Institute of Computing Technology, CAS

Research Center for Intelligent Computing Systems

CTA: Hardware-Software Co-design for Compressed Token Attention Mechanism

Haoran Wang, Haobo Xu, Ying Wang, Yinhe Han

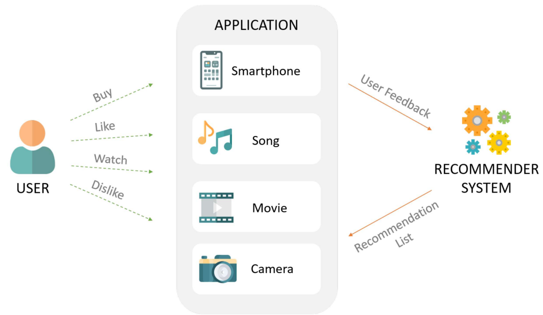
Research Center for Intelligent Computing Systems
Institute of Computing Technology, Chinese Academy of Sciences

HPCA 2023

Attention mechanism significantly improves neural network capability

- Transformer-based models have been widely applied
  - Natural Language Processing(NLP): machine translation, question answering, text classification, language modeling…
  - Computer Vision(CV): image segmentation, video classification, zero-shot image classification…
  - Recommendation system

Machine translation:
Google translate

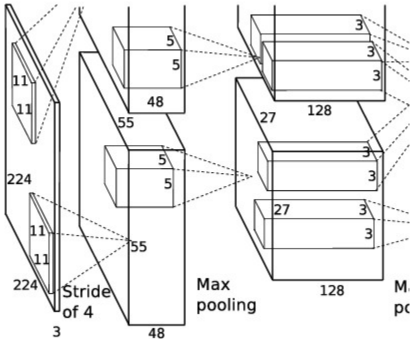


Online recommendations

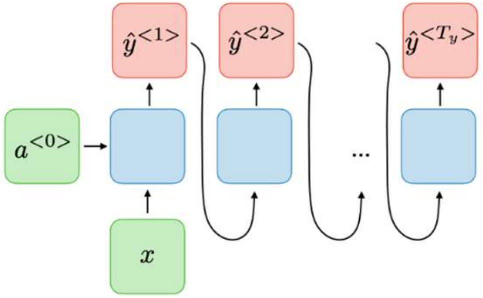
Language model for dialogue:
ChatGPT

Attention mechanism significantly improves neural network capability

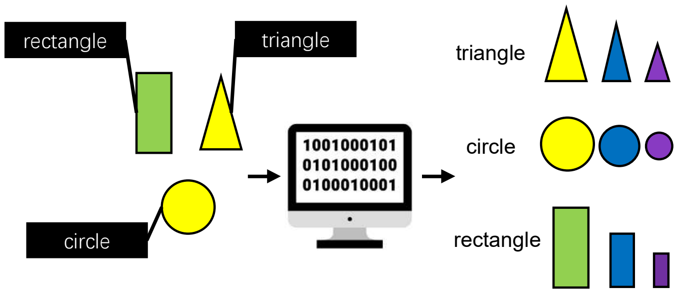
- The attention mechanism is the key enabler in Transformer-based models
  - Modeling relations among elements without regard to their distance in sequence
  - Parallelizing the sequence modeling
  - Enabling using larger amount of no-label data and achieves better performance



CNN models local relations by convolution but fail to model distant relations



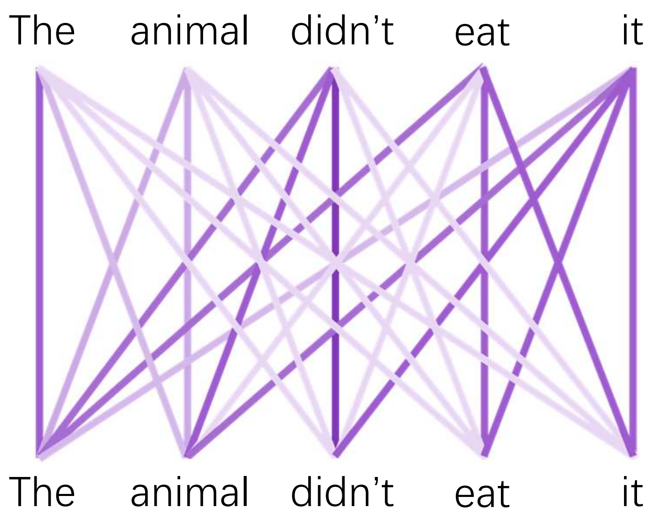
RNN utilizes sequential computation with less parallelism

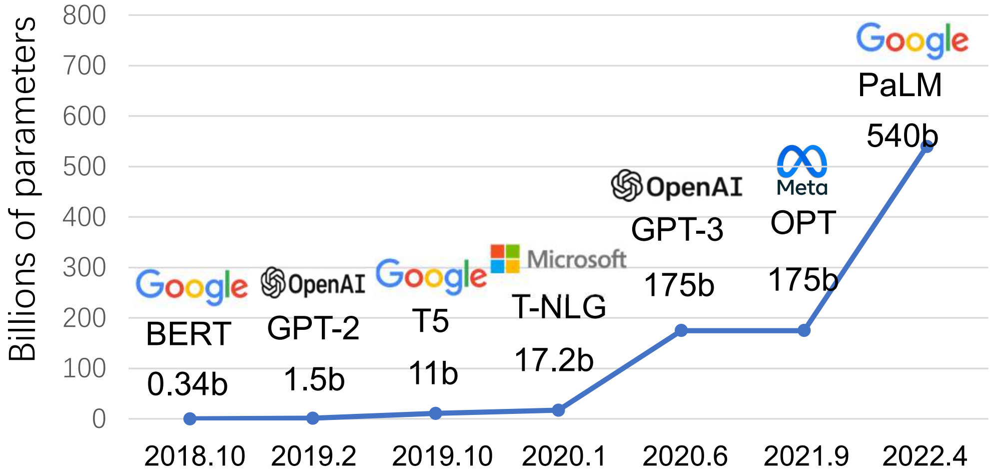


Previous NN models need more labeling to achieve intelligence

Efficient attention mechanism is important

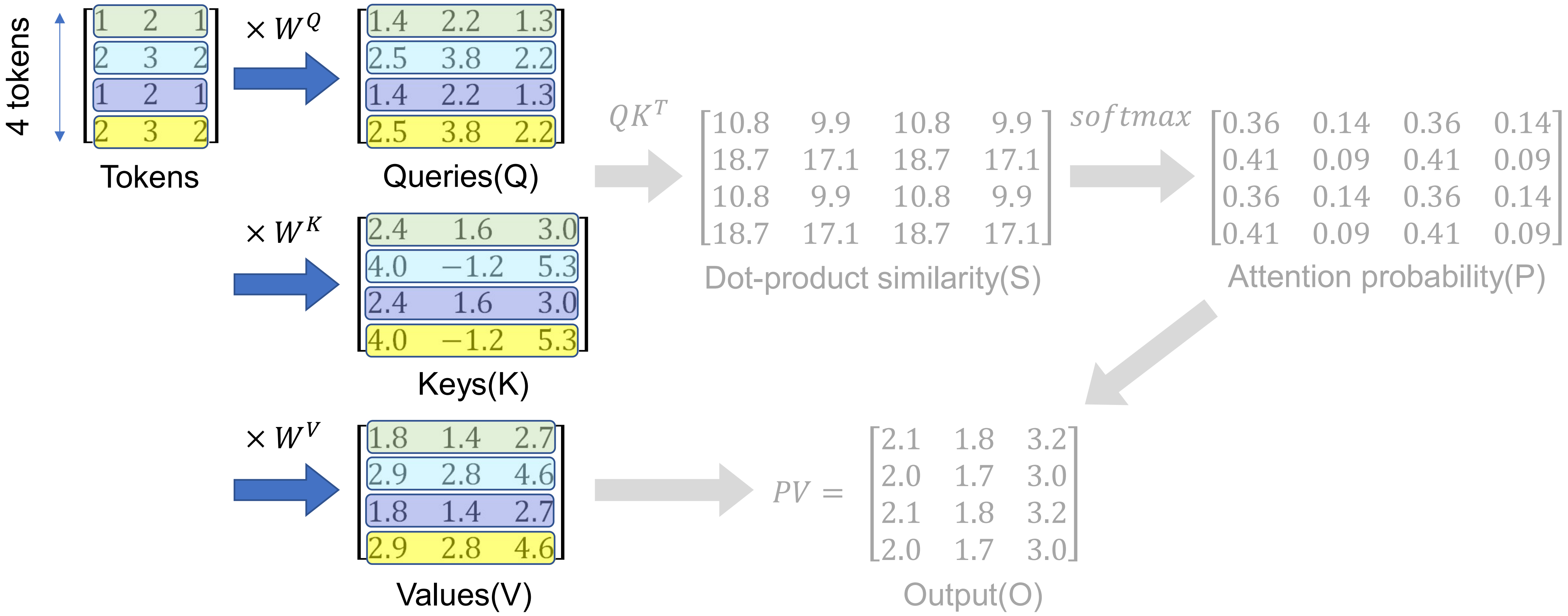
- Attention mechanism incurs high overhead
  - Attention mechanism models dense relations among sequence elements: 30%-50% of the computation time is spent on attention mechanism in Transformer-based models
  - Transformer-based models are growing recklessly in size





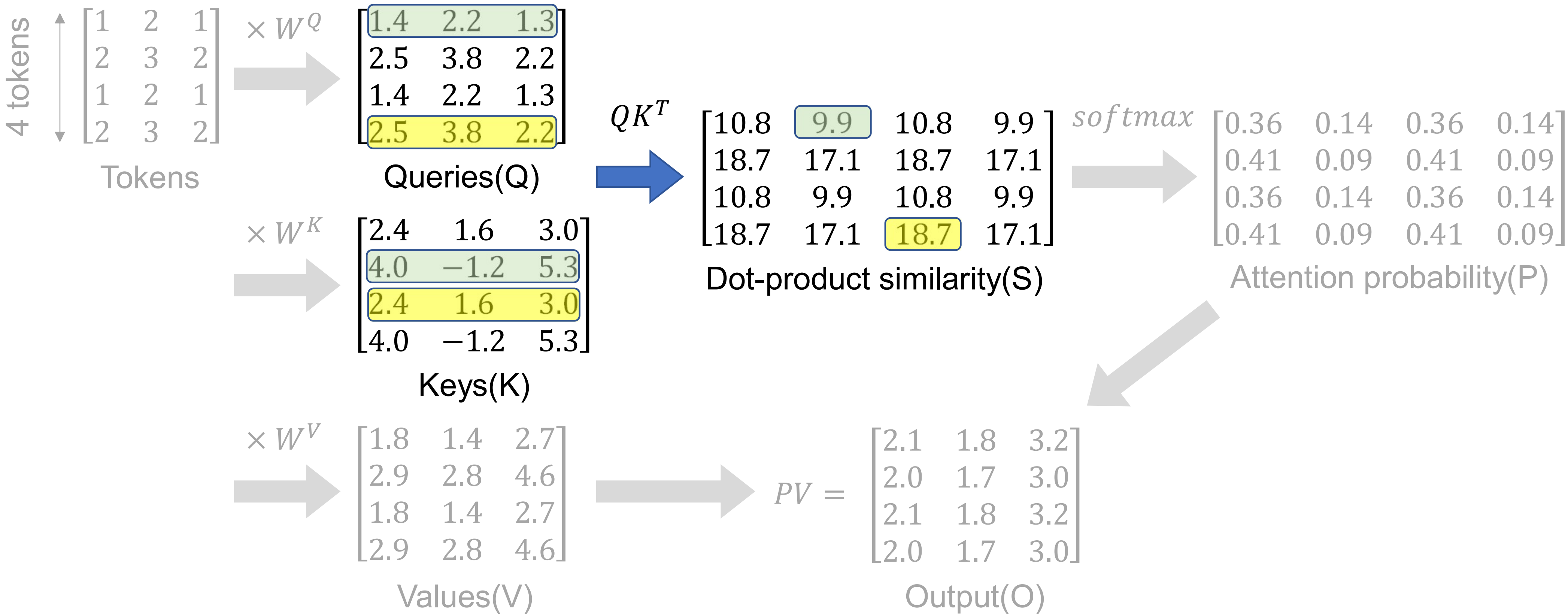
Attention mechanism overview

Step1: Three linear transformations are applied on tokens to generate queries, keys and values



