbol{v}^{(l)m}$ denote the Q-K-V embedding vectors after projection of the KB part. $\tilde{\mathbf{W}}_Q^{(l)} \in \mathbb{R}^{D \times D}, \tilde{\mathbf{W}}_K^{(l)}, \tilde{\mathbf{W}}_V^{(l)} \in \mathbb{R}^{D \times D_E}$ are the specific projection heads for the Q-K-V vectors of the KB. This rectangular attention have a time complexity of $\mathcal{O}\left((M+N) \cdot N \cdot D\right)$ and memory complexity of $\mathcal{O}\left((M+N) \cdot (N+D)\right)$. Its computational overhead would grow linearly with M, which is more efficient than standard self-attention.

However, as the size of KB M scales up heavily, the linearly growing time and memory complexity would still be a critical problem. AtlasKV can further improve the scalability of the KG augmented LLM with $\mathcal{O}\left((C_t \sqrt[3]{M} + N) \cdot N \cdot D\right)$ time complexity and $\mathcal{O}\left((C_m \sqrt[3]{M} + N) \cdot (N+D)\right)$ memory complexity, where C_t and C_m are constants that much smaller than M.

4 METHODOLOGY

Overview of AtlasKV. Augmenting LLMs with super large and complex external knowledge, e.g. general KGs with billions of triples, often struggles with low generalization abilities and unbearable computational and memory overhead (Wang et al., 2024; Zhang et al., 2024b; Jin et al., 2024). To overcome these fundamental challenges, we propose **AtlasKV**, a scalable, effective and general method to integrate massive KGs into LLMs through two key innovations: (1) **KG2KV**, a new KG integration paradigm that naturally converts KG triples into Q-K-V data, enabling LLMs to achieve both enhanced generalization performance and efficient knowledge integration, and (2) **HiKVP**, a hierarchical key-value pruning algorithm that dramatically reduces computational and memory overhead while maintaining high knowledge grounding accuracy during inference time.

4.1 KG TO KV

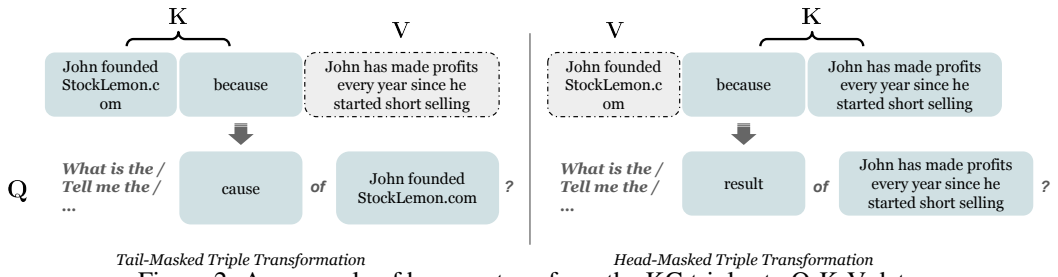


Figure 2: An example of how we transform the KG triples to Q-K-V data.

Building on the observation that every triple in KGs can be naturally decomposed into Q-K-V strings (Verga et al., 2020), which shares very similar structure with the Q-K-V vectors of self-attention networks in LLMs, we introduce the concept of KGKV and employ the KG2KV pipeline to transform each KG triple into Q-K-V strings and their corresponding sentence embedding vectors.

For a given KG triple (h, r, t), we firstly mask its head h or tail entity t (could be named entity, event or concept entity). Then the masked entity can be the value we need in this triple. And then we will rewrite r into a noun words according to the position we masked, which could be considered as the attribute of the entity that is not masked. For example, as shown in Figure 2, if we mask the tail entity in the triple, the relation "*because*" can be directly rewritten into its noun word "*cause*" through LLMs. And the key string of this tail-masked triple can be represented as "*the cause of John founded StockLemon.com*". If we mask the head entity in the triple, we need to rewrite the relation into its reversed noun word "*result*". And the key string of this head-masked triple can be represented as "*the result of John has made profits ...*". In this KG2KV pipline, we consider the masked entity as the value data, and the other entity as well as the relation as the key data, which complete KGKV data. Note that for the training data, we usually select named entities as the key to mask, and select

event entities and relations as the value due to the reasons elaborated in Section 5.3. Then through a sentence encoder, KGKVs can be compressed and encoded into sentence embeddings $\mathbf{k}^m, \mathbf{v}^m$ to integrate to the attention layers of LLMs.

We also need the query sentence of each triple to serve as the training data. The query string can be obtained by adding various questioning prefix to the key strings. For example, the questioning prefix can be "*What is ...*", "*Tell me ...*", or "*Provide details on ..*". This design can ensure the model would not overfit to a specific type of questioning way.

Compared to directly synthesizing Q-K-V data from documents with limited human-defined schemas, the massive relations from KGs can guarantee the diversity of the enquiry attributes in the constructed training data with KG2KV method. Besides, due to the only strings we need input into the LLMs in KG2KV is the masked position and the relation, KG2KV needs fewer token cost than directly synthetic method. We also provide the prompt template in Appendix H. As shown in Table 1, KG2KV holds a significantly higer diversity ratio (number of unique enquiry attributes divided by the total number of triples) of 7.864% and a lower average token cost of 165.7 than the synthetic method. We also provide some samples of the training data constructed by synthetic and KG2KV methods respectively in Appendix G.1.

Table 1: Comparison of the data diversity ratio and average token cost between KG2KV and synthetic method.

Method	Diversity Ratio ↑	Avg. Token Cost ↓
Synthetic	0.003%	349.9
KG2KV	**7.864%**	**165.7**

4.2 Augmenting LLM with KGKVs

In this section, we will introduce how AtlasKV integrates the KGKVs constructed in the previous section into the LLMs with the help of hierarchical key-value pruning (HiKVP), which makes it more scalable than other methods.

Hierarchical Clustering on KGKVs. Inspired by various works (Sarthi et al., 2024; Zhang et al., 2024a; Huang et al., 2025; Zhang et al., 2025b) that achieve some success by organizing hierarchical knowledge structure on textual chunks, which is an intuitive way to organize the world's knowledge, we employ the hierarchical clustering to cluster the keys of KGKVs into a hierarchical structure. This design aims to share the computational and memory burden during inference time on each layer of the hierarchical knowledge keys. Specifically, as previous works (Sarthi et al., 2024; Huang et al., 2025) did, we first employ Uniform Manifold Approximation and Projection (UMAP) to reduce the dimension of the knowledge keys, and then employ Gaussian Mixture Models (GMMs) to cluster the knowledge keys into a hierarchical structure. Each key vector in a higer layer is the pooling of the key vectors in the lower layer. In AtlasKV, we set the number of layers to be 3, which can also be larger according to the actual situation. We select 3 layers because that is the minimum number of layers to include all of the definitions we need in AtlasKV. To share the computational and memory burden equally, we set the size of clusters in each layer to be the same, which is $S = \left\lceil \sqrt[3]{M} \right\rceil$. Then we can have the base embeddings of three layer knowledge keys $\boldsymbol{k}_L^m \in \mathbb{R}^{M_L \times D_E}$, $\boldsymbol{k}_I^m \in \mathbb{R}^{M_I \times D_E}$, $\boldsymbol{k}_R^m \in \mathbb{R}^{M_R \times D_E}$, where $M_L = M, M_I = \left\lceil M^{\frac{2}{3}} \right\rceil, M_R = \left\lceil \left\lceil M^{\frac{2}{3}} \right\rceil M^{-\frac{1}{3}} \right\rceil$. And D_E is the embedding dimension of the sentence encoder.

