Handling Extremely Long Inputs with Inference Algorithms

Woomin Song

Korea Advanced Institute of Science and Technology (KAIST)

Self Introduction: Woomin Song

- Integrated M.S. + Ph.D. Student at KAIST

- Education
  - M.S. + Ph.D. in Artificial Intelligence, KAIST (advisor: Jinwoo Shin), Sep. 2022 – Current
  - B.S. in Electrical Engineering, KAIST, Mar. 2018 – Aug. 2022
    - Also studied Computer Science (Double Major) and Mathematics (Minor)

- Work Experience
  - Applied Scientist Intern, Amazon AGI team (Host: Sravan Bodapati), Aug. 2024 – Aug. 2025

Compress, Gather, and Recompute:
REFORMing Long-Context Processing in Transformers

Woomin Song[1,*], Sai Muralidhar Jayanthi[2], Srikanth Ronanki[2], Khanthashree Mysore Sathyendra[2], Jinwoo Shin[1], Aram Galstyan[2], Shubham Katiyar[2], Sravan Babu Bodapati[2]

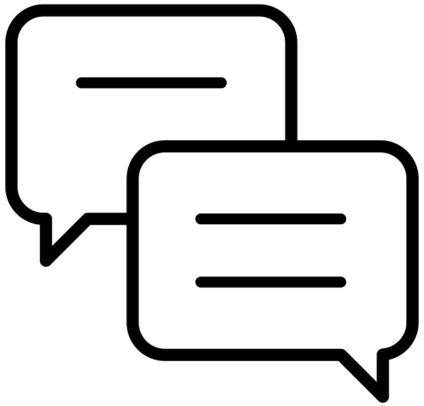
NeurIPS 2025

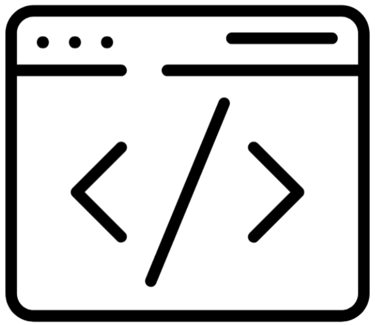
[1] KAIST, [2] Amazon AGI

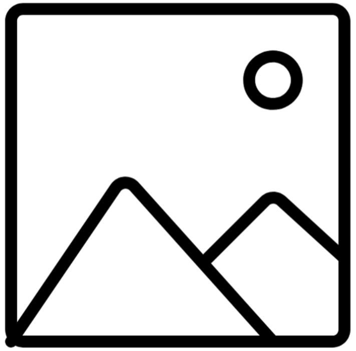
[*] Work done during an internship at Amazon

Long-Context Processing

- Long context processing is crucial for real-world applications
  - Processing life-long user interactions
  - Understanding and debugging repository-level codebases
  - Handling multi-modal inputs with extremely long contexts (e.g. videos)

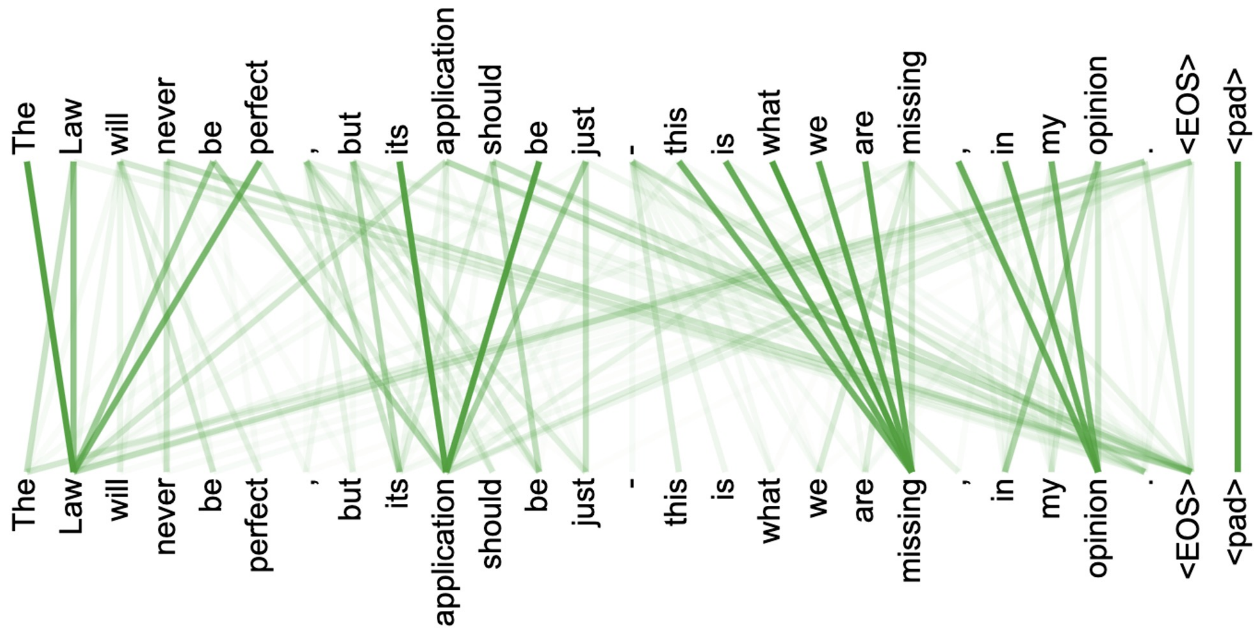






Key Challenges

- 1. Computational burden of self-attention
  - Computation grows quadratically with respect to the sequence length
  - Use of Key-Value (KV) cache further introduces high memory requirements

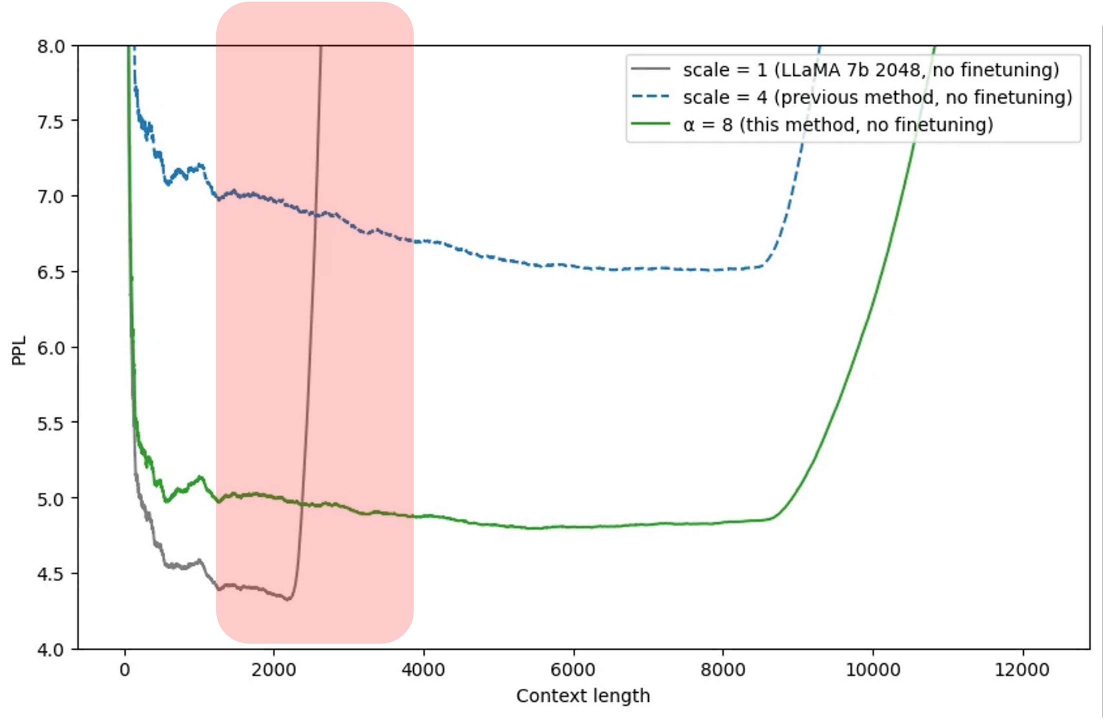


Visualization of Self-Attention [1]

[1] Vaswani et. al., Attention is all you need, NeuriPS 2017

Key Challenges

- 2. Pre-trained context limits of Transformers
  - LLMs often fail to generalize to inputs that are longer than their pre-trained context length
  - Extreme long-context training is challenging, due to computational costs and limited data

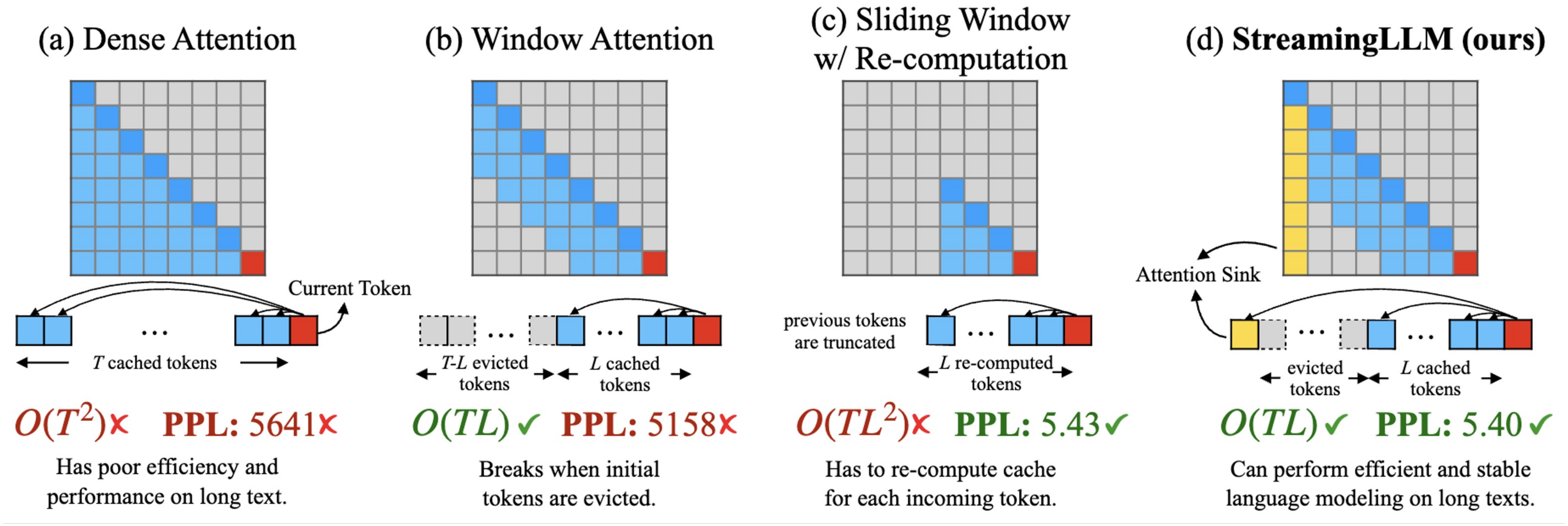


Degradation of Perplexity (PPL) from LLaMA[1]

[1] https://www.reddit.com/r/LocalLLaMA/comments/14lz7j5/ntkaware_scaled_rope_allows_llama_models_to_have/

Recurrent Compression Approaches

- Iteratively process tokens or 'chunks' based on compressed KV cache
  - StreamingLLM keeps the initial & most recent tokens in the cache
  - Reassign position ids to enable context extrapolation

