Universiteit Leiden

Master Computer Science

Training-Free Efficient Inference for Large Generative Models

Name:	Suju Li
Student ID:	4024354
Date:	29/01/2026
Specialisation:	Artificial Intelligence
1st supervisor:	Qinyu Chen
2nd supervisor:	Chang Gao (TU Delft)

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Training-Free Efficient Inference for Large Generative Models

Master's Thesis

for obtaining the degree of
Master at Leiden University,
under the authority of the Executive Board,
to be defended on *Thursday, 29 01 2026*

by

Suju Li
born in *China*
in *1999*

Promotor:	Dr. Qinyu Chen	
Co-promotor:	Dr. Chang Gao	(TU Delft)

Abstract

Large language models (LLMs) and vision-language models (VLMs) are increasingly deployed in settings where inference cost, rather than model accuracy, is the dominant bottleneck. While recent models support longer contexts and stronger reasoning capabilities, the memory and compute required for generative inference scale poorly due to the growth of key-value (KV) caches and dense visual token representations. Notably, such inefficiencies arise during inference and do not require modifying or retraining model weights, motivating training-free approaches that adapt to deployment constraints.

This thesis studies the problem of reducing inference cost in large-scale generative models through training-free and model-agnostic algorithms. We identify three distinct bottlenecks: (i) long-context inputs that expand the KV cache during prefill, (ii) chain-of-thought generation that inflates the cache during decoding, and (iii) vision–language inference where dense visual tokens dominate compute. Across these settings we introduce three mechanisms: (i) a geometry-aware KV pruning method based on query-key mapping that preserves long-context retrieval accuracy while discarding over 80% of keys, (ii) a redundancy-aware compression strategy for chain-of-thought decoding that leverages block-wise patterns in the generation process to prune verbose reasoning traces, and (iii) a gaze-guided token pruning framework that exploits physiological attention priors to reduce visual tokens for VQA without retraining.

Together, the results show that considerable memory and compute savings can be achieved at inference time through training-free mechanisms across text-only and multimodal workloads, helping bridge the gap between foundation model capability and practical on-device or resource-constrained deployment.

Contents

Chapter 1

Introduction

1.1 Background

Large Language Models (LLMs) have become central to modern AI systems, demonstrating strong capabilities across tasks such as conversational assistance [25, 14], autonomous agents [26], and multimodal reasoning [27, 15]. Their effectiveness largely arises from the transformer architecture and its ability to process long sequences via attention mechanisms. As these models scale in size and capability, the cost of inference has increasingly become a practical bottleneck, particularly in real-world deployments where memory, energy, and latency constraints are non-negotiable.

One major source of inefficiency comes from long-context processing. Applications such as document understanding, code analysis, and multi-turn dialogue require conditioning on long histories or large input documents. Recent advances have extended context lengths from tens of thousands to hundreds of thousands of tokens [25, 31, 14, 1], but this reveals severe memory scaling challenges: the key–value (KV) cache used for autoregressive decoding grows linearly with sequence length. In many configurations, the KV cache can occupy more GPU memory than the model parameters themselves. For instance, a single 8B-parameter model with a 128K-token context can require over 100 GB of cache memory [16], making long-context inference costly to deploy. Prior inference-time sparsification strategies [2, 17] prune tokens based on attention statistics, but static heuristics often fail in multi-step reasoning scenarios where token relevance changes dynamically over the generation process [16].

In addition to long-context inputs, LLMs increasingly serve as reasoning engines capable of chain-of-thought (CoT) style inference. Models such as DeepSeek-R1 can

generate tens of thousands of tokens of self-dialogue to solve complex tasks [7], but these traces have notable redundancy. This induces a different scaling problem: even when input prompts are short, the KV cache can explode due to excessively long decoding sequences. Furthermore, naïve attention-based pruning may either remove scattered but important reasoning steps, or preserve repetitive self-reflections that exhibit high self-attention [33]. This reveals a tension between preserving reasoning fidelity and serving models under realistic memory constraints.

Beyond text-only LLMs, multimodal vision-language models (VLMs) extend these capabilities to image captioning, visual question answering, and video understanding [13, 27, 15]. Deploying such models introduces an orthogonal bottleneck: visual representations are dense, and a single high-resolution image may produce hundreds of visual tokens [19]. Visual token processing alone may exceed 90% of the compute cost in typical image captioning pipelines [19]. This is problematic for edge deployments such as AR/VR headsets, where compute and battery resources are constrained. Interestingly, many of these devices integrate real-time eye-tracking modules [24, 18, 23], providing physiological signals about user attention. Human gaze thus offers a natural supervisory signal for guiding efficient multimodal inference [5].

Collectively, these trends highlight that the practical limitations of modern LLMs and VLMs increasingly arise not from model accuracy, but from the cost of inference. Improving efficiency without retraining has therefore become a critical research direction. In particular, *training-free* and *model-agnostic* inference methods provide a promising path for making state-of-the-art models deployable under realistic memory, energy, and latency constraints.

1.2 Research Questions

In this thesis, we investigate how to reduce the computational and memory cost of large-scale language and vision-language model inference without modifying model parameters or requiring additional training. This direction is motivated by the increasing deployment of LLMs in real-world settings where constraints arise from limited GPU memory, energy budgets, and latency-sensitive applications. To this end, we formulate three main research questions:

RQ1 **(Chapter 3) How can we reduce the KV cache memory footprint of long-context inference without sacrificing model accuracy?**

Long-context LLMs encounter severe memory scaling issues due to the linear

growth of the key–value (KV) cache. Existing sparsification-based solutions often prune tokens based on static attention heuristics, which may fail in multi-step reasoning scenarios. Therefore, the first research question asks whether it is possible to accelerate and reduce the memory footprint of long-context inference while retaining full information and without retraining the model.

RQ2 **(Chapter 4) How can we compress the KV cache during long chain-of-thought generation to fit within practical deployment constraints?**

Reasoning-oriented LLMs often generate excessively long and redundant self-dialogue, which inflates the KV cache during decoding rather than during prompt prefill. Unlike long-context input compression, output-side redundancy requires compression to preserve semantic reasoning steps while discarding verbose self-reflection. Thus, our second research question focuses on whether redundancy-aware, training-free compression mechanisms can retain reasoning fidelity while substantially decreasing memory consumption.

RQ3 **(Chapter 5) How can we reduce the inference cost of vision-language models by leveraging human physiological priors?**

VLMs incur significant computation due to dense visual tokenization. While pruning based on generic visual saliency is possible, AR/VR settings provide access to real-time eye gaze signals. This raises the question of whether gaze information can guide selective visual token reduction in a training-free and plug-and-play manner, thereby improving efficiency under constrained device budgets.

1.3 Contributions of this Thesis

The goal of this thesis is to improve the efficiency of large-scale language and vision-language model inference without additional training. To this end, the main contributions of this thesis are as follows:

1. **A training-free KV cache pruning mechanism for long-context inference.** We introduce a dynamic key selection strategy driven by query–query similarity during generation, enabling the model to retain only the most relevant KV entries. This substantially reduces memory footprint during long-context inference without retraining or modifying model weights, and preserves long-context task performance despite aggressive sparsification.

2. **A redundancy-aware KV cache compression method for chain-of-thought reasoning models.** We introduce a compression strategy that jointly considers token importance and token redundancy during autoregressive decoding. By retaining only a small subset of non-repetitive and semantically critical tokens, the method significantly reduces KV cache usage during long CoT generation while maintaining or even surpassing baseline reasoning accuracy.

3. **A gaze-guided token pruning framework for vision-language inference on resource-constrained devices.** We exploit human eye gaze as a physiological attention prior to crop task-relevant regions of input images, thereby reducing the number of visual tokens required for inference. This training-free framework enables efficient inference for visual question answering (VQA) in AR/VR settings while maintaining semantic coverage and reach similar accuracy in some configurations.

Collectively, these contributions demonstrate that substantial savings in memory and computation can be achieved through *training-free*, *plug-and-play* inference-time techniques. Moreover, the methods presented in this thesis are model-agnostic and applicable to open-source foundation models, which is an increasingly important deployment constraint in industry settings.

Chapter 2

Preliminaries

This chapter introduces the necessary background, terminology, and notation used throughout this thesis. We first review the transformer architecture with emphasis on attention and KV caching. We then discuss inference modes, chain-of-thought (CoT) reasoning, and vision-language tokenization.

2.1 Transformer Architecture

Large language models are primarily based on the transformer architecture, which processes a token sequence $x = (x_1, \ldots, x_T)$ using multi-head self-attention. Each input token is projected into queries, keys, and values:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V, \tag{2.1}$$

with $X \in \mathbb{R}^{T \times d}$ and $W_Q, W_K, W_V \in \mathbb{R}^{d \times d_h}$.

Scaled dot-product attention is defined as:

$$\mathrm{Attn}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_h}}\right) V. \tag{2.2}$$

2.2 Autoregressive Inference and KV Cache

During inference, the next token x_t is sampled conditioned on the previous tokens:

$$p(x_t \mid x_{<t}) = \mathrm{LM}(x_{<t}). \tag{2.3}$$

To avoid recomputing attention over the prefix, transformers store keys and values in a *KV cache*:

$$K^{\ell}_{1:t-1} = (k_1, \ldots, k_{t-1}), \qquad V^{\ell}_{1:t-1} = (v_1, \ldots, v_{t-1}) \tag{2.4}$$

for each layer ℓ.

Attention at step t becomes:

$$y_t = \mathrm{Attn}(q_t,\, K^{\ell}_{1:t-1},\, V^{\ell}_{1:t-1}). \tag{2.5}$$

The KV cache memory cost scales linearly with context length:

$$\mathrm{Mem}_{\mathrm{KV}} = \mathcal{O}(L \cdot H \cdot d_h \cdot T), \tag{2.6}$$

where L is number of layers, H heads, d_h head dimension, and T sequence length.

This scaling becomes a bottleneck for both:

1. long-context input prompts (large T during prefill)
2. chain-of-thought output sequences (large T during decode)

which motivates KV cache pruning strategies.

2.3 Prefill vs. Decode

Inference consists of two distinct phases. During *prefill*, the model processes the entire prompt in parallel, constructs the key–value (KV) cache, and emits the first token. After prefill, *decode* proceeds autoregressively, generating tokens one at a time while extending and reusing the KV cache.

2.4 Chain-of-Thought Reasoning

Tools such as self-reflection and deliberative reasoning produce intermediate traces:

$$x^{\mathrm{CoT}}_{1:T} = (\mathrm{step}_1, \ldots, \mathrm{step}_m, \mathrm{answer}). \tag{2.7}$$

These traces exhibit:

1. Redundancy: repeated argumentation and verification.

2. Sparse semantic contribution: many tokens have negligible influence on the final answer.

2.5 Vision-Language Tokenization

Vision-language models encode an image $I \in \mathbb{R}^{H \times W \times 3}$ into N patches:

$$I = (p_1, \ldots, p_N) \tag{2.8}$$

and embed them via

$$z_i = f_\theta(p_i) \in \mathbb{R}^d. \tag{2.9}$$

For high-resolution inputs, N may reach hundreds of tokens, dominating inference cost in multimodal tasks.

Chapter 3

Training-Free Q–K Mapping for Long-Context KV Pruning

In this chapter, we present a training-free method to reduce the key–value (KV) cache of large language models (LLMs) during long-context inference. Concretely, we instantiate our approach on a 7B-parameter LLaMA-2 chat model with a 4K context window (`llama2-7b-chat-4k`) and evaluate it on the LongBench benchmark. All experiments are run on a SLURM-managed GPU cluster using a single NVIDIA A100 GPU. The method builds a *query–key (Q–K) mapping* from attention statistics in the prompt (prefill) phase and reuses it to select a small, context-dependent subset of keys during decoding. This allows us to discard most KV entries from GPU memory while retaining competitive task performance on LongBench.

The key properties of the proposed approach are:

1. It is entirely training-free: no fine-tuning or gradient updates are required; we only modify the inference-time attention computation.

2. It is plug-and-play: it operates by wrapping the existing attention layer, without changing the underlying LLaMA weights or tokenizer.

3. It supports multiple variants (layer-wise sharing, and confidence-aware gating) within the same implementation framework and hardware setup.

3.1 Problem Setting

We consider an autoregressive transformer with L decoder layers, model dimension d_{model}, and H attention heads. At time step t, layer ℓ maintains a KV cache

$$K_{1:t-1}^{(\ell)} \in \mathbb{R}^{(t-1)\times d_k}, \qquad V_{1:t-1}^{(\ell)} \in \mathbb{R}^{(t-1)\times d_v},$$

where $d_k = d_v = d_{\text{model}}/H$.

For a single query token at step t, the standard scaled dot-product attention in layer ℓ is

$$A_t^{(\ell)} = \text{softmax}\left(\frac{Q_t^{(\ell)} {K_{1:t-1}^{(\ell)}}^\top}{\sqrt{d_k}}\right) \in \mathbb{R}^{H\times 1\times (t-1)}, \tag{3.1}$$

$$O_t^{(\ell)} = A_t^{(\ell)} V_{1:t-1}^{(\ell)} \in \mathbb{R}^{H\times 1\times d_v}. \tag{3.2}$$

The total KV memory at step t is proportional to

$$\sum_{\ell=1}^{L} (t-1)d_k + (t-1)d_v \;=\; 2L(t-1)d_k,$$

which grows linearly in the effective context length $(t-1)$. For long prompts and long chain-of-thought outputs, this KV memory becomes the dominant bottleneck.

Our goal is to define, for each layer ℓ and step t, a *retained index set*

$$S^{(\ell)}(t) \subseteq \{1, \ldots, t-1\}$$

such that:

1. only $K_{S^{(\ell)}(t)}^{(\ell)}$ and $V_{S^{(\ell)}(t)}^{(\ell)}$ need to be kept on GPU,

2. the key retention ratio

$$r_t^{(\ell)} = \frac{|S^{(\ell)}(t)|}{t-1}$$

   is small (e.g. 10%–20%),

3. and task performance on LongBench remains close to the full-KV baseline.