Knowledge Augmentation. During the tuning process of AtlasKV, we do not need to prune the KGKV pairs. Because due to the generalization capability of AtlasKV as verified in Section 5, we do not need such large scale KGKVs in the tuning process. And we use an equivalent attention method to replace the rectangular attention in KBLaM for knowledge augmentation. At the l-th attention layer and n-th token, it is computed as

$$\tilde{\mathbf{y}}_n^{(l)} = \lambda_{kg} \cdot \text{Softmax}\left(\text{logits}_{kg_L}\right) \cdot \tilde{\mathbf{v}}^{(l)} + \lambda_{seq} \cdot \text{Softmax}\left(\text{logits}_{seq}\right) \cdot \mathbf{v}^{(l)}, \tag{3}$$

where logits_{kg_L} is the KG part of the attention output, logits_{seq} is the sequence part of the attention output. For the weights λ_{kg} and λ_{seq} of these two softmax results, we have

$$\lambda_{kg} = \frac{\sum_{i=1}^{M} \exp(\text{logits}_{kg_L}^i)}{\sum_{i=1}^{M} \exp(\text{logits}_{kg_L}^i) + \sum_{i=1}^{n} \exp(\text{logits}_{seq}^i)}, \tag{4}$$

$$\lambda_{seq} = \frac{\sum_{i=1}^{n} \exp(\text{logits}_{seq}^i)}{\sum_{i=1}^{M} \exp(\text{logits}_{kg_L}^i) + \sum_{i=1}^{n} \exp(\text{logits}_{seq}^i)}. \tag{5}$$

And we also have

$$\text{logits}^i_{kg_L} = \langle \tilde{\mathbf{q}}^{(l)}_n, \tilde{\mathbf{k}}^{(l)i}_L \rangle / \sqrt{D}, \text{logits}^i_{seq} = \langle \mathbf{q}^{(l)}_n, \mathbf{k}^{(l)i} \rangle / \sqrt{D}, \tag{6}$$

where $\tilde{\mathbf{k}}^{(l)i}_L = \tilde{\mathbf{W}}^{(l)}_K \boldsymbol{k}^i_L$ and $\mathbf{k}^{(l)i} = \mathbf{W}^{(l)}_K \boldsymbol{k}^i$. The only learnable variables are the KG-specific query heads $\tilde{\mathbf{W}}_Q$ and KG projection heads $\tilde{\mathbf{W}}_K, \tilde{\mathbf{W}}_V$. It is also very intuitive that the attention output at both the prefilling process and each decoding step is a weighted combination of the KG part and the sequence part values. And the weights are determined dynamically according to the logits of the KG and sequence part. The equivalence to the rectangular attention can be proven in Appendix C.

Hierarchical Key-Value Pruning. During inference time, usually there are only a few relevant KG

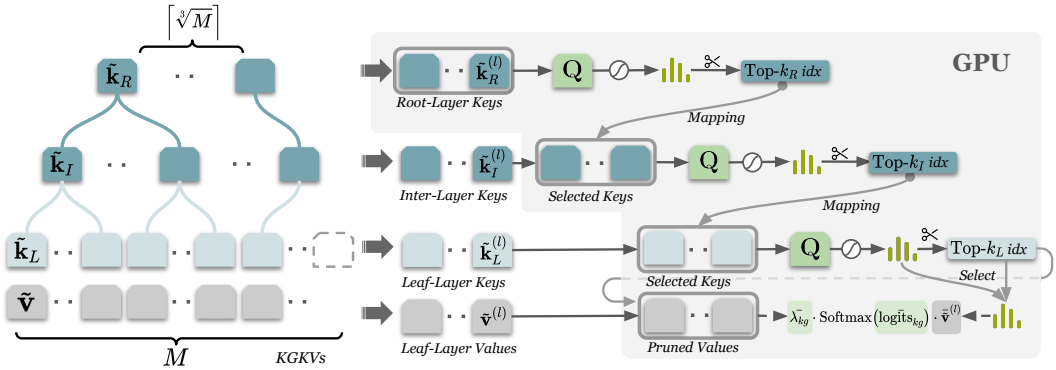


Figure 3: An overview of hierarchical key-value pruning (HiKVP) with three layers of knowledge keys at the l-th attention layer. The gray background indicate that the part is stored and computed in the GPU memory.

values needed for the query. And the injection of irrelevant knowledge would also introduce noise. So we use a hierarchical key-value pruning (HiKVP) pipeline as shown in Figure 3 to efficiently and scalably find the most relevant KGKVs for the query. And the attention will be computed as

$$\tilde{\mathbf{y}}^{(l)}_n = \bar{\lambda}_{kg} \cdot \text{Softmax}\left(\bar{\text{logits}}_{kg}\right) \cdot \bar{\tilde{\mathbf{v}}}^{(l)} + \lambda_{seq} \cdot \text{Softmax}\left(\text{logits}_{seq}\right) \cdot \mathbf{v}^{(l)}, \tag{7}$$

where the $\bar{\text{logits}}_{kg}$ and $\bar{\tilde{\mathbf{v}}}^{(l)}$ are the pruned KG logits and values in the leaf-layer. $\bar{\lambda}_{kg}$ is the weight based on the pruned KG logits. They can be obtained by the following steps.

Step 1. Initially, only the root-layer projected key vectors $\tilde{\mathbf{k}}^{(l)}_R$ are uploaded in the GPU memory, while other layer key and value vectors ($\tilde{\mathbf{k}}^{(l)}_I$, $\tilde{\mathbf{k}}^{(l)}_L$, and $\tilde{\mathbf{v}}^{(l)}$) remain in the CPU memory. We first compute attention weights between the query and root-layer keys:

$$\text{logits}_{kg_R} = \langle \tilde{\mathbf{q}}^{(l)}_n, \tilde{\mathbf{k}}^{(l)}_R \rangle / \sqrt{D}. \tag{8}$$

After calculating the softmax of it, we prune root keys by selecting the keys with top-k_R highest scores and use a mapping to obtain the included inter-layer keys $\bar{\tilde{\mathbf{k}}}^{(l)}_I \in \mathbb{R}^{(k_R S) \times D}$. And the root-layer keys will be offloaded back to the CPU memory.

Step 2. We conduct the similar process to the step 1, but with the selected inter-layer keys. We upload the selected inter-layer keys to the GPU memory, and compute the attention weights:

$$\bar{\text{logits}}_{kg_I} = \langle \tilde{\mathbf{q}}^{(l)}_n, \bar{\tilde{\mathbf{k}}}^{(l)}_I \rangle / \sqrt{D}. \tag{9}$$

Then we can prune the selected inter-layer keys and obtain the selected leaf-layer keys $\bar{\tilde{\mathbf{k}}}^{(l)}_L \in \mathbb{R}^{(k_I S) \times D}$ in the same way. And the selected inter-layer keys will be offloaded to the CPU memory.

Step 3. Finally, we upload the selected leaf-layer keys from the CPU to GPU memory. Then we compute the attention weights after the softmax calculation and prune the leaf-layer keys by directly selecting the corresponding logits with top-k_L highest softmax scores:

$$\bar{\text{logits}}_{kg} = \text{TopK_logits}(\text{Softmax}\left(\bar{\text{logits}}_{kg_L}\right), k_L) \in \mathbb{R}^{k_L \times 1}, \bar{\text{logits}}_{kg_L} = \langle \tilde{\mathbf{q}}^{(l)}_n, \bar{\tilde{\mathbf{k}}}^{(l)}_L \rangle / \sqrt{D}. \tag{10}$$

And through the mapping indices, we can also obtain the pruned values $\bar{\tilde{\mathbf{v}}}^{(l)} \in \mathbb{R}^{k_L \times D}$ and upload them to the GPU memory. Then $\bar{\text{logits}}_{kg}$, $\bar{\tilde{\mathbf{v}}}^{(l)}$ and $\bar{\lambda}_{kg}$ can be obtained and the attention output of Equation 7 during the inference time can be finally computed.

All these steps of HiKVP can be done in $\mathcal{O}\left((C_t \sqrt[3]{M} + N) \cdot N \cdot D\right)$ time complexity and $\mathcal{O}\left((C_m \sqrt[3]{M} + N) \cdot (N + D)\right)$ memory complexity, respectively, where C_t and C_m are constants that are much smaller than M. We also provide the detailed derivation in Appendix D.

5 Experiments

In this section, we report the performance of AtlasKV from GPU memory cost, knowledge grounding accuracy and the relevance of the generation results perspectives. And we also compare the performance of AtlasKV with other knowledge integration baseline methods to demonstrate the superiority of AtlasKV. We also report the training and evaluation details of AtlasKV in Appendix A.1 and Appendix A.2.

5.1 Experimental Settings

Baselines. Following the settings in KBLaM (Wang et al., 2024), we compare AtlasKV with both non-parametric and parametric methods. For non-parametric methods, we include **in-context learning (ICL)**, which is the basic knowledge augmentation paradigm used in RAG methods. For parametric methods, we include **KBLaM**, which is an advanced parametric knowledge augmentation paradigm. We also include **zero-shot learning** to provide some boundaries of our experimental results.