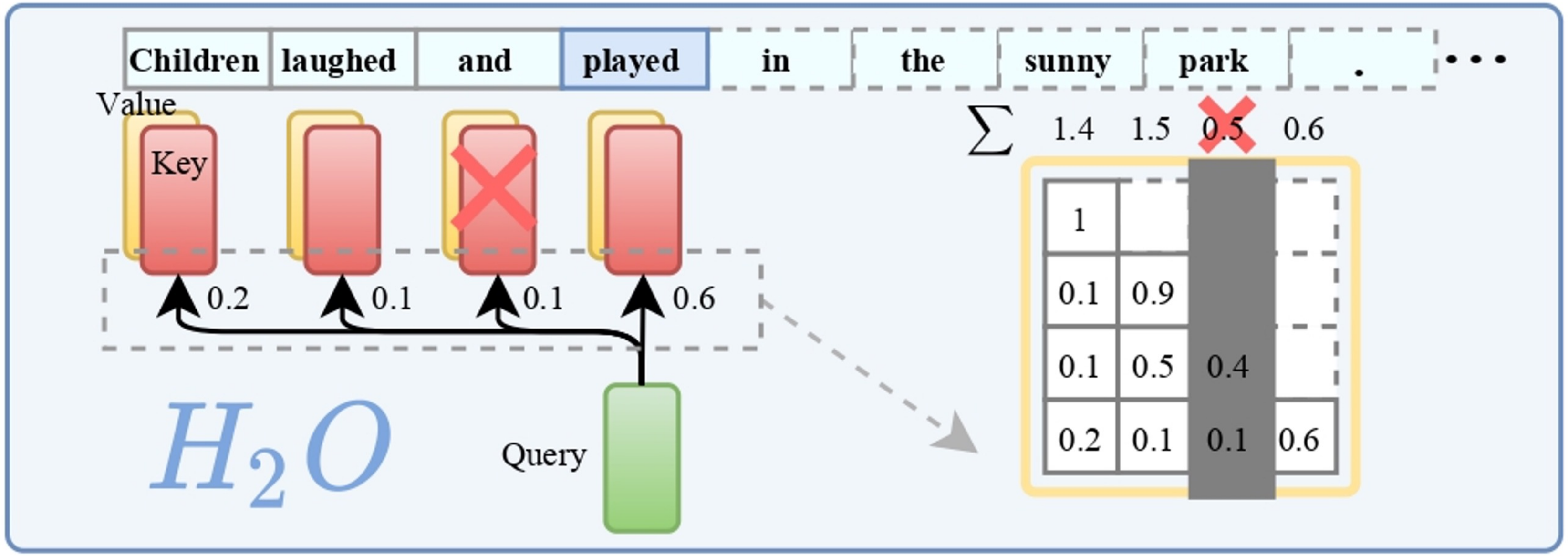


Concept of StreamingLLM

[1] Xiao et. al., Efficient Streaming Language Models with Attention Sinks, ICLR 2024

Recurrent Compression Approaches

- H2O further keeps the important information with selective eviction
  - Use cumulative attention scores as a proxy to measure token importance
  - Suffer from forgetting issues, where important context is lost during compression

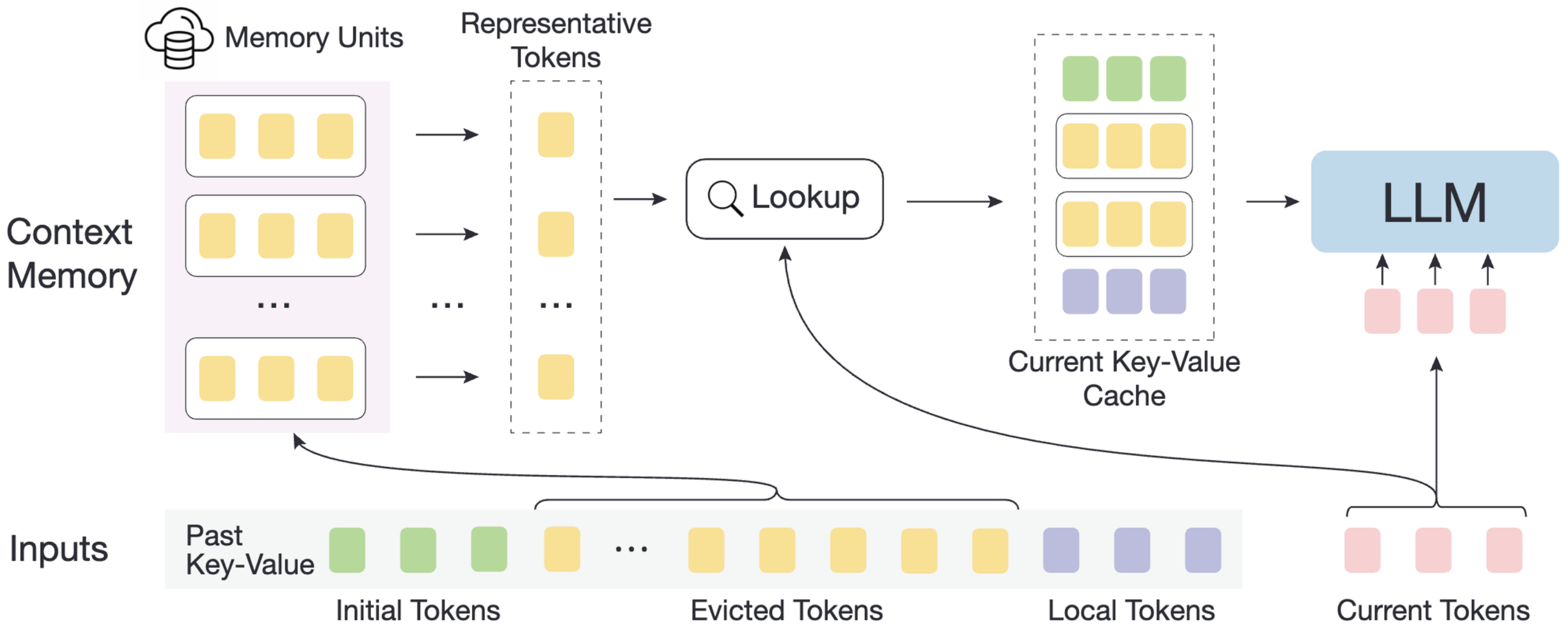


Token eviction criteria of H2O

[1] Zhang et. al., H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models, NeurIPS 2023

Random-Access Approaches

- InfLLM propose to keep the full KV cache, and retrieve tokens on-the-fly
  - Enables random access to any part of the previous input
  - Requires large memory, and adds latency due to memory offloading

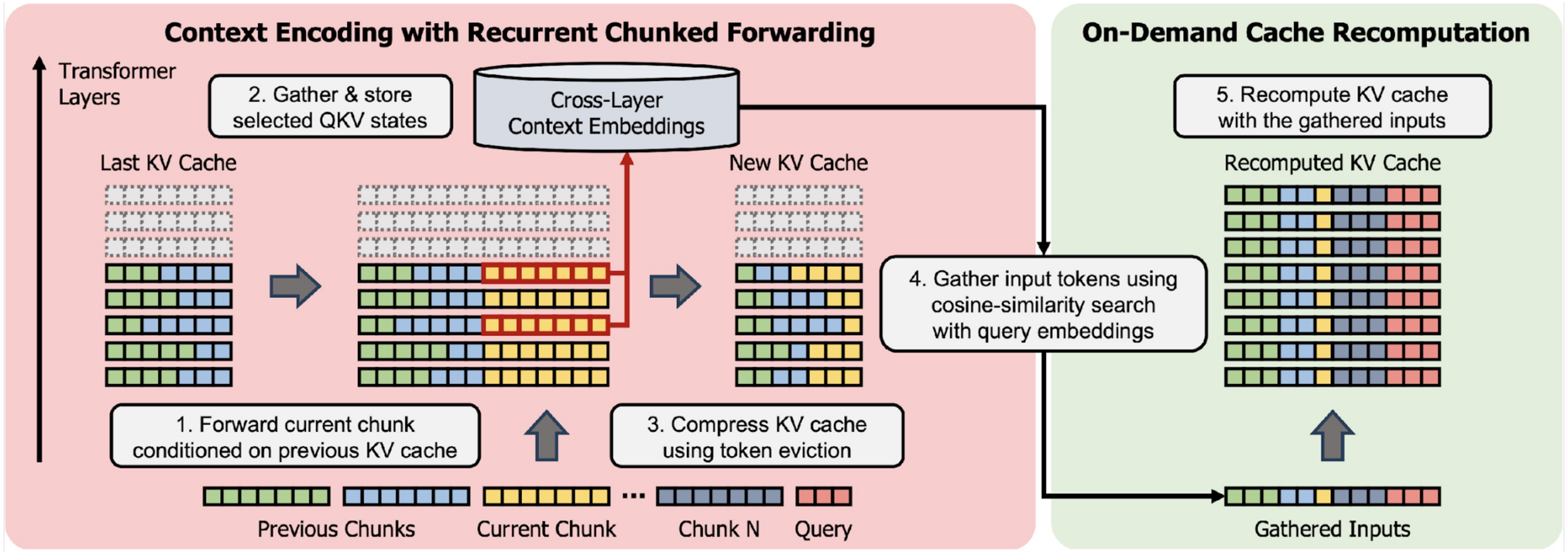


Overview of InfLLM

[1] Xiao et. al., InfLLM: Training-Free Long-Context Extrapolation for LLMs with an Efficient Context Memory, NeurIPS 2024

Our Solution: REFORM

- Efficient long-context inference with random access capabilities
  - **RE**current chunked **F**orwarding with **O**n-demand cache **R**eco**M**putation
  - *<u>Key idea</u>*: Recurrent context encoding + on-demand recomputation



Concept of REFORM

Observation: Attention Heads as Token Selectors

- Cosine similarity of attention QKV embeddings can identify the important tokens for answering a given question.
  - Works better than the attention scores, which are more commonly used as token selectors

This is irrelevant context from Wikipedia documents. This is ... (Passage 1) ... some more text ... (Passage 2) ... some more text ... What is the ... (question that can be answered by looking at both passages)?

Synthetic Multi-Hop QA Task

This is irrelevant context from Wikipedia documents. This is ... The value corresponding to leofksn21e is 19dksnlese. ... some more text ... What is the value for the key leofksn21e?

Synthetic Pattern Matching Task

$$\mathtt{MNR} = \frac{1}{\mathtt{len}(\mathtt{gold\_doc})} \sum_{t \in \mathtt{gold\_doc}} \frac{\mathtt{rank}(\mathtt{t})}{\mathtt{num\_tokens}}$$

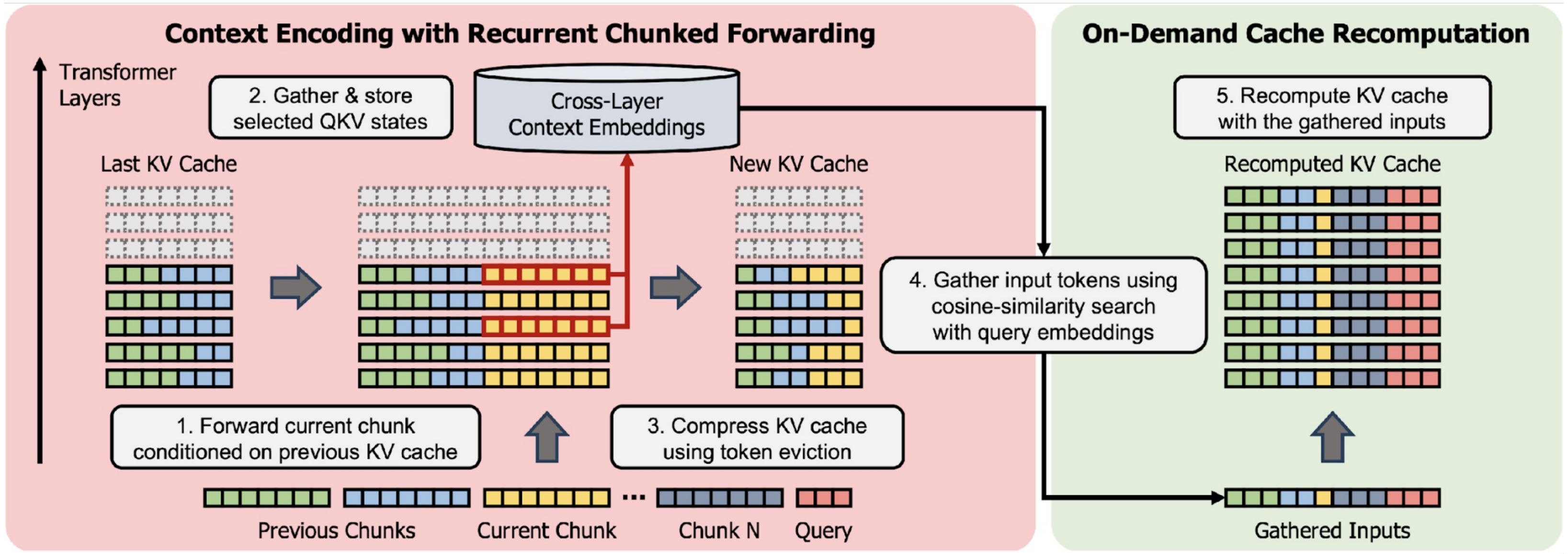
Definition of MNR (Mean Normalized Rank) Score