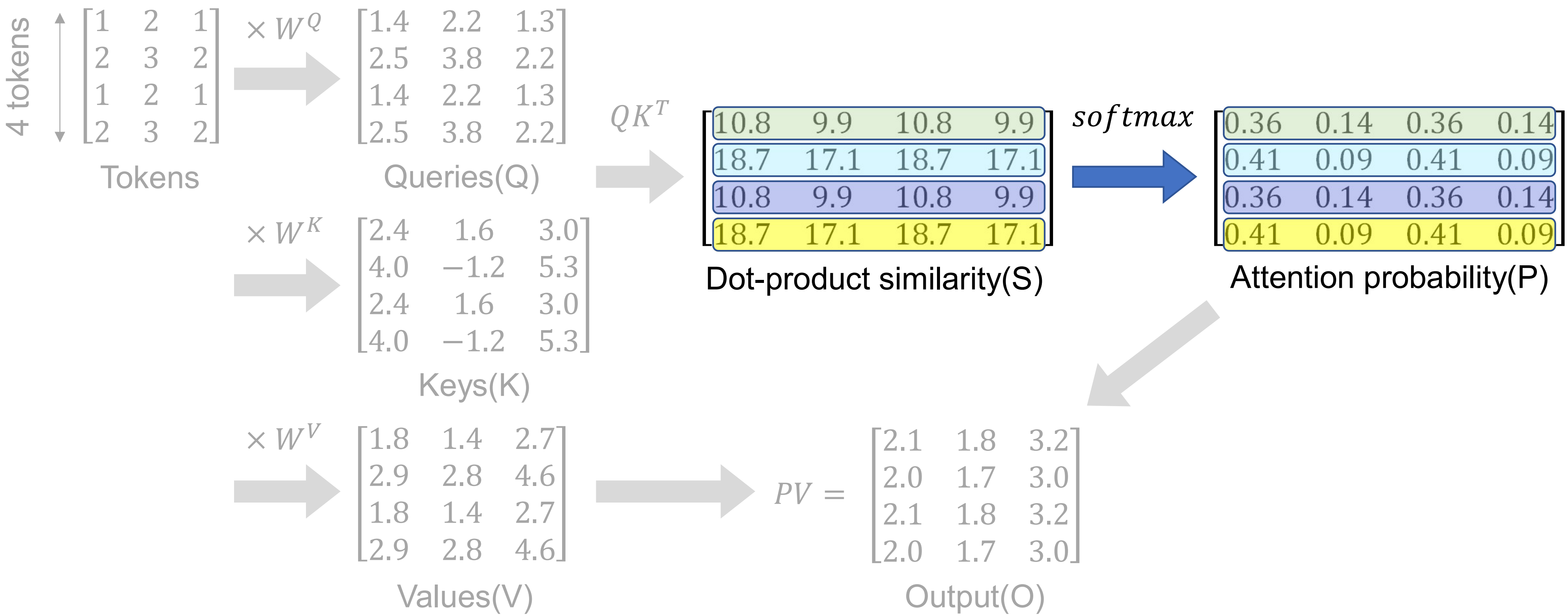
Attention mechanism overview

Step2: Calculate dot-product similarity score between each pair of query and key



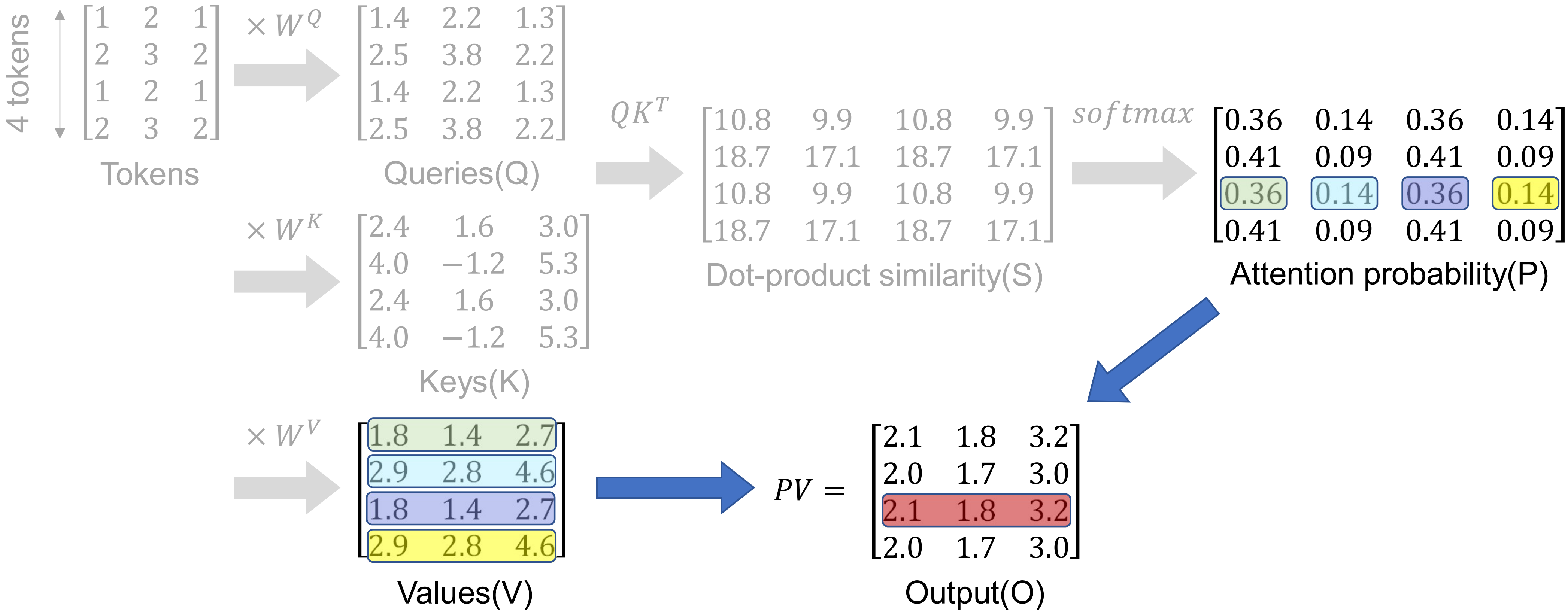
Attention mechanism overview

Step3: Apply softmax to normalize each row of similarity scores to generate attention probabilities



Attention mechanism overview

Step4: Using attention probabilities as weights to calculate weighted sum of values



Previous insights

KEY OBSERVATION: Given a query, keys do not contribute equally to the output.

After softmax normalization, for those keys that have small dot-product with this query, their attention probability is near 0 and can be pruned.

Queries

query1
[0.3 0.9 0.3]

query2
[-0.3 0.9 -0.3]

Keys

[-3.0 0.8 0.0]
[1.0 0.1 1.0]
[-1.0 0.0 0.9]
[1.4 0.2 0.5]

Scores for query1 → [-0.75 0.99 -1.17 1.05] → Softmax → [$\boxed{0.07}$ 0.43 $\boxed{0.05}$ 0.45]

Prune key1 and key3

Scores for query2 → [1.65 -0.81 1.17 -0.69] → Softmax → [0.56 $\boxed{0.05}$ 0.34 $\boxed{0.05}$]

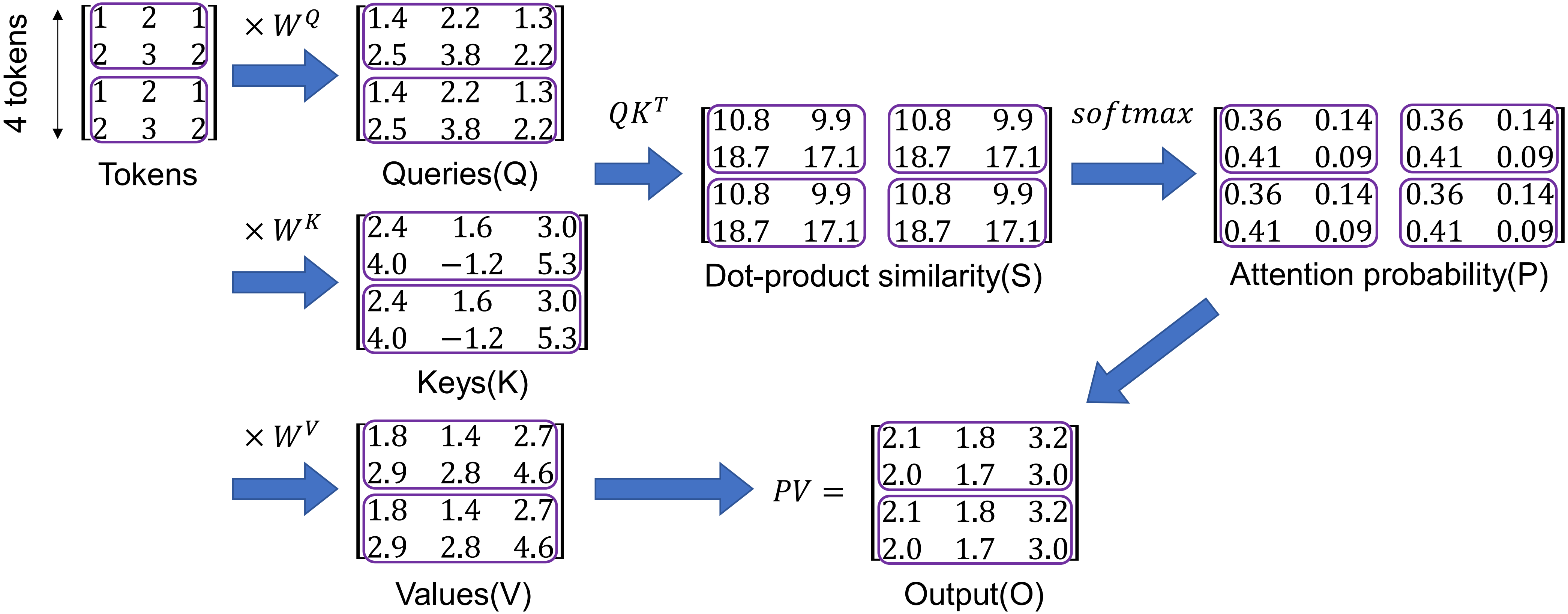
Prune key2 and key4

Query-specific pruning breaks the matrix multiplication formulation for multi-query processing.

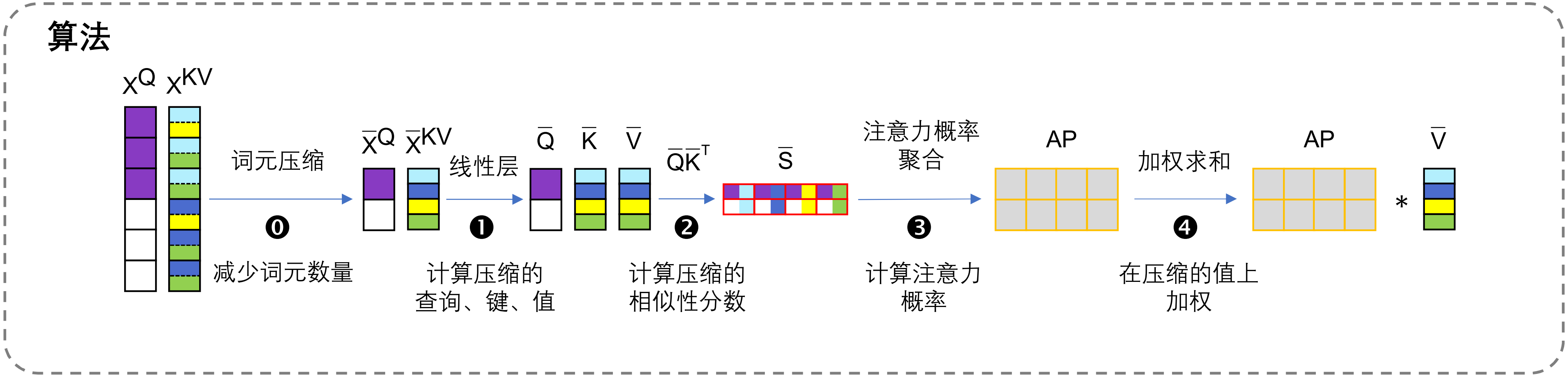
Our work keeps the matrix multiplication formulation with shrinking matrix scale.

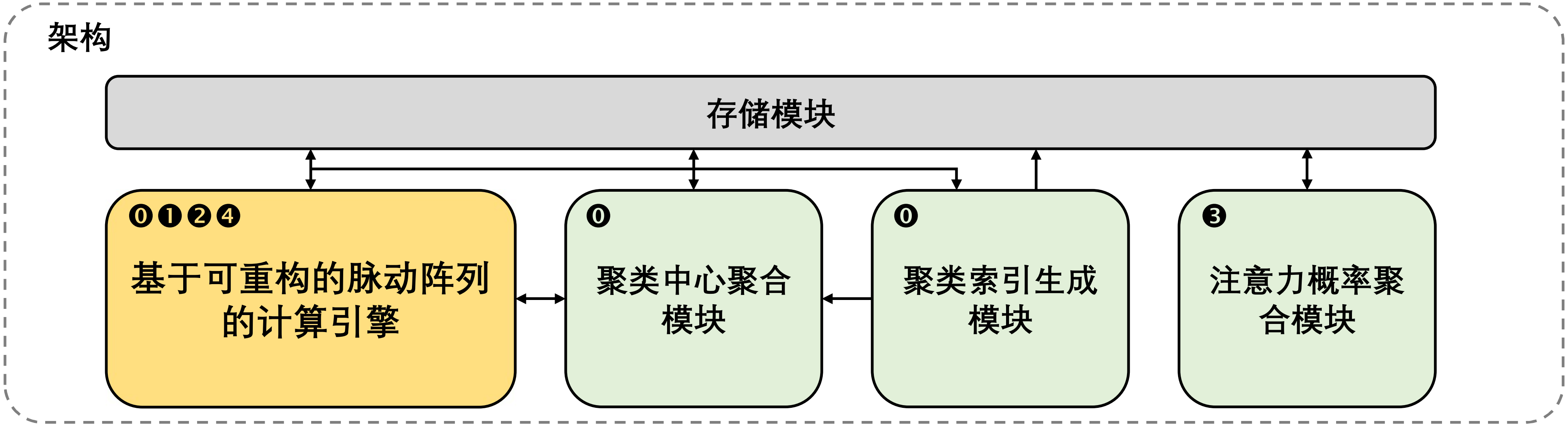
Motivation

KEY IDEA: Similar tokens lead to similar patterns throughout attention computation.
Large amount of computation can be avoided if we can remove semantic repetitions in tokens.

