NEURAL INFORMATION PROCESSING SYSTEMS

Efficient Low Rank Attention for Long-Context Inference in Large Language Models

Tenghui Li [1,2] Guoxu Zhou [1,6] Xuyang Zhao [3,4,5]
Yuning Qiu [2] Qibin Zhao [2]

[1]Guangdong University of Technology [2]RIKEN AIP [3]RIKEN iTHEMS [4]RIKEN IMS
[5]Chiba University [6]Key Laboratory of Intelligent Detection and the Internet of Things in Manufacturing, Ministry of Education, Guangzhou, CHINA

November 6, 2025

Introduction

- **Challenge:** Long context sequences (i.e., >32K) induce linearly increasing KV cache, causing out-of-memory (OOM) failures in LLM inference.

Introduction

- **Challenge:** Long context sequences (i.e., >32K) induce linearly increasing KV cache, causing out-of-memory (OOM) failures in LLM inference.
- **Options:**

Introduction

- **Challenge:** Long context sequences (i.e., >32K) induce linearly increasing KV cache, causing out-of-memory (OOM) failures in LLM inference.
- **Options:**
  - *Pruning:* Memory reduction via KV pair discarding (accuracy loss).

Introduction

- **Challenge:** Long context sequences (i.e., >32K) induce linearly increasing KV cache, causing out-of-memory (OOM) failures in LLM inference.
- **Options:**
  - *Pruning:* Memory reduction via KV pair discarding (accuracy loss).
  - *Offloading:* CPU storage of full KV cache (high data transfer overhead).

Introduction

- **Challenge:** Long context sequences (i.e., >32K) induce linearly increasing KV cache, causing out-of-memory (OOM) failures in LLM inference.
- **Options:**
  - *Pruning:* Memory reduction via KV pair discarding (accuracy loss).
  - *Offloading:* CPU storage of full KV cache (high data transfer overhead).
  - *Quantization:* Reduce the bits of floats in KV cache (precision loss).

Introduction

- **Challenge:** Long context sequences (i.e., >32K) induce linearly increasing KV cache, causing out-of-memory (OOM) failures in LLM inference.
- **Options:**
  - *Pruning:* Memory reduction via KV pair discarding (accuracy loss).
  - *Offloading:* CPU storage of full KV cache (high data transfer overhead).
  - *Quantization:* Reduce the bits of floats in KV cache (precision loss).
- **Target:** Efficient memory management for long-context LLM inference (maintain accuracy and avoid heavy data transfers).

Introduction

- **Challenge:** Long context sequences (i.e., >32K) induce linearly increasing KV cache, causing out-of-memory (OOM) failures in LLM inference.
- **Options:**
  - *Pruning:* Memory reduction via KV pair discarding (accuracy loss).
  - *Offloading:* CPU storage of full KV cache (high data transfer overhead).
  - *Quantization:* Reduce the bits of floats in KV cache (precision loss).
- **Target:** Efficient memory management for long-context LLM inference (maintain accuracy and avoid heavy data transfers).
- **Idea:** Offload KV cache to CPU but **only select** relevant pairs for GPU inference.

Introduction

- **Challenge:** Long context sequences (i.e., >32K) induce linearly increasing KV cache, causing out-of-memory (OOM) failures in LLM inference.
- **Options:**
  - *Pruning:* Memory reduction via KV pair discarding (accuracy loss).
  - *Offloading:* CPU storage of full KV cache (high data transfer overhead).
  - *Quantization:* Reduce the bits of floats in KV cache (precision loss).
- **Target:** Efficient memory management for long-context LLM inference (maintain accuracy and avoid heavy data transfers).
- **Idea:** Offload KV cache to CPU but **only select** relevant pairs for GPU inference.
- **Questions:** How to select relevant KV pairs efficiently?