3.2 Background: Long-Context Attention and KV Redundancy

Transformers cache past key and value tensors during autoregressive decoding to avoid recomputing attention over the full prefix. While this mechanism reduces per-token compute to linear time in the decode length, it incurs a memory cost proportional to the context length. As the prompt grows to thousands of tokens, the KV cache becomes a dominant memory component, particularly for multi-layer models with many attention heads.

Recent long-context benchmarks such as *LongBench* [3] reveal that only a small subset of prompt tokens meaningfully contributes to downstream outputs. Tasks such as narrative QA, scientific QA, and multi-hop reasoning require reading long documents, yet empirical analyses show that attention mass during generation is often concentrated on semantically relevant spans while large regions of the prompt receive negligible attention. This suggests substantial semantic redundancy in the KV cache: although all keys are stored for correctness, many have limited influence on the generation results.

This redundancy interacts with the inductive biases of transformer depth. Earlier layers often emphasize local and positional structure, while deeper layers tend to incorporate broader semantic context [28, 6]. Such depth-dependent retrieval patterns imply that relevance is unevenly distributed across both time and depth, creating opportunities for selective retention.

As context length increases, the KV cache becomes the dominant bottleneck in deployment scenarios such as retrieval augmented generation, chain-of-thought reasoning, multi-turn document chat, legal/financial analysis, and multimodal models with visual tokens. Without compression, throughput degrades or multi-GPU sharding is required, complicating serving.

Prior work on long-context efficiency has largely focused on modifying the attention mechanism to reduce computational cost. Methods such as Reformer [21], Longformer [4], and BigBird [32] introduce structured sparse attention patterns, while Linformer [29] employs low-rank projections to approximate dense self-attention, enabling efficient processing of longer sequences.

A separate line of work targets the KV cache itself. Several approaches perform inference-time eviction or retrieval of KV entries based on attention signals or learned importance scores [10, 12, 11]. Other methods reduce KV memory through compression or adaptive cache construction [9, 20], while system-level approaches jointly

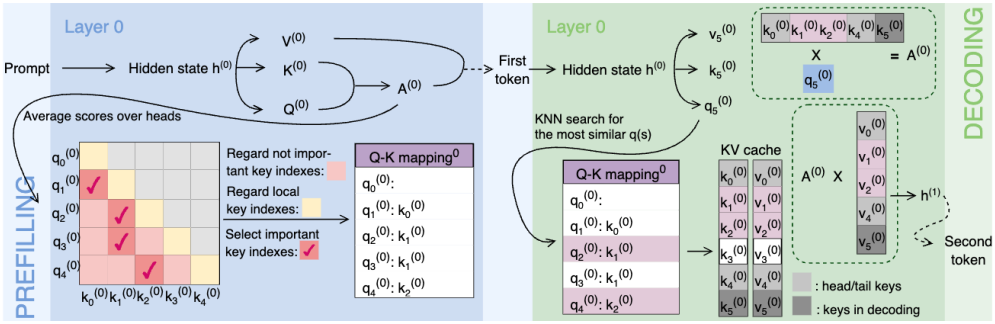


Figure 3.1: End-to-end illustration of the proposed training-free Q–K mapping approach for KV pruning. During **prefill** (left), the model processes the prompt with full attention, from which we average attention scores over heads and select non-local, semantically important key positions to construct a sparse query–key mapping. During **decoding** (right), each new query is projected into the prefill query space using a KNN search to retrieve similar prompt queries, whose mapped key sets are aggregated and augmented with structurally important head and tail regions. Only the retained keys and values are then used to compute attention, enabling substantial KV memory reduction without modifying model weights.

optimize compression and eviction to improve serving efficiency [8].

Finally, a complementary direction studies training-free KV reduction, leveraging prefill signals to select a compact subset of useful keys without retraining or altering model weights [16]. Our work follows this paradigm: we introduce a geometry-aware selection mechanism that uses prefill queries to construct a semantic key mapping, enabling aggressive KV sparsification during decoding while preserving downstream task performance.

3.3 Proposed Method: Training-Free Q–K Mapping

With the preliminaries in Sections 3.1 and 3.2, we now present our training-free Q–K mapping approach for KV pruning. Rather than modifying attention weights, we constrain where attention is allowed to look while keeping the standard mechanism for how attention is computed, and all changes are implemented as index selection over existing KV entries:

1. During **prefill**, we observe the model's full attention patterns once and transform them into a per-layer mapping from each prompt query position to a small set of semantically important key positions.

2. During **decoding**, each new query is projected into the prefill query space; its

nearest neighbours "vote" for a set of likely relevant keys based on the per-layer mapping we obtain in prefill, which we then augment with several structural regions (head, tail, and generated tokens) before computing attention.

3.3.1 Prefill: Building the Per-Layer Q–K Map

In the prefill stage, the model processes the entire prompt of length T with full attention, yielding per-layer attention tensors $A^{(\ell)} \in \mathbb{R}^{H\times T\times T}$ and query tensors $Q^{(\ell)} \in \mathbb{R}^{H\times T\times d_k}$ as described in Section 3.2. We use these tensors to construct a compressed, geometry-aware summary (Q–K Map) of the model's prompt-time behaviour.

Head-averaged attention. To remove head-specific noise and highlight keys that are consistently important across heads, we average attention weights:

$$\tilde{A}^{(\ell)}_{i,j} = \frac{1}{H}\sum_{h=1}^{H} A^{(\ell)}_{h,i,j}, \qquad i,j \in \{1,\dots,T\}. \tag{3.3}$$

Each row $\tilde{A}^{(\ell)}_{i,:}$ describes the global dependency pattern of query position i within the prompt at layer ℓ.

Tail masking to mitigate locality bias. Causal transformers tend to assign large attention mass to the most recent positions purely due to locality, even when more distant tokens are semantically crucial. If we directly extracted top-k keys from $\tilde{A}^{(\ell)}_{i,:}$, the resulting map would heavily overfit to this recency bias and underrepresent long-range dependencies (e.g. instructions or early evidence).

We therefore apply a simple tail masking heuristic: for each row i, we identify the subset of indices in its causal past and mask a small suffix near i by setting their scores to $-\infty$ before ranking. Intuitively, we force the mapping to focus on nontrivial, non-local dependencies and let locality be handled explicitly at decoding time.

Top-k per-query key selection. For each query position i, we then select a small set of important keys:

$$\mathcal{T}^{(\ell)}(i) = \mathrm{TopK}\big(\tilde{A}^{(\ell)}_{i,:}, k\big) \subseteq \{1,\dots,T\}, \tag{3.4}$$

where $k = \lfloor \alpha T \rfloor$ for some fraction $\alpha \in (0, 1]$. The set $\mathcal{T}^{(\ell)}(i)$ can be interpreted as the semantic support of query i under full attention at layer ℓ: keys that collectively carry most of its attention mass once trivial locality has been removed.

Collecting all such sets yields a sparse query–key graph,

$$\mathcal{M}^{(\ell)}_{\text{prefill}} : \{1, \dots, T\} \to 2^{\{1,\dots,T\}}, \quad i \mapsto \mathcal{T}^{(\ell)}(i),$$

which serves as a compact summary of prefill attention patterns. Importantly, $\mathcal{M}^{(\ell)}_{\text{prefill}}$ is constructed once per prompt and reused for all subsequent decoding steps.

Building a prefill queries space. To apply the prefill map to future decoding queries, we need a way to relate unseen queries to prefill queries. We therefore project prefill queries into a shared Euclidean space by concatenating all heads:

$$\hat{Q}^{(\ell)} \in \mathbb{R}^{T \times d_q}, \qquad d_q = H d_k,$$

where the i-th row is

$$\hat{q}^{(\ell)}_i = \text{concat}\big(q^{(\ell,1)}_i, \dots, q^{(\ell,H)}_i\big).$$

The rows of $\hat{Q}^{(\ell)}$ form a query manifold: positions with similar hidden representations at layer ℓ tend to share similar attention patterns. We build a nearest-neighbour index over $\hat{Q}^{(\ell)}$ via FAISS, which will later be used to transfer prefill key preferences to decoding queries via nearest-neighbour voting.

Algorithm 1 Prefill: Building Query–Key Mapping

Input: Layer index ℓ, prompt length T, per-head attention $A^{(\ell)}$, per-head queries $Q^{(\ell)}$, fraction $\alpha \in (0, 1]$.
Output: Mapping $\mathcal{M}^{(\ell)}_{\text{prefill}}$, flattened queries $\hat{Q}^{(\ell)}$, nearest-neighbour index.
Compute head-averaged attention:
$\tilde{A}^{(\ell)}_{i,j} \leftarrow \frac{1}{H} \sum_{h=1}^{H} A^{(\ell)}_{h,i,j}$
Apply tail masking to rows of $\tilde{A}^{(\ell)}$
$k \leftarrow \lfloor \alpha T \rfloor$
for $i = 1$ to T **do**
    $\mathcal{T}^{(\ell)}(i) \leftarrow \text{TopK}(\tilde{A}^{(\ell)}_{i,:}, k)$
end for
Flatten into $\hat{Q}^{(\ell)} \in \mathbb{R}^{T \times d_q}$
Build NN index on rows of $\hat{Q}^{(\ell)}$
return $\mathcal{M}^{(\ell)}_{\text{prefill}}, \hat{Q}^{(\ell)}, \text{index}$

3.3.2 Decoding: Query-Space Neighbours and KV Selection

During decoding, the model generates tokens one by one. At step t, layer ℓ produces a new query $q_t^{(\ell)}$ (per head); we again flatten it to

$$\hat{q}_t^{(\ell)} \in \mathbb{R}^{d_q},$$

and use the prefill objects to define a compact retained set $S^{(\ell)}(t)$.

Neighbour queries and semantic voting. We first retrieve the n nearest neighbours of $\hat{q}_t^{(\ell)}$ among prefill queries:

$$\mathcal{N}^{(\ell)}(t) = \mathrm{KNN}\big(\hat{q}_t^{(\ell)}; \hat{Q}^{(\ell)}, n\big), \qquad n = \lfloor \beta T \rfloor,$$

for $\beta \in (0, 1]$. Each neighbour $i \in \mathcal{N}^{(\ell)}(t)$ contributes its precomputed key set $\mathcal{T}^{(\ell)}(i)$, and we aggregate these via a union:

$$S_{\mathrm{map}}^{(\ell)}(t) = \bigcup_{i \in \mathcal{N}^{(\ell)}(t)} \mathcal{T}^{(\ell)}(i). \tag{3.5}$$

This semantic vote captures keys that were important for prefill queries geometrically similar to the current decoding query. Intuitively, if $\hat{q}_t^{(\ell)}$ lies near positions that consistently attend to a particular evidence span or instruction, that span is likely to reappear in $S_{\mathrm{map}}^{(\ell)}(t)$.

Head and tail regions as structural anchors. The semantic map alone is not sufficient: some regions of the prompt are structurally critical even if their instantaneous attention mass is modest. We therefore define two global index sets:

1. A **head region** $S_{\mathrm{head}} = \{1, \ldots, H_{\mathrm{head}}\}$ containing the first H_{head} tokens. These typically encode system prompts, instructions, or task definitions. Dropping them can lead to catastrophic failures (e.g. ignoring the question or changing the requested format), even if their attention weights at deep layers are low.

2. A **tail region** $S_{\mathrm{tail}} = \{T - H_{\mathrm{tail}} + 1, \ldots, T\}$ containing the last H_{tail} prompt tokens. This region captures the most recent user turns or clarifications. While tail tokens are deliberately excluded from map construction by the tail mask (to avoid degeneracy), we always include them at decoding time to preserve recency and disambiguation cues.

Note that head and tail preservation are intentionally symmetric: the head stabilizes global behaviour; the tail preserves recent context, and the semantic map focuses on long-range, nontrivial dependencies in between.

Always keeping generated tokens. Let T denote the prompt length, so that positions $T+1, \ldots, t-1$ correspond to generated tokens before step t. We define

$$S_{\text{gen}}(t) = \{T+1, \ldots, t-1\}.$$

We always retain $S_{\text{gen}}(t)$ for two reasons:

1. **Generation continuity.** Many LongBench tasks require coherent generation over long output spans. During decoding, the model repeatedly attends to previously generated tokens to maintain stylistic and contextual consistency across the sequence. If earlier generated tokens were pruned, the model would lose access to preceding generation context, leading to degraded continuity and unstable outputs.

2. **Asymmetric budget.** In long-context settings, T (prompt length) dominates $t - T$ (decode length) for most of the generation. Pruning aggressively on the prompt side while keeping all generated tokens therefore yields a strong memory reduction with negligible additional cost from $S_{\text{gen}}(t)$.

In addition, keeping all generated tokens implicitly captures short-range locality on the decode side: recent tokens that the model is likely to re-attend to remain available, without needing to be re-identified by the semantic map.

Final retained set and induced sparsity. Combining the semantic map, structural anchors, and decode-side locality yields:

$$S^{(\ell)}(t) = S^{(\ell)}_{\text{map}}(t) \cup S_{\text{head}} \cup S_{\text{tail}} \cup S_{\text{gen}}(t). \tag{3.6}$$

Attention at step t and layer ℓ is then computed only over $S^{(\ell)}(t) \subseteq \{1, \ldots, t-1\}$ instead of the full history:

$$A^{(\ell)}_{t,j} = 0 \quad \text{for } j \notin S^{(\ell)}(t),$$

and the corresponding keys and values outside $S^{(\ell)}(t)$ need not be kept on GPU for that layer and step. The resulting key retention ratio

$$r_t^{(\ell)} = \frac{|S^{(\ell)}(t)|}{t-1}$$

can be tuned via α, β, and $(H_{\text{head}}, H_{\text{tail}})$, enabling a continuum of trade-offs between KV memory and approximation quality.