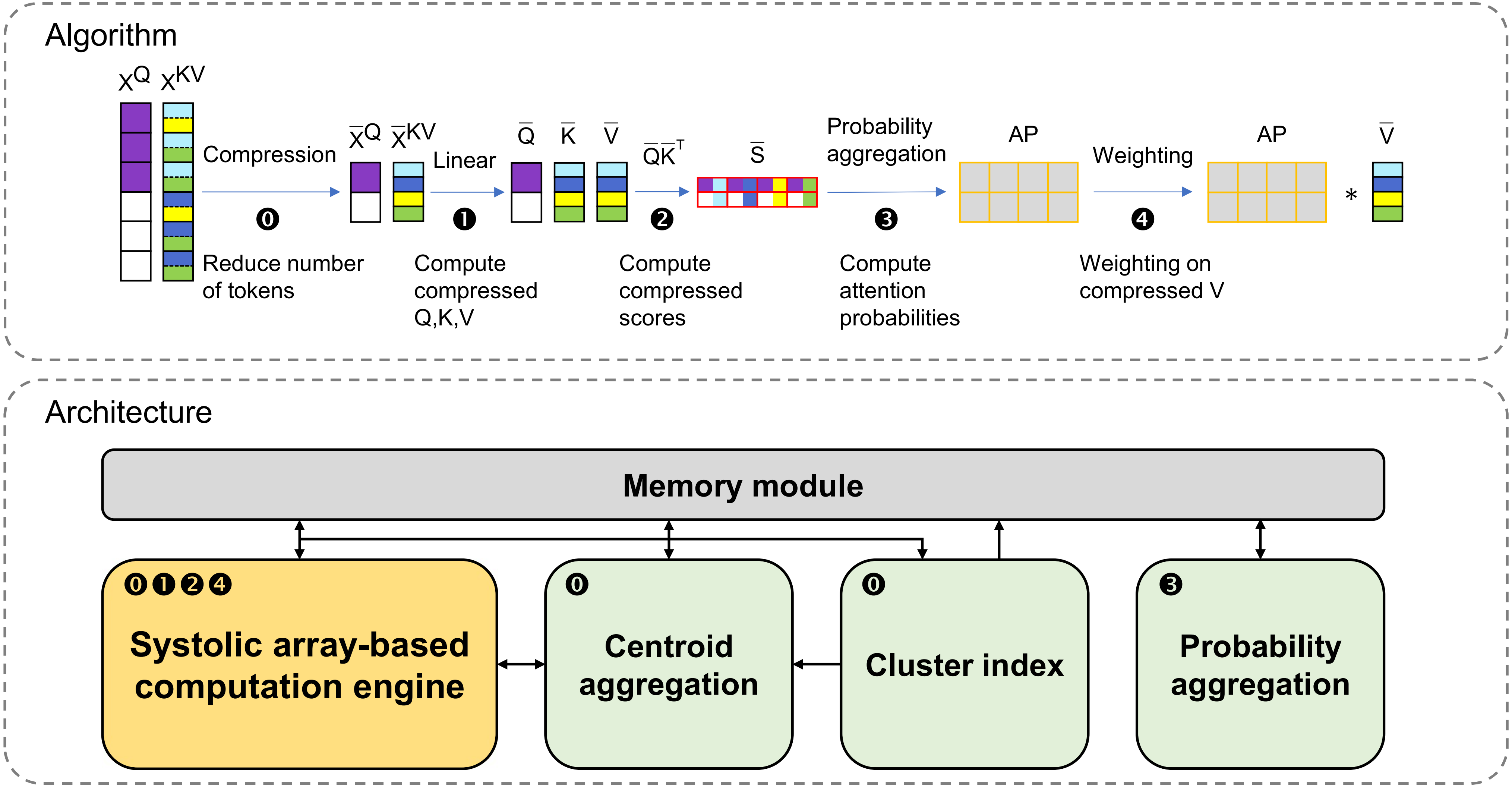


Overview: Compressed Token Attention Mechanism (CTA)



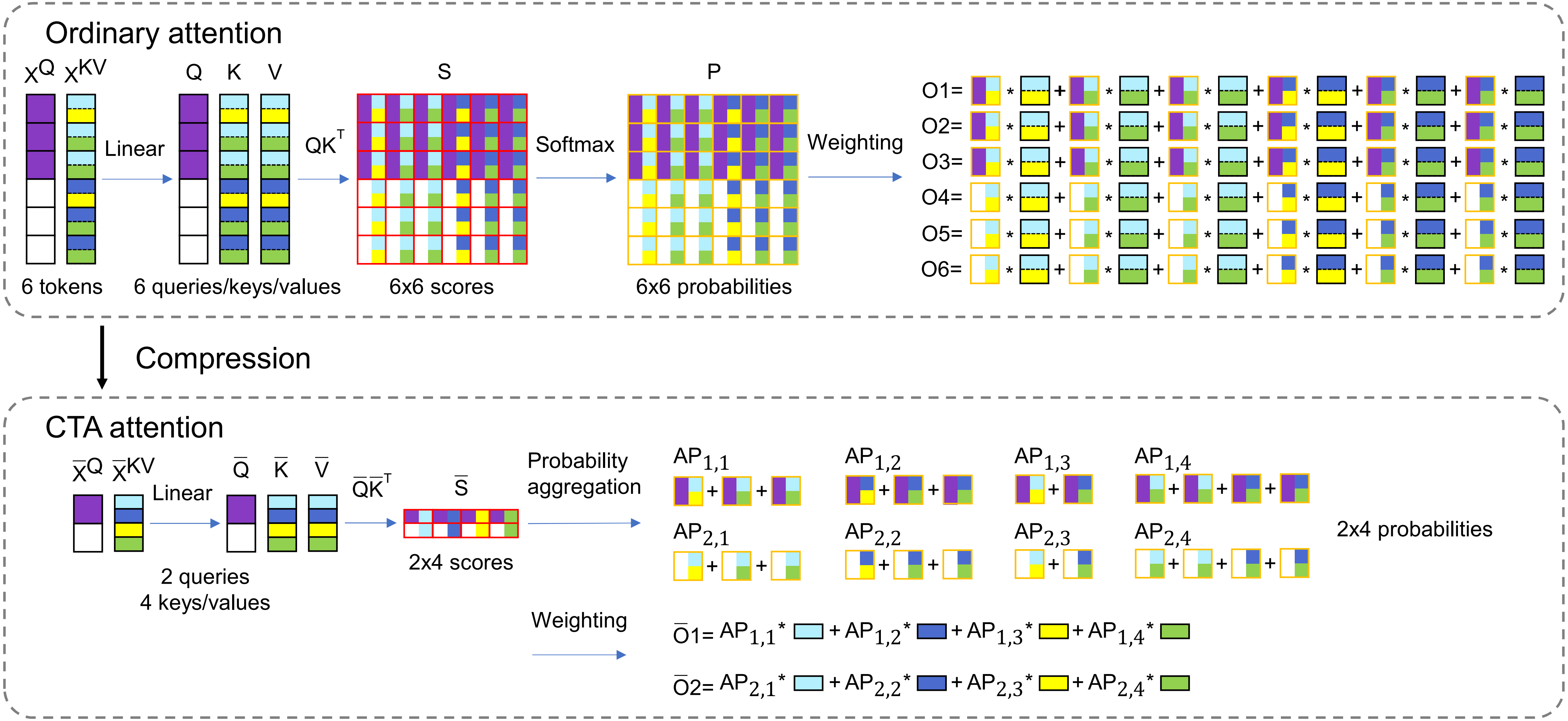


Overview: Compressed Token Attention Mechanism (CTA)



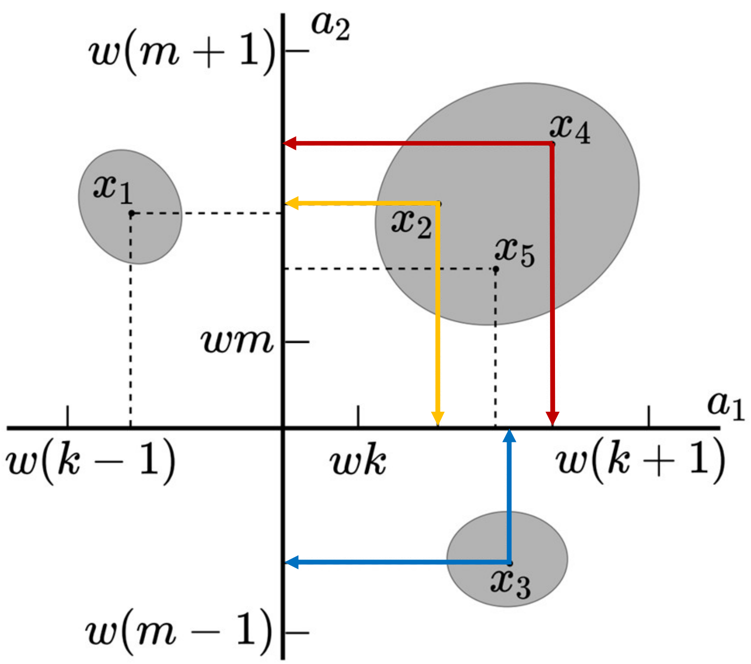
Algorithm overview

Compress tokens to remove repeating semantics, saving attention computation.



Efficiently remove semantic repetitions in tokens

Goal: group similar tokens and find appropriate representation for grouped tokens



a_1 a_2

$h(x_2)$ = (k , m)

$h(x_3)$ = (k , $m-1$)

$h(x_4)$ = (k , m)

Tokens (X) projects to which interval on direction axes (A)

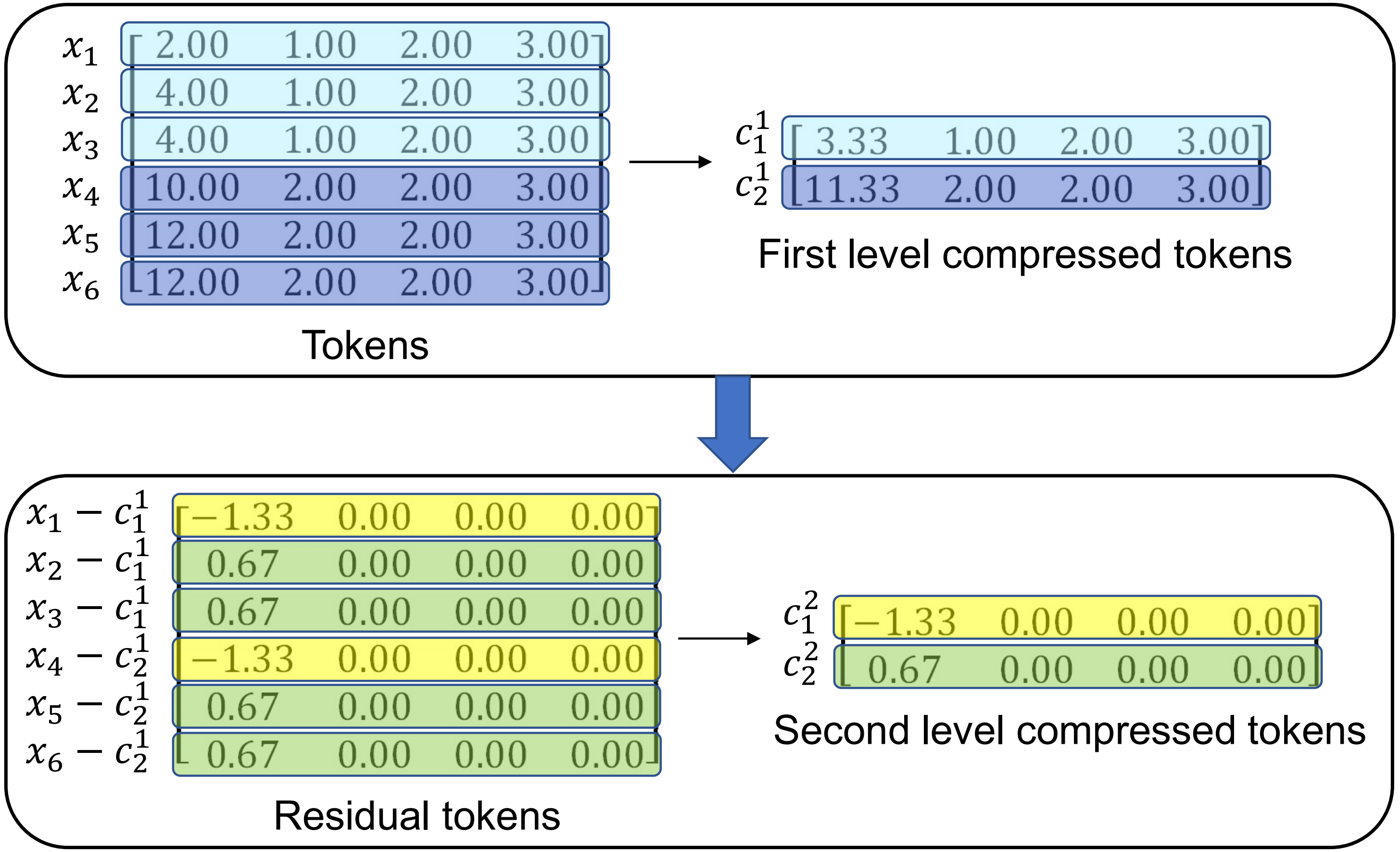
$$H = \lfloor (A \cdot X^T + B)/w \rfloor$$