Type	Dim.	Top-1	Top-2	Top-3	Avg.
Attention	160	6.91	7.70	7.81	7.47
Cosine-HS	5120	9.40	9.63	9.80	9.61
Cosine-Q	160	6.48	6.74	6.93	6.72
Cosine-K	160	6.77	7.31	7.41	7.16
Cosine-V	160	5.77	6.57	6.57	6.30

MNR score of different token selection methods (Lower is better)

Key Idea

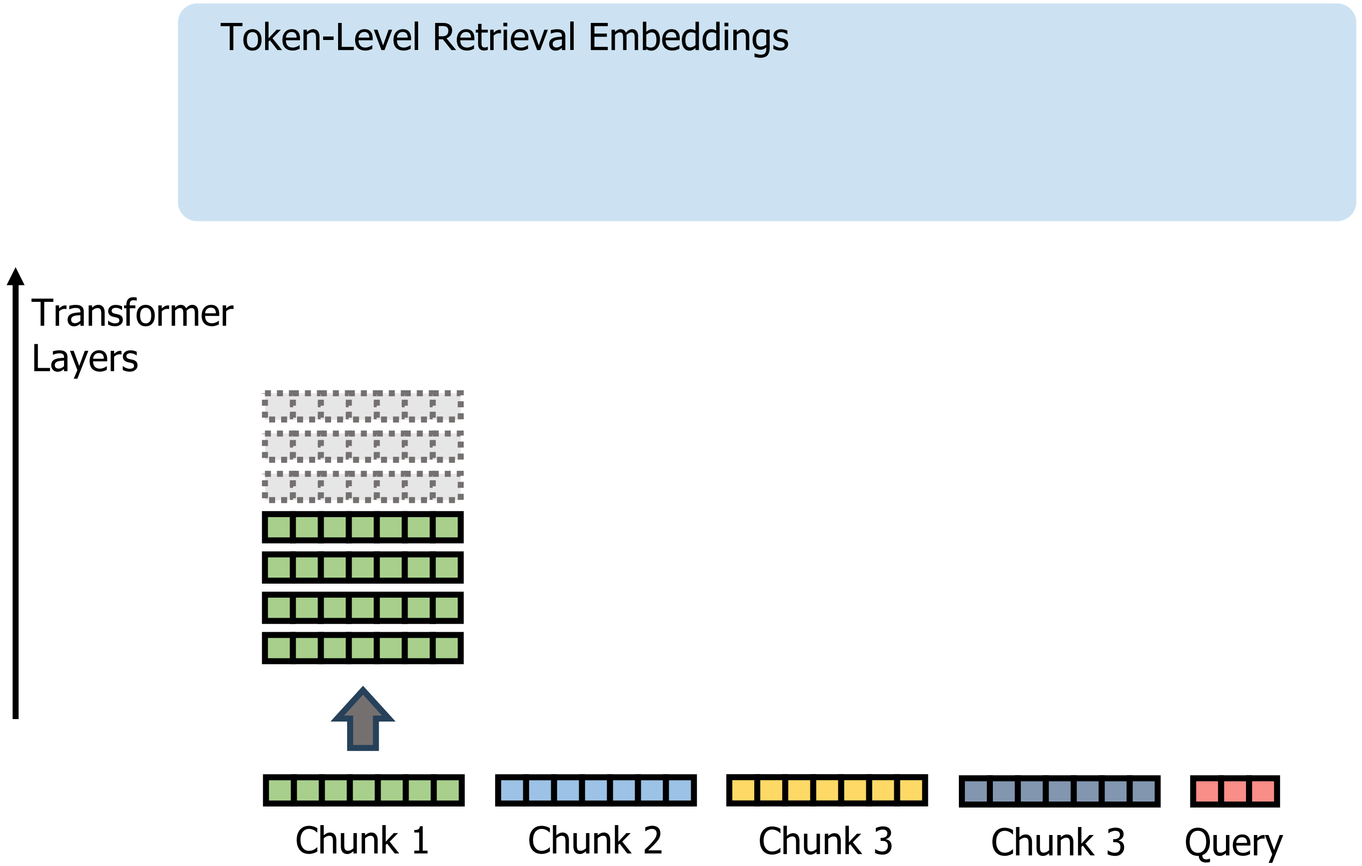
- Propose a 3-stage approach
  - **Compress:** Use recurrent compression to generate query-unaware, token-level embeddings
  - **Gather:** Use the embeddings to identify the core tokens from the long input
  - **Recompute:** Recompute the KV cache with the selected tokens



Concept of REFORM

Compress: Embedding Extraction Stage

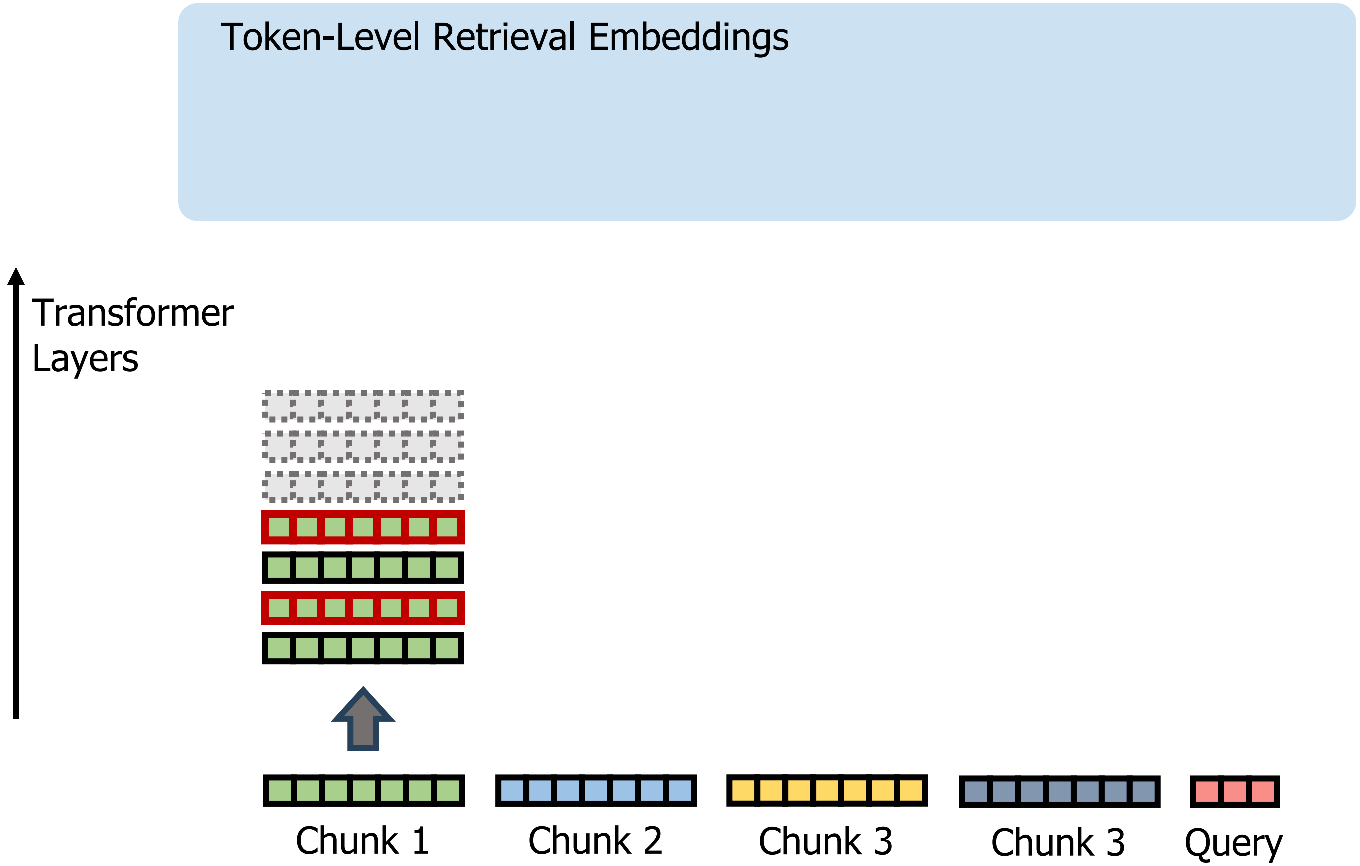
- Use recurrence to generate query-unaware, token-level embeddings



Segment the input into chunks.
Input is passed through the first few layers (early-exit).

Compress: Embedding Extraction Stage

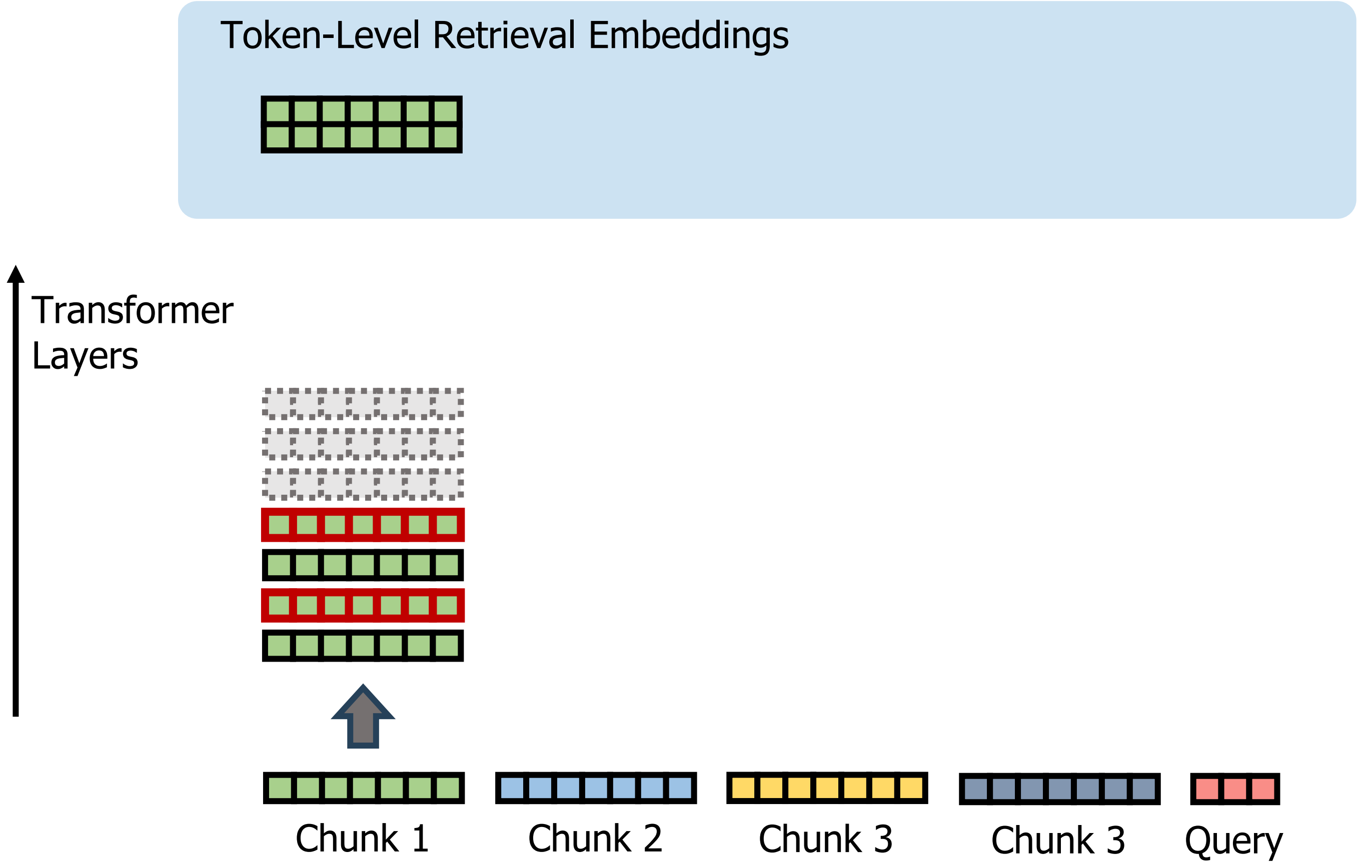
- Use recurrence to generate query-unaware, token-level embeddings



Gather selected QKV heads, and save them as token-level retrieval embeddings.

Compress: Embedding Extraction Stage

- Use recurrence to generate query-unaware, token-level embeddings



Gather selected QKV heads, and save them as token-level retrieval embeddings.