Algorithm 2 Decoding: Training-Free KV Selection per Layer

1: **Input:** Layer ℓ, time step t, prompt length T, flattened queries $\hat{Q}^{(\ell)}$, prefill mapping $\mathcal{M}^{(\ell)}_{\text{prefill}}$, KNN index, current query $q_t^{(\ell)}$.
2: **Output:** Retained index set $S^{(\ell)}(t)$.
3: Flatten query: $\hat{q}_t^{(\ell)} \leftarrow \text{flatten}(q_t^{(\ell)})$.
4: Retrieve neighbours $\mathcal{N}^{(\ell)}(t)$ in query space.
5: Aggregate map votes: $S^{(\ell)}_{\text{map}}(t) \leftarrow \bigcup_{i \in \mathcal{N}^{(\ell)}(t)} \mathcal{M}^{(\ell)}_{\text{prefill}}(i)$.
6: Construct regions S_{head}, S_{tail}, and $S_{\text{gen}}(t)$.
7: Combine into final set $S^{(\ell)}(t)$ via Eq. (3.6).
8: **return** $S^{(\ell)}(t)$.

In summary, the proposed method uses prefill-time attention to learn a compact semantic index over the prompt, then reuses this index at decoding time through query-space nearest neighbours, while explicitly preserving structurally important regions. All mechanisms are implemented as index selection on top of standard attention, making the approach training-free, model-agnostic, and compatible with existing long-context LLaMA-style deployments.

3.4 Variants of the Mapping Strategy

Within the framework of Sections 3.3–3.3.2, we describe three orthogonal extensions that enrich the base mapping mechanism along semantic, architectural, and confidence dimensions:

1. *position-invariant mapping*, which constructs the Q–K mapping using queries without rotary positional embedding to emphasize semantic similarity over token position;

2. *layer-shared mappings*, which reduces the cost of building query–key maps across layers; and

3. *confidence-aware gating*, which discards unreliable query–key recommendations coming from nearly uniform attention.

3.4.1 Position-Invariant Mapping

The base Q–K mapping relies on query representations that combine semantic content with positional information due to the rotary positional embedding (RoPE) applied during attention. This positional component causes queries that are semantically similar but located at different places in the prompt to be separated in the query space, even when they attend to similar evidence spans.

To decouple content from position, we introduce a *position-invariant* variant that constructs the mapping from queries before applying RoPE. This adjustment causes the query space to cluster tokens by semantic similarity rather than by token location, making it more likely that repeated entities, concepts, or evidence scattered throughout long documents vote for the same key regions.

The structural retention strategy remains unchanged (head, tail, and generated tokens are always preserved), while only the geometry of the query space is altered. Empirically, this variant improves retrieval-style and multi-hop tasks where relevant information may appear at diverse positions in the context, as demonstrated in Section 3.6.

3.4.2 Layer-Shared Mapping

The base method builds a separate prefill mapping $\mathcal{M}^{(\ell)}_{\text{prefill}} : i \mapsto \mathcal{T}^{(\ell)}(i)$ for every layer ℓ. In practice, we observe that attention patterns in early, middle, and late layers are structurally similar: within each depth band, queries tend to look at comparable regions of the prompt. This motivates a layer-shared variant that reuses the expensive part of the mapping across layers.

Let $\mathcal{B} \subset \{1, \dots, L\}$ be a small set of base layers (e.g. $\mathcal{B} = \{1, \ell_{\text{mid}}, L\}$). For each base layer $b \in \mathcal{B}$ we run Algorithm 1 and obtain

$$\mathcal{M}^{(b)}_{\text{prefill}} \quad \text{and} \quad \hat{Q}^{(b)} \in \mathbb{R}^{T \times d_q}.$$

For every non-base layer $\ell \notin \mathcal{B}$, we assign it to the nearest base layer in depth,

$$b^\star(\ell) = \arg\min_{b \in \mathcal{B}} |\ell - b|,$$

and reuse its query–key map:

$$\mathcal{T}^{(\ell)}(i) \equiv \mathcal{T}^{(b^\star(\ell))}(i), \qquad \forall i \in \{1, \dots, T\}.$$

In addiction, only the mapping $\mathcal{M}^{(\ell)}_{\text{prefill}}$ is shared; the query geometry used at decoding remains layer-specific. For each layer ℓ, we still flatten its own queries $Q^{(\ell)}$ into $\hat{Q}^{(\ell)} \in \mathbb{R}^{T \times d_q}$ and build a nearest-neighbour index on $\hat{Q}^{(\ell)}$. Thus, during decoding, a new query $q_t^{(\ell)}$ is always compared against queries from the same layer, but the votes it collects refer to key sets $\mathcal{T}^{(b^\star(\ell))}(i)$ learned in the corresponding base layer.

This design reduces the cost of prefill mapping construction roughly by a factor of $L/|\mathcal{B}|$ (only base layers need to compute top-k sets from full attention), while keeping the decoding stage fully layer-aware. Empirically we find that this approximation has negligible impact on LongBench accuracy when the base layers cover early, middle, and late depths.

Algorithm 3 Layer-Shared Prefill Mapping

Input: Layers $1, \dots, L$; base-layer set $\mathcal{B}$; attention tensors $A^{(\ell)}$; queries $Q^{(\ell)}$; fraction α.
Output: Layer mappings $\mathcal{M}^{(\ell)}_{\text{prefill}}$, query indices for all ℓ.
for each base layer $b \in \mathcal{B}$ **do**
    Run Algorithm 1 on $(A^{(b)}, Q^{(b)}, \alpha)$ to obtain $\mathcal{M}^{(b)}_{\text{prefill}}$ and $\hat{Q}^{(b)}$.
end for
for each non-base layer $\ell \notin \mathcal{B}$ **do**
    Find nearest base layer $b^\star(\ell) \leftarrow \arg\min_{b \in \mathcal{B}} |\ell - b|$.
    Share mapping: $\mathcal{M}^{(\ell)}_{\text{prefill}} \leftarrow \mathcal{M}^{(b^\star(\ell))}_{\text{prefill}}$.
    Flatten layer-specific queries: $\hat{Q}^{(\ell)} \in \mathbb{R}^{T \times d_q}$.
    Build a nearest-neighbour index on rows of $\hat{Q}^{(\ell)}$.
end for
return all $\{\mathcal{M}^{(\ell)}_{\text{prefill}}, \hat{Q}^{(\ell)}\}_{\ell=1}^{L}$.

3.4.3 Confidence-Aware Gating

The base mapping from Eq. (3.4) assumes that the head-averaged attention $\tilde{A}^{(\ell)}_{i,:}$ is sufficiently peaked so that its top-k entries identify meaningful key positions. In practice, however, some queries exhibit nearly uniform or noisy attention—for example, tokens that play a purely syntactic role or appear far from any task-relevant content. For such rows, forcing a non-empty $\mathcal{T}^{(\ell)}(i)$ yields essentially arbitrary key recommendations, which can harm pruning quality and unnecessarily increases the number of

query–key pairs stored in the map, leading to a longer and less efficient $Q \to K$ lookup table.

We therefore introduce a *confidence-aware gating* mechanism that detects diffuse attention rows and discards their contributions to the map. Given a row $\tilde{A}^{(\ell)}_{i,:}$ and its top-k set $\mathcal{T}^{(\ell)}(i)$, we first define normalized probabilities

$$p_{i,j} = \frac{\tilde{A}^{(\ell)}_{i,j}}{\sum_{k=1}^{T} \tilde{A}^{(\ell)}_{i,k} + \varepsilon}, \qquad j = 1, \ldots, T,$$

and let $N_i = |\{j : \tilde{A}^{(\ell)}_{i,j} > 0\}|$ be the number of active entries.

We then consider two scalar confidence scores:

1. **Top-k mass ratio**

$$\rho_i = \frac{\sum_{j \in \mathcal{T}^{(\ell)}(i)} \tilde{A}^{(\ell)}_{i,j}}{\sum_{j=1}^{T} \tilde{A}^{(\ell)}_{i,j} + \varepsilon},$$

   which measures how much of the row's mass is concentrated on the selected keys.

2. **Normalized entropy**

$$H_i = -\sum_{j=1}^{T} p_{i,j} \log p_{i,j}, \qquad \bar{H}_i = \frac{H_i}{\log N_i},$$

   which lies in $[0, 1]$ and approaches 1 when the distribution is close to uniform.

If ρ_i is small or $\bar{H}_i$ is large, the attention pattern is too diffuse to support a reliable semantic recommendation. Instead of committing to arbitrary keys, we set $\mathcal{T}^{(\ell)}(i) = \emptyset$ and let this query rely solely on the structural priors in Eq. (3.6): the global head region S_{head}, the prompt tail S_{tail}, and the generated-token region $S_{\text{gen}}(t)$. During decoding, such low-confidence queries simply cast no votes in Eq. (3.5), which empirically reduces over-pruning on LongBench without increasing the KV ratio.

3.5 Experimental Setup

3.5.1 Datasets, Prompts, and Metrics

We follow the LongBench protocol and report results on seven representative tasks that require processing long contexts:

1. NarrativeQA (**NQA**) for narrative reading comprehension;

Algorithm 4 Confidence-Aware Filtering of a Query Row

Input: Row $\tilde{A}_{i,:}$, candidate set $\mathcal{T}(i)$, thresholds $\tau_{\text{mass}}, \tau_{\text{ent}}$, metric type (TOPK_MASS or ENTROPY).
Compute row sum $Z \leftarrow \sum_j \tilde{A}_{i,j}$ and probabilities $p_j \leftarrow \tilde{A}_{i,j}/(Z+\varepsilon)$.
Compute top-k mass ratio $\rho_i \leftarrow \big(\sum_{j\in\mathcal{T}(i)} \tilde{A}_{i,j}\big)/(Z+\varepsilon)$.
Let $N_i \leftarrow \#\{j : \tilde{A}_{i,j} > 0\}$ and normalized entropy $\bar{H}_i \leftarrow -\big(\sum_j p_j \log p_j\big)/(\log N_i + \varepsilon)$.
if metric is TOPK_MASS **and** $\rho_i < \tau_{\text{mass}}$ **then**
    $\mathcal{T}(i) \leftarrow \emptyset$.
else if metric is ENTROPY **and** $\bar{H}_i > \tau_{\text{ent}}$ **then**
    $\mathcal{T}(i) \leftarrow \emptyset$.
end if
return $\mathcal{T}(i)$.

2. Qasper (**Qsp**) and MultifieldQA-en (**MFQA**) for long-context question answering over academic or multi-field documents;

3. HotpotQA (**Hp**) and 2WikiMultihopQA (**2Wk**) for multi-hop reasoning;

4. Passage retrieval in English (**PR**) for long-context retrieval; and

5. TREC (**TRC**) for long-context classification.

We adopt the standard LongBench metrics: F1 for question answering, ROUGE-style scores for reading comprehension, retrieval accuracy for passage retrieval, and classification accuracy for TREC. For each instance, the official evaluation procedure compares the model output against all reference answers, keeps the best-matching reference, and averages scores across the dataset.

3.5.2 Q–K Mapping Hyperparameters and Variants

Our method introduces a mapping from queries to keys that is constructed during the prompt-processing (prefill) stage and reused during decoding to decide which keys to retain. The mapping is controlled by two global fractions:

1. a *mapping fraction* that specifies what proportion of keys are selected as important for each prompt query, and

2. a *neighbour fraction* that specifies what proportion of prompt queries are retrieved as nearest neighbours in query space for each decoding query.

These fractions correspond to the parameters α and β in Section 3.3. They are set to control the key retention ratio across all pruned configurations; the effective key retention is then further shaped by structural design choices (such as always keeping head/tail regions and generated tokens) and by when and where the mapping is applied.

During generation we also log, for each layer, the ratio between the number of unique keys retained by our method and the full sequence length. Averaging these ratios across layers and tokens yields an empirical key-retention range for each configuration (e.g. 10–12% or 17–19%), which is reported in Table 3.1.

We compare the full-KV baseline with five pruned variants:

1. **Full KV (L2-7B-chat-4k)**: the unmodified model using full attention over the entire KV cache; this serves as the reference.

2. **Q–K**: the basic form of query–key mapping in which a semantic mapping is constructed during the prompt-processing stage at every transformer layer, and decoding queries attend only to mapped prompt keys together with all previously generated tokens.

3. **Q–K Pro**: a structurally enhanced variant that always retains a head segment at the beginning of the prompt, a tail segment at the end, and all generated tokens. The mapping is constructed only on the remaining prompt span in order to avoid purely locality-driven keys and to preserve reasoning continuity across the generated outputs.

4. **Non-positional Q–K Pro**: identical to Q–K Pro in structural design, but the mapping is built from query and key representations with positional information removed so that the query space reflects primarily semantic similarity rather than token position. This adjustment encourages clustering by content rather than proximity and improves robustness on tasks where evidence may appear at highly variable locations within the input.

5. **Layer-wise Q–K Pro**: a layer-sparse variant of Q–K Pro in which only a small subset of designated transformer layers construct and apply the mapping, while the remaining layers rely solely on the structural retention of head, tail, and generated tokens. This reduces the computational overhead of building multiple mapping indices and probes how layer depth affects the formation of semantic relevance.

6. **Confidence Q–K Pro**: a confidence-aware extension of Q–K Pro in which the mapping is gated on the sharpness of the model's attention distribution. When attention is confidently concentrated, the mapping is applied; when it is diffuse, the method falls back to structural retention alone. This mechanism prevents aggressive pruning in ambiguous contexts and is particularly relevant for multi-hop queries where the model may need to preserve multiple competing evidence spans.

3.6 Results on LongBench

3.6.1 Compression–Performance Trade-off on Qasper

To characterize how Q–K pruning behaves under different memory budgets, we evaluate Qasper while varying the retained-key fraction from 12% to 17%, keeping the structural retention strategy (head/tail + generated tokens) fixed. Figure 3.2 reports accuracy as a function of key retention.

Within the compressed range (12–17%), Qasper accuracy increases monotonically as additional prefill-selected keys are preserved. At approximately 17% retention, accuracy approaches the full-KV baseline while using over 80% fewer keys. This indicates that pruning can remove substantial amounts of KV state without harming multi-hop evidence retrieval.

Beyond the near-linear improvement observed between 12% and 17% retention, the curve suggests a diminishing-return regime as the retained-key fraction approaches the full-KV setting. In other words, a relatively small subset of prefill-selected keys already captures most of the task-relevant context, while additional keys primarily provide marginal refinements rather than qualitative gains. This behavior supports the hypothesis that long-context reasoning in Qasper relies on a sparse set of salient evidence spans rather than uniformly distributed information. Consequently, Q–K pruning offers an effective operating point where memory consumption can be drastically reduced while maintaining accuracy close to the uncompressed baseline, making it particularly attractive for long-context inference under strict memory constraints.