- Group tokens with the same hash value into a cluster
- Compress tokens to their cluster centroids - average of tokens of their cluster

$$x_1, x_2, x_3, x_4, x_5 \xrightarrow{\text{compression}} x_1, \frac{x_2 + x_4 + x_5}{3}, x_3$$

Step ❶: Token compression

Efficiently extracting semantic features in tokens, yielding compressed tokens with little information loss compared to original tokens



First level compression: tokens are compressed

Second level compression: residual tokens are further compressed

Step ⓿: Token compression

Approximate original tokens with compressed tokens utilizing only first level compressed tokens or the sum of two levels' compressed tokens

$$
\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{array}
\begin{bmatrix}
2.00 & 1.00 & 2.00 & 3.00 \\
4.00 & 1.00 & 2.00 & 3.00 \\
4.00 & 1.00 & 2.00 & 3.00 \\
10.00 & 2.00 & 2.00 & 3.00 \\
12.00 & 2.00 & 2.00 & 3.00 \\
12.00 & 2.00 & 2.00 & 3.00
\end{bmatrix}
\longrightarrow
\begin{array}{c} c_1^1 \\ c_2^1 \end{array}
\begin{bmatrix}
3.33 & 1.00 & 2.00 & 3.00 \\
11.33 & 2.00 & 2.00 & 3.00
\end{bmatrix} \text{ First level}
\quad
\begin{array}{c} c_1^2 \\ c_2^2 \end{array}
\begin{bmatrix}
-1.33 & 0.00 & 0.00 & 0.00 \\
0.67 & 0.00 & 0.00 & 0.00
\end{bmatrix} \text{ Second level}
$$

6 tokens

4 compressed tokens

Compression

$$
\begin{array}{c} c_1^1 \\ c_1^1 \\ c_1^1 \\ c_2^1 \\ c_2^1 \\ c_2^1 \end{array}
\begin{bmatrix}
3.33 & 1.00 & 2.00 & 3.00 \\
3.33 & 1.00 & 2.00 & 3.00 \\
3.33 & 1.00 & 2.00 & 3.00 \\
11.33 & 2.00 & 2.00 & 3.00 \\
11.33 & 2.00 & 2.00 & 3.00 \\
11.33 & 2.00 & 2.00 & 3.00
\end{bmatrix}
\qquad
\begin{array}{c} c_1^1 + c_1^2 \\ c_1^1 + c_2^2 \\ c_1^1 + c_2^2 \\ c_2^1 + c_1^2 \\ c_2^1 + c_2^2 \\ c_2^1 + c_2^2 \end{array}
\begin{bmatrix}
2.00 & 1.00 & 2.00 & 3.00 \\
4.00 & 1.00 & 2.00 & 3.00 \\
4.00 & 1.00 & 2.00 & 3.00 \\
10.00 & 2.00 & 2.00 & 3.00 \\
12.00 & 2.00 & 2.00 & 3.00 \\
12.00 & 2.00 & 2.00 & 3.00
\end{bmatrix}
$$

Only one level

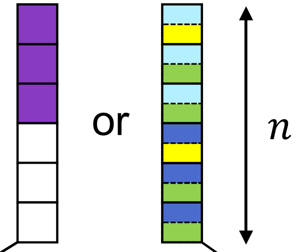
Two level

Approximating original tokens with compressed tokens

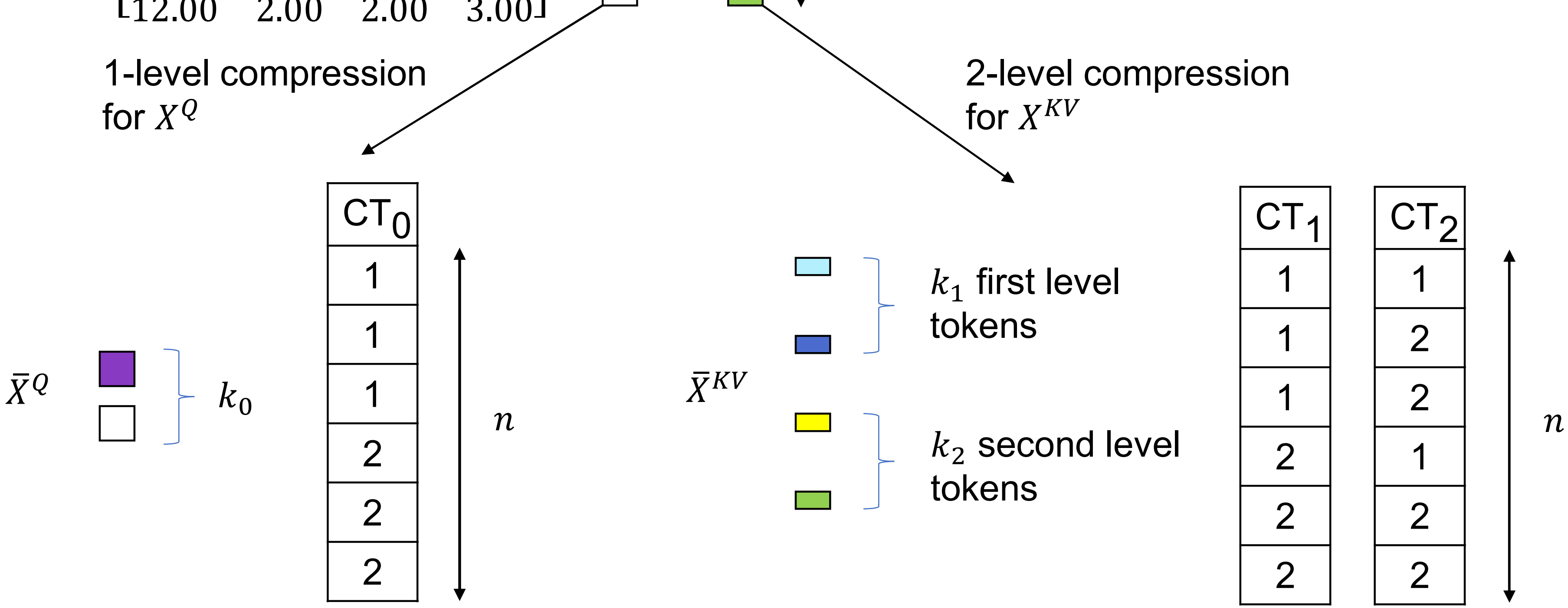
Step ❶: Token compression

In this example,

$$X^Q = X^{KV} = \begin{bmatrix} 2.00 & 1.00 & 2.00 & 3.00 \\ 4.00 & 1.00 & 2.00 & 3.00 \\ 4.00 & 1.00 & 2.00 & 3.00 \\ 10.00 & 2.00 & 2.00 & 3.00 \\ 12.00 & 2.00 & 2.00 & 3.00 \\ 12.00 & 2.00 & 2.00 & 3.00 \end{bmatrix} \approx$$

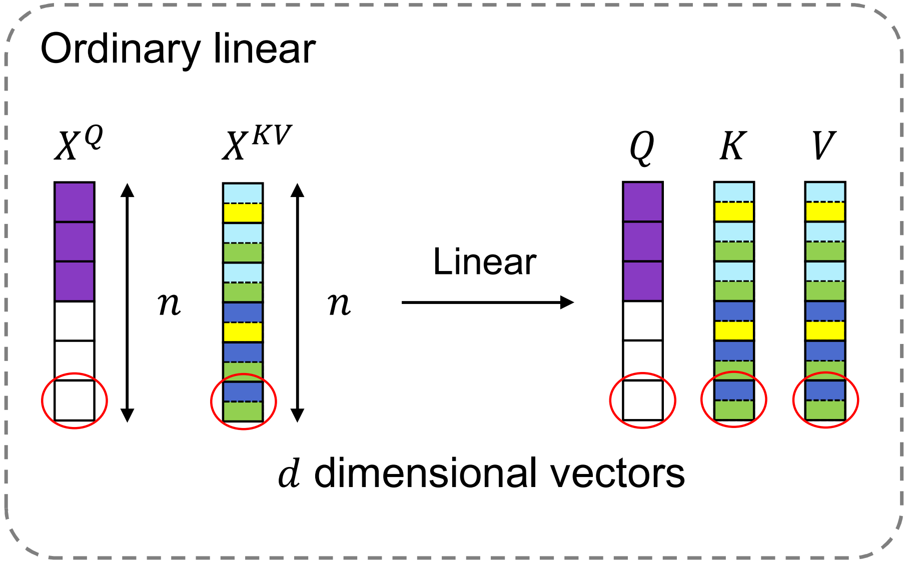


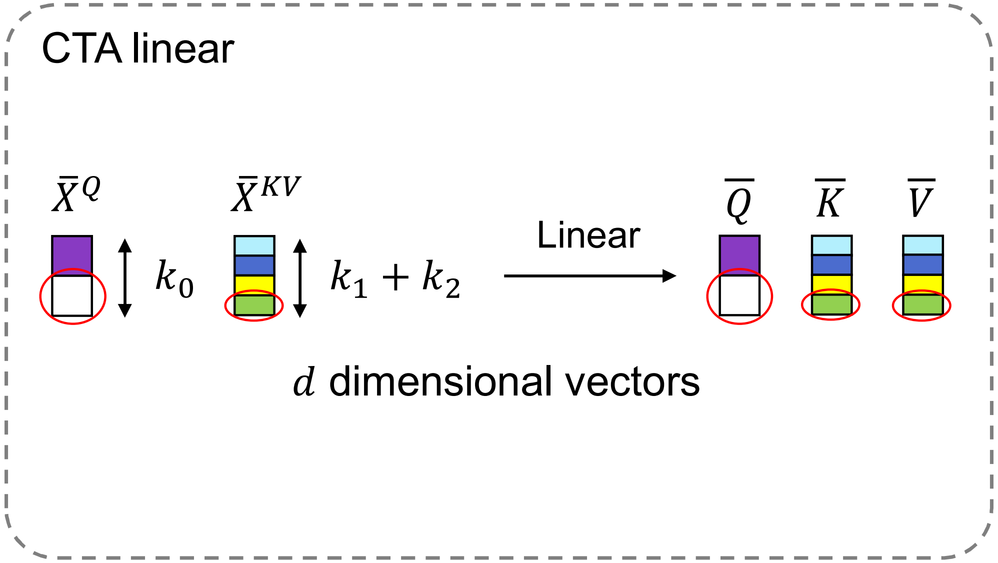
METHOD
1-level compression for query tokens
2-level compression for key-value tokens



Step ❶: Linear transformations on compressed tokens

Linear transformations maps compressed tokens to compressed queries, keys and values





$$Q = X^Q \cdot W^Q, K = X^{KV} \cdot W^K, V = X^{KV} \cdot W^V$$

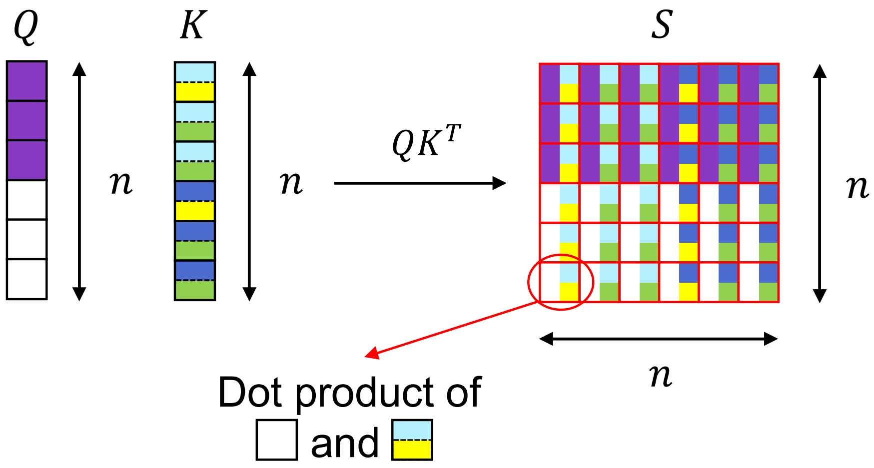
$$\overline{Q} = \bar{X}^Q \cdot W^Q, \overline{K} = \bar{X}^{KV} \cdot W^K, \overline{V} = \bar{X}^{KV} \cdot W^V$$

Computation for queries is reduced from nd^2 to $k_0 d^2$
Computation for keys/values is reduced from nd^2 to $(k_1 + k_2)d^2$

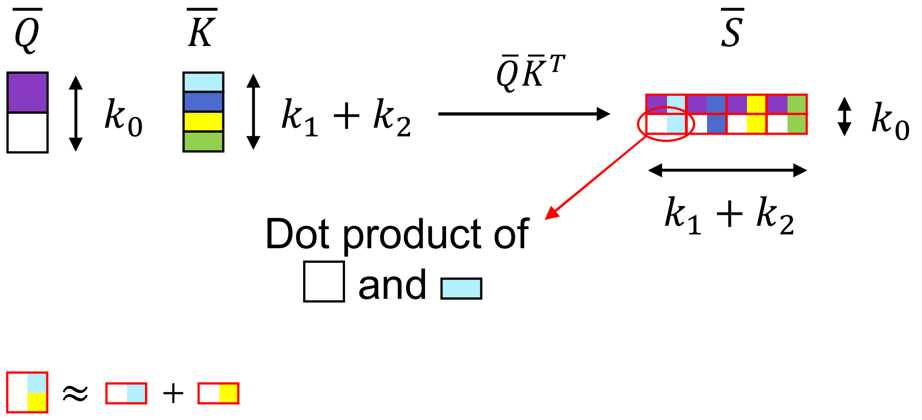
Step ❷: Similarities between compressed queries and keys