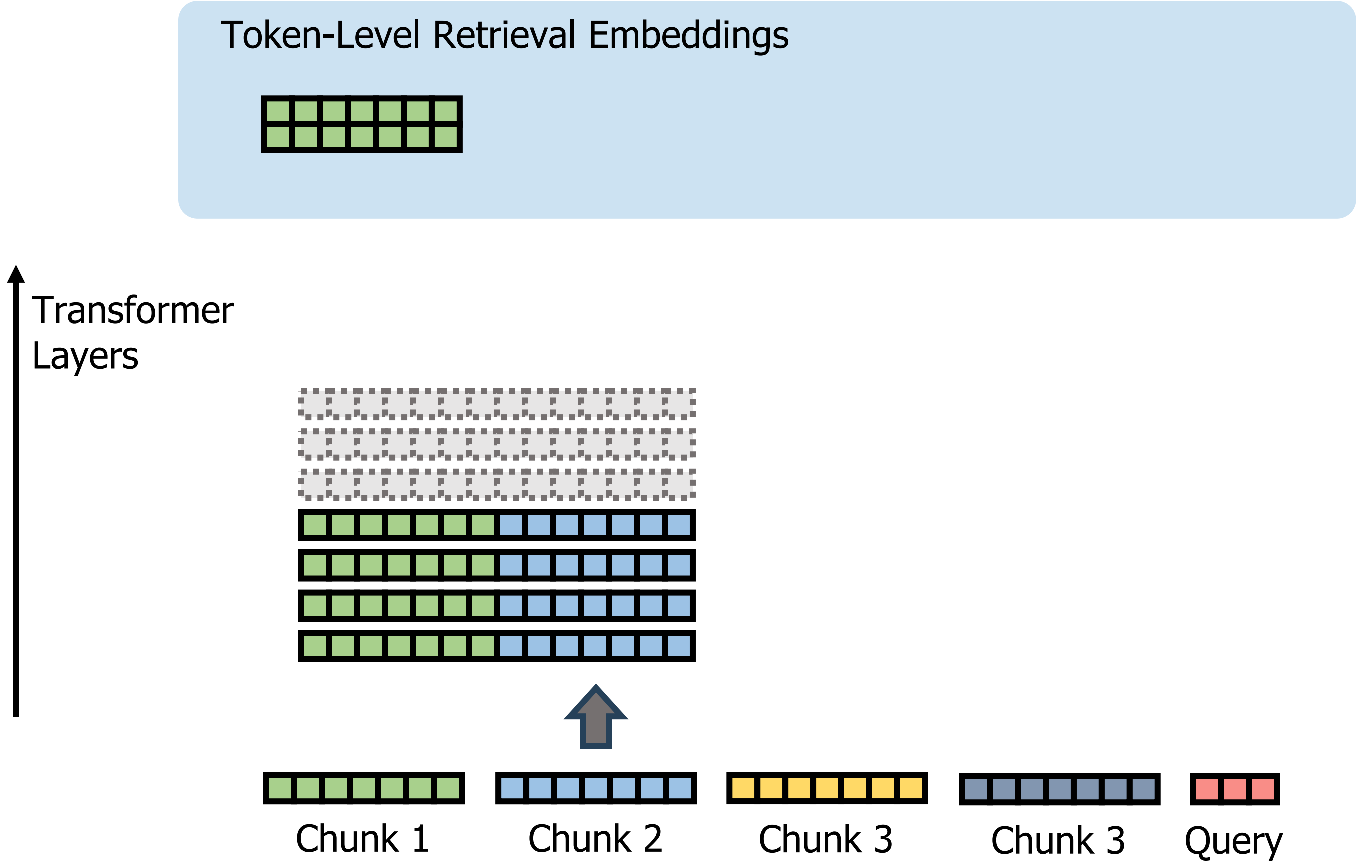
Compress: Embedding Extraction Stage

- Use recurrence to generate query-unaware, token-level embeddings
  - Empirically, we identify four QKV heads using the synthetic datasets (2 from each)
  - We normalize & concatenate them, resulting in a single, cross-layer embeddings
    - Note: Cosine similarity search is equivalent to average cosine similarity scores of the four QKV heads

$$e_{\mathtt{comb}} = \mathtt{concat}\left(\left\{\frac{e_{\mathtt{i}}}{||e_{\mathtt{i}}||}, \mathtt{i} \in \mathtt{selected\_heads}\right\}\right)$$

Compress: Embedding Extraction Stage

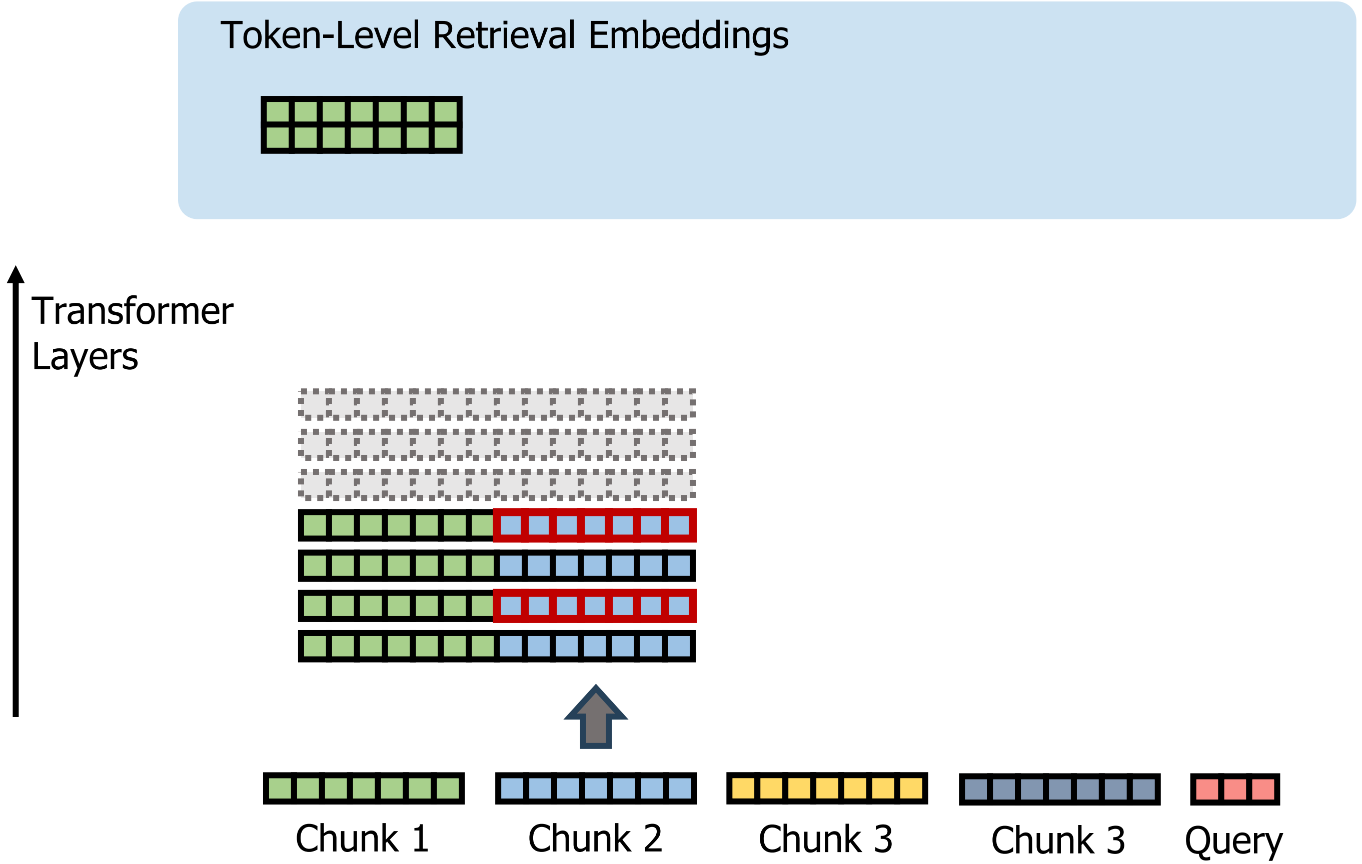
- Use recurrence to generate query-unaware, token-level embeddings



Forward the next chunk, conditioned on the previous chunk's KV cache

Compress: Embedding Extraction Stage

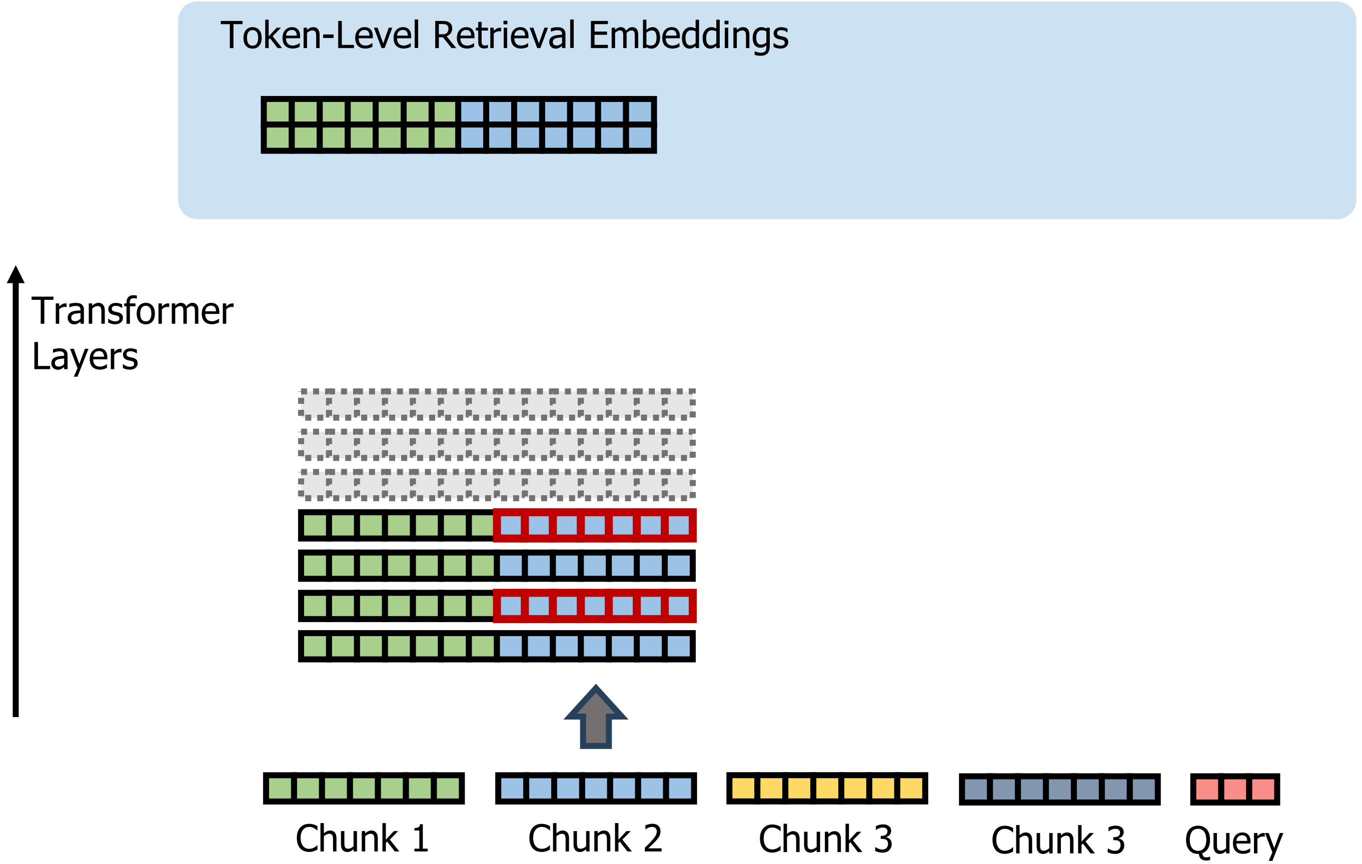
- Use recurrence to generate query-unaware, token-level embeddings



Gather selected QKV heads, and save them as token-level retrieval embeddings.