3.6.2 Benchmark Comparison on LongBench

Table 3.1 reports results across seven LongBench tasks for the full-KV baseline and five Q–K pruning variants. Although the pruned configurations retain only 10–19% of the KV cache, several maintain competitive accuracy relative to the unpruned model.

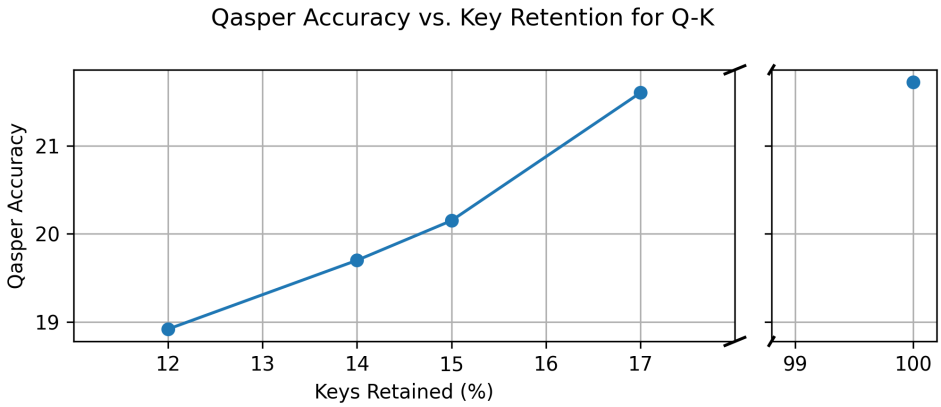


Figure 3.2: Qasper accuracy as a function of key retention for Q–K pruning. The left panel shows monotonic accuracy gains as the retained-key fraction increases from 12% to 17% for a head/tail-preserving Q–K variant. The slim right panel isolates the full-KV baseline at 100% of keys, showing that several compressed configurations outperform the unpruned model. The broken x-axis highlights the gap between the compressed regime and the full-KV point.

This demonstrates that substantial KV sparsification is feasible without retraining and without catastrophic accuracy degradation.

The **Full KV** configuration achieves the strongest scores on NarrativeQA, Qasper, 2WikiMultihopQA, and TREC, reflecting the advantage of dense context for tasks requiring long-range retrieval or structured classification. At the same time, multiple pruned variants outperform Full KV on other workloads: **QK** attains the best results on MultifieldQA-en and HotpotQA, while **Pro-NP** is highest on passage retrieval. The fact that these gains occur under 80–90% KV reduction indicates that dense KV storage contains substantial redundancy that can be removed without harming—and in some cases improving—semantic retrieval and reasoning.

Overall, these results suggest that training-free KV pruning can reduce memory requirements significantly while preserving performance across heterogeneous long-context tasks.

3.6.3 Ablation: Positional vs. Non-Positional Mapping

We compare *Non-positional Q–K Pro* (Pro-NP) against the standard *Q–K Pro* to isolate the role of positional information during prefill. Both variants share the same structural retention (head/tail + generated tokens), but Pro-NP removes rotary positional embeddings before nearest-neighbor lookup, producing a more content-based

Model	Keys(%)	NQA	Qsp	MFQA	Hp	2Wk	PR	TRC
Full KV	100	**19.13**	**21.72**	36.68	27.72	**31.78**	5.5	**64.5**
QK	17–19	18.2	20.81	**37.62**	**31.02**	27.49	7.5	**64.5**
Pro	11–13	17.72	20.26	35.65	27.03	30.18	7.0	**64.5**
Pro-NP	10–12	17.79	21.12	37.15	27.83	30.08	**8.5**	64.0
Pro-LW	11–13	17.50	18.00	31.84	26.82	29.93	3.75	55.5
Pro-CF	11–13	17.24	20.22	35.36	26.64	31.08	7.0	**64.5**

Table 3.1: LongBench performance comparison for six KV pruning configurations. **Full KV** uses unpruned attention. **QK** is the basic query–key mapping applied at every layer. **Pro** adds structural head/tail & generated-token retention. **Pro-NP** removes positional information when constructing the mapping ("non-positional"). **Pro-LW** applies mapping sparsely at layers 0, 5, 14, and 20 ("layer-wise"). **Pro-CF** introduces entropy-based gating ("confidence"). Columns: Keys = retained key percentage. Best results are in **bold**; second-best are underlined.

query space.

Despite retaining fewer keys (10–12% vs. 11–13%), Pro-NP achieves higher scores on Qasper (21.12 vs. 20.26), MultifieldQA-en (37.15 vs. 35.65), and passage retrieval (8.5 vs. 7.0). These improvements indicate that position-invariant similarity helps to find semantically related evidence that may appear at distant or irregular positions.

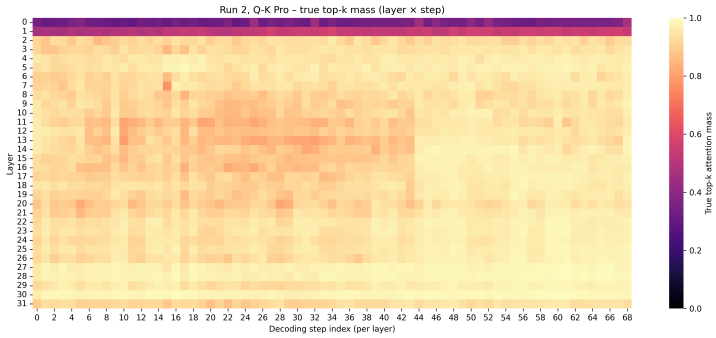
3.6.4 Ablation: Layer Sharing

Layer-wise Q–K Pro applies mapping only at layers 0, 5, 14, and 20, reducing mapping overhead but leaving intermediate layers without semantic key selection. The performance degradation is substantial: MultifieldQA-en drops from 35.65 to 31.84, passage retrieval from 7.0 to 3.75, and TREC from 64.5 to 55.5. These reductions suggest that relevance decisions evolve with depth: early layers capture local and syntactic patterns, while deeper layers integrate global semantic relations. Limiting mapping to a sparse set of layers prevents intermediate layers from refining which keys to retain, which negatively impacts retrieval-heavy tasks.

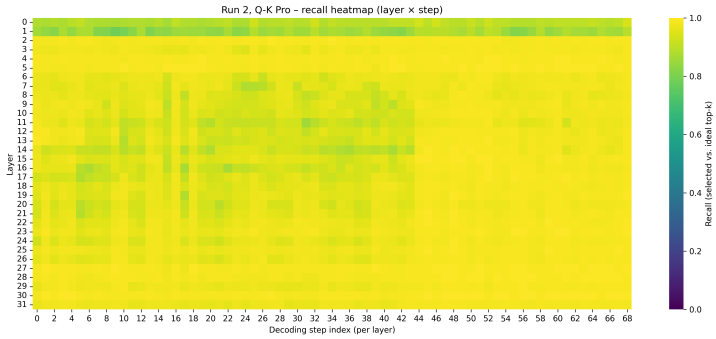
3.6.5 Ablation: Confidence Gating

Confidence Q–K Pro adds an entropy-based gate that applies mapping when attention is sharp and disables it when attention is diffuse. This prevents aggressive pruning in cases where evidence spans are ambiguous or distributed.

Relative to Pro, the differences are modest: MultifieldQA-en and HotpotQA de-



(a) True top-k attention mass



(b) Pruned vs. full recall

Figure 3.3: Layer–step heatmaps for **Q–K Pro** on Qasper (sample 2), which retains only $\sim$ **12**% of keys. Top: true top-k attention mass (using top $\sim 12\%$ keys) shows how concentrated the model's ideal attention distribution is across depth and decoding time. Bottom: recall between pruned keys and true top $\sim 12\%$ keys. High recall aligns with regions of concentrated attention mass, indicating that Q–K Pro preserves salient keys even under aggressive sparsification.

crease slightly, while 2WikiMultihopQA improves (31.08 vs. 30.18) and passage retrieval remains stable. The improvement on 2WikiMultihopQA is consistent with the dataset's dispersed multi-hop evidence, where suppressing uncertain pruning prevents removal of relevant context.

3.6.6 Depth–Time Dynamics of Attention Mass and Recall

To analyze how pruning decisions evolve during long-context decoding, we examine the dynamics of attention mass and the corresponding retention of ground true keys as a function of both layer depth and generation step. The top panel of Fig. 3.3 shows

that true top-k attention mass is not uniform: early layers exhibit diffuse attention, while deeper layers concentrate mass on a small subset of positions, consistent with evidence aggregation and multi-hop reasoning.

The bottom panel reports the recall between the pruned key set and the oracle top-k keys. Recall remains consistently high (≥ 0.9 for the majority of layer–step pairs), demonstrating that sparsification largely preserves the keys that the unpruned model relies on. This aligns with the strong task-level results in Table 3.1, despite retaining only 10–13% of KV entries.

Importantly, localized dips in recall coincide with regions where the oracle attention redistributes its mass, typically at boundaries between reasoning phases. These correspond to semantic transitions (e.g., retrieving new evidence spans) where the model temporarily shifts which tokens are relevant. Such transitions highlight that effective pruning must adapt to both depth and time rather than rely on static filters.

Overall, this analysis reveals that KV redundancy exhibits structured depth–time dependencies: attention mass becomes more concentrated in deeper layers, and pruning accuracy must follow these transitions to avoid degrading semantic retrieval. Q–K Pro succeeds at this alignment, explaining its ability to aggressively reduce memory while preserving long context performance.

3.7 Discussion

The results show that training-free Q–K mapping is a viable approach for KV pruning in long-context LLaMA-2 inference. Several variants operate with retention ratios as low as 10–13% while remaining competitive with the full-KV baseline across a diverse set of LongBench tasks. This indicates that dense KV caching contains substantial redundancy and that only a small subset of prompt keys is needed to support long-context retrieval and reasoning.

A second observation concerns the geometry of the query space. The non-positional variant outperforms the positional one on evidence-heavy tasks such as Qasper, MultiFieldQA, and passage retrieval, despite retaining slightly fewer keys. Removing rotary positional components before nearest-neighbor mapping yields a more content-based semantic similarity measure, which is beneficial when relevant evidence may appear at distant or variable locations within long documents.

A third observation relates to depth and uncertainty. Layer-wise ablations show that relevance estimation evolves across transformer depth, and that applying the mapping at all layers is important for preserving retrieval and classification accuracy.

Confidence gating further stabilizes pruning in settings where attention is diffuse, improving performance on multi-hop benchmarks by avoiding premature elimination of uncertain evidence spans. These results suggest that both depth and confidence are meaningful axes for controlling inference-time sparsification policies.

Overall, Q–K mapping functions as more than a KV compression mechanism. By removing weakly relevant context and retaining semantically informative keys, it regularizes attention during decoding without modifying model parameters. This behavior enables substantial reductions in KV memory footprint while maintaining accuracy, making the approach compatible with zero-shot and production inference settings where GPU memory allocated to KV caching is a primary deployment constraint.

Chapter 4

BlocKV: Blockwise Query Similarity for KV Cache Pruning

In this chapter we focus on *long reasoning* rather than long input contexts. Modern reasoning models such as DeepSeek-R1 can generate tens of thousands of chain-of-thought (CoT) tokens even when the initial prompt is short. In these settings, the dominant bottleneck is not the prefill phase but the autoregressive decoding phase: the key–value (KV) cache grows linearly with the number of generated tokens, quickly exhausting GPU memory and slowing down inference.

We propose a *training-free*, *plug-and-play* attention modification, which we call **BlocKV**, to reduce the KV cache during long reasoning. The core idea is to group consecutive decoding steps into blocks whose queries are geometrically similar and show highly similar attention patterns. For tokens inside such a block, we restrict attention to a compact set of keys discovered when the block was created, so that only a fraction of the KV cache needs to be accessed and stored. Our implementation wraps a LLaMA-style attention layer and is instantiated on AIME 2024 and MATH-500 math reasoning tasks.

A central empirical observation is that, during long reasoning, *query similarity and attention similarity follow similar patterns over time.* This chapter formalizes that observation and uses it as the foundation for block construction and pruning.

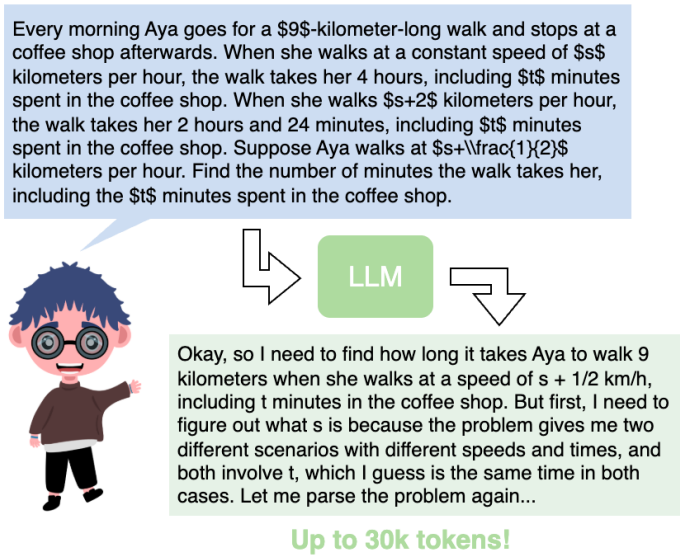


Figure 4.1: Illustration of a long-form mathematical reasoning query and model response. The input describes a structured word problem that requires the model to set up unknown variables, consider multiple hypothetical scenarios, and integrate constraints over several steps before producing the final answer. Such tasks benefit from extended context windows (up to 30k tokens) that allow models to retain intermediate deductions and revisit earlier premises during multi-stage reasoning.