Introduction

- **Challenge:** Long context sequences (i.e., >32K) induce linearly increasing KV cache, causing out-of-memory (OOM) failures in LLM inference.
- **Options:**
  - *Pruning:* Memory reduction via KV pair discarding (accuracy loss).
  - *Offloading:* CPU storage of full KV cache (high data transfer overhead).
  - *Quantization:* Reduce the bits of floats in KV cache (precision loss).
- **Target:** Efficient memory management for long-context LLM inference (maintain accuracy and avoid heavy data transfers).
- **Idea:** Offload KV cache to CPU but **only select** relevant pairs for GPU inference.
- **Questions:** How to select relevant KV pairs efficiently?
- Find a lightweight proxy attention to approximate the original attention.

Method Overview

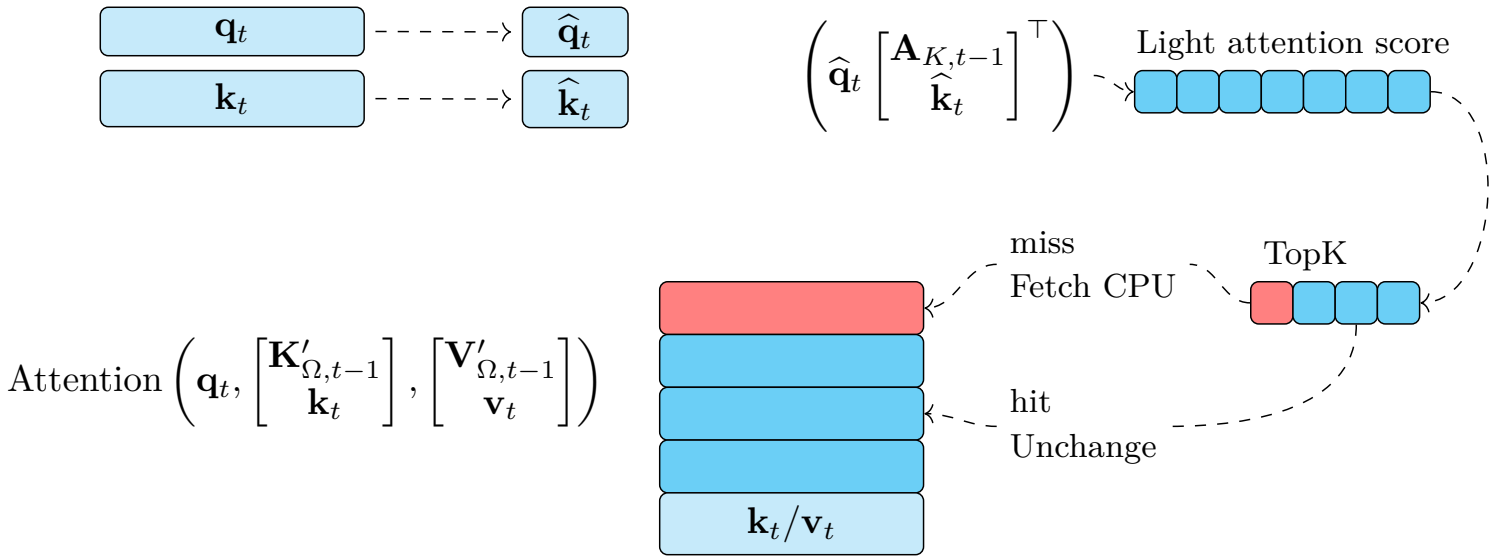
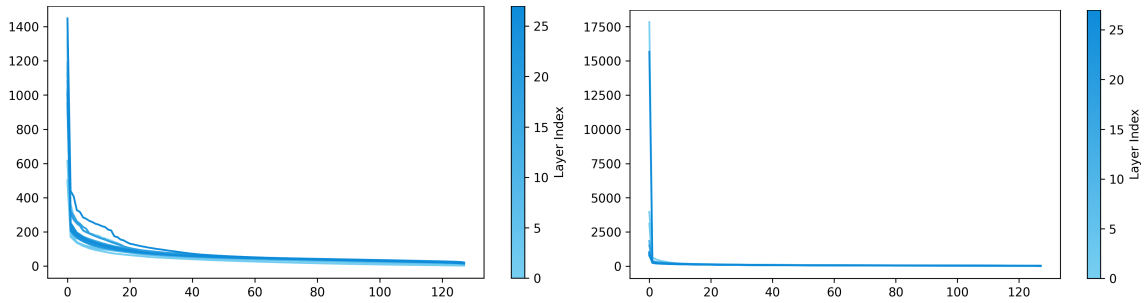