Compress: Embedding Extraction Stage

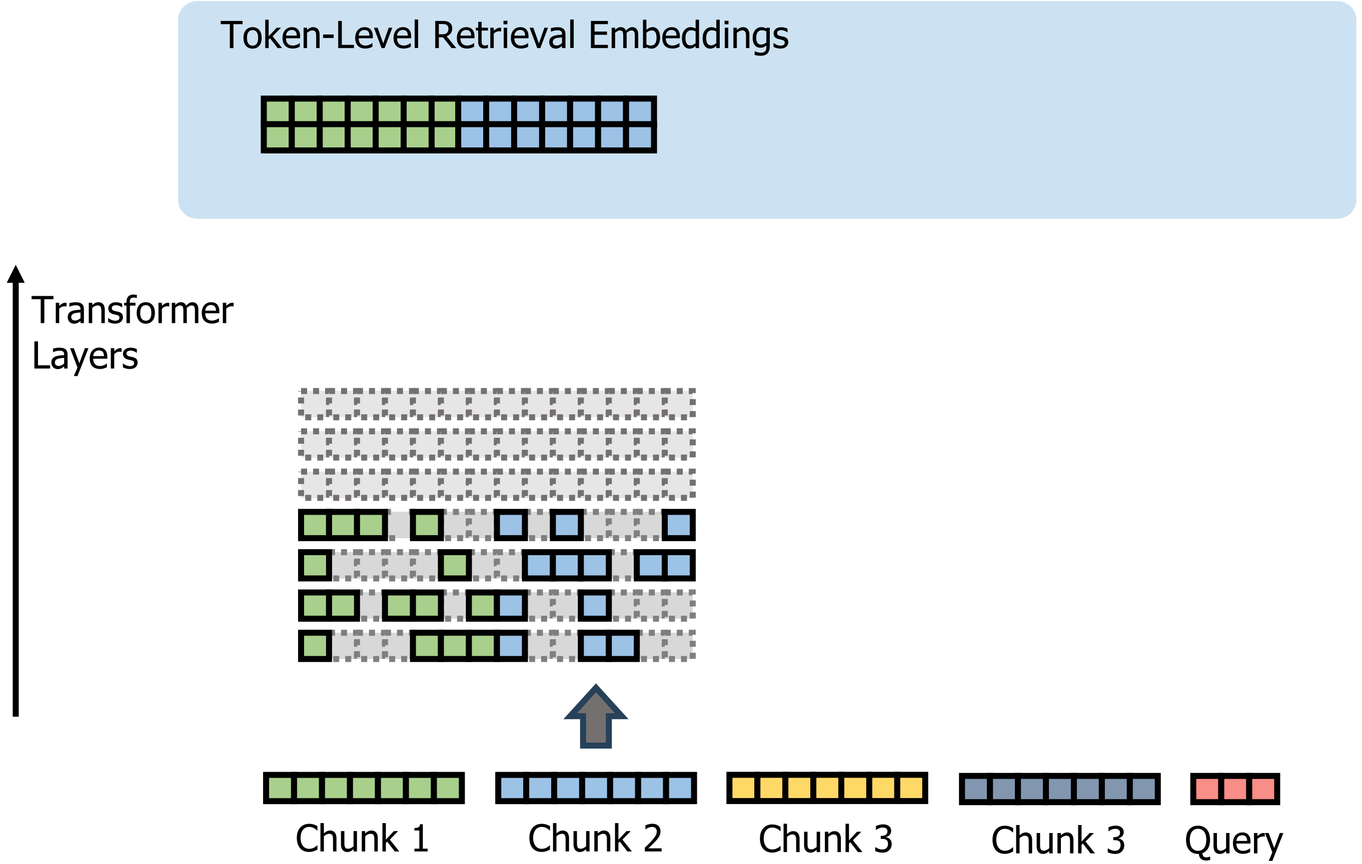
- Use recurrence to generate query-unaware, token-level embeddings



Gather selected QKV heads, and save them as token-level retrieval embeddings.

Compress: Embedding Extraction Stage

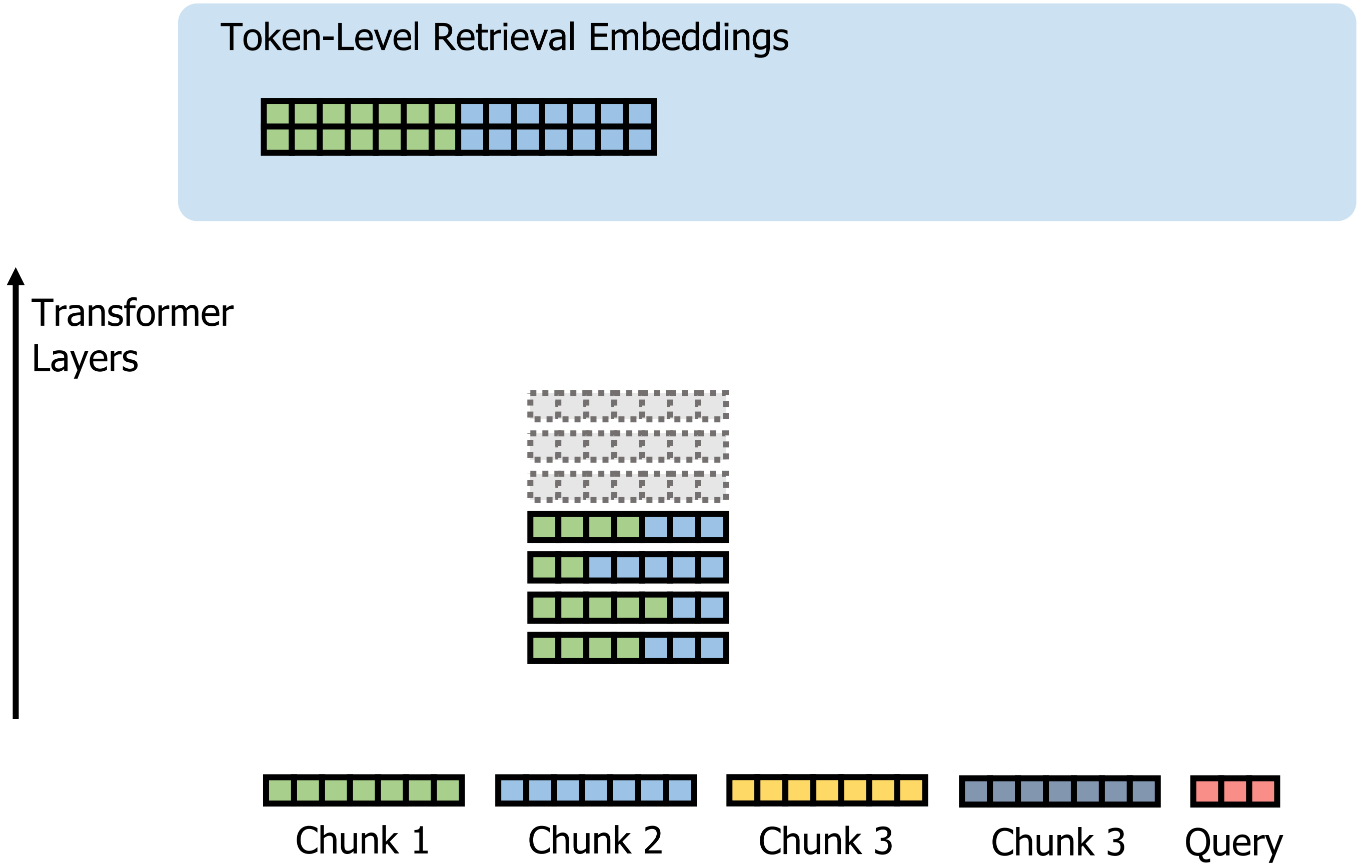
- Use recurrence to generate query-unaware, token-level embeddings



Select surviving tokens from the KV cache in a query-independent way (H2O)

Compress: Embedding Extraction Stage

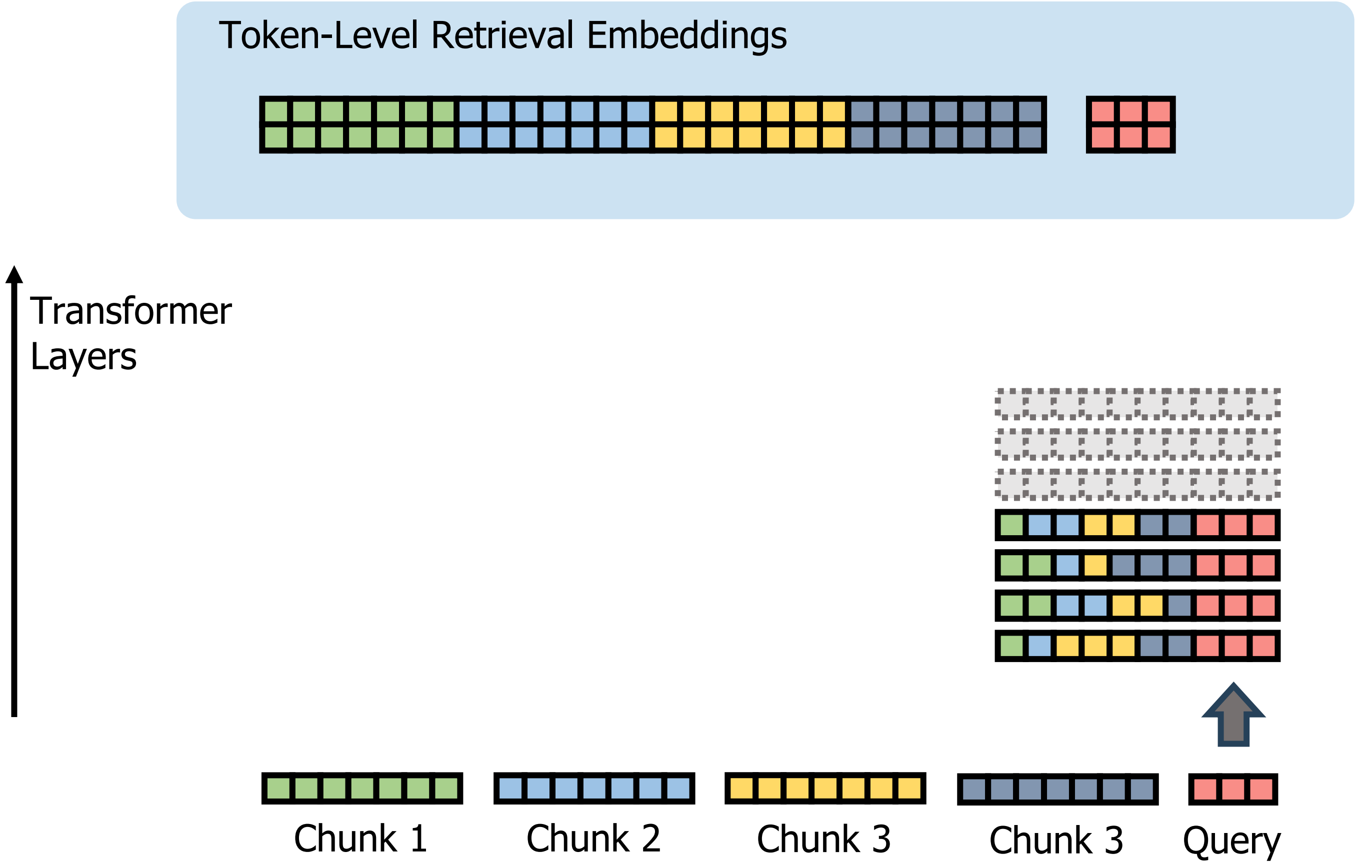
- Use recurrence to generate query-unaware, token-level embeddings



Perform token eviction to compress the KV cache.