4.1 Problem Setting

We consider an autoregressive transformer with L decoder layers, hidden size d, and H attention heads. Let $x_{1:T_0} = (x_1, \ldots, x_{T_0})$ denote the (short) input prompt, and let $y_{T_0+1:T}$ denote the chain-of-thought tokens generated by the model up to step T, where typically $T \gg T_0$. At decoding step $t > T_0$, the model outputs a new token

$$y_t \sim p_\theta(\cdot \mid x_{1:T_0}, y_{T_0+1:t-1}),$$

and we write $t = 1$ for the first decoding step ($t = T_0 + 1$ in absolute indexing) when convenient.

At layer $\ell \in \{1, \ldots, L\}$ and step t, the multi-head self-attention module computes

$$Q_t^{(\ell)} \in \mathbb{R}^{H \times 1 \times d_h}, \tag{4.1}$$

$$K_{1:t-1}^{(\ell)} \in \mathbb{R}^{H \times (t-1) \times d_h}, \tag{4.2}$$

$$V_{1:t-1}^{(\ell)} \in \mathbb{R}^{H \times (t-1) \times d_h}, \tag{4.3}$$

where $d_h = d/H$ is the per-head dimension. The (causal) attention distribution and output are

$$A_t^{(\ell)} = \mathrm{softmax}\left(\frac{Q_t^{(\ell)} {K_{1:t-1}^{(\ell)}}^\top}{\sqrt{d_h}}\right) \in \mathbb{R}^{H\times 1\times(t-1)}, \tag{4.4}$$

$$O_t^{(\ell)} = A_t^{(\ell)} V_{1:t-1}^{(\ell)} \in \mathbb{R}^{H\times 1\times d_h}. \tag{4.5}$$

The KV cache at layer ℓ after t decoding steps has size

$$\underbrace{(t-1)Hd_h}_{K\text{ cache}} + \underbrace{(t-1)Hd_h}_{V\text{ cache}} = 2(t-1)d,$$

so that total KV memory over all layers scales as

$$M_{\mathrm{KV}}(t) = 2L(t-1)d \quad \propto \quad t.$$

When t reaches 10K–30K CoT tokens, the KV cache can exceed the parameter footprint, dominating both memory and runtime.

Our goal is to define, for each layer ℓ and decoding step t, a *retained key index set*

$$S^{(\ell)}(t) \subseteq \{1, \ldots, t-1\}$$

such that attention accesses only $K^{(\ell)}_{S^{(\ell)}(t)}$ and $V^{(\ell)}_{S^{(\ell)}(t)}$:

$$\tilde{A}_t^{(\ell)} = \mathrm{softmax}\left(\frac{Q_t^{(\ell)} {K^{(\ell)}_{S^{(\ell)}(t)}}^\top}{\sqrt{d_h}}\right), \qquad \tilde{O}_t^{(\ell)} = \tilde{A}_t^{(\ell)} V^{(\ell)}_{S^{(\ell)}(t)}, \tag{4.6}$$

with a small *retention ratio*

$$r_t^{(\ell)} = \frac{|S^{(\ell)}(t)|}{t-1} \ll 1,$$

while preserving task accuracy on long reasoning benchmarks such as AIME 2024.

4.2 Background: Long Reasoning and KV Redundancy

Unlike the LongBench setting in Chapter 3, where the context is long but decoding is short, AIME 2024-style tasks often use short prompts but extremely long CoT traces. Let $T_0 \leq 512$ and $T \geq 10{,}000$. Then the cumulative KV memory over the entire

generation is dominated by decode-side tokens:

$$\frac{M_{\text{KV, decode}}}{M_{\text{KV, prefill}}} \approx \frac{T - T_0}{T_0} \gg 1.$$

In practice we observe that more than 90% of the KV cache belongs to generated tokens for long reasoning runs.

The structure of long CoT also differs from long document prompts:

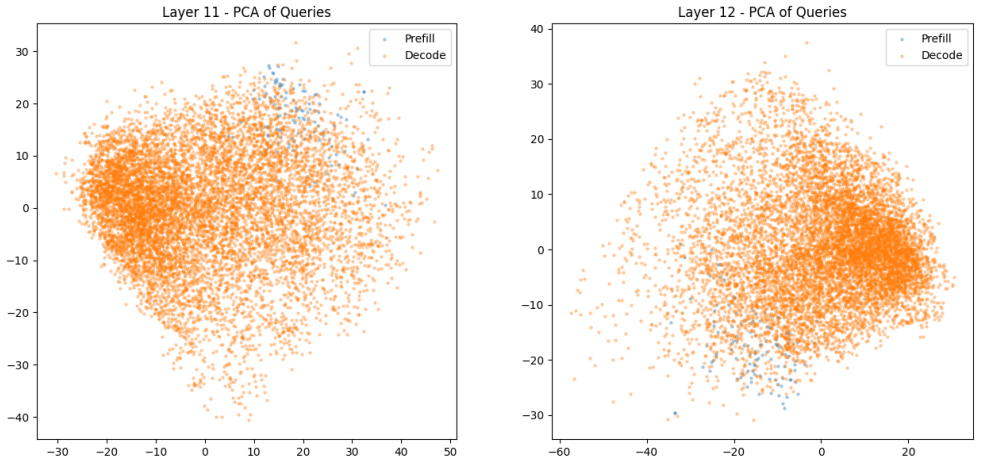
1. Reasoning unfolds in *phases*: the model focuses on one subproblem or intermediate quantity for dozens of tokens, then jumps to another. Within a phase, attention patterns change slowly; across phases, they change abruptly.

2. Many tokens inside a phase are syntactic glue or verbal elaborations, whose attention patterns are almost identical to neighbouring tokens. This suggests high redundancy in the KV cache.

These observations align with recent findings in efficient LLM inference, which show that KV cache growth during autoregressive decoding is a primary bottleneck in long-generation settings, motivating selective compression or eviction strategies [30, 9, 12]. At the same time, prior analyses of transformer attention reveal that neighbouring tokens often exhibit highly correlated attention behaviour, implying substantial redundancy in token-level representations [6]. A naive pruning strategy based solely on per-token attention mass may misinterpret these patterns: syntactic tokens with locally high self-attention could be preserved, while equally important neighbours get dropped. Instead, we look for a more block-wise signal: the temporal structure of query and attention similarity.

4.3 Empirical Patterns: Query–Query and Attention–Attention Similarity

To characterize the internal dynamics of long chain-of-thought decoding, we measure how attention distributions and intermediate representations evolve over time across layers. We find that both queries and attention rows exhibit extended intervals of high self-similarity punctuated by sharp transitions, forming blockwise structures that are synchronized across depth. Moreover, the temporal patterns of query similarity and attention similarity are closely aligned, indicating that the geometry of the query space provides a compact proxy for changes in attention. The remainder of this section



(a) Layer 11 **(b)** Layer 12

Figure 4.2: PCA projection of query vectors during prefilling (blue) and decoding (orange) on a long chain-of-thought example. Decoding queries depart from the prefilling cluster and expand into new regions of the space, indicating geometric drift during generation.

quantifies these phenomena using layerwise projections, pairwise cosine metrics, and lagged similarity heatmaps.

4.3.1 Geometric Drift of Queries Across Prefill and Decoding

To examine how the geometry of the query space evolves during long chain-of-thought decoding, we project all query vectors from both the prefilling and decoding phases into a two-dimensional subspace using PCA. For each layer ℓ, we collect the per-step (head-averaged) queries

$$q_1^{(\ell)}, \ldots, q_{T_{\text{prefill}}}^{(\ell)}, \quad q_{T_{\text{prefill}}+1}^{(\ell)}, \ldots, q_{T_{\text{prefill}}+T_{\text{decode}}}^{(\ell)},$$

fit a PCA model on the concatenated set, and label points by whether they originate from prefilling or decoding.

Figure 4.2 shows representative layers. In both layers, the prefilling queries occupy a relatively compact region, whereas decoding queries expand into a larger and directionally biased region that departs from the prefilling manifold. This reveals a geometric drift as the model generates intermediate reasoning steps.

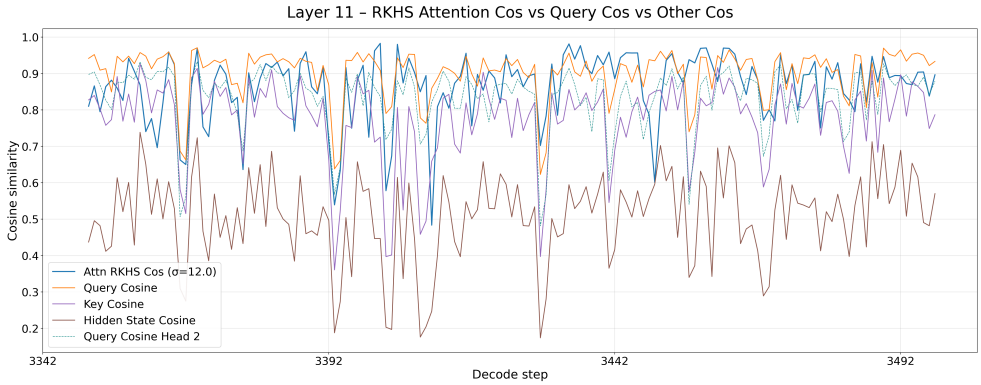


Figure 4.3: Cosine similarity between consecutive decoding steps for problem 67 at layer 11. We plot RKHS-smoothed attention similarity (*Attn RKHS Cos*), mean query cosine, mean key cosine, hidden-state cosine, and a representative per-head query cosine (head 2). All curves exhibit a block-wise pattern: extended intervals of high similarity punctuated by sharp drops, indicating transitions between reasoning segments. The close alignment between the attention and query/key curves suggests that changes in the attention pattern are largely driven by the underlying query–key geometry.

Interpretation. The decoding query distribution is not well-approximated by the prefilling distribution alone. Specifically: (i) decoding queries explore new directions absent during prefilling, and (ii) the decoding support is strictly larger. Consequently, KV pruning strategies that depend solely on prefilling queries cannot fully capture which keys remain relevant deep into decoding. In later sections we address this by incorporating decoding-side query geometry when selecting key subsets.

4.3.2 Layer-wise Similarity Dynamics During Long Reasoning

To characterize how attention patterns and internal representations evolve during long chain-of-thought decoding, we compute stepwise cosine similarities between consecutive decoding states at each transformer layer ℓ. For a decoding sequence $(x_t)_{t=1}^{T}$, we examine four signals at every layer:

$$a_t^{(\ell)}, \qquad q_t^{(\ell)}, \qquad k_t^{(\ell)}, \qquad h_t^{(\ell)},$$

where $a_t^{(\ell)} \in \mathbb{R}^{L_t}$ is the head-averaged attention row over L_t positions, and $q_t^{(\ell)}, k_t^{(\ell)}, h_t^{(\ell)} \in \mathbb{R}^d$ denote the mean query, mean key, and hidden state vectors (with per-head query vectors $q_t^{(\ell,h)} \in \mathbb{R}^{d_h}$).

4.3. Empirical Patterns: Query–Query and Attention–Attention Similarity

RKHS cosine for attention. Direct cosine on $a_t^{(\ell)}$ is sensitive to low-mass positions and sequence-length effects. We therefore compute a position-aware similarity in an RKHS induced by a Gaussian positional kernel. After trimming boundary tokens and selecting the union of top-k salient positions for $(a_t^{(\ell)}, a_{t+1}^{(\ell)})$, we form

$$K_{ij} = \exp\Big(-\tfrac{(p_i-p_j)^2}{2\sigma^2}\Big), \qquad \sigma = 12,$$

where p_i indexes token positions. The RKHS cosine is then

$$\cos_t^{\mathrm{RKHS},(\ell)} = \frac{(a_t^{(\ell)})^\top K\, a_{t+1}^{(\ell)}}{\sqrt{(a_t^{(\ell)})^\top K\, a_t^{(\ell)}}\,\sqrt{(a_{t+1}^{(\ell)})^\top K\, a_{t+1}^{(\ell)}}}. \tag{4.7}$$

Vector cosine for queries, keys, and states. For the vector components we use the standard cosine similarity:

$$\cos_t^{X,(\ell)} = \frac{\langle x_t^{(\ell)},\ x_{t+1}^{(\ell)}\rangle}{\|x_t^{(\ell)}\|_2\,\|x_{t+1}^{(\ell)}\|_2}, \qquad X \in \{Q, K, H\}, \tag{4.8}$$

with an analogous per-head form

$$\cos_t^{Q,h,(\ell)} = \frac{\langle q_t^{(\ell,h)},\ q_{t+1}^{(\ell,h)}\rangle}{\|q_t^{(\ell,h)}\|_2\,\|q_{t+1}^{(\ell,h)}\|_2}.$$

Empirical observation. Across layers $\ell = 0, \ldots, L-1$ and across multiple long reasoning problems, the four similarity traces exhibit consistent structure. In particular: (i) the RKHS attention cosine forms a block-wise pattern, staying high for extended spans and then dropping sharply before stabilizing again, (ii) the mean query and mean key cosines closely track the RKHS curve across blocks, and (iii) the hidden-state cosine fluctuates more strongly while remaining correlated with attention and query–key similarity. These trends persist throughout depth, indicating that attention evolution during long decoding is strongly coupled to the geometry of the query–key space.

Example at layer 11. Figure 4.3 illustrates the signals for layer 11 on problem 67 from AIME 2024 dataset. The RKHS attention cosine (blue) remains near 1 except for occasional drops, while the query and key cosines (orange, purple) follow almost the same profile. The hidden-state cosine (brown) changes more aggressively, reflecting

that token embeddings absorb larger semantic updates even when attention structure remains stable. Per-head cosine traces show similar structure with additional head-specific variability.

In summary, long chain-of-thought decoding progresses through a regime where attention, queries, and keys evolve in a coordinated and gradually-varying manner at all layers. This layer-synchronous behavior suggests that attention updates are not arbitrary but are tightly constrained by the incremental drift of the underlying representation space.

4.3.3 Sliding-Window Similarity Heatmaps

The one-step similarities in §4.3.2 quantify how representations evolve locally from t to $t{+}1$. To study stability over longer ranges, we construct *lagged* similarity matrices at each layer ℓ.