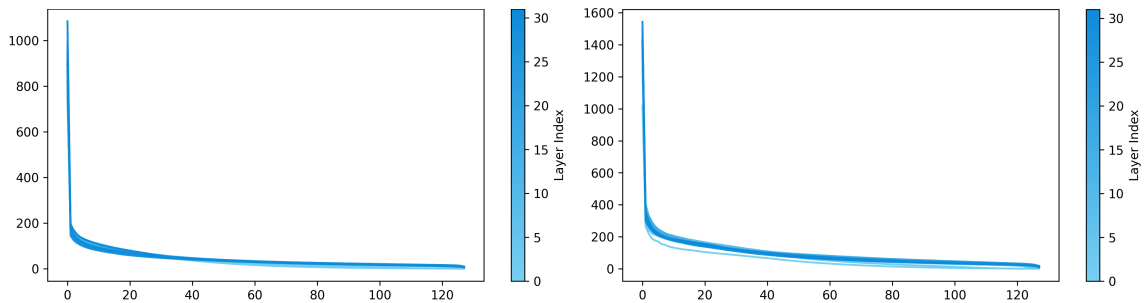
Figure: Overview of the LRQK method. Subscript Ω denotes selected tokens, t denotes current token. $\mathbf{q}_t, \mathbf{k}_t$ are original query and key, $\widehat{\mathbf{q}}_t, \widehat{\mathbf{k}}_t$ are approximated query and key.

Low Rank Structure of $\mathbf{QK}^\top$

$$\text{rank}(\mathbf{QK}^\top) \leq \min\left(\text{rank}(\mathbf{Q}), \text{rank}(\mathbf{K}^\top)\right) = \min\left(\text{rank}(\mathbf{Q}), \text{rank}(\mathbf{K})\right).$$



(a) Qwen2.5 **Q** (b) Qwen2.5 **K**



(c) LLaMA3 **Q** (d) LLaMA3 **K**

Figure: Examples of the mean of singular values of the query and key matrix over different layers on Qwen2.5-7B and LLaMA-3-8B-1M models. The singular values are summed over batches and attention heads.

Proposed

Identify relevant KV pairs $\rightarrow$ *Joint Low-Rank Approximation*: For $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{l \times d}$ (consider last two dimensions), approximate interaction matrix as:

$$\mathbf{Q}\mathbf{K}^\top \approx \mathbf{A}_Q \mathbf{A}_K^\top \quad \text{with} \quad \mathbf{A}_Q, \mathbf{A}_K \in \mathbb{R}^{l \times r}, \; r < d$$

Proposed

Identify relevant KV pairs $\rightarrow$ *Joint Low-Rank Approximation*: For $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{l \times d}$ (consider last two dimensions), approximate interaction matrix as:

$$\mathbf{Q}\mathbf{K}^\top \approx \mathbf{A}_Q \mathbf{A}_K^\top \quad \text{with} \quad \mathbf{A}_Q, \mathbf{A}_K \in \mathbb{R}^{l \times r}, \; r < d$$

Avoid reconstruction errors $\rightarrow$ *Precision-Preserving Attention:* Maintain full-precision computation using original keys/values,

$$\text{Attention}(\mathbf{q}_t, \mathbf{K}_{\Omega,t}, \mathbf{V}_{\Omega,t})$$

Avoid reconstruction as,

$$\mathbf{K}_{\Omega,t} \leftarrow \text{rec}(\text{factors } \mathbf{K}, \Omega_t) \text{ or } \mathbf{V}_{\Omega,t} \leftarrow \text{rec}(\text{factors } \mathbf{V}, \Omega_t).$$