Compressed queries and keys generates compressed similarities

Ordinary similarities



CTA similarities



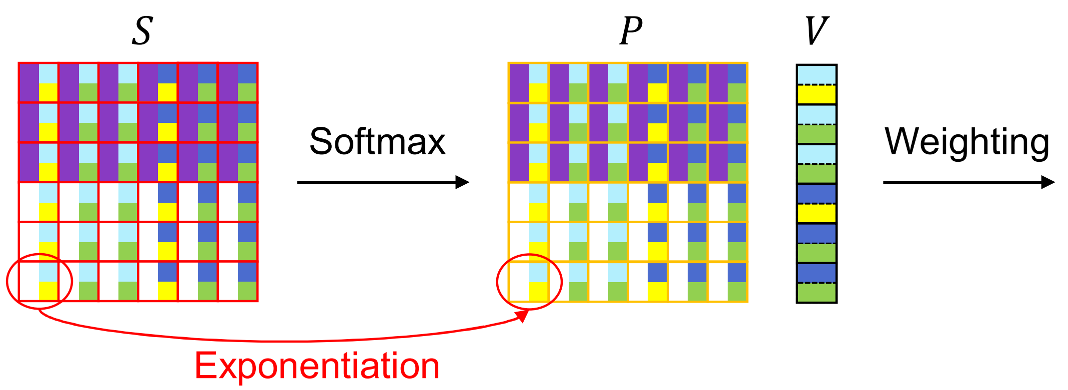
$$S = (Q \cdot K^T)/\sqrt{d}$$

$$\bar{S} = (\bar{Q} \cdot \bar{K}^T)/\sqrt{d}$$

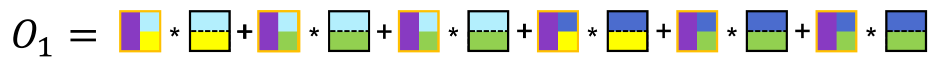
Computation is reduced from $n^2 d$ to $k_0(k_1 + k_2)d$

Step ❸: Attention probability aggregation

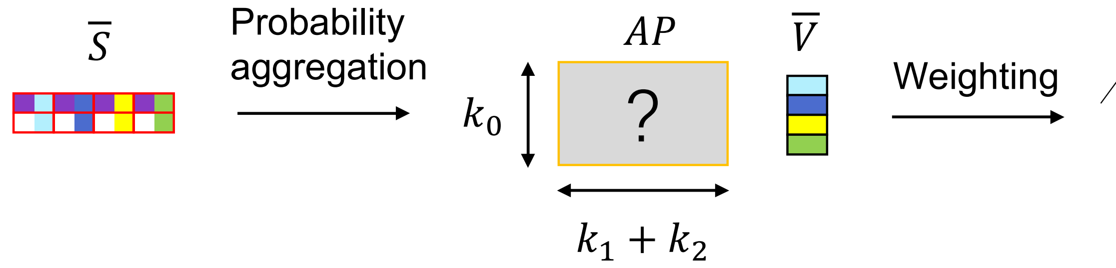
Ordinary Softmax and weighting



We omit softmax division in step ❸, because it can be equivalently put off to the end of step ❹



CTA attention probability aggregation and weighting

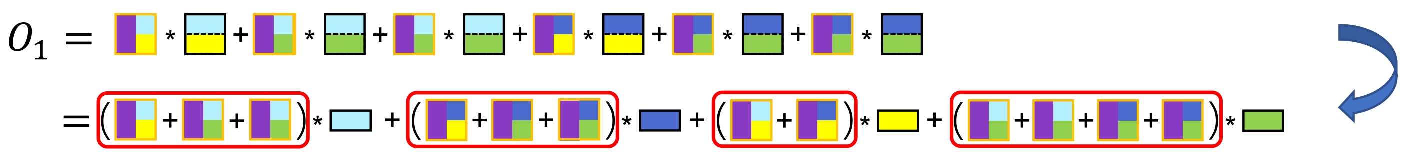


Key idea:
Find a way of weighting compressed values that can approximate ordinary output

Step ❸: Attention probability aggregation

Reform output into the form of weighted sum of compressed values

Observation

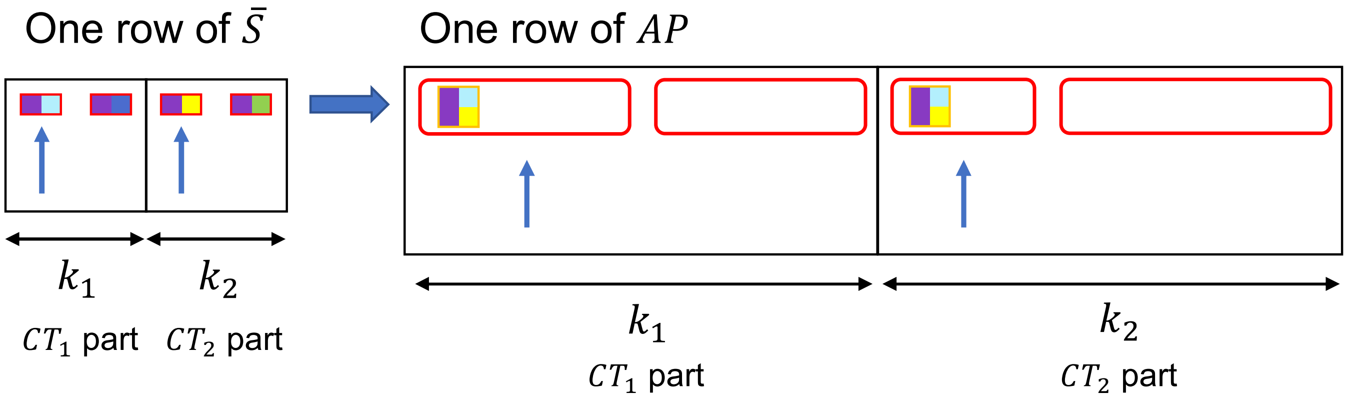


- Split value vectors into sum of compressed value vectors
- Merge coefficients

Aggregation algorithm

CT_1	CT_2
1	1
1	2
1	2
2	1
2	2
2	2

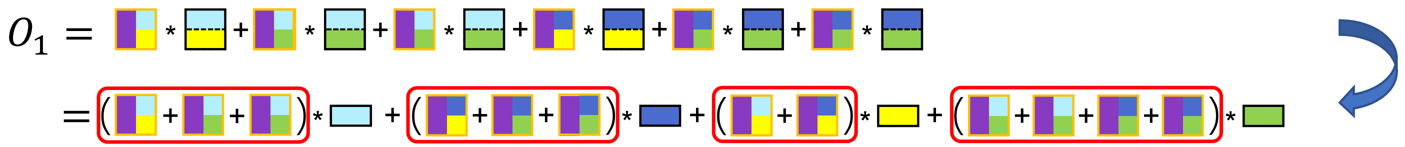
Each time select two scores from $\bar{S}$ according to CT_1 and CT_2, take the exponent of their sum and add to the same positions in AP



Step ❸: Attention probability aggregation

Reform output into the form of weighted sum of compressed values

Observation

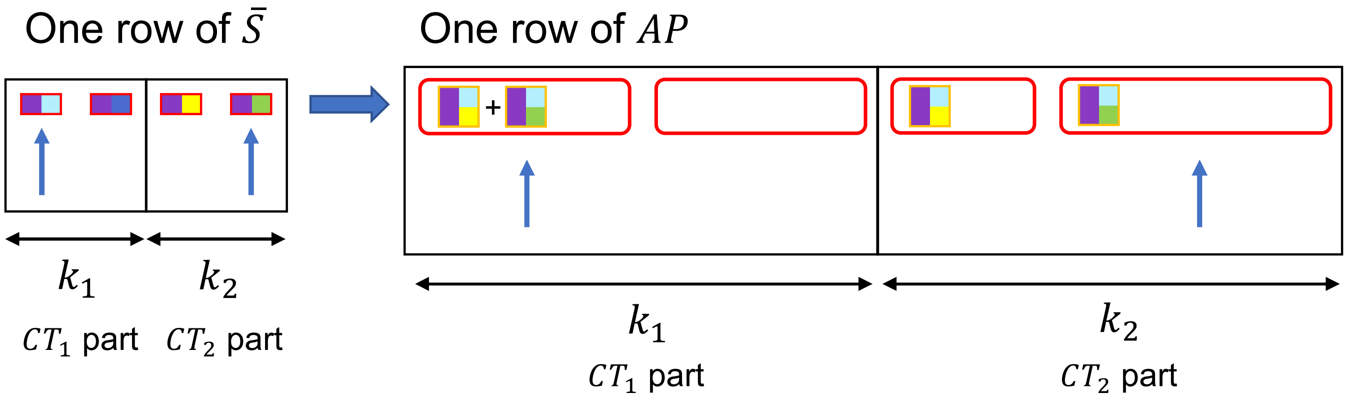


- Split value vectors into sum of compressed value vectors
- Merge coefficients

Aggregation algorithm

CT_1	CT_2
1	1
1	2
1	2
2	1
2	2
2	2

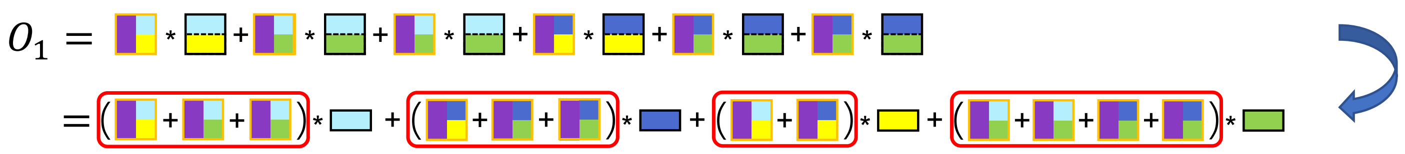
Each time select two scores from $\bar{S}$ according to CT_1 and CT_2, take the exponent of their sum and add to the same positions in AP



Step ❸: Attention probability aggregation

Reform output into the form of weighted sum of compressed values

Observation

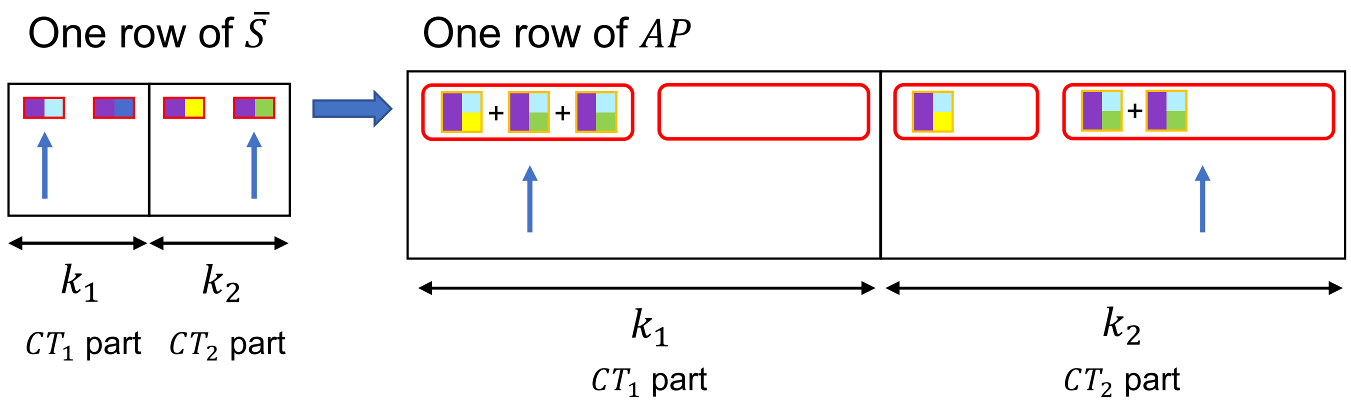


- Split value vectors into sum of compressed value vectors
- Merge coefficients

Aggregation algorithm

CT_1	CT_2
1	1
1	2
1	2
2	1
2	2
2	2

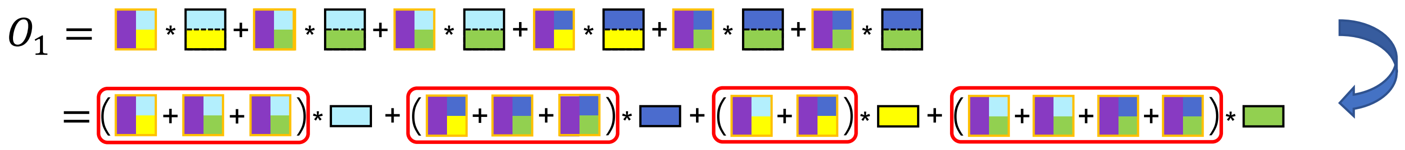
Each time select two scores from $\bar{S}$ according to CT_1 and CT_2, take the exponent of their sum and add to the same positions in AP