For head-averaged queries $\bar{q}_t^{(\ell)} = \frac{1}{H}\sum_{h=1}^{H} q_t^{(\ell,h)}$ and head-averaged attention rows $\bar{a}_t^{(\ell)} = \frac{1}{H}\sum_{h=1}^{H} A_{t,:}^{(\ell,h)}$, we fix a maximum lag K (here $K = 100$) and define, for $t = K{+}1, \ldots, T$ and $1 \le k \le K$,

$$S_q^{(\ell)}(t,k) = \cos\big(\bar{q}_t^{(\ell)},\ \bar{q}_{t-k}^{(\ell)}\big), \tag{4.9}$$

$$S_a^{(\ell)}(t,k) = \cos^{\mathrm{RKHS}}\big(\bar{a}_t^{(\ell)},\ \bar{a}_{t-k}^{(\ell)}\big), \tag{4.10}$$

where $\cos^{\mathrm{RKHS}}$ is the Gaussian-kernel similarity from Eq. (4.7). Each matrix is visualized as a heatmap with rows indexed by the current step t and columns indexed by the lag k.

Figure 4.4 shows the resulting matrices for layer 10 on AIME 2024 problem 67. The left panel plots the RKHS attention similarity $S_a^{(\ell)}(t,k)$; the right panel plots the mean-query similarity $S_q^{(\ell)}(t,k)$. Both heatmaps display a pronounced block-wise structure: within a block, similarities remain high over a wide range of lags, forming bright diagonal bands; at block boundaries, similarity drops simultaneously across many lags, producing sharp horizontal seams.

Repeating this construction across layers and for individual heads (using $q_t^{(\ell,h)}$ instead of $\bar{q}_t^{(\ell)}$) yields the same pattern: long reasoning phases correspond to high-similarity blocks in both query and attention space, while semantic shifts produce coherent breaks in the heatmaps. This supports the view that query geometry provides a stable proxy for attention phases, which we later exploit for block segmentation and KV pruning.

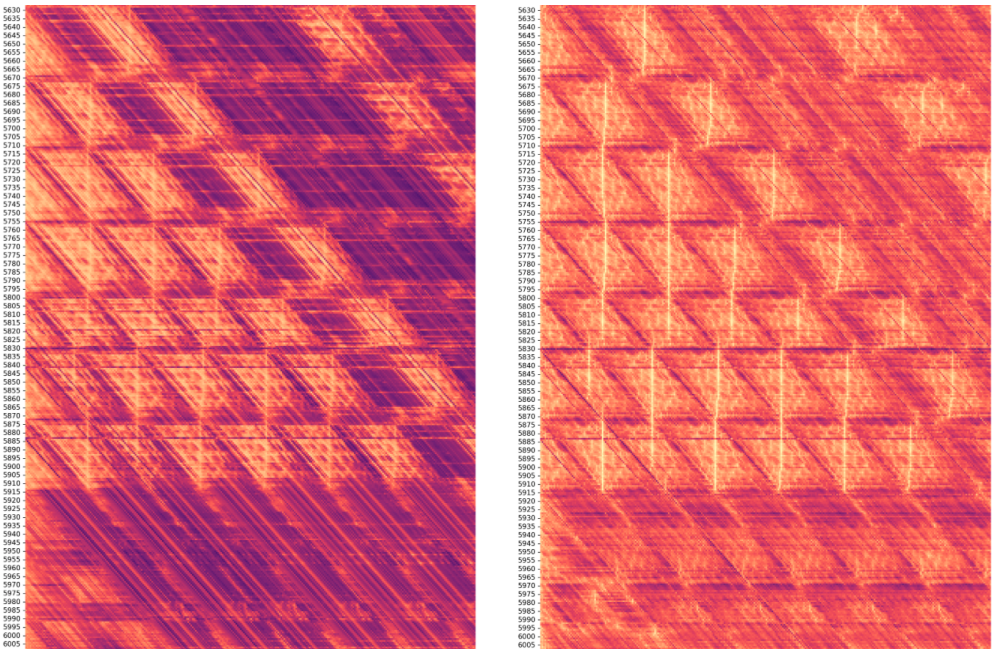


Figure 4.4: Sliding-window similarity heatmaps for layer 10 on AIME 2024 problem 67. **Left:** RKHS attention similarity $S_a^{(\ell)}(t,k)$ from Eq. (4.10). **Right:** mean-query cosine similarity $S_q^{(\ell)}(t,k)$ from Eq. (4.9).

4.4 Proposed Method: BlocKV

We now describe our BlocKV mechanism, which uses the patterns in Section 4.3 to prune KV accesses during long reasoning.

4.4.1 Blocks, Reference Queries, and Thresholds

At each layer ℓ we maintain a list of blocks

$$\mathcal{B}^{(\ell)} = \{B_1^{(\ell)}, B_2^{(\ell)}, \dots\}, \qquad B_m^{(\ell)} \subset \mathbb{N},$$

where each block is a contiguous interval of decoding positions. The current block is $B_{m^\star}^{(\ell)}$.

Each block stores:

1. a set of *reference queries* $R_m^{(\ell)} \subseteq \{\bar{q}_t^{(\ell)} : t \in B_m^{(\ell)}\}$, typically the first B_{ref} queries in the block;

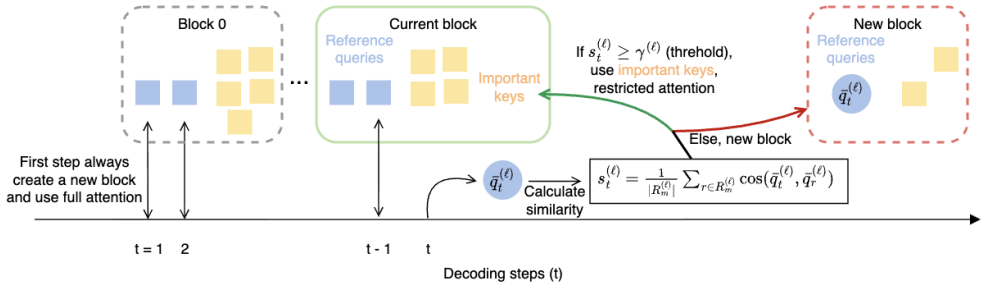


Figure 4.5: Block construction and restricted attention at layer ℓ. At decoding step t, the mean query $\bar{q}_t^{(\ell)}$ is compared against the reference queries $R_m^{(\ell)}$ of the current block $B_m^{(\ell)}$ to obtain a similarity score $s_t^{(\ell)}$. If $s_t^{(\ell)} \geq \gamma^{(\ell)}$ and $|B_m^{(\ell)}| < B_{\max}$, the block is extended and restricted attention is applied using the stored important keys $K_m^{(\ell)}$. Otherwise a new block is created at t, full attention is used, and a fresh set of important keys is initialized. The first decoding step always forms a new block and uses full attention.

2. a set of *important keys* $K_m^{(\ell)} \subseteq \mathbb{N}$ (positions in the sequence);
3. the maximum decode index seen in the block, $t_{m,\max}^{(\ell)} = \max B_m^{(\ell)}$.

Given a new query $\bar{q}_t^{(\ell)}$ and current block $B_m^{(\ell)}$, we define the block similarity as

$$s_t^{(\ell)} = \frac{1}{|R_m^{(\ell)}|} \sum_{r \in R_m^{(\ell)}} \cos\big(\bar{q}_t^{(\ell)}, \bar{q}_r^{(\ell)}\big). \tag{4.11}$$

We then compare $s_t^{(\ell)}$ to a layer-specific threshold $\gamma^{(\ell)}$ (e.g. $\gamma^{(\ell)} = 0.80$ for most layers and slightly higher for early and final layers):

1. If $s_t^{(\ell)} \geq \gamma^{(\ell)}$ and the block size is below a maximum $B_{\max}$, we *extend* the current block and run restricted attention using the existing important keys $K_m^{(\ell)}$.
2. Otherwise we *start a new block* at t, run full attention, and initialize a fresh set of important keys.

The first decoding step always starts a new block and is processed with full attention.

4.4.2 Discovering Important Keys from Attention

Whenever we process a token with full attention (block creation or refresh), we use its attention row to update the block's key set $K_m^{(\ell)}$. Let $\alpha_t^{(\ell)} \in \mathbb{R}^{t-1}$ denote the head-

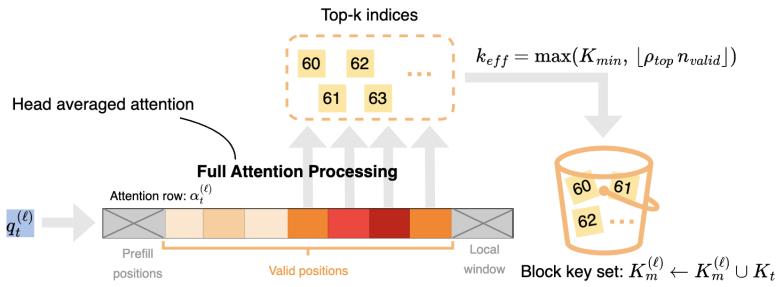


Figure 4.6: Updating block keys during full attention. At decoding step t we compute the head-averaged attention row $\alpha_t^{(\ell)}$, extract the top-k_{eff} scoring positions among the valid region, and store them as a temporary key set K_t. These keys are then merged into the block key set $K_m^{(\ell)}$ for future restricted-attention updates.

averaged attention over keys (last dimension of $A_t^{(\ell)}$). We define an *entropy-aware top-k* procedure that extracts a set of important key indices $\mathcal{K}_t$:

1. Mask out prefill positions and the most recent tail window of size W_{tail} to avoid trivial locality bias during key discovery.
2. We select $k_{\text{eff}} = \max(K_{\min}, \lfloor \rho_{\text{top}} \cdot n_{\text{valid}} \rfloor)$ indices with the highest $\alpha_{t,j}^{(\ell)}$ among valid positions.

The block's key set is updated by

$$K_m^{(\ell)} \leftarrow K_m^{(\ell)} \cup \mathcal{K}_t.$$

In practice, $K_m^{(\ell)}$ contains at most a few hundred unique indices, even for blocks spanning hundreds of tokens.

4.4.3 Restricted Attention Inside a Block

For a step t that stays within the current block $B_m^{(\ell)}$ we do not recompute important keys from full attention; instead, we restrict attention using the existing $K_m^{(\ell)}$. Let T_0 be the prefill length. We define three structural regions:

1. **Prefill region**: $S_{\text{prefill}} = \{1, \ldots, T_0\}$.
2. **Block tail**: a local window anchored at the block's reference decode step,

$$S_{\text{tail}}^{(\ell)}(t) = \{j : j \geq j_{\text{ref}}^{(\ell)}\},$$

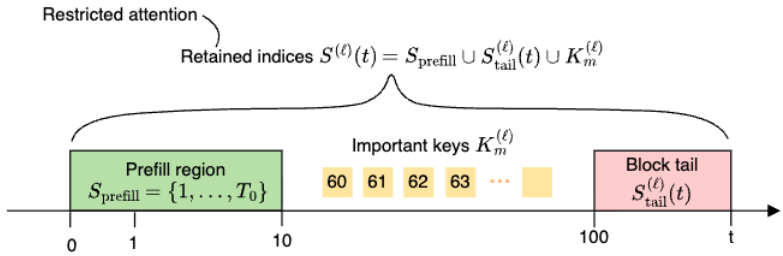


Figure 4.7: Illustration of the retained index set used during restricted attention at step t. The model keeps three disjoint subsets of token positions: the prefill region $S_{\text{prefill}} = \{1, \ldots, T_0\}$, the current block's important keys $K^{(\ell)}_m$, and the block tail $S^{(\ell)}_{\text{tail}}(t)$. Their union defines the retained index set $S^{(\ell)}(t) = S_{\text{prefill}} \cup S^{(\ell)}_{\text{tail}}(t) \cup K^{(\ell)}_m$, over which restricted attention is performed instead of full attention.

where $j^{(\ell)}_{\text{ref}}$ is the key index corresponding to $t^{(\ell)}_{m,\max}$.

3. **Important keys**: $K^{(\ell)}_m$.

The final retained set is

$$S^{(\ell)}(t) = S_{\text{prefill}} \cup S^{(\ell)}_{\text{tail}}(t) \cup K^{(\ell)}_m. \tag{4.12}$$

We then compute restricted attention as in Eq. (4.6), using only keys and values at indices in $S^{(\ell)}(t)$.

Algorithm 5 corresponds to our detailed implementation of BlocKV, where the per-layer object stores block metadata, important keys, and the prefill length.

4.5 Experimental Setup

4.5.1 Model and Implementation

We implement the proposed **BlocKV** mechanism by replacing the model's native attention layer. Prefilling uses standard full attention, whereas during decoding BlocKV forms similarity-based blocks and restricts attention to a small set of relevant keys while falling back to full attention when needed. The mechanism is applied to an 8B-parameter LLaMA-style model in `bfloat16`, and experiments are run on a single NVIDIA A100 GPU.

Algorithm 5 BlocKV Attention (per layer)

Require: Layer ℓ, prefill length T_0, thresholds $\gamma^{(\ell)}$, $B_{\max}$
Initialize block list $\mathcal{B}^{(\ell)}$
Prefill: run full attention and set `prefill_len` $\leftarrow T_0$
for $t = 1$ to T_{decode} **do**
    Compute $Q_t^{(\ell)}$ and mean query $\bar{q}_t^{(\ell)}$
    if no active block **then**
        Start new block m, set $B_m^{(\ell)} \leftarrow \{t\}$ and $m^\star \leftarrow m$
        Run full attention; extract keys $\mathcal{K}_t$ and set $K_m^{(\ell)} \leftarrow \mathcal{K}_t$, $R_m^{(\ell)} \leftarrow \{\bar{q}_t^{(\ell)}\}$
    else
        Let $m \leftarrow m^\star$
        Compute block similarity $s_t^{(\ell)}$
        if $s_t^{(\ell)} < \gamma^{(\ell)}$ or $|B_m^{(\ell)}| \geq B_{\max}$ **then**
            Start new block (as above) with full attention
        else
            Extend $B_m^{(\ell)}$ with t
            Form $S^{(\ell)}(t)$ and run restricted attention
        end if
    end if
end for

4.5.2 Datasets and Metrics

We evaluate on two math reasoning datasets that induce long chain-of-thought traces:

1. AIME 2024 — a set of contest-style problems requiring multi-step symbolic and numerical reasoning.

2. MATH-500 — a curated subset of the MATH benchmark containing 500 problems across algebra, combinatorics, number theory, and geometry.