Proposed

Identify relevant KV pairs $\rightarrow$ *Joint Low-Rank Approximation*: For $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{l \times d}$ (consider last two dimensions), approximate interaction matrix as:

$$\mathbf{Q}\mathbf{K}^\top \approx \mathbf{A}_Q \mathbf{A}_K^\top \quad \text{with} \quad \mathbf{A}_Q, \mathbf{A}_K \in \mathbb{R}^{l \times r}, \; r < d$$

Avoid reconstruction errors $\rightarrow$ *Precision-Preserving Attention:* Maintain full-precision computation using original keys/values,

$$\text{Attention}(\mathbf{q}_t, \mathbf{K}_{\Omega,t}, \mathbf{V}_{\Omega,t})$$

Avoid reconstruction as,

$$\mathbf{K}_{\Omega,t} \leftarrow \text{rec}(\text{factors } \mathbf{K}, \Omega_t) \text{ or } \mathbf{V}_{\Omega,t} \leftarrow \text{rec}(\text{factors } \mathbf{V}, \Omega_t).$$

Memory management $\rightarrow$ *Hybrid Memory Architecture:* CPU-GPU coordination with asynchronous transfers and hierarchical caching.

Algorithm Prefill

Within the standard LLM inference framework, prompt processing (prefill) followed by autoregressive generation (decode). This approach introduces a low-rank attention and a dynamic CPU-GPU KV management.

Prefill (prompt processing): To derive a low rank representation $\mathbf{A}_Q, \mathbf{A}_K \in \mathbb{R}^{l\times r}$, and $\mathbf{B}_Q, \mathbf{B}_K \in \mathbb{R}^{r\times d}$ from the original query and key matrices $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{l\times d}$,

$$\underset{\mathbf{A}_Q,\mathbf{B}_Q,\mathbf{A}_K,\mathbf{B}_K}{\arg\min} \frac{1}{2}\left\|\mathbf{Q}\mathbf{K}^\top - \mathbf{A}_Q\mathbf{A}_K^\top\right\|_{\mathrm{F}}^2, \text{s.t. } \mathbf{Q} = \mathbf{A}_Q\mathbf{B}_Q, \mathbf{K} = \mathbf{A}_K\mathbf{B}_K.$$

Algorithm Decode

Decode (autoregressive generation): Update the low rank factors during inference. At step t, continually compute smaller $\widehat{\mathbf{q}}_t, \widehat{\mathbf{k}}_t \in \mathbb{R}^{1\times r}$ from current input $\mathbf{q}_t, \mathbf{k}_t \in \mathbb{R}^{1\times d}$ and update the low rank factors $\mathbf{B}_{Q,t}, \mathbf{B}_{K,t} \in \mathbb{R}^{r\times d}$,

$$\arg\min_{\widehat{\mathbf{q}}_t, \widehat{\mathbf{k}}_t} \frac{1}{2} \left\| \widehat{\mathbf{q}}_t \mathbf{B}_{Q,t-1} - \mathbf{q}_t \right\|_{\mathrm{F}}^2 + \frac{1}{2} \left\| \widehat{\mathbf{k}}_t \mathbf{B}_{K,t-1} - \mathbf{k}_t \right\|_{\mathrm{F}}^2,$$

$$\text{s.t. } \widehat{\mathbf{q}}_t \widehat{\mathbf{k}}_t^\top = \mathbf{q}_t \mathbf{k}_t^\top, \widehat{\mathbf{q}}_t \mathbf{A}_{K,\Omega,t-1}^\top = \mathbf{q}_t \mathbf{K}_{\Omega,t-1}^\top.$$