Step ❸: Attention probability aggregation

Reform output into the form of weighted sum of compressed values

Observation

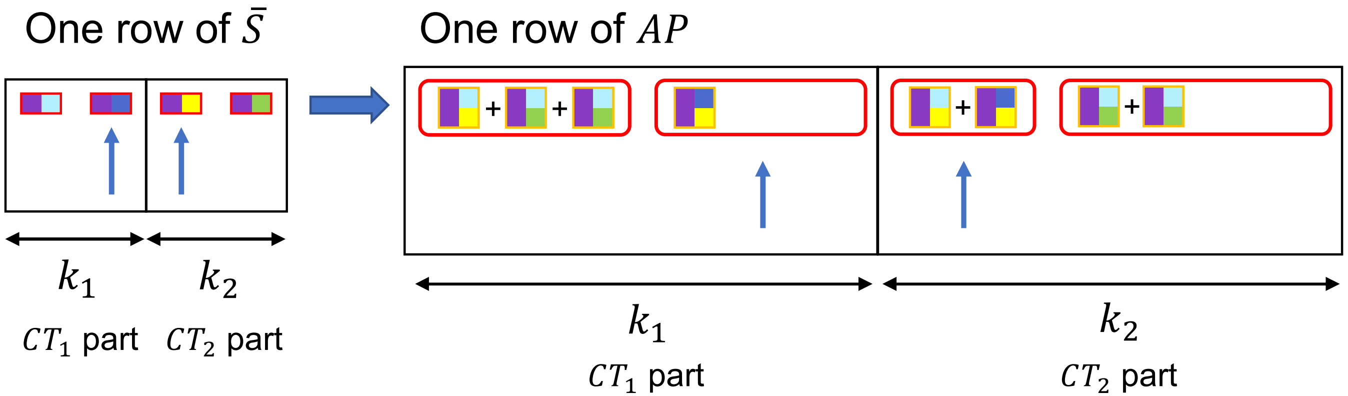


- Split value vectors into sum of compressed value vectors
- Merge coefficients

Aggregation algorithm

CT_1	CT_2
1	1
1	2
1	2
2	1
2	2
2	2

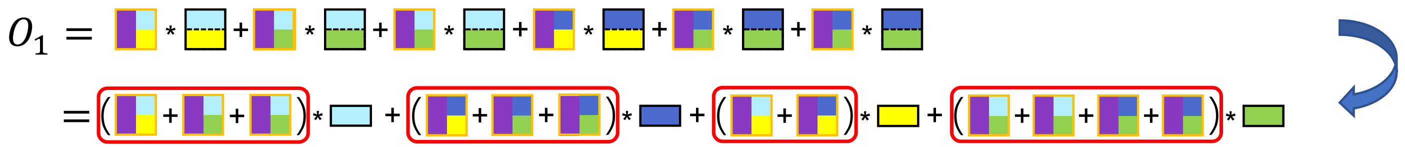
Each time select two scores from $\bar{S}$ according to CT_1 and CT_2, take the exponent of their sum and add to the same positions in AP



Step ❸: Attention probability aggregation

Reform output into the form of weighted sum of compressed values

Observation

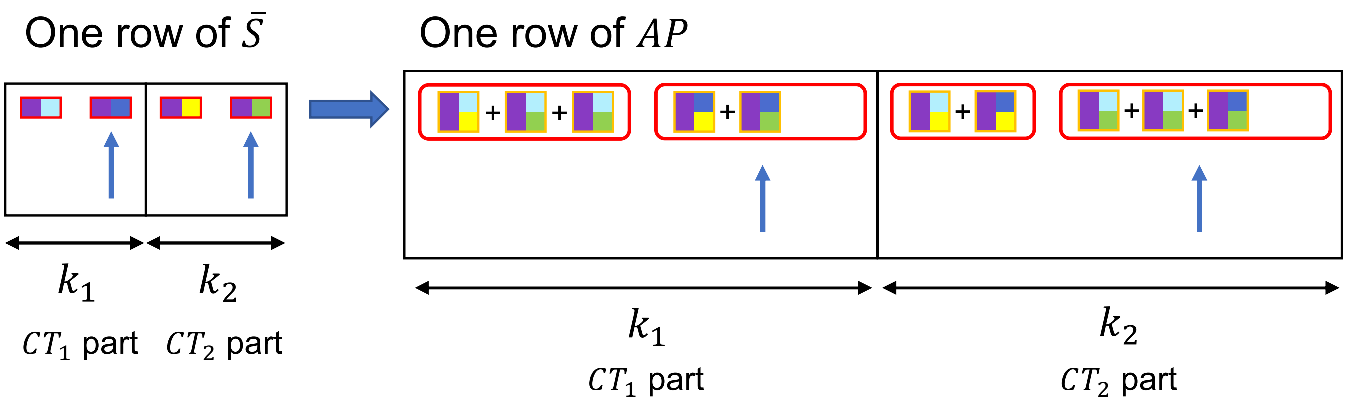


- Split value vectors into sum of compressed value vectors
- Merge coefficients

Aggregation algorithm

CT_1	CT_2
1	1
1	2
1	2
2	1
2	2
2	2

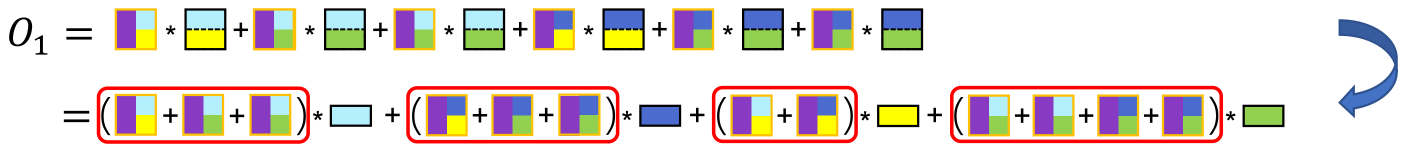
Each time select two scores from $\bar{S}$ according to CT_1 and CT_2, take the exponent of their sum and add to the same positions in AP



Step ❸: Attention probability aggregation

Reform output into the form of weighted sum of compressed values

Observation



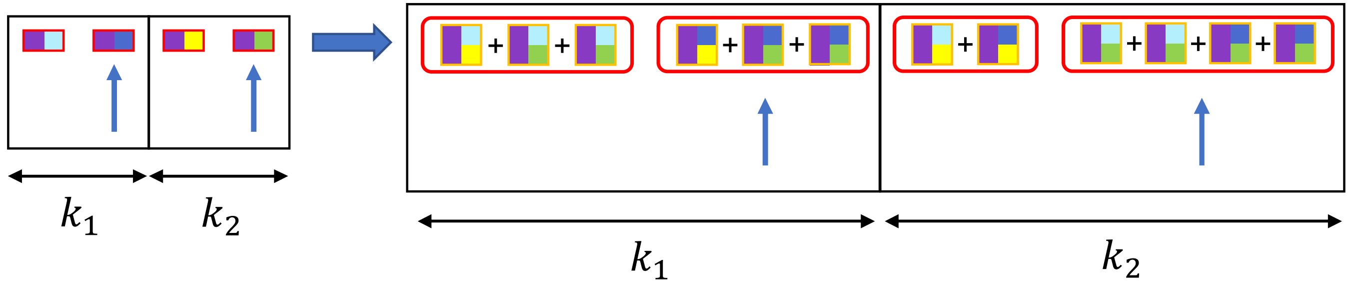
- Split value vectors into sum of compressed value vectors
- Merge coefficients

Aggregation algorithm

CT_1	CT_2
1	1
1	2
1	2
2	1
2	2
2	2

Each time select two scores from $\bar{S}$ according to CT_1 and CT_2, take the exponent of their sum and add to the same positions in AP

One row of $\bar{S}$ One row of AP

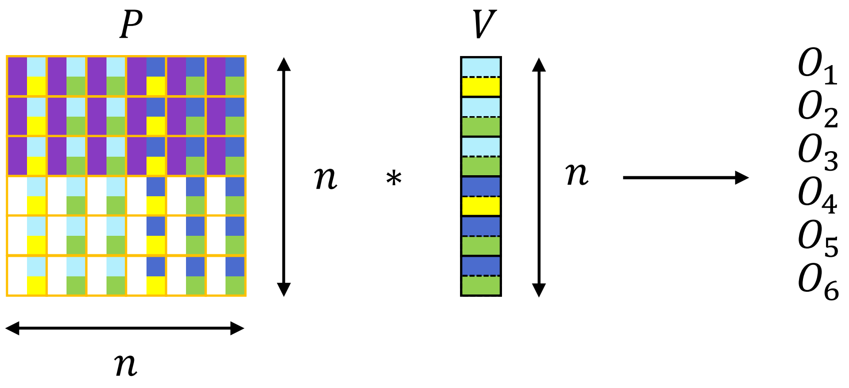


n exponent operations per row, the total number is reduced from n^2 to $k_0 n$

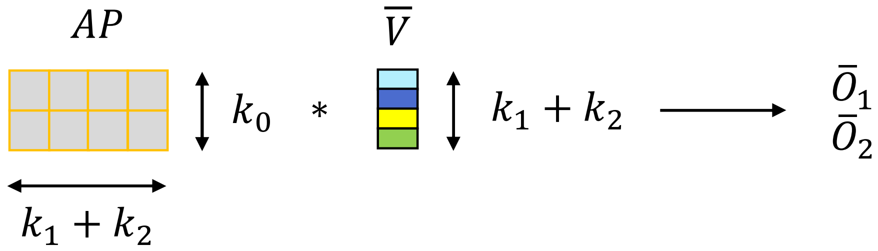
Step ❹: Output weighting

Given the aggregated attention probability matrix AP, output can be calculated as $AP \cdot \overline{V}$

Ordinary weighting



CTA weighting



$$O = P \cdot V$$

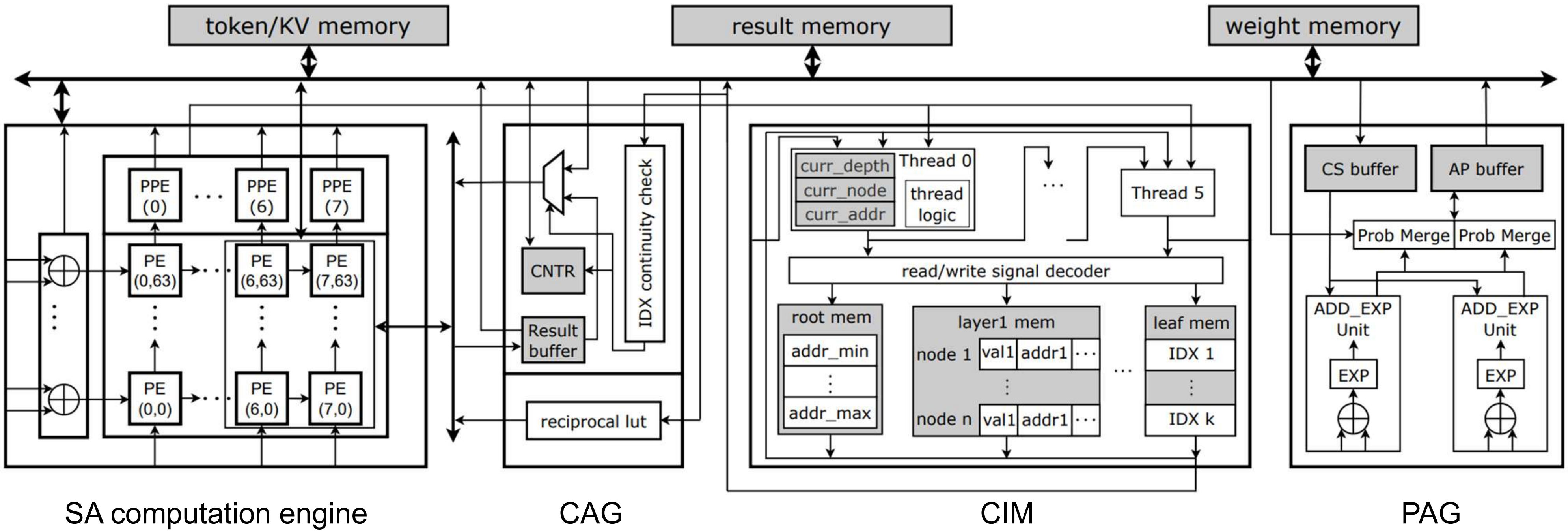
$$\overline{O} = AP \cdot \overline{V}$$

Computation is reduced from $n^2 d$ to $k_0(k_1 + k_2)d$

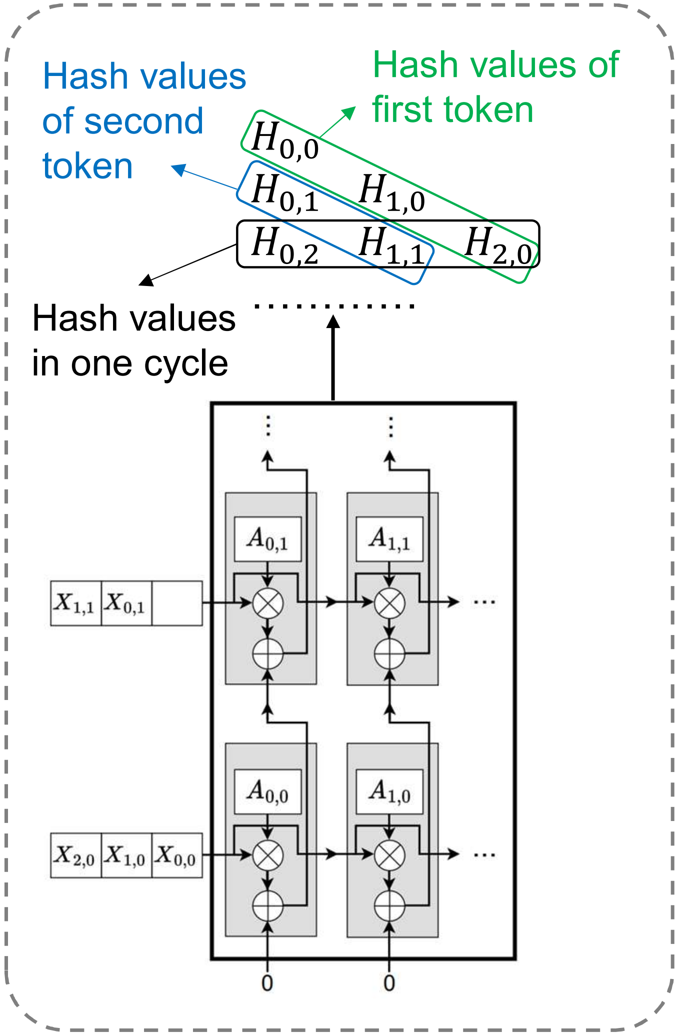
Architecture overview: Specialized accelerator for CTA