Training Datasets. In AtlasKV, we construct the Q-K-V training dataset **ATLAS-Wiki-QKV** with **ATLAS-Wiki**, which is one of ATLAS (Bai et al., 2025) family KGs with 900+ million nodes and 5.9 billion edges containing both event and named entities. Because it offer sufficiently large KGs that enable us to train the model and comprehensively assess the performance of AtlasKV and other baseline methods across various KG sizes. And we also use the **Synthetic** training dataset used in KBLaM (Wang et al., 2024) to compare with.

Evaluation Datasets. To comprehensively assess the performance of AtlasKV and other baseline methods in a scenario closer to the real world, we test all methods in the OOD settings. We not only include the **Enron** dataset (Klimt & Yang, 2004), which is an OOD dataset used in KBLaM (Wang et al., 2024), but also introduce the **ATLAS-CC-QKV** dataset (Bai et al., 2025) and **ATLAS-Pes2o-QKV** dataset (Bai et al., 2025), which are also constructed from **ATLAS-CC** and **ATLAS-Pes2o** in ATLAS (Bai et al., 2025) with the KG2KV method, respectively, to evaluate the performance of different methods in a more comprehensive way. Note that ATLAS-CC-QKV and ATLAS-Pes2o-QKV are much harder than Enron dataset because they are constructed from more complex KGs and include much more unique enquiry attributes that are closer to the real world scenarios. With these OOD datasets, we can better evaluate the generalization capabilities of various methods.

5.2 Experimental Results

Method	Time Complexity	Memory Complexity
ICL	$\mathcal{O}\left((MT + N)^2 \cdot D\right)$	$\mathcal{O}\left((MT + N) \cdot (MT + N + D)\right)$
RAG	$\mathcal{O}\left(M + RT + (RT + N)^2\right) \cdot D$	$\mathcal{O}\left((RT + N) \cdot (RT + N + D)\right)$
KBLaM	$\mathcal{O}\left((M + N) \cdot N \cdot D\right)$	$\mathcal{O}\left((M + N) \cdot (N + D)\right)$
AtlasKV	$\mathcal{O}\left((C_t \sqrt[3]{M} + N) \cdot N \cdot D\right)$	$\mathcal{O}\left((C_m \sqrt[3]{M} + N) \cdot (N + D)\right)$

Table 2: Comparison of the time and memory complexity of AtlasKV, KBLaM, RAG, and ICL methods, where the parts marked in teal color represent they could be very large.

AtlasKV is more scalable with HiKVP. To verify the scalability of AtlasKV with HiKVP, we compare the GPU memory usage at inference time of AtlasKV and other methods across a wide range of KG sizes from 1 to 1B triples. As shown in Figure 4, the colored areas represent how much VRAM is saved compared with the other method. It demonstrates that with the help of HiKVP, AtlasKV can save a huge amount of VRAM compared with ICL as well as KBLaM and achieve a much lower GPU memory cost. With the increasing of KG scale, the VRAM usage of AtlasKV even just a little bit higher than the zero-shot generation. And in AtlaskV, less than 20GB VRAM is required to augment LLMs with 1B triples. However, in KBLaM, over 40GB VRAM is required to deal with even 100K triples. The key reason why AtlasKV can achieve this is that HiKVP significantly reduces the time

and memory complexity of KBLaM from linear to sub-linear, as demonstrated in Table 2, where T denotes the average token length of the triples. In ICL-based RAG methods, when the size of relevant triples R scales up, the inference latency and VRAM usage caused by long context prior dependence would grow heavily. And their performance will also influenced by the lost-in-the-middle dilemma (Liu et al., 2023), which would not exist in AtlasKV.

Method	Steps	10^1 Triples ACC@1	10^1 Triples ACC@5	10^2 Triples ACC@1	10^2 Triples ACC@5	10^3 Triples ACC@1	10^3 Triples ACC@5	10^4 Triples ACC@1	10^4 Triples ACC@5
Eval on Enron									
KBLaM	3e3	**100.0**	**100.0**	50.9	76.4	29.1	56.4	9.1	20.0
	2e4	90.0	**100.0**	50.9	83.6	25.5	47.3	7.3	23.6
AtlasKV (128-64-16)	3e3	**100.0** (+0.0)	**100.0** (+0.0)	67.3 (+16.4)	90.9 (+7.3)	41.8 (+12.7)	50.9	21.8 (+12.7)	32.7 (+12.7)
AtlasKV w/o HiKVP	3e3	**100.0** (+0.0)	**100.0** (+0.0)	**76.4** (+25.5)	**92.7** (+9.1)	**56.4** (+27.3)	**80.0** (+23.6)	**27.3** (+18.2)	**47.3** (+27.3)
Eval on ATLAS-Pes2o-QA									
KBLaM	3e3	40.0	80.0	16.4	45.5	5.5	14.5	0.0	3.6
	2e4	50.0	80.0	25.5	52.7	3.6	14.5	0.0	5.5
AtlasKV (128-64-16)	3e3	90.0 (+40.0)	100.0 (+20.0)	87.3 (+61.8)	92.7 (+40.0)	52.7 (+47.2)	70.9 (+56.4)	16.4 (+16.4)	49.0 (+43.5)
AtlasKV w/o HiKVP	3e3	**100.0** (+50.0)	**100.0** (+20.0)	**92.7** (+67.2)	**100.0** (+47.3)	**72.7** (+67.2)	**90.9** (+76.4)	**47.3** (+47.3)	**67.2** (+61.7)
Eval on ATLAS-CC-QA									
KBLaM	3e3	60.0	90.0	21.8	38.2	12.7	23.6	3.6	10.9
	2e4	50.0	**100.0**	23.6	56.4	10.9	21.8	3.6	10.9
AtlasKV (128-64-16)	3e3	**100.0** (+0.0)	**100.0** (+0.0)	89.1 (+65.5)	90.9 (+34.5)	61.8 (+49.1)	74.5 (+50.9)	40.0 (+36.4)	54.5 (+43.6)
AtlasKV w/o HiKVP	3e3	**100.0** (+0.0)	**100.0** (+0.0)	**96.4** (+72.8)	**100.0** (+43.6)	**83.6** (+70.9)	**96.4** (+72.8)	**61.8** (+58.2)	**81.8** (+70.9)

Table 3: The knowledge grounding performance of AtlasKV against KBLaM with all-MiniLM-L6-v2 as the sentence encoder on three OOD evaluation datasets across various tuning steps and KG sizes. We defaultly set the top-k in HiKVP to 128, 64, and 16 for the k_R, k_I, k_L respectively.

AtlasKV is more accurate and generalizable with KGKVs. Not only can AtlasKV save a huge amount of inference cost with strong scalability, but it can also maintain a higher knowledge grounding performance with strong generalization ability. To quantitatively assess the knowledge grounding performance of AtlasKV, we extract the averaged-over-heads KG part post-softmax attention scores at the 15th layer due to the reason described in Appendix A.2, and then we can obtain the Top-1 and Top-5 accuracy of the knowledge grounding performance. As shown in Table 3, across all three OOD datasets and a wide range of KG sizes, AtlasKV achieves significantly higher Top-1 and Top-5 accuracy than KBLaM with the KGKVs as the training data. Especially in ATLAS-Pes2o-QKV and ATLAS-CC-QKV datasets, which are much harder due to their complex and diverse enquiry attributes, KBLaM performs very bad because there are too limited enquiry attributes in Synthetic training data to make it more generalizable. However, only 20K KGKV samples as the training data are needed to make AtalsKV much more accurate and generalizable. It also suggests KGKVs can make the training process more efficient with only 3K steps, compared with the 20K training steps reported in KBLaM. We also experiment with different top-k settings in HiKVP in Appendix B.2. Another interesting observation is that, even with HiKVP, there is not a big performance drop of AtlasKV and it still performs better than KBLaM. This is mainly because the specific heads trained in AtlasKV have the capabilities to conduct fuzzy retrieval at different layers of semantic granularity. Besides, we observe the training dynamics of AtlasKV in Appendix E, which is that **from a specific training step, the model regularly start learning to retrieve relevant knowledge from the external KG triples, instead of brute force over-fitting**. We also report the results with a larger model as the sentence encoder in Appendix B.1.

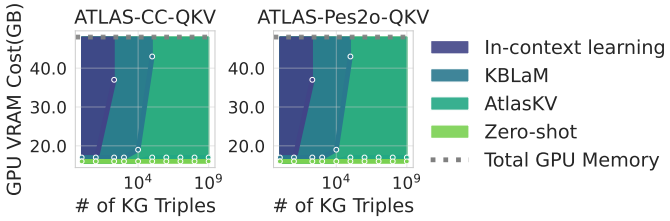


Figure 4: GPU memory usage comparison of AtlasKV and other methods across various KG sizes from 1 to 1B triples.