Algorithm Decode

Decode (autoregressive generation): Update the low rank factors during inference. At step t, continually compute smaller $\widehat{\mathbf{q}}_t, \widehat{\mathbf{k}}_t \in \mathbb{R}^{1\times r}$ from current input $\mathbf{q}_t, \mathbf{k}_t \in \mathbb{R}^{1\times d}$ and update the low rank factors $\mathbf{B}_{Q,t}, \mathbf{B}_{K,t} \in \mathbb{R}^{r\times d}$,

$$\arg\min_{\widehat{\mathbf{q}}_t, \widehat{\mathbf{k}}_t} \frac{1}{2} \left\| \widehat{\mathbf{q}}_t \mathbf{B}_{Q,t-1} - \mathbf{q}_t \right\|_{\mathrm{F}}^2 + \frac{1}{2} \left\| \widehat{\mathbf{k}}_t \mathbf{B}_{K,t-1} - \mathbf{k}_t \right\|_{\mathrm{F}}^2,$$

$$\text{s.t. } \widehat{\mathbf{q}}_t \widehat{\mathbf{k}}_t^\top = \mathbf{q}_t \mathbf{k}_t^\top, \widehat{\mathbf{q}}_t \mathbf{A}_{K,\Omega,t-1}^\top = \mathbf{q}_t \mathbf{K}_{\Omega,t-1}^\top.$$

- Select top-k relevant tokens using proxy attention:

$$\Omega_k = \mathrm{topk}\left(\widehat{\mathbf{q}}_t \mathbf{A}_{K,\Omega,t-1}^\top,\ k\right)$$

Algorithm Decode

Decode (autoregressive generation): Update the low rank factors during inference. At step t, continually compute smaller $\widehat{\mathbf{q}}_t, \widehat{\mathbf{k}}_t \in \mathbb{R}^{1\times r}$ from current input $\mathbf{q}_t, \mathbf{k}_t \in \mathbb{R}^{1\times d}$ and update the low rank factors $\mathbf{B}_{Q,t}, \mathbf{B}_{K,t} \in \mathbb{R}^{r\times d}$,

$$\arg\min_{\widehat{\mathbf{q}}_t, \widehat{\mathbf{k}}_t} \frac{1}{2} \left\| \widehat{\mathbf{q}}_t \mathbf{B}_{Q,t-1} - \mathbf{q}_t \right\|_{\mathrm{F}}^2 + \frac{1}{2} \left\| \widehat{\mathbf{k}}_t \mathbf{B}_{K,t-1} - \mathbf{k}_t \right\|_{\mathrm{F}}^2,$$

$$\text{s.t. } \widehat{\mathbf{q}}_t \widehat{\mathbf{k}}_t^\top = \mathbf{q}_t \mathbf{k}_t^\top, \widehat{\mathbf{q}}_t \mathbf{A}_{K,\Omega,t-1}^\top = \mathbf{q}_t \mathbf{K}_{\Omega,t-1}^\top.$$

- Select top-k relevant tokens using proxy attention:

$$\Omega_k = \mathrm{topk}\left(\widehat{\mathbf{q}}_t \mathbf{A}_{K,\Omega,t-1}^\top,\ k \right)$$

- Fetch corresponding $\{\mathbf{k}_i, \mathbf{v}_i\}_{i\in\Omega_k}$ from CPU cache; merge with GPU-resident KV cache.

Algorithm Decode

Decode (autoregressive generation): Update the low rank factors during inference. At step t, continually compute smaller $\widehat{\mathbf{q}}_t, \widehat{\mathbf{k}}_t \in \mathbb{R}^{1\times r}$ from current input $\mathbf{q}_t, \mathbf{k}_t \in \mathbb{R}^{1\times d}$ and update the low rank factors $\mathbf{B}_{Q,t}, \mathbf{B}_{K,t} \in \mathbb{R}^{r\times d}$,

$$\arg\min_{\widehat{\mathbf{q}}_t, \widehat{\mathbf{k}}_t} \frac{1}{2}\left\|\widehat{\mathbf{q}}_t \mathbf{B}_{Q,t-1} - \mathbf{q}_t\right\|_{\mathrm{F}}^2 + \frac{1}{2}\left\|\widehat{\mathbf{k}}_t \mathbf{B}_{K,t-1} - \mathbf{k}_t\right\|_{\mathrm{F}}^2,$$

$$\text{s.t. } \widehat{\mathbf{q}}_t \widehat{\mathbf{k}}_t^\top = \mathbf{q}_t \mathbf{k}_t^\top, \widehat{\mathbf{q}}_t \mathbf{A}_{K,\Omega,t-1}^\top = \mathbf{q}_t \mathbf{K}_{\Omega,t-1}^\top.$$