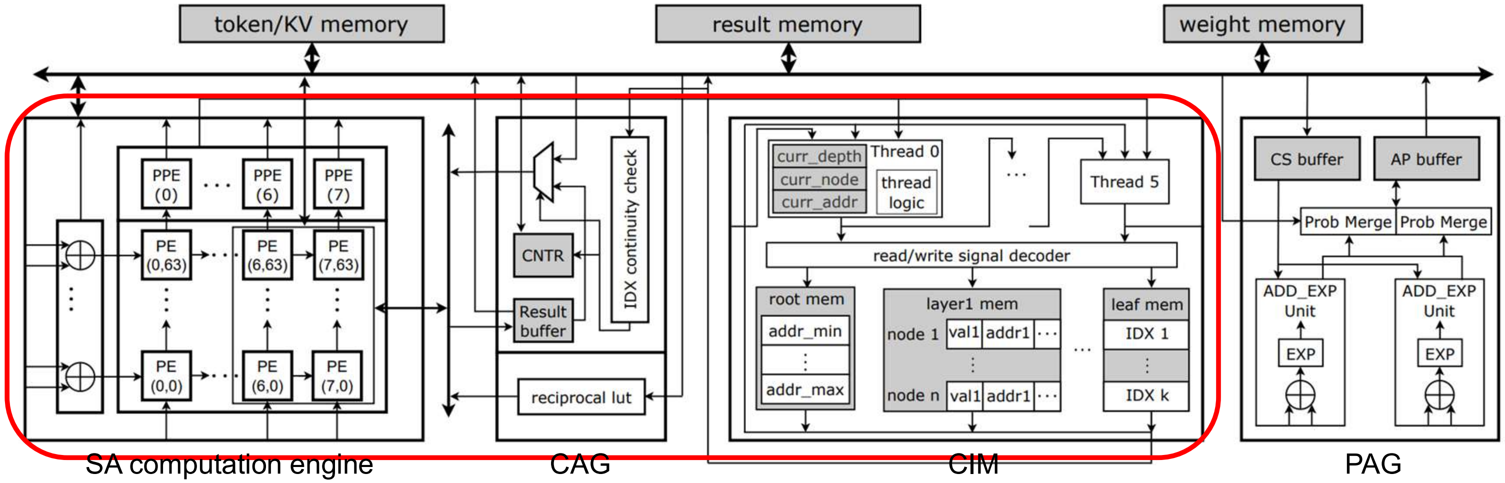
Four modules:

- Systolic array computation engine: accommodate major matrix multiplication operations
- CAG: averaging tokens assigned to the same cluster to compute compressed tokens
- CIM: separate tokens into clusters by assigning cluster index according to their hash values
- PAG: calculate aggregated attention probabilities from compressed attention scores



Dedicated accelerator for step ⓿

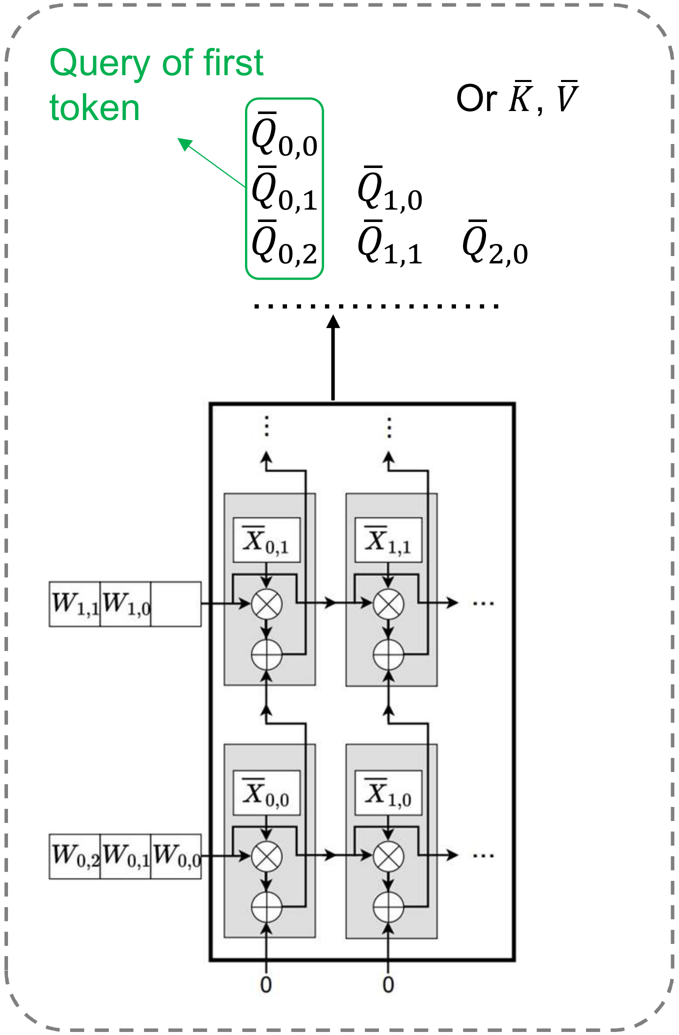


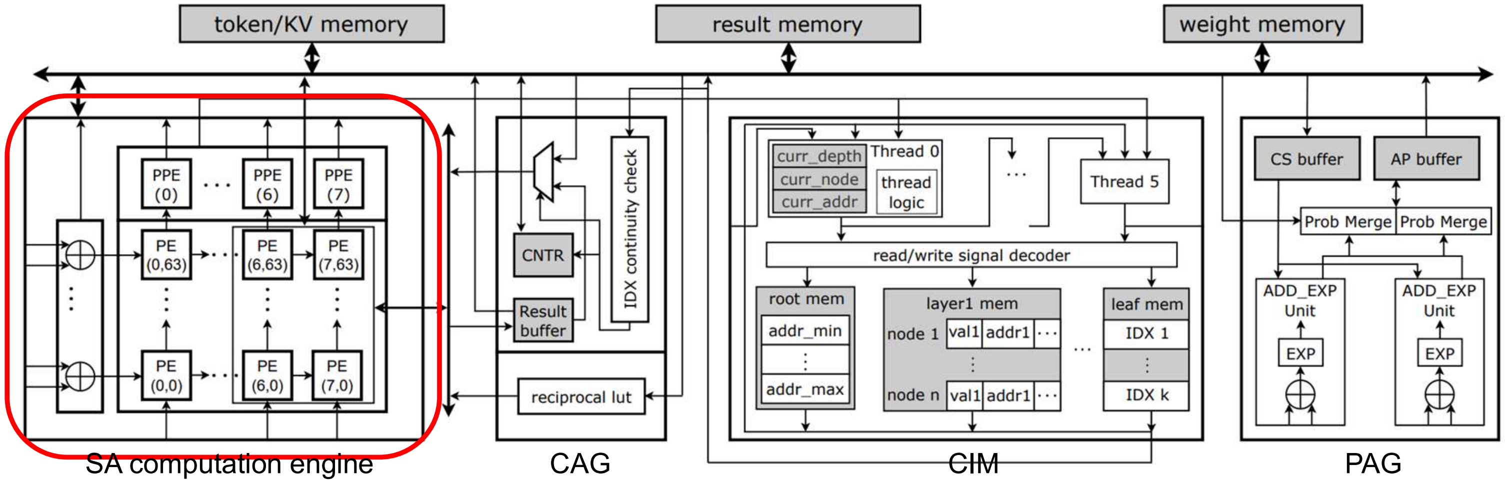


$$H = \lfloor (A \cdot X^T + B)/w \rfloor$$

- Each column of PEs accumulate the inner products between a direction vector and tokens.
- Each PPE adds bias to the result and divide it by w, then take floor.
- CIM separates tokens according to their hash values and CAG computes the average.

Dedicated accelerator for step ❶

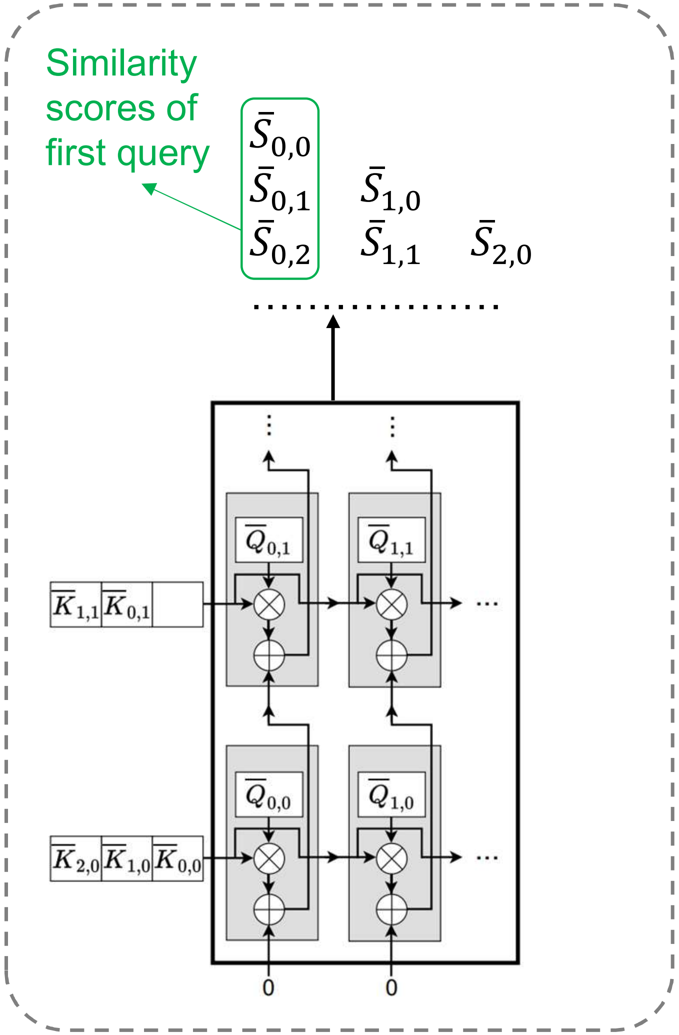


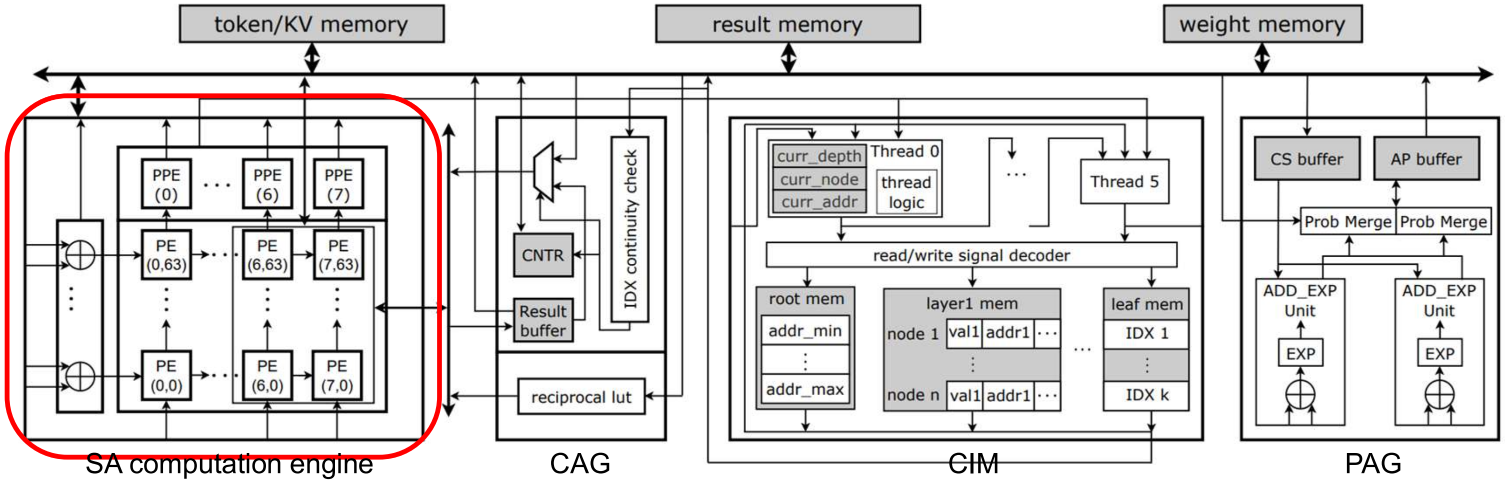


$$\bar{Q} = \bar{X}^{Q} \cdot W^{Q}, \bar{K} = \bar{X}^{KV} \cdot W^{K}, \bar{V} = \bar{X}^{KV} \cdot W^{V}$$

- Each column of PEs accumulate the inner products between a compressed token and weight matrix columns to compute the transformed query/key/value for this compressed token.