We also use GPT-4o (Hurst et al., 2024) to score the relevance between the ground truth and generated answers. As shown in Figure 5, AtlasKV achieves significantly higher GPTScores than KBLaM. Although ICL can generate a very accurate result with over 0.9 scores, it is super time-consuming to put all external knowledge into the context of LLMs. Expecially when there are more than 100 triples in a KG, over 48GB VRAM is required and can not be run on the limited GPU memory. And KBLaM also performs poorly on the two difficult datasets, which is also due to the reasons we explained above. **Remarkably, despite having only limited training samples with enquiry**

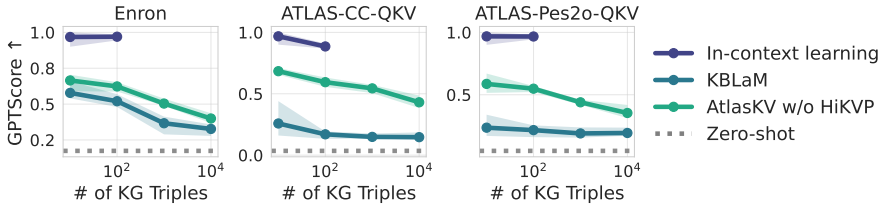


Figure 5: Scored by GPT-4o between 0 and 1, the shaded area exhibits the standard error over 5 random seeds. The score of each random seed is also the average of 5 generation results.

attributes similar to Enron in ATLAS-Wiki-QKV, AtlasKV still outperforms KBLaM on both knowledge grounding accuracy and answer relevance metrics, even though KBLaM's training data contains exactly the same enquiry attributes as Enron. This is mainly due the the diversity of the enquiry attributes in ATLAS-Wiki-QKV, which is constructed by KG2KV module. We have compared that with fully synthetic method in Table 1 and it makes AtlasKV have the capability to be generalized to more unseen enquiry attributes in complex scenarios. We also provide some samples of ATLAS-Wiki-QKV and Synthetic dataset in Appendix G.1.

5.3 Ablation Study

Method	Steps	10^1 Triples		10^2 Triples		10^3 Triples		10^4 Triples	
		ACC@1	ACC@5	ACC@1	ACC@5	ACC@1	ACC@5	ACC@1	ACC@5
Eval on ATLAS-Pes2o-QA									
AtlasKV w/o HiKVP	3e3	**100.0**	**100.0**	**92.7**	**100.0**	**72.7**	**90.9**	**47.3**	**67.2**
AtlasKV w/o HiKVP & Event	3e3	90.0	**100.0**	80.0	89.1	34.5	63.6	9.1	36.4
AtlasKV w/o HiKVP & Entity	3e3	**100.0**	**100.0**	49.0	67.3	20.0	30.9	3.6	5.5
Eval on Enron									
AtlasKV w/o HiKVP	3e3	**100.0**	**100.0**	**76.4**	**92.7**	**56.4**	**80.0**	**27.3**	**47.3**
AtlasKV w/o HiKVP & Event	3e3	80.0	**100.0**	73.6	84.5	48.0	66.0	10.9	38.2
AtlasKV w/o HiKVP & Entity	3e3	40.0	**100.0**	40.0	74.5	16.4	27.3	1.8	9.1

Table 4: The knowledge grounding performance of different variants of AtlasKV with all-MiniLM-L6-v2 as the sentence encoder on three OOD evaluation datasets across various tuning steps and KG sizes.

Cooperating named and event entities together in KG2KV process helps with the model's learning. To analyze the reasonability of the design described in Section 4.1, we conduct experiments on the variant with only KG2KV component, which are denoted as "AtalsKV w/o HiKVP". Because we need focus on the ablations of training data and avoid the influence of pruning process. We compare the training data containing both named and event entities with the variants containing only named or event entities, which are denoted as "AtalsKV w/o HiKVP & Event" and "AtalsKV w/o HiKVP & Entity". Here is the analysis based on the results in Table 4: (1) Without any one kind of entities, there will be a drop of the knowledge grounding performance. (2) Especially when there are only event entities, due to the high complex semantics in both key and value strings, it becomes very hard for the specific heads to learn from scratch, which leads to a huge performance drop. (3) When only named entities are employed in KG2KV, the performance drop is smaller because the key and value strings of them are shorter and the semantics are simpler, which makes it easier for the specific heads to learn from scratch. But the worse performance with only named entities suggests that we still need some complex semantics in event entities to help the model to learn better.

6 Conclusion

In this paper, we presented AtlasKV, a scalable, effective, and general framework to augment LLMs with billion-scale knowledge graphs under very low GPU memory budgets. Compared with non-parametric methods, AtlasKV requires no external retrievers and does not depend on long context prior, which could lead to substantial inference latency. Compared with traditional parametric methods, AtlasKV can be adapted to new knowledge in a training-free manner. We achieve that by (1) KG2KV which naturally converts KG triples into Q-K-V data, enabling LLMs to achieve both enhanced generalization performance and efficient knowledge integration, and (2) HiKVP which conducts hierarchical key-value pruning to dramatically reduces computational and memory costs while maintaining high performance during inference time.

7 ETHICS STATEMENT

We affirm adherence to the ICLR Code of Ethics. This work develops AtlasKV does not involve human subjects or interventions. We use publicly available datasets (e.g., ATLAS family KGs, Synthetic, and Enron datasets) under their licenses. We construct Q-K-V data via KG2KV without collecting new personal data or attempting deanonymization. We also comply with model or API providers' terms (e.g., LLaMA3.1-8B-Instruct, GPT-4o, and GPT-4o-mini) without uploading proprietary or sensitive information. No undisclosed conflicts of interest exist. All experiments were performed on a single 48GB GPU, and we encourage energy-efficient configurations.

8 REPRODUCIBILITY STATEMENT

LLMs for Training and Evaluation. Three LLMs are used in our AtlasKV experiments: LLaMA3.1-8B-Instruct, GPT-4o, and GPT-4o-mini. We use LLaMA3.1-8B-Instruct as the backbone of AtlasKV and it can be obtained at Hugging Face. GPT-4o and GPT-4o-mini are used to score the generated answers and rewrite the relations in KG2KV process, respectively. They can be accessed via OpenAI API calls.

Training and Evaluation Details. We provide comprehensive descriptions about our training and evaluation settings in Appendix A, including the hyper-parameter settings and detailed processing steps. All of the datasets we used in our work can be obtained through public resources (Bai et al., 2025; Wang et al., 2024; Klimt & Yang, 2004). We will also release our source code soon after the submission.

REFERENCES

Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D Manning. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 344–354, 2015.

Jiaxin Bai, Wei Fan, Qi Hu, Qing Zong, Chunyang Li, Hong Ting Tsang, Hongyu Luo, Yauwai Yim, Haoyu Huang, Xiao Zhou, et al. Autoschemakg: Autonomous knowledge graph construction through dynamic schema induction from web-scale corpora. *arXiv preprint arXiv:2505.23628*, 2025.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Jiaqi Cao, Jiarui Wang, Rubin Wei, Qipeng Guo, Kai Chen, Bowen Zhou, and Zhouhan Lin. Memory decoder: A pretrained, plug-and-play memory for large language models. *arXiv preprint arXiv:2508.09874*, 2025.

Ge Chang, Jinbo Su, Jiacheng Liu, Pengfei Yang, Yuhao Shang, Huiwen Zheng, Hongli Ma, Yan Liang, Yuanchun Li, and Yunxin Liu. Grail: Learning to interact with large knowledge graphs for retrieval augmented reasoning. *arXiv preprint arXiv:2508.05498*, 2025.

John Dagdelen, Alexander Dunn, Sanghoon Lee, Nicholas Walker, Andrew S Rosen, Gerbrand Ceder, Kristin A Persson, and Anubhav Jain. Structured information extraction from scientific text with large language models. *Nature communications*, 15(1):1418, 2024.

Shizhe Diao, Tianyang Xu, Ruijia Xu, Jiawei Wang, and Tong Zhang. Mixture-of-domain-adapters: Decoupling and injecting domain knowledge to pre-trained language models memories. *arXiv preprint arXiv:2306.05406*, 2023.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. Ragas: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pp. 150–158, 2024.