- Select top-k relevant tokens using proxy attention:

$$\Omega_k = \mathrm{topk}\left(\widehat{\mathbf{q}}_t \mathbf{A}_{K,\Omega,t-1}^\top,\ k\right)$$

- Fetch corresponding $\{\mathbf{k}_i, \mathbf{v}_i\}_{i\in\Omega_k}$ from CPU cache; merge with GPU-resident KV cache.
- Update basis matrices $\mathbf{B}_{Q,t}, \mathbf{B}_{K,t}$ and asynchronously offload $\mathbf{k}_t, \mathbf{v}_t$ to CPU.

Method Overview

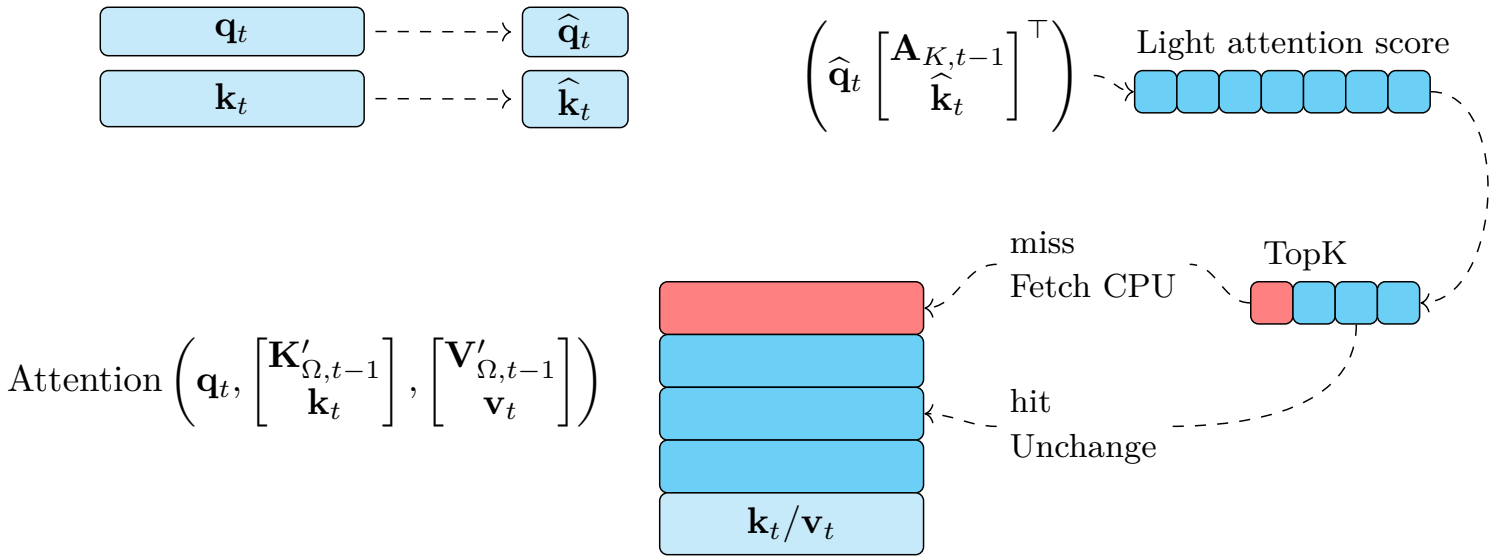


Figure: Overview of the LRQK method. Subscript Ω denotes selected tokens, t denotes current token. $\mathbf{q}_t, \mathbf{k}_t$ are original query and key, $\widehat{\mathbf{q}}_t, \widehat{\mathbf{k}}_t$ are approximated query and key.