Compress: Embedding Extraction Stage

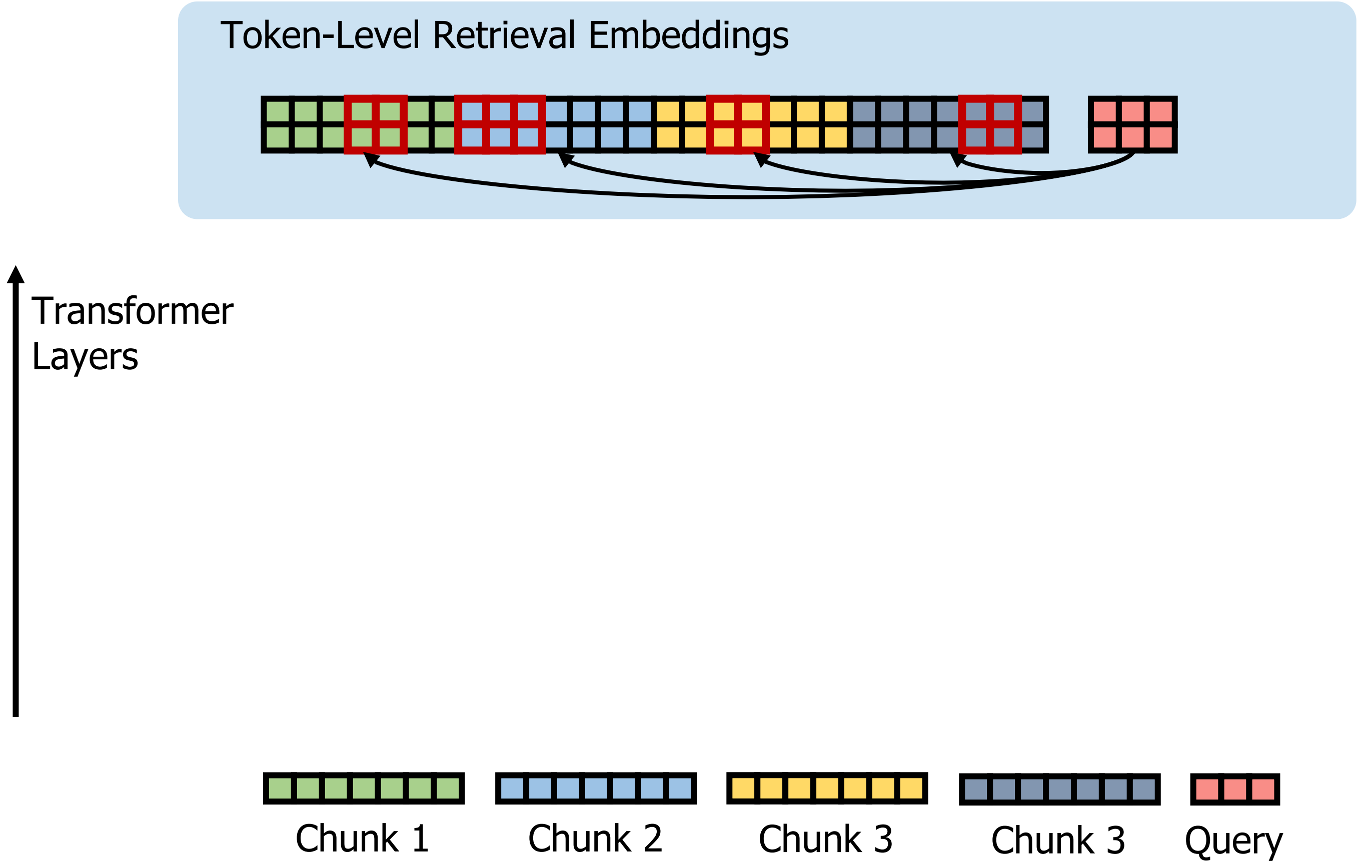
- Use recurrence to generate query-unaware, token-level embeddings



Repeat the process until the end.

Gather: Token Identification Stage

- Use the embeddings to identify the core tokens from the long input

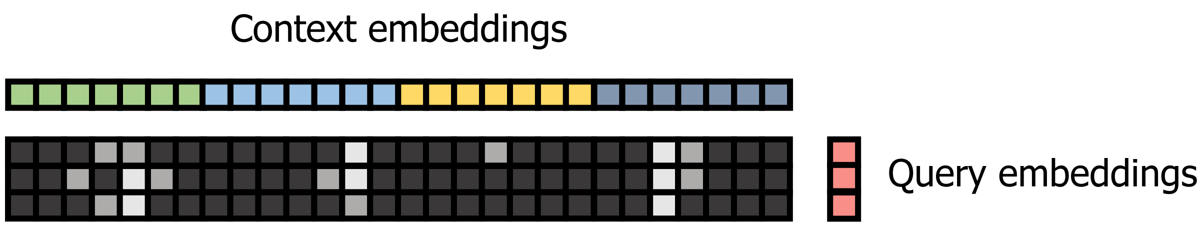


Important input segments are identified using cosine similarity search on the token-level embeddings.

Gather: Token Identification Stage

- How are the significance scores computed?

1. Compute cosine similarity scores

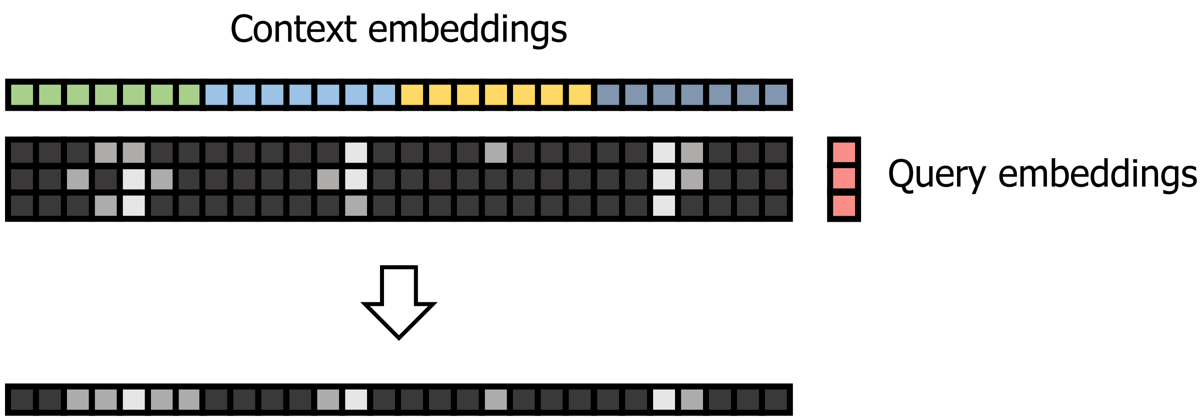


Gather: Token Identification Stage

- How are the significance scores computed?

1. Compute cosine similarity scores

2. Max aggregation along query dimension



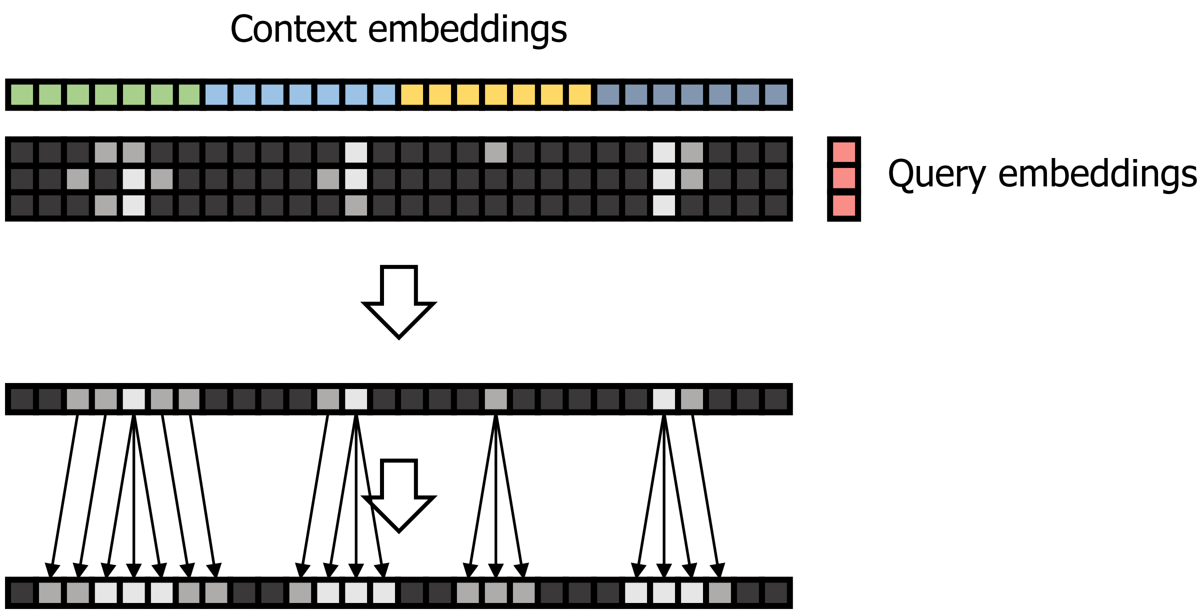
Gather: Token Identification Stage

- How are the significance scores computed?

1. Compute cosine similarity scores

2. Max aggregation along query dimension

3. Max pooling with neighboring tokens



Gather: Token Identification Stage

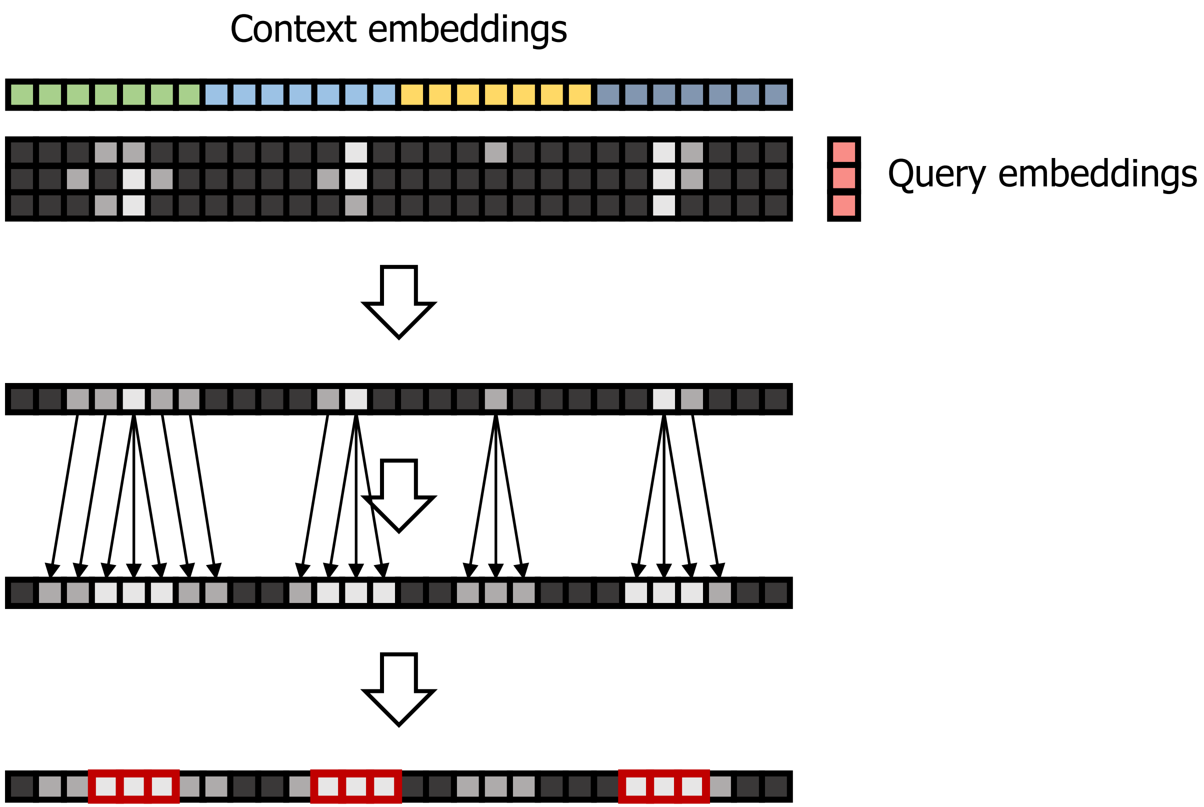
- How are the significance scores computed?