Wenqi Fan, Yujuan Ding, Liangbo Ning, Shijie Wang, Hengyun Li, Dawei Yin, Tat-Seng Chua, and Qing Li. A survey on rag meeting llms: Towards retrieval-augmented large language models. In *Proceedings of the 30th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 6491–6501, 2024.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1), 2023.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. *arXiv preprint arXiv:2012.14913*, 2020.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. Lightrag: Simple and fast retrieval-augmented generation. *arXiv preprint arXiv:2410.05779*, 2024.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. Don't stop pretraining: Adapt language models to domains and tasks. *arXiv preprint arXiv:2004.10964*, 2020.

Junxian He, Graham Neubig, and Taylor Berg-Kirkpatrick. Efficient nearest neighbor language models. *arXiv preprint arXiv:2109.04212*, 2021a.

Ruidan He, Linlin Liu, Hai Ye, Qingyu Tan, Bosheng Ding, Liying Cheng, Jia-Wei Low, Lidong Bing, and Luo Si. On the effectiveness of adapter-based tuning for pretrained language model adaptation. *arXiv preprint arXiv:2106.03164*, 2021b.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

Haoyu Huang, Chong Chen, Zeang Sheng, Yang Li, and Wentao Zhang. Can llms be good graph judge for knowledge graph construction? *arXiv preprint arXiv:2411.17388*, 2024.

Haoyu Huang, Yongfeng Huang, Junjie Yang, Zhenyu Pan, Yongqiang Chen, Kaili Ma, Hongzhi Chen, and James Cheng. Retrieval-augmented generation with hierarchical knowledge. *arXiv preprint arXiv:2503.10150*, 2025.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning, 2021. URL https://arxiv.org/abs/2112.09118.

Yixin Ji, Kaixin Wu, Juntao Li, Wei Chen, Mingjie Zhong, Xu Jia, and Min Zhang. Retrieval and reasoning on kgs: Integrate knowledge graphs into large language models for complex question answering. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 7598–7610, 2024.

Zhengbao Jiang, Frank F Xu, Jun Araki, and Graham Neubig. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438, 2020.

Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in Neural Information Processing Systems*, 37:59532–59569, 2024.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. Ragcache: Efficient knowledge caching for retrieval-augmented generation. *arXiv preprint arXiv:2404.12457*, 2024.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. *arXiv preprint arXiv:1911.00172*, 2019.

Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *European conference on machine learning*, pp. 217–226. Springer, 2004.

Tiffany H Kung, Morgan Cheatham, Arielle Medenilla, Czarina Sillos, Lorie De Leon, Camille Elepaño, Maria Madriaga, Rimel Aggabao, Giezel Diaz-Candido, James Maningo, et al. Performance of chatgpt on usmle: potential for ai-assisted medical education using large language models. *PLoS digital health*, 2(2):e0000198, 2023.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Junyi Li, Tianyi Tang, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. Pre-trained language models for text generation: A survey. *ACM Computing Surveys*, 56(9):1–39, 2024.

Haochen Liu, Song Wang, Yaochen Zhu, Yushun Dong, and Jundong Li. Knowledge graph-enhanced large language models via path selection. *arXiv preprint arXiv:2406.13862*, 2024.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Costas Mavromatis and George Karypis. Gnn-rag: Graph neural retrieval for large language model reasoning. *arXiv preprint arXiv:2405.20139*, 2024.

Belinda Mo, Kyssen Yu, Joshua Kazdan, Proud Mpala, Lisa Yu, Chris Cundy, Charilaos Kanatsoulis, and Sanmi Koyejo. Kggen: Extracting knowledge graphs from plain text with language models. *arXiv preprint arXiv:2502.09956*, 2025.

John X Morris, Chawin Sitawarin, Chuan Guo, Narine Kokhlikyan, G Edward Suh, Alexander M Rush, Kamalika Chaudhuri, and Saeed Mahloujifar. How much do language models memorize? *arXiv preprint arXiv:2505.24832*, 2025.

Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H Miller, and Sebastian Riedel. Language models as knowledge bases? *arXiv preprint arXiv:1909.01066*, 2019.

Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331, 2023.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. Raptor: Recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations*, 2024.

Xiangqing Shen, Fanfan Wang, and Rui Xia. Reason-align-respond: Aligning llm reasoning with knowledge graphs for kgqa. *arXiv preprint arXiv:2505.20971*, 2025.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. Replug: Retrieval-augmented black-box language models. *arXiv preprint arXiv:2301.12652*, 2023.

Xingwei Tan, Yuxiang Zhou, Gabriele Pergola, and Yulan He. Set-aligning framework for auto-regressive event temporal graph generation. *arXiv preprint arXiv:2404.01532*, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Pat Verga, Haitian Sun, Livio Baldini Soares, and William W Cohen. Facts as experts: Adaptable and interpretable neural memory over symbolic knowledge. *arXiv preprint arXiv:2007.00849*, 2020.

Wenhui Wang, Hangbo Bao, Shaohan Huang, Li Dong, and Furu Wei. Minilmv2: Multi-head self-attention relation distillation for compressing pretrained transformers. *arXiv preprint arXiv:2012.15828*, 2020a.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788, 2020b.

Xi Wang, Taketomo Isazawa, Liana Mikaelyan, and James Hensman. Kblam: Knowledge base augmented language model. *arXiv preprint arXiv:2410.10450*, 2024.

Hongming Zhang, Xin Liu, Haojie Pan, Yangqiu Song, and Cane Wing-Ki Leung. Aser: A large-scale eventuality knowledge graph. In *Proceedings of the web conference 2020*, pp. 201–211, 2020.

Nan Zhang, Prafulla Kumar Choubey, Alexander Fabbri, Gabriel Bernadett-Shapiro, Rui Zhang, Prasenjit Mitra, Caiming Xiong, and Chien-Sheng Wu. Sirerag: Indexing similar and related information for multihop reasoning. *arXiv preprint arXiv:2412.06206*, 2024a.

Qinggang Zhang, Junnan Dong, Hao Chen, Daochen Zha, Zailiang Yu, and Xiao Huang. Knowgpt: Knowledge graph based prompting for large language models. *Advances in Neural Information Processing Systems*, 37:6052–6080, 2024b.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025a.

Yaoze Zhang, Rong Wu, Pinlong Cai, Xiaoman Wang, Guohang Yan, Song Mao, Ding Wang, and Botian Shi. Leanrag: Knowledge-graph-based generation with semantic aggregation and hierarchical retrieval. *arXiv preprint arXiv:2508.10391*, 2025b.

Appendix of AtlasKV

CONTENTS

A Training and Evaluation Details of AtlasKV **15**
A.1 Training Settings 15
A.2 Evaluation Settings 15

B Extended Experiments **16**
B.1 Knowledge Grounding with Different Encoders 16
B.2 Influence of Various Top-K in HiKVP 16

C Derivation of the attention mechanism in AtlasKV **17**

D Derivation of the time and memory complexity of HiKVP **17**

E Training Dynamics in AtlasKV **18**

F Where is the Trade-Off? **19**

G Case Study **19**
G.1 Differences between Synthetic KB and KGKVs 19
G.2 Sample Q&A 20

H Prompt Template **21**

I The Usage of LLMs **21**

A TRAINING AND EVALUATION DETAILS OF ATLASKV

For the reproducibility of our work, we provide the training and evaluation details of AtlasKV in this section. All training and evaluation experiments are conducted in a single 48GB GPU under bfloat16 precision.

A.1 TRAINING SETTINGS

We use the same training settings and methods to construct training samples as in KBLaM (Wang et al., 2024), where we also use the instruction fine-tuned version of LLaMA3.1-8B (Dubey et al., 2024), which is represented as LLaMA3.1-8B-Instruct. As an essential part in AtalsKV, any sentence encoder can be employed in our method to compute the base key and value embeddings. We conduct experiments with open-source all-MiniLM-L6-v2 (Wang et al., 2020b) ($D_E = 384$) and closed-source text-embedding-3-large ($D_E = 3072$) through API respectively to serve as the sentence encoders.