Each problem is prompted to first produce a detailed solution and then output a final boxed answer in a standardized format. Dataset-specific maximum output lengths range between 8k and 30k tokens.

For each dataset we report:

1. Answer accuracy (fraction of problems for which the final boxed answer matches the ground truth);

2. Output length (average number of decoded tokens);

3. KV retention ratio r_{KV} (fraction of keys retained by BlocKV relative to a full key–value cache).

Dataset	Method	Acc. (%) ↑	Output len.	KV ratio r_{KV}
AIME 2024	Full KV	53.3	10k	1.00
	BlocKV (ours)	43.3	15k	0.6
MATH-500	Full KV	79.8	4k	1.00
	BlocKV (ours)	68	8k	0.5

Table 4.1: BlocKV on long reasoning tasks. We report answer accuracy, average output length in tokens, average KV retention ratio across layers.

Generation uses temperature sampling ($T = 0.6$) with nucleus filtering ($p = 0.95$) and a single sample per problem. Prefill and decode token counts are recorded independently to enable accurate efficiency measurements.

4.6 Results

Table 4.1 reports accuracy, average output length, and key–value retention ratio for **BlocKV** compared to a full key–value cache. Across both datasets, BlocKV substantially reduces the retained key budget—retaining roughly 50–60% of the stored keys—but this comes at the cost of a noticeable drop in answer accuracy. On **AIME 2024**, accuracy decreases from 53.3% to 43.3%, and on **MATH-500** from 79.8% to 68.0%. BlocKV also induces longer generated solutions, with output lengths increasing by 1–2× depending on the task.

Overall, these results indicate that while BlocKV effectively reduces the memory footprint of long decoding, the current implementation does not fully preserve the model's reasoning fidelity. In particular, the extended chain-of-thought traces suggest that the reduced key set may impair the model's ability to maintain global context, leading to more verbose but less accurate solutions. Future variants may require improved key selection strategies, tighter similarity thresholds, or additional fallbacks in order to approach full-cache accuracy while retaining the memory benefits.

4.7 Discussion

While BlocKV shows that blockwise query geometry can be used to prune the KV cache during long-form reasoning, several limitations remain.

(1) Imperfect alignment between queries and attention. Although query trajectories and attention distributions exhibit similar blockwise structure, the correspon-

dence is not exact. In particular, attention often assigns small but crucial mass to positions whose queries appear redundant under cosine similarity. This mismatch can result in accuracy loss, suggesting that query-similarity alone is not a sufficient proxy for attention relevance.

(2) Local reference signals without global planning. BlocKV evaluates new tokens against a small set of block-internal reference queries. This captures local coherence but ignores longer-term reasoning dependencies and future reuse. Additional global signals may be needed to distinguish transient redundancy from genuinely dispensable context.

(3) Dependence on full attention for key discovery. BlocKV requires periodic full-attention steps to extract top-ranked keys for future pruning. These refreshes introduce latency and partially offset the efficiency gains during restricted attention. An interesting direction is to amortize or predict key importance without recomputing dense attention, or to learn sparse discovery layers.

Overall, these limitations indicate that KV pruning for long-form reasoning requires more than local geometric cues. Promising future directions include hybrid relevance estimators, learning-based pruning policies, and reasoning-aware segmentation that better aligns with the global structure of mathematical problem solving.

Chapter 5

Gaze-Guided Token Pruning for Vision–Language Models

In this chapter, we study how human eye gaze can be used as a training-free supervisory signal to reduce the cost of vision–language model (VLM) inference. We focus on visual question answering (VQA) in augmented reality (AR)–like settings, where a head-mounted device records both the egocentric scene and the user's gaze trace. Our goal is to exploit this physiological prior to prune visual tokens before they are passed to a large VLM, thereby reducing the number of visual tokens, total context tokens, and floating-point operations (FLOPs) without sacrificing answer quality.

Building on the gaze-driven region-of-interest (ROI) cropping framework introduced by Qinyu [5], we integrate gaze-aware cropping into Qwen2.5-VL and extend it with a novel *multi-ROI* scheme that better captures dispersed gaze patterns. Concretely:

1. We adopt Qinyu's single-ROI method, which converts a gaze heatmap into one gaze-conditioned bounding box and feeds a single cropped image to the VLM.

2. We propose a new **multi-ROI cropping strategy** that decomposes the high-density gaze region into multiple smaller, non-overlapping gaze patches. These patches are jointly fed into the VLM through a multi-image prompt that explicitly instructs the model to integrate information across all crops.

3. We adopt Qinyu's unified benchmarking pipeline that measures per-sample FLOPs, vision and total token counts, and answer quality using a GPT-4o–based auto-

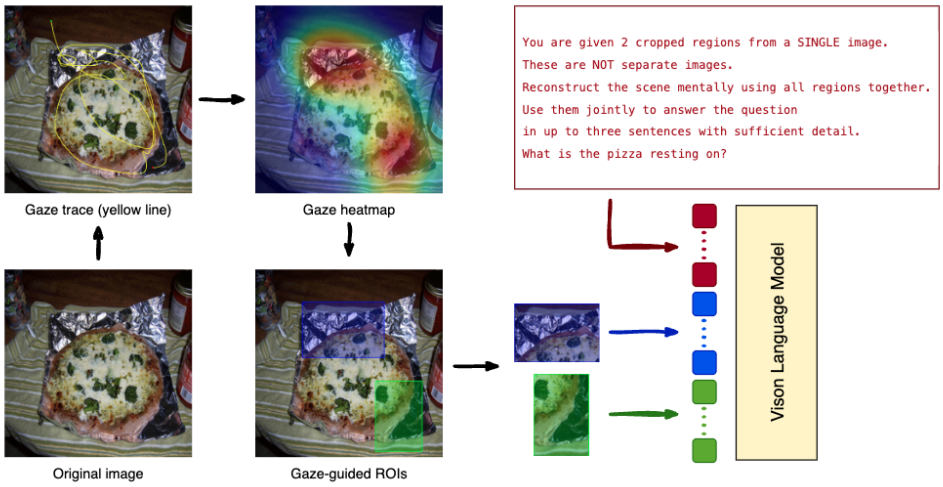


Figure 5.1: End-to-end overview of the gaze-guided multi-ROI inference pipeline. Gaze traces are converted into heatmaps, from which one or more regions of interest (ROIs) are extracted. The crops are passed to a vision–language model together with an instruction prompt that explains that all crops originate from the same image and should be jointly integrated using spatial reasoning.

matic judge.

Experiments on the VOILA gaze–VQA dataset demonstrate that gaze-guided cropping significantly reduces visual tokens, and that our multi-ROI variant produces similar answer quality at lower computational cost.

5.1 Problem Setting

We consider a VLM f_θ that takes as input a set of images $\{I_m\}_{m=1}^{M}$ and a text question q, and generates an answer $\hat{a} = f_\theta(\{I_m\}, q)$. The model is implemented by Qwen2.5-VL-3B and 7B , which tokenize each image into a sequence of visual tokens and interleave them with text tokens in a transformer-based decoder.

In our setting, the model receives an RGB frame $I \in \mathbb{R}^{H\times W\times 3}$ together with a normalized gaze trace

$$\mathcal{G} = \{(x_t, y_t)\}_{t=1}^{T}, \qquad (x_t, y_t) \in [0,1]^2.$$

The trace is converted into a gaze heatmap $H \in [0,1]^{H'\times W'}$ using a standard Gaussian

smoothing procedure provided by the VOILA dataset. Intuitively, high values in H indicate regions that attract sustained human attention.

Our objective is to construct from H a set of image crops $\{I_m^{\text{crop}}\}_{m=1}^{M}$ such that:

1. The number of visual tokens L_{vis} and the total context length L_{ctx} are significantly reduced compared to the full-image baseline.
2. The model's answer quality, measured by a GPT-4o judge relative to ground-truth answers, remains comparable or improves.
3. The method is training-free and does not modify any parameters of f_θ.

In this chapter we consider three inference configurations:

1. **Baseline:** The full 896×896 frame I is resized and passed as a single image to Qwen2.5-VL.
2. **Single-ROI:** A single gaze-conditioned bounding box is extracted from H and used as a stand-alone crop.
3. **Multi-ROI (ours):** Multiple gaze-conditioned bounding boxes are extracted from H, producing a set of crops $\{I_m^{\text{crop}}\}_{m=1}^{M}$ which are all fed to the VLM with an explicit instruction to integrate them.

5.2 Background: Gaze-Guided ROI Cropping

Eye gaze has long been studied as a proxy for human attention in perception and human–computer interaction. Recent works leverage gaze data for saliency modelling, user interface optimization, and gaze-aware compression [5]. VOILA [22] extends this idea to VQA by recording head-mounted eye traces while participants answer visual questions.

Qinyu et al. [5] propose *GazeVLM*, a training-free framework that uses VOILA-style gaze heatmaps to crop a single ROI around the most attended region. The resulting crop is upsampled and fed to a frozen VLM, either alone or in combination with a global low-resolution image. This approach already reduces visual tokens by over 50% while often improving answer quality, suggesting that VLMs can benefit from human attention priors.

However, a single bounding box is not always sufficient: in many VOILA scenes, human gaze alternates between multiple semantically distinct objects (e.g., the question mentions "the man" and "the sign in the background"). A single ROI must either

grow large enough to cover all attended regions—undoing the benefits of cropping—or commit to one subset of them. Our multi-ROI method is designed to handle such cases more gracefully.

5.3 Single-ROI Cropping

We briefly summarize the single-ROI cropping algorithm originally proposed by Qinyu and adopted in our implementation.[1]

Given a heatmap $H \in [0,1]^{H' \times W'}$, the algorithm selects a minimal bounding box that captures a fixed fraction $\rho \in (0,1]$ of the total gaze mass. Let h_{ij} denote the heatmap value at pixel (i,j) and $Z = \sum_{i,j} h_{ij}$ the total mass [5].

1. Flatten H into a vector, sort values in descending order, and find the smallest index k such that the cumulative sum of the top k entries exceeds ρZ.

2. Let $\mathcal{S}_\rho$ be the set of pixels corresponding to these top k entries. The binary mask $1_{\mathcal{S}_\rho}$ highlights the most attended region.

3. Compute the tight bounding box (y_0, y_1, x_0, x_1) that encloses all non-zero pixels of $1_{\mathcal{S}_\rho}$, and enlarge it slightly by a small padding margin.

4. Snap the box to a grid of size $a \times a$ (with $a = 14$ or 28 in our experiments), ensuring that both height and width are at least a minimum size $s_{\min}$.

The resulting bounding box is then used to crop the original image:

$$I^{\text{roi}} = I[y_0 : y_1, x_0 : x_1],$$

which is resized to match the VLM's expected resolution. We denote by ROI-size the area of this box in pixels.

5.4 Proposed Method: Multi-ROI Cropping

The single-ROI method assumes that the gaze heatmap is unimodal, or that a single bounding box can adequately cover all attended content. In VOILA, we frequently observe *multi-modal* gaze patterns where participants inspect multiple objects or text

[1] All design choices in Sections 5.3 follow Qinyu's method; our contributions start in Section 5.4.

regions. In such cases, forcing a single box leads either to large crops (and therefore many visual tokens) or to the exclusion of important context.

To address this, we propose **multi-ROI cropping**: instead of collapsing the top-ρ mass region into one bounding box, we decompose it into several connected components and extract a separate crop for each. These crops are then jointly passed to the VLM with a tailored instruction that encourages holistic reasoning across all regions.

5.4.1 Top-Mass Connected Components

Let $H \in [0,1]^{H' \times W'}$ be the heatmap and $Z = \sum_{i,j} h_{ij}$ its total mass. We first determine a threshold τ such that the set of pixels above threshold accounts for the top ρ fraction of mass.

$$\sum_{(i,j):h_{ij} \geq \tau} h_{ij} \approx \rho Z, \qquad 0 < \rho \leq 1. \tag{5.1}$$

In practice, we flatten H into a vector, sort entries in descending order, and pick τ as the value at the smallest index where the cumulative sum exceeds ρZ. We then form a binary mask

$$M_{ij} = 1[h_{ij} \geq \tau],$$

and run 8-connected component labeling on M to obtain regions $\mathcal{C}_1, \ldots, \mathcal{C}_K$.

For each component $\mathcal{C}_k$, we compute the tight bounding box $(y_0^{(k)}, y_1^{(k)}, x_0^{(k)}, x_1^{(k)})$ and enlarge it to satisfy minimal size and grid alignment constraints, mirroring the single-ROI case. We also compute a component score given by the total heatmap mass inside the box:

$$s_k = \sum_{(i,j) \in \mathrm{box}(\mathcal{C}_k)} h_{ij}. \tag{5.2}$$

We sort components in descending order of s_k and keep the top N components:

$$\mathcal{B} = \left\{ \left(y_0^{(k)}, y_1^{(k)}, x_0^{(k)}, x_1^{(k)} \right) \right\}_{k=1}^{N}.$$

Each bounding box yields a cropped image

$$I_k^{\mathrm{roi}} = I[y_0^{(k)} : y_1^{(k)}, x_0^{(k)} : x_1^{(k)}].$$

5.4.2 Multi-Image Prompting

Qwen2.5-VL natively supports multiple images in its chat template. We exploit this to feed the set of ROI crops

$$\{I_k^{\text{roi}}\}_{k=1}^{N}$$

as separate images in a single conversation turn. To steer the model towards integrating information across all regions, we prepend the question with a short instruction. In code, this takes the form

```
``You are given {$N} cropped regions from a single image.
They are not separate images.  Mentally reconstruct the scene
using all regions together, and answer the question in at
most three sentences with sufficient detail.''
```

This instruction is injected into the text portion of the multimodal input, after the visual tokens for all ROI crops. The model remains frozen; no fine-tuning is performed.