1. Compute cosine similarity scores

2. Max aggregation along query dimension

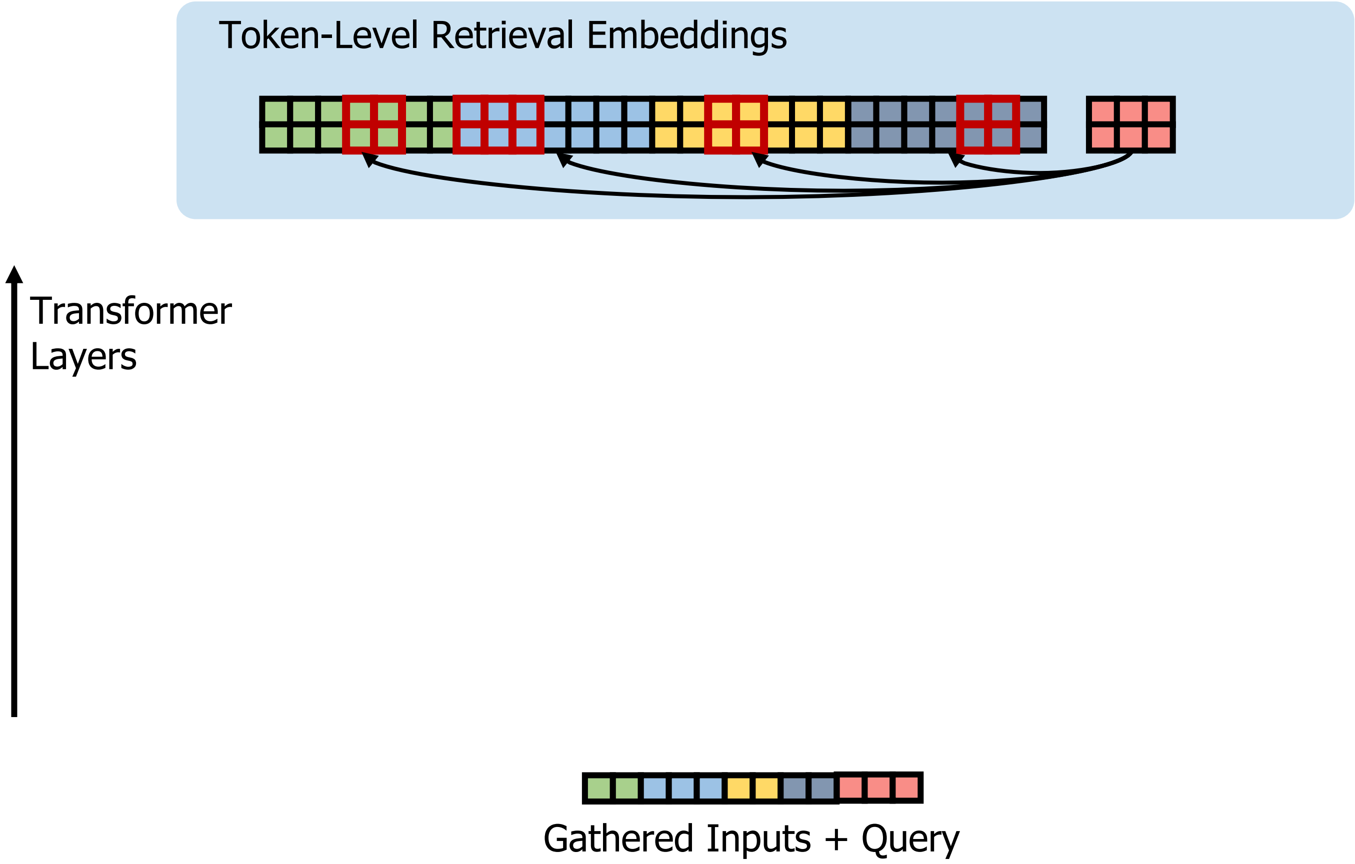
3. Max pooling with neighboring tokens

4. Select N tokens with the highest score



Gather: Token Identification Stage

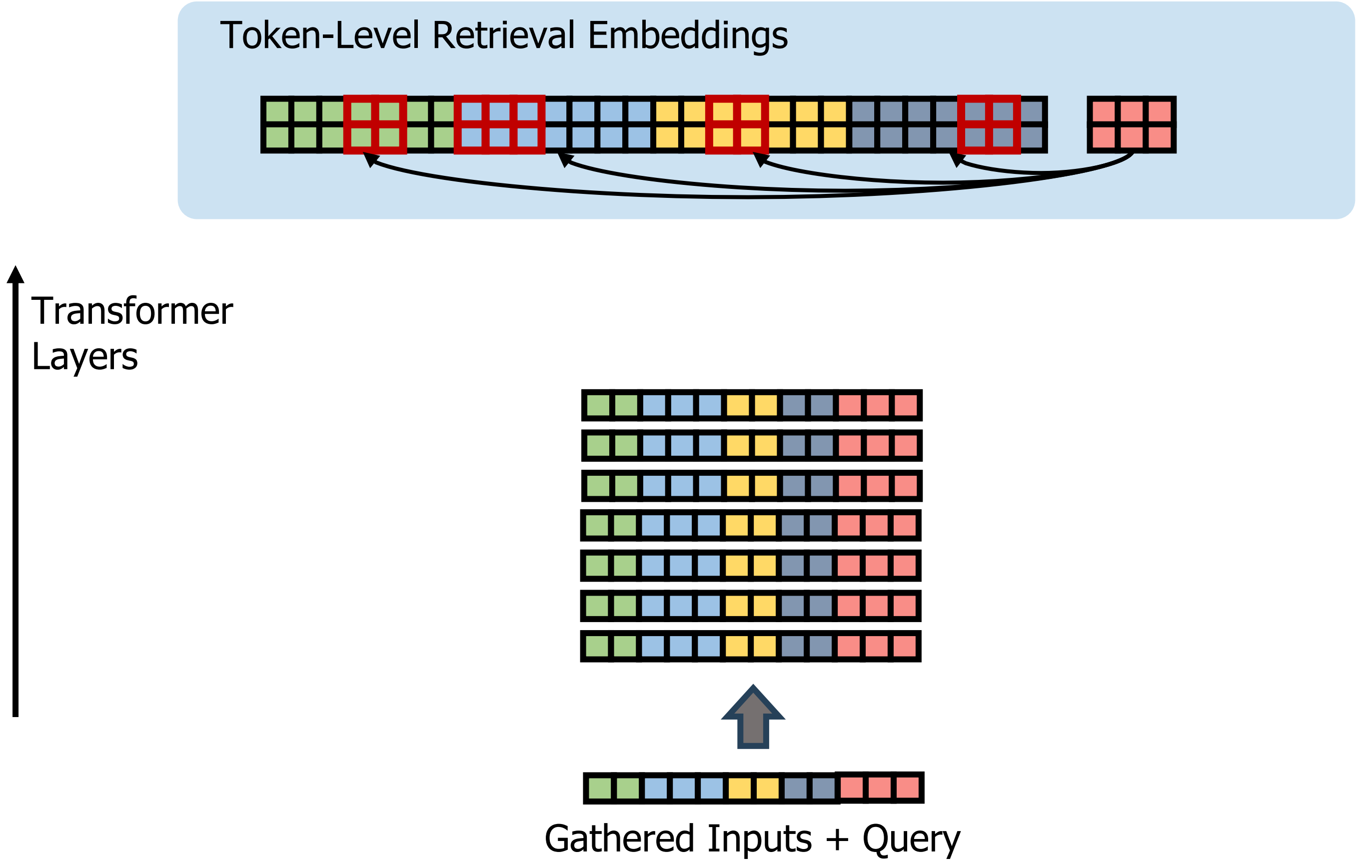
- Use the embeddings to identify the core tokens from the long input



Gather the corresponding input tokens.

Recompute: Cache Recomputation Stage

- Recompute the KV cache with the selected tokens



Recompute the KV cache by forwarding the gathered inputs.
This cache is used for generating further response.

Evaluations: Needle-In-a-Haystack

- REFORM shows perfect recall up to 1M tokens
  - Task setup: Hide a 'needle' sentence in irrelevant text (Paul Graham's essays)
  - All baselines struggle at longer contexts

Needle: The best thing to do in San Francisco is to eat a sandwich and sit in Dolores Park on a sunny day.

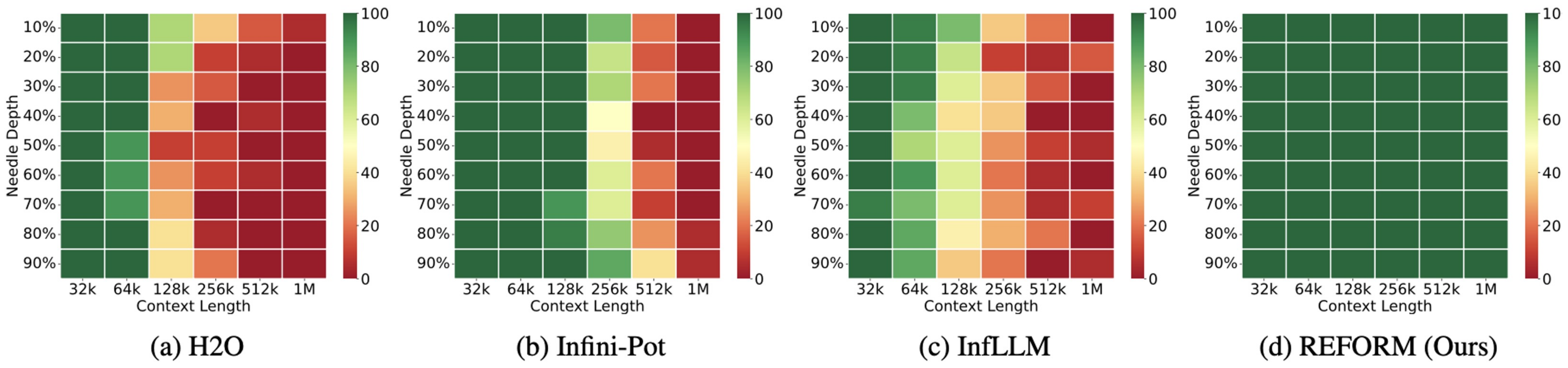


Figure 3. **Needle-In-A-Haystack Evaluation.** We compare the precise retrieval ability of different long-context handling approaches by visualizing the needle-in-a-haystack performance at different depth and context lengths. All experiments were done with Qwen2.5-7B-Instruct model, and the performance is averaged over 20 samples.

Evaluations: Synthetic Benchmarks

- REFORM outperforms the baselines on two synthetic benchmarks
  - RULER contains more diverse and challenging needle-in-a-haystack tasks, along with aggregation & question answering
  - BABILong introduces more challenging tasks (e.g. multi-hop, multi-arg reasoning)

Table 2: **Evaluation on RULER and BABILong.** We measure the performance on an extended version of the RULER [17] and BABILong [18] benchmark. We report the averaged performance of all tasks at different context lengths. The best values are highlighted in **bold**.

	RULER							BABILong				
	64k	128k	200k	300k	400k	500k	1M	64k	128k	256k	512k	1M
Mistral-Nemo-Instruct-2407												
Truncation	32.6	20.4	17.8	15.2	12.3	12.5	10.8	32.2	26.2	17.0	13.6	14.0
StreamingLLM	27.6	13.8	11.6	9.3	7.2	7.1	4.7	38.8	23.4	15.4	11.0	6.2
TOVA	21.6	15.3	14.0	11.8	7.9	8.7	4.6	37.8	23.6	14.4	9.6	3.4
H2O	15.1	7.4	7.8	5.6	4.2	5.7	3.6	38.0	25.2	16.2	7.2	3.6
InfiniPot	26.9	19.4	15.6	14.5	12.7	13.4	12.0	39.6	26.8	18.6	11.2	8.8
InfLLM	52.7	39.7	28.5	24.9	20.9	22.0	23.3	40.6	34.0	23.6	13.0	9.6
REFORM (Ours)	**79.9**	**81.1**	**83.0**	**84.6**	**84.1**	**83.5**	**75.5**	**57.4**	**51.4**	**50.6**	**47.6**	**48.8**
Qwen2.5-7B-Instruct												
Truncation	46.3	25.1	21.8	17.4	14.9	15.2	11.3	48.4	33.4	27.4	20.0	15.6
StreamingLLM	43.5	25.3	18.7	17.3	11.8	11.8	9.1	53.4	40.6	33.2	23.8	19.6
TOVA	66.2	27.7	25.7	25.8	21.9	20.4	17.0	56.0	46.6	40.6	29.4	21.8
H2O	51.8	20.9	18.5	17.1	11.6	12.1	8.7	57.0	41.6	36.4	24.6	18.8
InfiniPot	65.7	51.7	39.2	33.9	27.8	26.7	23.7	59.6	51.0	53.4	48.2	40.2
InfLLM	47.1	34.2	29.2	24.0	22.0	23.2	23.8	43.0	29.2	20.4	15.4	11.4
REFORM (Ours)	**78.2**	**75.8**	**74.7**	**74.9**	**74.9**	**73.0**	**75.1**	**61.6**	**60.4**	**59.8**	**58.8**	**58.8**

Evaluations: Diverse Domains & Modalities

- Infinite-Bench constructs tasks from more realistic text: long books and dialogues

Table 3: **Evaluation on ∞-Bench.** We evaluate each method on more 10 datasets from ∞-Bench [19]. We did not evaluate on C.Run and M.Calc datasets since no method was capable of achieving a nonzero score with these models. The best values are highlighted in **bold**.