To initialize the parameters we need to train in AtlasKV, the KG-specific query heads $\tilde{\mathbf{W}}_Q^{(l)}$ are initialized from $\mathbf{W}_Q^{(l)}$ at each layer and the KG projection heads $\tilde{\mathbf{W}}_K, \tilde{\mathbf{W}}_V$ are initialized randomly. We sample and select only 20K triples from ATLAS-Wiki, which contains 1.492B triples, to construct the Q-K-V training dataset. We use AdamW (Loshchilov & Hutter, 2017) as the optimizer with a step size of 1×10^{-3} and a cosine learning rate decay to 1×10^{-5} for 3K iterations. Each iteration consists a batch size of 10. And the KG sizes at each iteration will increase 4 every 100 iterations. The reason why we only need a small number of training steps (KG sizes for training) is that we found AtlasKV also exhibits some generalization capabilities across various KG sizes as verified in Section 5.2. For example, within 3K iterations, even though the maximum size of KG that is used to train the AtlasKV is only 120. However, it can already perform well with larger scale of KGs. And we integrate the KGKVs into the LLM's attention every 3 layers for efficient training and inference.

Note that all of the base embeddings of the KGKVs are computed offline. So during both of the training and inferencing processes, we only need to load them from hard disk and project some of them into the embedding space of LLMs.

A.2 EVALUATION SETTINGS

In the knowledge grounding experiments, to verify AtlasKV can exhibit superior knowledge grounding accuracy even with small sentence encoders, we employ a lightweight open-source sentence transformer (Reimers & Gurevych, 2019) all-MiniLM-L6-v2 (Wang et al., 2020b) [1] to serve as the sentence encoder here. In the generation relevance experiments, we need stronger sentence encoder to let the value embeddings in KGKV2 have enough semantics. So in these experiments, we select a bigger OpenAI sentence encoder text-embedding-3-large through API. And we also demonstrate in Appendix B.1 that with text-embedding-3-large as the encoder, AtlasKV can also achieve a higher knowledge grounding accuracy.

For knowledge grounding performance evaluation, we extract the averaged-over-heads attention scores of the KG parts that are computed by softmax at the 15th attention layer (for LLaMA3.1-8B-Instruct that has attention layers from 0-31). We did that due to this attention layer is mainly responsible to retreive the accurate external knowledge keys and the external knowledge key embeddings after this attention layer show a higher degree of variation, which has been verified in KBLaM (Wang et al., 2024).

For generation relevance evaluation, like many previous works did (Edge et al., 2024; Huang et al., 2025; Guo et al., 2024; Es et al., 2024), we employ GPT-4o as the evaluator to score the relevance between the generated results and ground truth answers. And the prompt template is shown in Figure 10. To make that statistically significant, we run 5 random seeds for each experiment and we also generate the score 5 times for each seed to get the average score.

[1] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

B EXTENDED EXPERIMENTS

B.1 KNOWLEDGE GROUNDING WITH DIFFERENT ENCODERS

In this section, we conduct experiments with text-embedding-3-large as the sentence encoder to verify that AtlasKV can achieve superior knowledge grounding accuracy with different sentence encoders. Because the output dimension of text-embedding-3-large is 3072, which is much larger than the output dimension of all-MiniLM-L6-v2 (384), we increase the training steps of AtlasKV from 3K to 10K to make sure the training process can well converge. As shown in Table 5, with text-embedding-3-large as the sentence encoder, AtlasKV can still achieve a higher knowledge grounding accuracy than KBLaM at most of the cases. It further demonstrates the adaptivities of AtlasKV to various sentence encoders.

Method	Steps	10^1 Triples		10^2 Triples		10^3 Triples		10^4 Triples	
		ACC@1	ACC@5	ACC@1	ACC@5	ACC@1	ACC@5	ACC@1	ACC@5
Eval on Enron									
KBLaM	1e4	80.0	**100.0**	31.0	69.0	32.0	56.0	20.7	25.9
AtlasKV (128-64-16)	1e4	**100.0**	**100.0**	77.6	89.7	36.2	50.0	19.0	25.9
AtlasKV w/o HiKVP	1e4	**100.0**	**100.0**	**86.2**	**96.6**	**62.1**	**84.5**	**36.2**	**46.6**
Eval on ATLAS-Pes2o-QA									
KBLaM	1e4	60.0	90.0	20.7	56.9	13.8	34.5	6.9	15.5
AtlasKV (128-64-16)	1e4	**100.0**	**100.0**	93.0	98.2	49.1	63.2	28.1	43.9
AtlasKV w/o HiKVP	1e4	**100.0**	**100.0**	**96.5**	**100.0**	**71.9**	**91.2**	**36.8**	**59.6**
Eval on ATLAS-CC-QA									
KBLaM	1e4	80.0	**100.0**	43.9	73.7	17.5	35.1	10.5	12.8
AtlasKV (128-64-16)	1e4	**100.0**	**100.0**	85.5	96.4	54.5	74.5	52.7	65.5
AtlasKV w/o HiKVP	1e4	**100.0**	**100.0**	**85.5**	**98.2**	**80.0**	**92.7**	**65.5**	**81.8**

Table 5: The knowledge grounding accuracy of AtlasKV against KBLaM with text-embedding-3-large as the sentence encoder across various tuning steps and KG sizes.

B.2 INFLUENCE OF VARIOUS TOP-K IN HIKVP

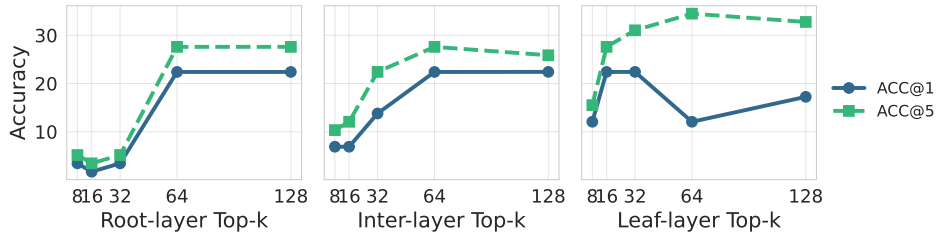


Figure 6: The knowledge grounding accuracy of AtlasKV on ATLAS-CC-QKV with different top-k settings at each layer.

In the default settings of our previous experiments, we set the k_R, k_I, k_L to 128, 64, and 16 respectively. In this section, we conduct experiments to investigate how different top-k settings at each layer of HiKVP influence the knowledge grounding accuracy. We test that based on our default settings, and we change one of k_R, k_I, k_L to different values to see the influence of the top-k settings on the knowledge grounding accuracy of HiKVP. Specifically, we set k_R, k_I, k_L to 128, 64, 32, 16, and 8, respectively, while keeping the other two layers the same as our default settings. And we set the candidate triples to 10^5. As shown in Figure 6, we can see that the knowledge grounding accuracy of AtlasKV will be significantly improved if we increase k_R. And the performance will first improve and then slightly decrease when we increase k_I or k_L. This suggests that the accurate retrieval ability of AtlasKV is stronger than the fuzzy retrieval ability of it. And the reason why too large k_I or k_L will hurt the performance might be that the noise candidate keys selected in early attention layers would influence the retrieval accuracy of the later attention layers.

C DERIVATION OF THE ATTENTION MECHANISM IN ATLASKV

Here we give the details of how the attention mechanism in AtlasKV is derived from the standard rectangular attention in KBLaM.

Proof. First, the rectangular attention at the l-th attention layer and n-th token in KBLaM can be simply rewritten as:

$$\tilde{\mathbf{y}}_n^{(l)} = \frac{\sum_{i=1}^{M} \exp(\langle \tilde{\mathbf{q}}_n^{(l)}, \tilde{\mathbf{k}}^{(l)i} \rangle / \sqrt{D}) \tilde{\mathbf{v}}^{(l)i}}{\sum_{i=1}^{M} \exp(\langle \tilde{\mathbf{q}}_n^{(l)}, \tilde{\mathbf{k}}^{(l)i} \rangle / \sqrt{D}) + \sum_{i=1}^{n} \exp(\langle \mathbf{q}_n^{(l)}, \mathbf{k}^{(l)i} \rangle / \sqrt{D})} + \tag{11}$$

$$\frac{\sum_{i=1}^{n} \exp(\langle \mathbf{q}_n^{(l)}, \mathbf{k}^{(l)i} \rangle / \sqrt{D}) \mathbf{v}^{(l)i}}{\sum_{i=1}^{M} \exp(\langle \tilde{\mathbf{q}}_n^{(l)}, \tilde{\mathbf{k}}^{(l)i} \rangle / \sqrt{D}) + \sum_{i=1}^{n} \exp(\langle \mathbf{q}_n^{(l)}, \mathbf{k}^{(l)i} \rangle / \sqrt{D})}. \tag{12}$$