5.5 Implementation Details

We instantiate all methods on top of Qwen2.5-VL-3B and 7B Instruct models as provided by Hugging Face. The models are loaded in `bfloat16` with `flash_attention_2` enabled and run on a single NVIDIA A100 GPU.

Dataset. We use the VOILA dataset, which provides egocentric frames, question–answer pairs, and gaze traces. For each sample, we reconstruct the Flamingo-normalized image from the stored tensors, resize it to 896×896, and resample the gaze heatmap to the same resolution. Since our multi-ROI method is designed to operate on heatmaps with spatially separable fixation clusters, we restrict evaluation to samples whose gaze heatmaps produce at least two non-overlapping ROI components under the top-mass selection rule (Section 5.4.1). Samples that produce only a single ROI are excluded from the analysis, as no meaningful multi-ROI decomposition is possible in such cases.[2]

Token and FLOPs accounting. For each input configuration we record:

- L_{vis}: the number of visual tokens produced by the processor,
- L_{text}: the number of text tokens in the prompt,

[2]These samples are still valid for single-ROI baselines, but do not constitute informative comparisons for multi-ROI reasoning.

- $L_{\text{ctx}} = L_{\text{vis}} + L_{\text{text}}$: the total context length.

We estimate prefill and decoding FLOPs using closed-form expressions based on model depth L, hidden size d, number of heads H, and feed-forward size d_{ff}:

$$\text{FLOPs}_{\text{prefill}}(L_{\text{ctx}}) \approx L\big[4HL_{\text{ctx}}^2 d_h + 4L_{\text{ctx}} d^2 + 2L_{\text{ctx}} d d_{\text{ff}}\big], \tag{5.3}$$

$$\text{FLOPs}_{\text{decode-step}}(L_{\text{ctx}}) \approx L\big[4HL_{\text{ctx}} d_h + 4d^2 + 2d d_{\text{ff}}\big], \tag{5.4}$$

where $d_h = d/H$. The total FLOPs for generating T_{gen} new tokens is

$$\text{FLOPs}_{\text{total}} = \text{FLOPs}_{\text{prefill}} + T_{\text{gen}} \cdot \text{FLOPs}_{\text{decode-step}}.$$

5.6 Evaluation Protocol

Answer quality is evaluated using GPT-4o as an automatic judge. For each image–question pair we obtain candidate answers $a^{(1)}$ and $a^{(2)}$ from two different configurations (e.g., baseline vs. single-ROI, or single-ROI vs. multi-ROI). We then prompt GPT-4o with the question, caption, ground-truth answer, and both candidates.

Pairwise preference. The judge is asked to decide which answer is better, returning -1 if Answer 1 is better, 1 if Answer 2 is better, and 0 for a tie. Repeating this for all examples yields a *win rate*:

$$\text{WinRate} = \frac{\#\{\text{wins for configuration B}\}}{\#\{\text{wins for A or B}\}},$$

where ties are excluded from the denominator.

5.7 Results

Table 5.1 compares gaze-guided cropping strategies on Qwen2.5-VL-3B across different cropping ratios ρ. Both single-ROI and multi-ROI configurations substantially reduce the number of visual tokens and total FLOPs relative to the baseline, confirming that attention-guided cropping can effectively lower computational cost. Across all settings, larger values of ρ correspond to weaker compression and higher computational overhead.

At low cropping ratios ($\rho = 0.10$), both methods achieve aggressive token reduction, with multi-ROI pruning more than 84% of visual tokens and reducing FLOPs by over

Model	ρ	Visual tokens ↓	Win-rate (%) ↑	FLOPs (G) ↓
Qwen2.5-VL-3B				
Baseline	–	4096	–	2756.9
1-ROI	0.10	1290.3 (−68.5%)	46.3	902.5 (−64.7%)
N-ROI (ours)	0.10	643.5 (−84.3%)	38.1	628.7 (−75.4%)
1-ROI	0.20	1665 (−59.3%)	48.8	1136.7 (−58.8%)
N-ROI (ours)	0.20	1066 (−74.0%)	50.5	955.2 (−65.3%)
1-ROI	0.30	2011 (−50.9%)	58.5	1356.0 (−50.8%)
N-ROI (ours)	0.30	1765 (−56.9%)	60.1	1120.0 (−59.3%)

Table 5.1: Comparison of gaze-guided cropping strategies with Qwen2.5-VL. We report relative reduction in visual tokens, GPT-4o judged win rate, and total FLOPs in gigaFLOPs. All numbers are averaged over the VOILA test split.

75%. However, this regime also exhibits the lowest win rates, suggesting that excessive cropping can remove semantically important visual content and harm answer quality. Single-ROI performs more conservatively in this regime, retaining higher win rates at the cost of additional computation.

As ρ increases, answer quality improves for both methods while maintaining meaningful efficiency gains. Notably, multi-ROI consistently outperforms single-ROI in win rate at comparable ρ, indicating that distributing attention across multiple regions helps preserve relevant visual context under compression. At $\rho = 0.30$, multi-ROI achieves the highest win rate while still reducing FLOPs by nearly 60%, representing a favorable balance between efficiency and answer quality.

Overall, these results highlight a clear trade-off between computational savings and response quality. Multi-ROI cropping offers a more flexible mechanism for navigating this trade-off, enabling stronger compression at lower ρ and superior quality at moderate compression levels compared to single-ROI.

5.8 Discussion

The results in Section 5.7 show that gaze-guided cropping can substantially reduce visual tokens and computational cost while retaining reasonable answer quality at moderate compression levels. However, these gains also reveal several important limitations that point to directions for future work.

First, gaze provides only a partial signal of visual relevance. Regions that receive little gaze attention may nonetheless be essential for correct reasoning, particularly for

tasks involving counting, spatial relationships, or fine-grained attribute comparisons. As a result, cropping decisions based solely on gaze can remove information that the model later requires, leading to irreversible errors.

Second, the current approach applies cropping statically before generation. Once visual tokens are removed, the model cannot revisit discarded regions even if later reasoning steps indicate their importance. Future work could explore adaptive or iterative cropping schemes, where the set of retained visual tokens is refined in response to intermediate model states or uncertainty signals.

Third, gaze reflects saliency rather than task-specific utility. Although multi-ROI selection improves robustness compared to a single crop, ROI selection remains heuristic and coarse. Incorporating learned relevance signals, cross-modal attention scores, or joint vision–language criteria may allow more precise identification of visually informative regions.

Finally, the method is entirely inference-time and training-free. While this simplifies integration with existing models, it prevents the model from learning to reason under partial visual context. Training or distillation strategies that expose the model to cropped inputs may help improve robustness and better align the model with gaze-guided pruning.

Overall, gaze-guided cropping is a promising step toward efficient multimodal inference, but fully exploiting it will require more adaptive selection mechanisms and tighter integration between visual pruning and language reasoning.

Chapter 6

Conclusion

This thesis investigated training-free strategies for reducing the computational and memory cost of large language models (LLMs) and vision–language models (VLMs) at inference time. Motivated by the growing deployment of foundation models under strict memory, energy, and latency constraints, the work focused on identifying and exploiting structured redundancy in attention, reasoning traces, and visual inputs—without modifying model parameters or requiring additional training. Across three settings—long-context understanding, long chain-of-thought reasoning, and multimodal inference—the thesis demonstrated that substantial efficiency gains are possible by leveraging observable inference-time signals.

Answer to RQ1: KV pruning for long-context inference. The first research question asked whether the KV cache memory footprint of long-context inference can be reduced without sacrificing model accuracy. Chapter 3 shows that this is indeed feasible in many settings. By constructing a query–key mapping from prompt-time attention statistics and reusing it during decoding via query-space nearest-neighbor matching, the proposed method retains only a small fraction of prompt keys while preserving competitive performance on LongBench. Importantly, this approach is entirely training-free and model-agnostic. The results suggest that long-context prompts contain substantial semantic redundancy, and that query geometry provides a useful proxy for identifying which parts of the context remain relevant throughout generation. However, the ablation studies also indicate that relevance evolves across depth and that overly aggressive layer sharing can degrade performance, highlighting the need for depth-aware pruning strategies.

Answer to RQ2: KV compression for long chain-of-thought reasoning. The second research question addressed KV cache growth during long chain-of-thought generation, where redundancy arises primarily from the model's own outputs rather than the input prompt. In Chapter 4, we identified a consistent empirical pattern: query similarity and attention similarity evolve in synchronized, blockwise phases during long reasoning. Based on this observation, the **BlocKV** method groups decoding steps into blocks and restricts attention within each block to a compact set of keys. While BlocKV successfully reduces KV retention on AIME 2024 and MATH-500, the current implementation incurs a noticeable accuracy drop and produces longer reasoning traces. This outcome provides a clear answer to RQ2: redundancy can be identified and exploited, but long-form reasoning is highly sensitive to approximation errors. Local geometric similarity alone is not sufficient to preserve global reasoning fidelity, and future approaches will likely require richer signals or adaptive fallbacks to approach full-KV accuracy.

Answer to RQ3: Gaze-guided efficiency for vision–language models. The third research question explored whether human physiological priors—in particular eye gaze—can guide efficient multimodal inference. In Chapter 5, gaze heatmaps were used to prune visual tokens via region-of-interest cropping. The proposed multi-ROI strategy extends prior single-ROI methods by handling multi-modal gaze patterns and encouraging holistic reasoning across multiple attended regions. Experiments on the VOILA dataset show that gaze-guided cropping can substantially reduce visual tokens and FLOPs, while maintaining reasonable answer quality at moderate compression levels. At the same time, the results highlight an inherent limitation: aggressive cropping can irreversibly remove information required for correct answers. Thus, gaze is a powerful but imperfect supervisory signal that benefits from careful integration and adaptive use.

Overall perspective and future directions. Taken together, the three chapters present a coherent view of training-free efficiency: inference-time signals—attention distributions, query geometry, temporal similarity, and human gaze—encode rich information about redundancy and relevance. When these signals are stable and well aligned with task requirements, aggressive pruning is possible with little loss in performance. When alignment breaks down, as in long-form reasoning or heavily compressed multimodal inputs, approximation errors accumulate and accuracy degrades.

Several directions for future work emerge naturally. First, combining multiple rel-

evance signals (e.g., query similarity, attention confidence, and uncertainty estimates) may yield more robust pruning decisions than any single proxy. Second, adaptive or iterative mechanisms that allow models to recover pruned context when needed could mitigate irreversible errors. Finally, while this thesis focused on training-free methods, lightweight distillation or curriculum-based exposure to pruned contexts may help models learn to reason more robustly under constrained inference budgets.

In conclusion, this thesis demonstrates that meaningful reductions in memory and computation are achievable without retraining large models, but also shows that efficiency is deeply intertwined with representation dynamics and task structure. Understanding and exploiting this structure is key to making foundation models practical under real-world deployment constraints.

Bibliography

[1] 01.AI. Yi: Open foundation models by 01.ai. 2024.

[2] M André et al. Sparse and continuous attention mechanisms. 2020.

[3] Y Bai et al. Longbench: A benchmark for long-context language models. *arXiv preprint arXiv:2308.14508*, 2023.

[4] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. In *ACL*, 2020.

[5] Qinyu Chen et al. Eye gaze tells you where to compute: Gaze-driven efficient vlms. *arXiv preprint: 2509.16476*, 2025.

[6] K Clark et al. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

[7] DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2026.

[8] Feng et al. Evicpress: Joint kv-cache compression and eviction for efficient llm serving. *arXiv preprint arXiv:2512.14946*, 2025.

[9] Ge et al. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2024.

[10] Kim et al. Fast kvzip: Efficient and accurate llm inference with gated kv eviction. *arXiv preprint arXiv:2601.17668*, 2026.

[11] Wang et al. Fier: Fine-grained kv cache retrieval for efficient long-context inference. *arXiv preprint arXiv:2508.08256*, 2025.

[12] Wang et al. Llms know what to drop: Self-attention guided kv cache eviction for efficient long-context inference. *arXiv preprint arXiv:2503.08879*, 2025.

[13] F Florian et al. An introduction to vision-language modeling. *arXiv preprint arXiv:2405.17247*, 2024.

[14] Touvron H et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[15] Lin Jingyang et al. Unleashing hour-scale video training for long video-language understanding. *arXiv preprint arXiv:2506.05332*, 2025.

[16] Hao Jitai et al. Omnikv: Dynamic context selection for efficient long-context llms. 2026.

[17] M Junlin et al. Sals: Sparse attention in latent space for kv cache compression. *arXiv preprint arXiv:2510.24273*, 2025.

[18] B Kenan et al. Gaze-enabled activity recognition for augmented reality feedback. *Computers Graphics*, 2024.

[19] L Kevin et al. Inference optimal vlms need fewer visual tokens and more parameters. *arXiv preprint arXiv:2411.03312*, 2025.

[20] Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. KVzip: Query-agnostic KV cache compression with context reconstruction. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

[21] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *ICLR*, 2020.

[22] Yan Kun et al. Voila-a: Aligning vision-language models with user's gaze attention. In *NeurIPS*, 2024.

[23] Liwan Lin et al. Recent progress on eye-tracking and gaze estimation for ar/vr applications: A review. 2025.

[24] Y Mingqiang et al. Wearable eye-tracking system for synchronized multimodal data acquisition. *IEEE Transactions on Circuits and Systems for Video Technology*, 34(6):5146–5159, 2024.

[25] OpenAI et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[26] Wu Qingyun et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

[27] Sapkota Ranjan et al. Vision-language-action models: Concepts, progress, applications and challenges. *arXiv preprint arXiv:2505.04769*, 2025.

[28] E Voita et al. Analyzing the source and target contributions to predictions in neural machine translation. *In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 2021.

[29] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[30] G Xiao et al. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

[31] W Xindi et al. Beyond the limits: A survey of techniques to extend the context length in large language models. *arXiv preprint arXiv:2402.02244*, 2024.

[32] Manzil Zaheer, Guru Guruganesh, Kumar Dubey, Samuel Ainsworth, Chris Alberti, Santiago Ontanon, Philip Pham, Arijit Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In *NeurIPS*, 2020.

[33] Cai Zefan et al. R-kv: Redundancy-aware kv cache compression for reasoning models. *arXiv preprint arXiv:2505.24133*, 2025.