	R.PK	R.Num	R.KV	En.Sum	En.QA	En.MC	En.Dia	Zh.QA	C.Debug	M.Find	Avg.
Mistral-Nemo-Instruct-2407											
Truncation	27.1	21.4	3.6	13.7	16.0	51.1	11.5	25.2	28.9	20.3	21.9
StreamingLLM	28.1	15.3	0.0	12.5	12.6	45.9	6.5	19.2	27.2	0.0	16.7
TOVA	82.2	47.0	0.0	12.3	13.8	47.2	8.0	6.6	25.1	4.6	24.7
H2O	31.5	9.5	0.0	14.2	17.6	49.3	6.0	21.2	26.1	15.7	19.1
InfiniPot	84.1	13.6	0.0	11.9	17.1	52.0	7.0	11.3	27.4	15.7	24.0
InfLLM	100.0	100.0	1.0	16.9	17.4	58.1	7.0	24.5	24.1	27.1	37.6
REFORM (Ours)	100.0	100.0	88.2	18.2	18.0	70.3	18.5	26.7	25.9	36.0	**50.2**
Qwen2.5-7B-Instruct											
Truncation	27.1	27.1	7.4	29.0	13.3	43.2	15.0	9.34	37.1	45.7	25.4
StreamingLLM	28.8	28.8	6.0	29.2	8.6	52.4	14.5	9.51	32.5	28.6	23.9
TOVA	100.0	100.0	1.2	29.4	8.6	56.8	15.0	10.65	34.3	42.6	39.8
H2O	93.1	85.4	0.0	31.0	11.0	56.3	15.5	11.97	34.8	44.6	38.4
InfiniPot	100.0	99.8	0.8	30.6	11.3	59.0	17.0	9.99	36.6	44.9	41.0
InfLLM	100.0	99.8	1.6	27.6	9.6	38.0	12.0	10.41	29.7	45.1	37.4
REFORM (Ours)	100.0	100.0	32.8	27.8	16.5	61.6	21.5	11.81	33.0	21.7	**42.7**

Evaluations: Diverse Domains & Modalities

- RepoEval is a repository-level code completion benchmark
  - Model: Qwen2.5-Coder-1.5B/7B-Instruct
- MM-NIAH showcases performance on multi-modal applications
  - Model: Pixtral-12B-2409

Table 4: **Evaluation on RepoEval and MM-NIAH.** For RepoEval, we report the edit similarity (ES) score on RepoEval api-level completion task and line-level completion task with 1.5B and 7B models. For MM-NIAH, we report normalized performance across input lengths to ensure equal contribution from each context length range. We do not run multi-modal evaluation for InfLLM, as its implementation only supports text-based models. Best results are in **bold**.

Method	RepoEval				MM-NIAH			
	1.5B API	1.5B Line	7B API	7B Line	Retrieval	Counting	Reasoning	Avg.
Truncate	54.8	63.9	59.2	59.5	72.2	18.7	51.2	47.4
StreamingLLM	55.0	62.7	59.9	58.4	71.9	17.8	49.8	46.5
TOVA	54.7	62.2	59.7	59.8	82.9	18.8	54.1	52.0
H2O	55.1	63.4	61.2	59.6	83.3	18.9	53.5	51.9
InfiniPot	59.4	68.4	66.2	63.8	85.4	18.8	54.7	53.0
InfLLM	61.8	66.8	64.3	66.3	N/A	N/A	N/A	N/A
REFORM (Ours)	**65.3**	**72.4**	**68.7**	**69.4**	**89.2**	**22.0**	**61.3**	**57.5**

Efficiency Analysis

- REFORM improves both inference time and memory requirements
  - Most gains come from the early-exit strategy
  - Upper-layer computations are skipped, and corresponding KV cache is not required

Table 7. **Efficiency Analysis.** We compare the peak memory usage and inference time required for generating 10 tokens conditioned on 256k inputs. All measurements are made with the Mistral-NeMo-Instruct-2407 model on a single H100 GPU, and are averaged over 10 samples. The best values are highlighted in **bold**.

	Inference Time (sec.)	Peak Memory (GB)
StreamingLLM	36.58	37.34
H2O	41.33	37.85
TOVA	39.46	37.06
InfiniPot	40.90	37.06
InfLLM	129.14	51.62
REFORM (Ours)	**27.24**	**35.00**

Efficiency Analysis

- REFORM improves both inference time and memory requirements
  - Lowest latency is achieved across different input length and processing phase

Table 8: **Latency Breakdown.** (a) Time to first token (seconds) measurements and (b) time per output token (seconds) measurements (Mistral-Nemo-Instruct-2407, single H200, averaged over 20 runs and 200 tokens generated per measurement).

(a) Time to first token (seconds).

Model	256k	512k	1M
StreamingLLM	30.59	68.22	143.57
InfiniPot	35.77	73.67	149.64
InfLLM	95.71	213.23	474.96
REFORM (Ours)	**26.24**	**53.68**	**108.64**
- Compress + Gather	25.84	53.28	108.24
- Recompute	0.40	0.40	0.40

(b) Time per output token (seconds)

Model	256k	512k	1M
StreamingLLM	0.111	0.111	0.111
InfiniPot	0.256	0.256	0.256
InfLLM	0.259	0.267	0.329
REFORM (Ours)	**0.040**	**0.040**	**0.040**

Comparison with RAG

- REFORM has several benefits over RAG
  - Avoids context fragmentation from chunking, by using KV cache compression
  - Architecture-level solution: has broad applicability, and can be seamlessly applied to diverse domain & modalities
  - Does not require an external retriever
- Furthermore, combining them can achieve even higher performance

Table 6. **Comparison with RAG.** We compare the performance of RAG methods and REFORM on four groups of needle-in-a-haystack datasets from RULER at 300k contexts, using Mistral-NeMo-Instruct-2407 model.

	Single	Multikey	Multivalue	Multiquery
Sparse RAG	86.7	77.3	88.5	90.0
Dense RAG	87.3	57.3	82.5	78.0
REFORM	**99.3**	93.3	98.5	**100.0**
REFORM + RAG	**99.3**	**94.7**	**99.0**	**100.0**

Ablation Studies

- REFORM is compatible with diverse compression approaches.
- Head selection is important for performance.
- Max pooling with adjacent tokens is crucial for maintaining the integrity of cache.

	RULER	BABILong
REFORM (Ours)	**84.6**	**47.6**
w/ StreamingLLM	82.7	44.6
w/ TOVA	81.4	46.8
w/ Random heads	80.3	43.0
w/ Worst heads	44.7	22.8
w/ Kernel size 5	18.4	36.8
w/ Kernel size 17	39.4	45.8

Ablation results.

Appendix

Algorithm

Algorithm 1 Overview of REFORM

```
procedure FORWARDCHUNK(chunk, cache, emb)
    /* Initialize hidden states */
    hs ← input
    /* Forward with early exit */
    for layer in model_layers[:early_exit_layer] do
        hs, cache, qkv ← layer.Forward(hs, cache)
        /* Save selected embeddings */
        emb.SaveSelected(qkv)
    end for
    /* Evict less important tokens */
    cache ← Compress(cache)
    return cache, emb
end procedure
procedure REFORM(input)
    /* Initialize */
    cache, emb ← EmptyInit()
    /* Prepare input chunks */
    context, query ← SplitQuery(input)
    chunks ← ChunkInputs(context) + [query]
    /* Recurrent chunked forwarding */
    for c_i in chunks do
        cache, emb ← ForwardChunk(c_i, cache, emb)
    end for
    /* Gather relevant inputs */
    relevant_inputs ← GatherRelevant(input, emb)
    /* On-demand recomputation */
    cache ← model.Forward(relevant_inputs)
    return cache
end procedure
```

Head Selection

For Mistral-NeMo-Instruct-2407, the following heads are used:

1. Query head 9 at layer 15
2. Value head 5 at layer 19
3. Value head 0 at layer 27
4. Value head 7 at layer 27

For Qwen2.5-7B-Instruct, the following heads are used:

1. Value head 3 at layer 7
2. Key head 0 at layer 14
3. Value head 3 at layer 14
4. Value head 0 at layer 19

For Qwen2.5-Coder-1.5B-Instruct, the following heads are used:

1. Query head 3 at layer 8
2. Value head 1 at layer 11
3. Key head 0 at layer 14
4. Value head 0 at layer 15

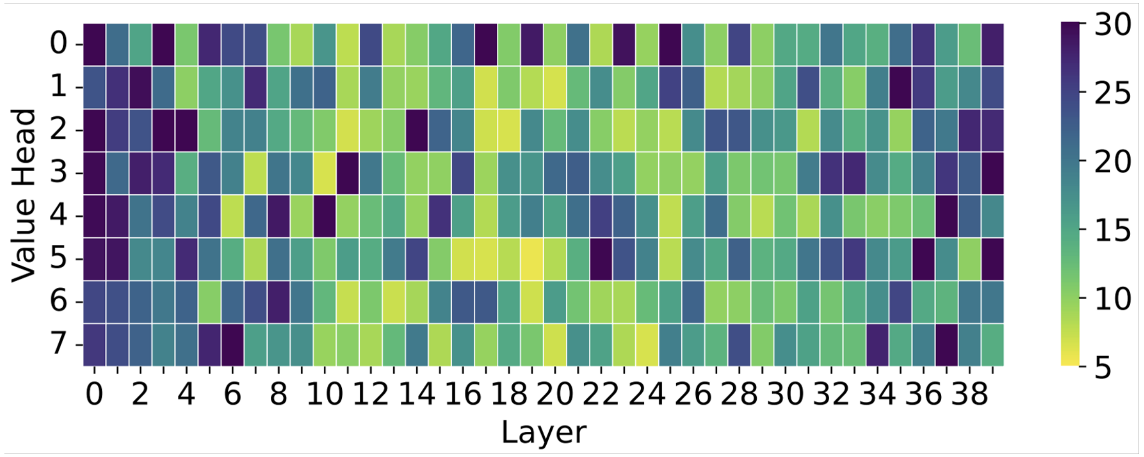
For Qwen2.5-Coder-7B-Instruct, the following heads are used:

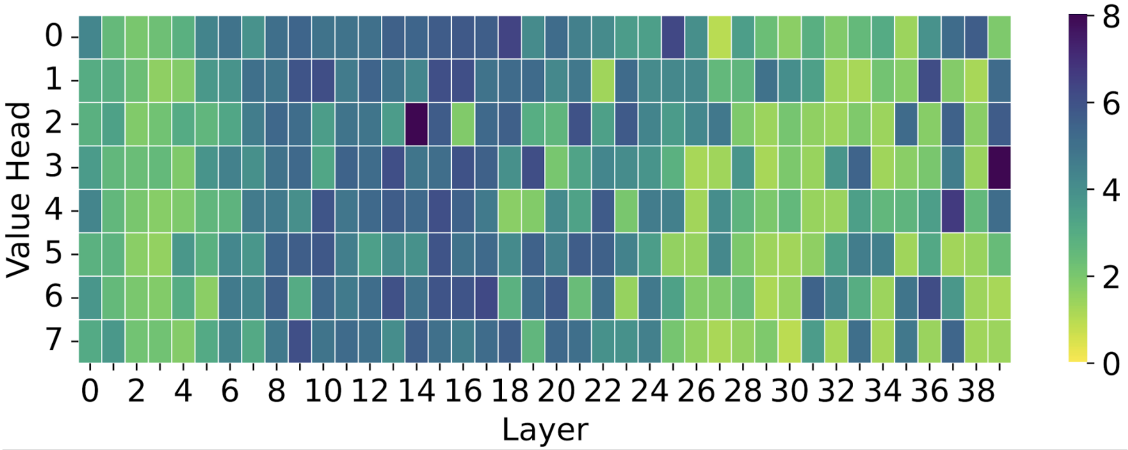
1. Value head 2 at layer 13
2. Key head 0 at layer 14
3. Value head 3 at layer 14
4. Query head 4 at layer 14

For Pixtral-12B-2409, the following heads are used:

1. Value head 3 at layer 10
2. Value head 5 at layer 19
3. Value head 0 at layer 27
4. Value head 7 at layer 27

MNR Score Distribution





Qwen2.5-32B-Instruct Evaluations

Table 9: **Performance of Qwen2.5-32B-Instruct.** We report the performance on key long-context benchmarks for H2O, InfiniPot, and REFORM. The best values are highlighted in **bold**.

	RULER Single 300k	RULER Multikey 300k	RULER Multivalue 300k	RULER Multiquery 300k	BABILong 256k
H2O	38.7	2.7	14.0	6.0	31.0
InfiniPot	69.3	19.3	40.0	74.0	54.2
REFORM (Ours)	**100.0**	**90.0**	**96.0**	**100.0**	**67.6**