Then we replace the original formulation with the terms logits_{kb} and logits_{seq} as follows:

$$\tilde{\mathbf{y}}_n^{(l)} = \frac{\sum_{i=1}^{M} \exp(\text{logits}_{kb}^i) \tilde{\mathbf{v}}^{(l)i}}{\sum_{i=1}^{M} \exp(\text{logits}_{kb}^i) + \sum_{i=1}^{n} \exp(\text{logits}_{seq}^i)} + \tag{13}$$

$$\frac{\sum_{i=1}^{n} \exp(\text{logits}_{seq}^i) \mathbf{v}^{(l)i}}{\sum_{i=1}^{M} \exp(\text{logits}_{kb}^i) + \sum_{i=1}^{n} \exp(\text{logits}_{seq}^i)}, \tag{14}$$

where $\text{logits}_{kb}^i = \langle \tilde{\mathbf{q}}_n^{(l)}, \tilde{\mathbf{k}}^{(l)i} \rangle / \sqrt{D}$ and $\text{logits}_{seq}^i = \langle \mathbf{q}_n^{(l)}, \mathbf{k}^{(l)i} \rangle / \sqrt{D}$. Then we can calculate the softmax of the KB and sequence parts separately as follows:

$$\tilde{\mathbf{y}}_n^{(l)} = \underbrace{\frac{\sum_{i=1}^{M} \exp(\text{logits}_{kb}^i)}{\sum_{i=1}^{M} \exp(\text{logits}_{kb}^i) + \sum_{i=1}^{n} \exp(\text{logits}_{seq}^i)}}_{\lambda_{kb}} \cdot \underbrace{\frac{\sum_{i=1}^{M} \exp(\text{logits}_{kb}^i) \tilde{\mathbf{v}}^{(l)i}}{\sum_{i=1}^{M} \exp(\text{logits}_{kb}^i)}}_{\text{Softmax}(\text{logits}_{kb}^i) \tilde{\mathbf{v}}^{(l)i}} + \tag{15}$$

$$\underbrace{\frac{\sum_{i=1}^{n} \exp(\text{logits}_{seq}^i)}{\sum_{i=1}^{M} \exp(\text{logits}_{kb}^i) + \sum_{i=1}^{n} \exp(\text{logits}_{seq}^i)}}_{\lambda_{seq}} \cdot \underbrace{\frac{\sum_{i=1}^{n} \exp(\text{logits}_{seq}^i) \mathbf{v}^{(l)i}}{\sum_{i=1}^{n} \exp(\text{logits}_{seq}^i)}}_{\text{Softmax}(\text{logits}_{seq}^i) \mathbf{v}^{(l)i}}, \tag{16}$$

where λ_{kb} and λ_{seq} are the weights of the two parts. And in this way, the attention computation of the KB and sequence parts can be separated so that we can improve the scalability as well as efficiency of the KB part individually in a more intuitive way.

In our implementation, we replace the attention of the KB part with the KG part and replace λ_{kb} with λ_{kg} in a more scalable way as we described. □

D DERIVATION OF THE TIME AND MEMORY COMPLEXITY OF HIKVP

Here we proof the time and memory complexity of HiKVP are $\mathcal{O}\left((C_t \sqrt[3]{M} + N) \cdot N \cdot D\right)$ and $\mathcal{O}\left((C_m \sqrt[3]{M} + N) \cdot (N + D)\right)$, where $C_t = 1 + k_R + k_I$ and $C_m = \max(1 + k_R + k_I)$ are constants that are much smaller than M.

Proof. **First, we analyze the time and memory complexity of HiKVP step by step.** For the process of calculating the softmax of attention scores with the root-layer keys at each step and selecet top-k_R relevant root-layer keys with a heap of size k_R to fetch the connected inter-layer keys we need in the next step, the time complexity is:

$$\mathcal{O}\left(\sqrt[3]{M} D + \sqrt[3]{M} \log(k_R)\right). \tag{17}$$

The memory complexity to store and calculate the attention scores of the root-layer keys of this process is (before offloading the root-layer keys to the CPU memory):

$$\mathcal{O}\left(\sqrt[3]{M}(N + D)\right). \tag{18}$$

Then we repeat that process with the selected inter-layer keys that are connected to the pruned root-layer keys and fetch the top-k_I relevant inter-layer keys with a heap of size k_I to obtain the connected leaf-layer keys we need in the next step, which has a time complexity of:

$$\mathcal{O}\left(k_R \sqrt[3]{M} D + k_R \sqrt[3]{M} \log(k_I)\right). \tag{19}$$

The memory complexity to store and calculate the attention scores of the selected inter-layer keys of this process is (after offloading the root-layer keys to the CPU memory and before offloading the selected inter-layer keys to the CPU memory):

$$\mathcal{O}\left(k_R \sqrt[3]{M}(N + D)\right). \tag{20}$$

Finally, we compute the softmax of the attention scores of the selected leaf-layer keys that are connected to the pruned inter-layer keys. Similarly, we also need to fetch the top-k_L relevant leaf-layer keys with a heap of size k_L, which has a time complexity of:

$$\mathcal{O}\left(k_I \sqrt[3]{M} D + k_I \sqrt[3]{M} \log(k_L)\right). \tag{21}$$

The memory complexity to store the selected leaf-layer keys and pruned KG values, and to calculate their attention scores is (after offloading the selected inter-layer keys to CPU memory and before offloading the selected leaf-layer keys to GPU memory):

$$\mathcal{O}\left(k_I \sqrt[3]{M}(N + D)\right). \tag{22}$$

Note that $\bar{\text{logits}}_{kg}$ can be obtained by simplying selecting from $\bar{\text{logits}}_{kg_L}$ with the top-k_L softmax scores indices. So this process would not take any additional time or memory complexity.

Then we can synthesize the time and memory complexity of HiKVP at each step. HiKVP has a total time complexity of:

$$\mathcal{O}\left((1 + k_R + k_I)D\sqrt[3]{M} + \left(\sqrt[3]{M}\log(k_R) + k_R\sqrt[3]{M}\log(k_I) + k_I\sqrt[3]{M}\log(k_L)\right)\right), \tag{23}$$

which can be simplified to:

$$\mathcal{O}\left((1 + k_R + k_I)D\sqrt[3]{M} + (\log(k_R) + k_R\log(k_I) + k_I\log(k_L))\sqrt[3]{M}\right). \tag{24}$$

And because usually $D \gg (\log(k_R) + k_R\log(k_I) + k_I\log(k_L))$, we can further simplify it to:

$$\mathcal{O}\left(C_t D\sqrt[3]{M}\right), \tag{25}$$

where $C_t = 1 + k_R + k_I$ is a constant. Then the total time complexity of both HiKVP part and the sequence part at all steps can be represented as:

$$\mathcal{O}\left((C_t\sqrt[3]{M} + N)ND\right), \tag{26}$$

Then for the total memory complexity of the both HiKVP and the sequence part at all steps, we have:

$$\mathcal{O}\left(\left(\max(1 + k_R + k_I)\sqrt[3]{M} + N\right)(N + D)\right), \tag{27}$$

which can be simplified to:

$$\mathcal{O}\left((C_m\sqrt[3]{M} + N)(N + D)\right), \tag{28}$$

where $C_m = \max(1 + k_R + k_I)$ is a constant. □

E Training Dynamics in AtlasKV

We also observed dynamics in the training processes of AtlasKV, which can suggest ***the model regularly start learning to retrieve relevant knowledge from the external KG triples, instead of brute force over-fitting, from a specific training step***. As shown in Figure 7, we trained AtlasKV on both correct and randomly paired KGKVs (denoted as "+ RandomKV") of ATLAS-Wiki-QKV dataset across three different sentence encoders, including all-MiniLM-L6-v2, text-embedding-ada-002, and text-embedding-3-large, respectively. We can find that before a specific training step, the training loss on these two variants of the dataset are almost the same. However, after that, the training loss on the correct variant of the dataset drops significantly while the training loss on the random variant of the dataset continues to decrease slowly. This suggests that AtlasKV starts to generalize by retrieving relevant knowledge from the external KG triples, instead of brute force over-fitting through neural parameters, from a specific training step. And this phenomenon can also support the experimental results that AtlasKV can achieve strong generalization abilities in OOD scenarios.

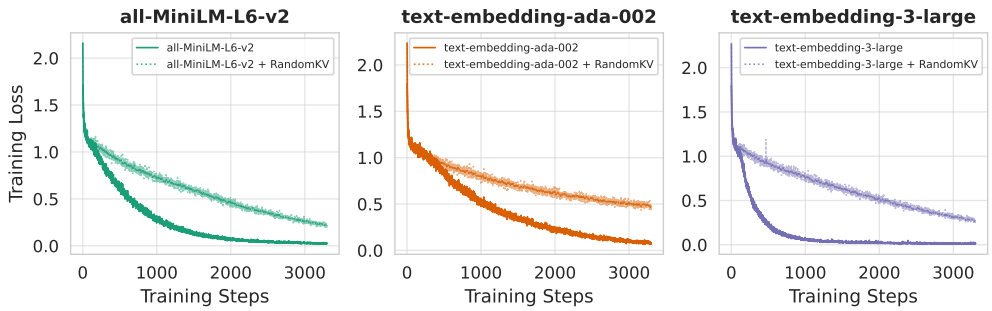


Figure 7: The training loss curves of AtlasKV with correct and random paired key-value embeddings (KGKVs) across three different sentence encoders.

F WHERE IS THE TRADE-OFF?

The trade-off of AtlasKV is essentially between the performance and scalability. As expressed by Equation 26 and Equation 28, we can select different k_R, k_I according to the specific needs to trade off the performance and scalability. For example, if we want to achieve higher performance, we can set a larger k_R, k_I. However, if we need higher scalability, we can set a smaller k_R, k_I like our default settings in our previous experiments. Note that our experiments have suggested that AtlasKV can still achieve superior performance even with small k_R, k_I while maintaining a good scalability. And if we do not prune any key at each layer, the scalability of AtlasKV will degenerate to that of the standard rectangular attention in KBLaM.

G CASE STUDY

G.1 DIFFERENCES BETWEEN SYNTHETIC KB AND KGKVS

In this section, we give some samples of the Q-K-V training data constructed by synthetic and KG2KV methods, respectively, to further demonstrate the differences between them. As shown in Table 6, we demonstrate the Q-K-V strings constructed by synthetic method and there are very limited and fixed enquiry attributes. However, as shown in Table 7, the Q-K-V strings constructed by KG2KV method have much more varied and flexible enquiry attributes, which are much more near to the real-world scenarios.

Q	***K***	***V***
What is the *description* of Elara Moonshadow?	the *description* of Elara Moonshadow	The *description* of Elara Moonshadow is a skilled botanist with a passion for rare plants.
Describe the *description* of Thorne Blackwood?	the *description* of Thorne Blackwood	The *description* of Thorne Blackwood is a renowned chef known for his innovative culinary techniques.
Provide details on the *objectives* of Zara Nightingale?	the *objectives* of Zara Nightingale	The *objectives* of Zara Nightingale is to perform in prestigious concert halls worldwide.
Can you let me know the *purpose* of Lyra Starfire?	the *purpose* of Lyra Starfire	The *purpose* of Lyra Starfire is to preserve marine biodiversity.
Can you explain the *description* of Jaxon Wildheart?	the *description* of Jaxon Wildheart	The *description* of Jaxon Wildheart is a tech entrepreneur with a knack for innovative solutions.
What insights can you provide about the *objectives* of Kaelith Silverwind?	the *objectives* of Kaelith Silverwind	The *objectives* of Kaelith Silverwind is to document endangered animals.

Table 6: Samples from Synthetic dataset. The enquiry attributes have been marked in *italics*.

Q	***K***	***V***
What is the *explanation* of Postsocialist scholars?	the *explanation* of Postsocialist scholars	The *explanation* of Postsocialist scholars is the developments as a backlash against the 'feminizing' nature of the socialist state.
Can you explain the *cause* of World records?	the *cause* of World records	The *cause* of World records is World records in Paralympic powerlifting are ratified by the International Paralympic Committee.
Can you elaborate on the *rank* of Ramble On?	the *rank* of Ramble On	The *rank* of Ramble On is number 5 on the list of the 40 greatest Led Zeppelin songs.
How would you describe the *favorite* of Dick the Mockingbird?	the *favorite* of Dick the Mockingbird	The *favorite* of Dick the Mockingbird is among at least four mockingbirds the president had while in office.
Can you inform me about the *publication* of Hensley?	the *publication* of Hensley	The *publication* of Hensley is Fifty Miles from Tomorrow , a memoir of Alaska and the real people.
Tell me about the *threat* of African coral reefs?	the *threat* of African coral reefs	The *threat* of African coral reefs is industrial run-offs and pollutants, untreated sewage and the increasing sediment flows in rivers.

Table 7: Samples from ATLAS-Wiki-QKV dataset. The enquiry attributes have been marked in *italics*.

Sample Outputs.

Relevant Triple: (*MOROCCO*; *consider*; *synthetic biology should be considered as a new and emerging issue*)
Q: *Can you elaborate on the opinion of MOROCCO?*
K: *the opinion of MOROCCO*
V: *The opinion of MOROCCO is synthetic biology should be considered as a new and emerging issue.*

AtlasKV Output:
The opinion of MOROCCO is issue of synthetic biology should be considered as a new frontier.

KBLaM Output:
I'm not sure what you mean. Can you provide more context?

ICL Output:
The opinion of MOROCCO is synthetic biology should be considered as a new and emerging issue.

Figure 8: A sample Q&A of AtlasKV, KBLaM, and ICL.

G.2 SAMPLE Q&A

As shown in Figure 8, we provide Q&A samples of AtlasKV, KBLaM and ICL with 100 triples in the ATLAS-CC-QKV dataset as candidates. We can tell that AtlasKV can generate a very relevant answer, which is almost close to the ICL's answer. However, KBLaM can not generate a relevant answer and even cannot provide any usefull information. This is mainly because KBLaM is limited by the fully synthetic training data and cannot be generalized to this unseen enquiry attribute. AtlasKV can achieve a higher relevant answer because of a higher diversity of the training data constructed by our KG2KV method.

Prompt Template for KG2KV.

System Message:

Task: Convert relation phrase to natural noun based on missing entity position.
Rules:
- **Missing head**: Passive relations → agent nouns ("govern" → "governor", "is participated by" → "participation")
- **Missing tail**: Active relations → object nouns ("produces" → "product", "achieves" → "achievement")

Output: Natural noun only.

Examples:
- ("is participated by", "head") → "participation"
- ("is participated by", "tail") → "participant"
- ("produces", "head") → "producer"
- ("produces", "tail") → "product"

User Message:

relation: **{relation}**, missing: **{missing}**

Figure 9: The prompt template to rewrite the relation phrase to natural noun based on missing entity position in KG2KV process.

H PROMPT TEMPLATE

In this section, we give the prompt template we use to conduct the evaluations and KG2KV process. As shown in Figure 9, we use LLMs to generate the KGKVs from the text. And in this prompt template, we only need to provide the relation phrase and the missing entity position to generate the natural noun, which is token efficient. And this process usually do not need very powerful LLMs, which are cheaper. As shown in Figure 10, we use LLMs to score the relevance between the generated text and the ground truth answer. This process usually need powerful LLMs like GPT-4o, because it needs to evaluate the results with high quality.

I THE USAGE OF LLMS

In this work, we use the LLM Claude-4-sonnet to polish statements and to check grammars in our paper. We also use that to help with our software developing, such as finding some issues in the codes and giving some advice to make the code structure better.

Prompt Template for Relevance Scoring.

System Message:
You are an AI system that evaluates the quality of generated text. You will be given a text and a ground truth answer, your goals is to return a score between 0 and 1.

User Message:
Given a text and a ground truth answer, evaluate the quality of the text. Return a score of 1 if the text is exactly the same as the ground truth answer. Return a score of 0 if the text is completely wrong. Return a score between 0 and 1 if the text is partially correct. A more correct text should have a higher score. Do NOT generate anything else.
Example:
Model output: "The sky is blue."
True answer: "The sky is blue."
Score: **1.0**

Example 2:
Model output: "The color of Alexandria is blue."
True answer: "The color of Alexandria is green."
Score: **0.0**

Example 3:
Model output: "The purpose of Alexandria is to extract knowledge."
True answer: "The color of Alexandria is to discover and organize knowledge into a structured form."
Score: **0.9**

Important: Only generate a number.

Figure 10: The prompt template for the GPT-4o to score the relevance between the generated text and the ground truth answer.