Dedicated accelerator for step ❷





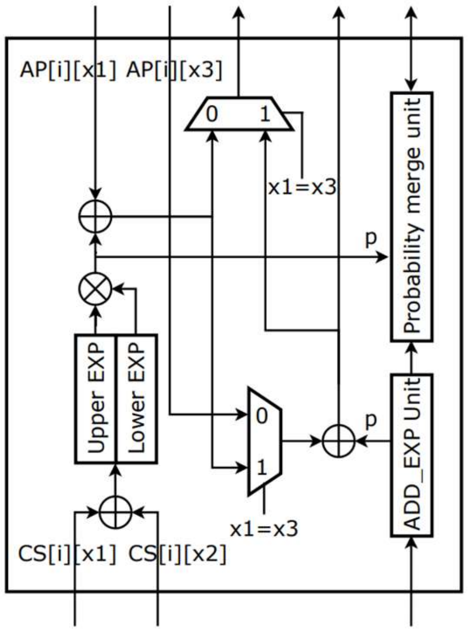
$$\bar{S} = (\bar{Q} \cdot \bar{K}^T)/\sqrt{d}$$

- Each column of PEs accumulate the inner products between a compressed query and compressed keys to compute the similarity scores of this query.
- Each PPE counts the largest first level scores (first k_1 scores), then subtract the maximum from the following scores.

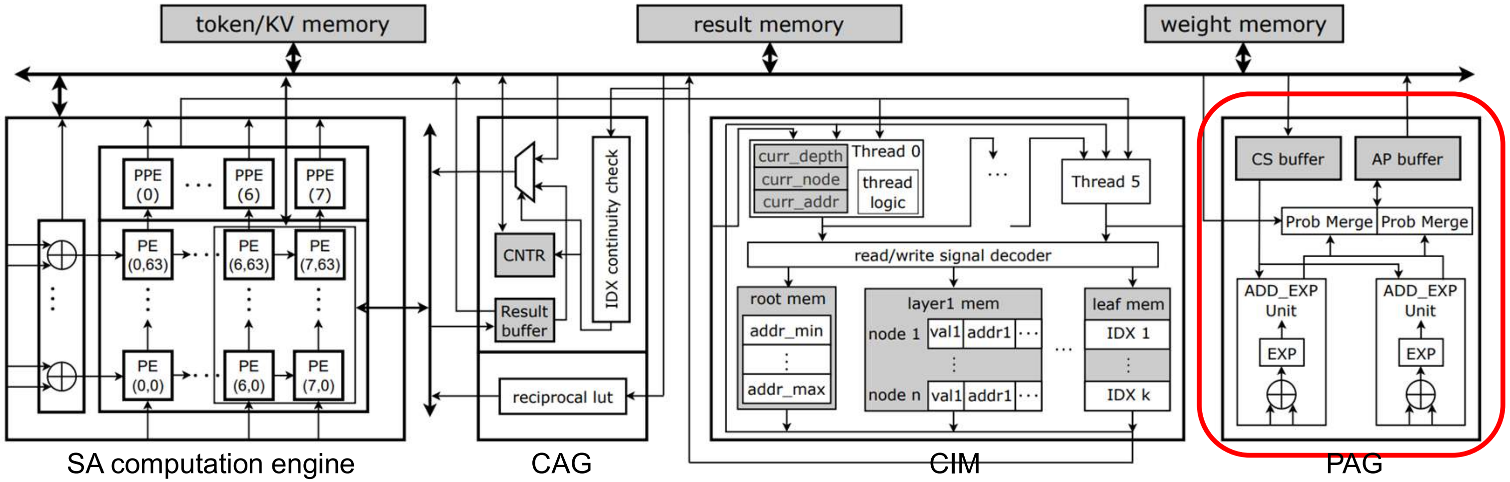
Dedicated accelerator for step ❸

One tile of PAG

Aggregated probabilities



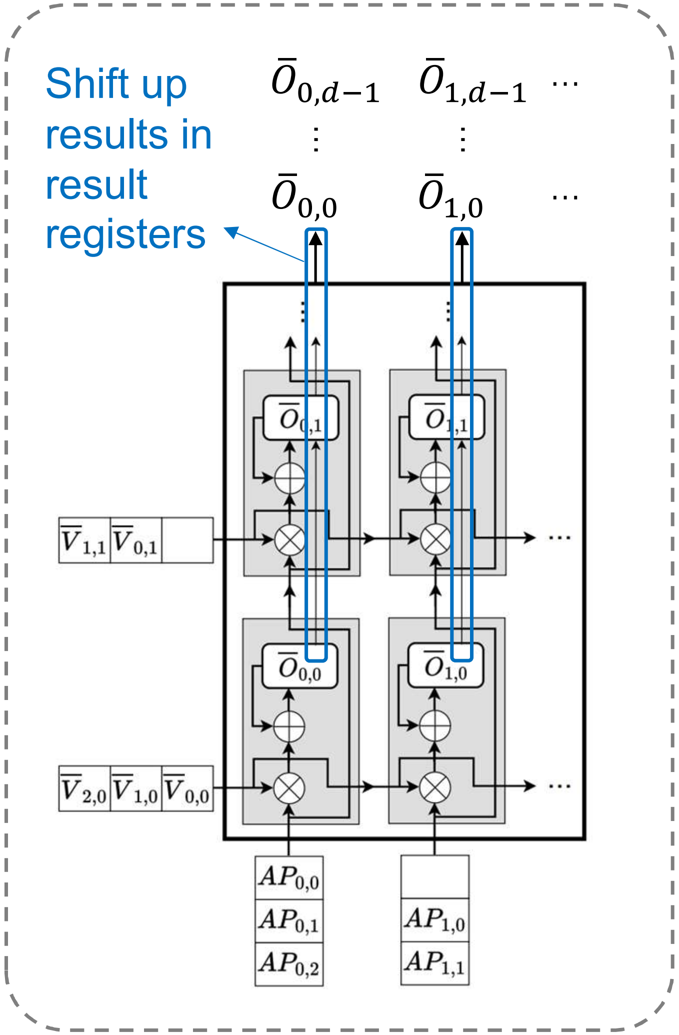
Compressed similarities

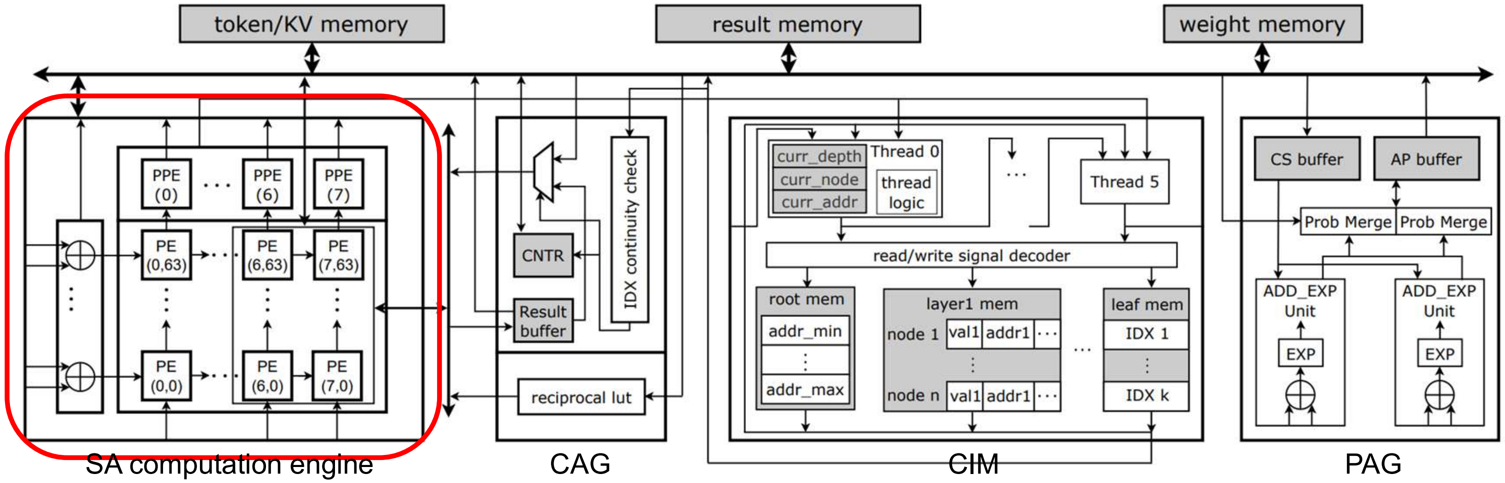


$$AP = PAG(\bar{S}, CT_1, CT_2)$$

- PAG reads pairs of compressed similarity scores indexed by CT_1 and CT_2, sum the scores and take the exponent.
- PAG accumulates the result to the same positions of aggregated probabilities indexed by CT_1 and CT_2.

Dedicated accelerator for step ❹

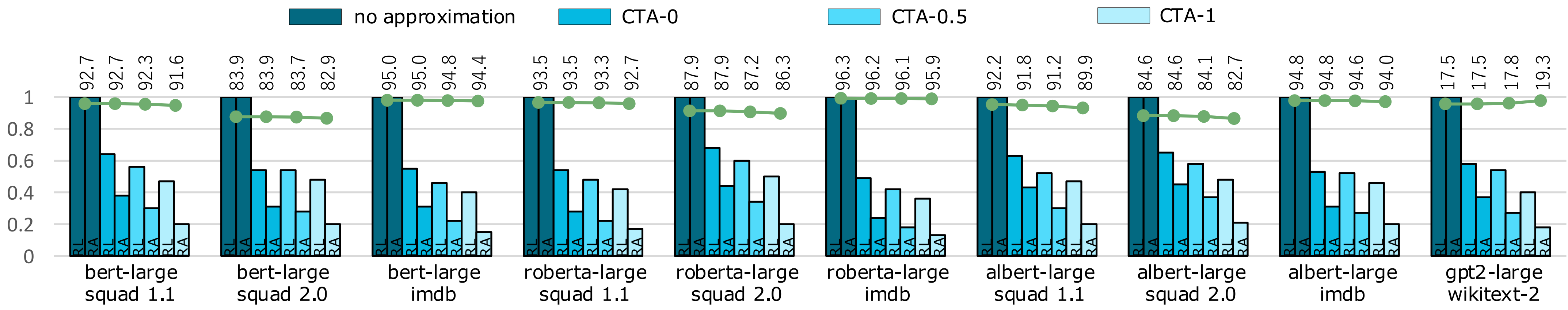




$$\bar{O} = AP \cdot \bar{V}$$

- Each column of PEs accumulate an output vector in their result registers.
- Each PPE sums the aggregated probabilities, divides the results by half of the sum when shifting up the results.

Evaluation: accuracy and computation reduction

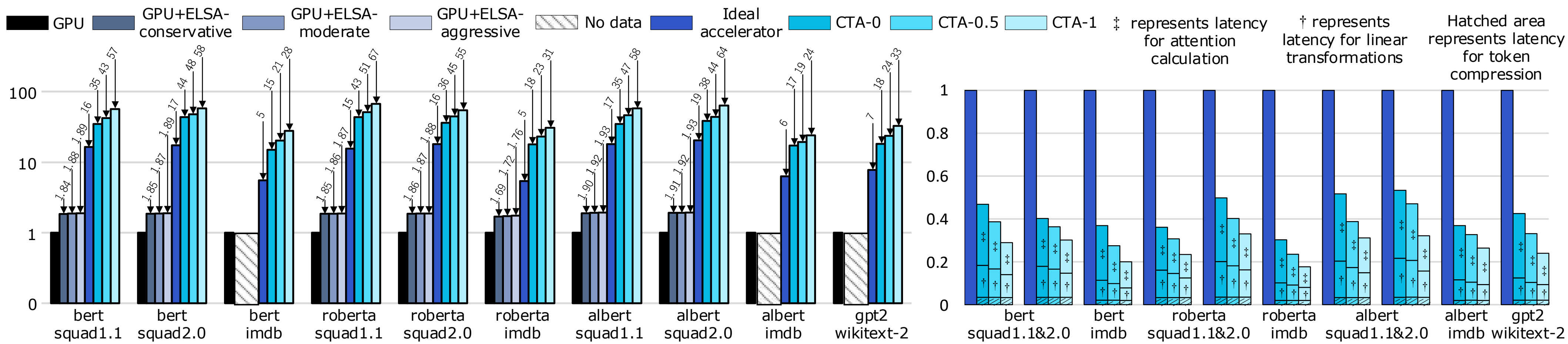


Green line: accuracy
RL, RA: (computation after compression) / (computation before compression)
RL is for linear transformations (step1), RA is for the rest steps in attention (step2-4)

Average results

CTA-0:	virtually no accuracy loss	&	RL=58.3%, RA=35.2%
CTA-0.5:	<0.5% accuracy loss	&	RL=52.2%, RA=27.5%
CTA-1:	<1% accuracy loss	&	RL=44.4%, RA=18.4%

Evaluation: throughput and latency



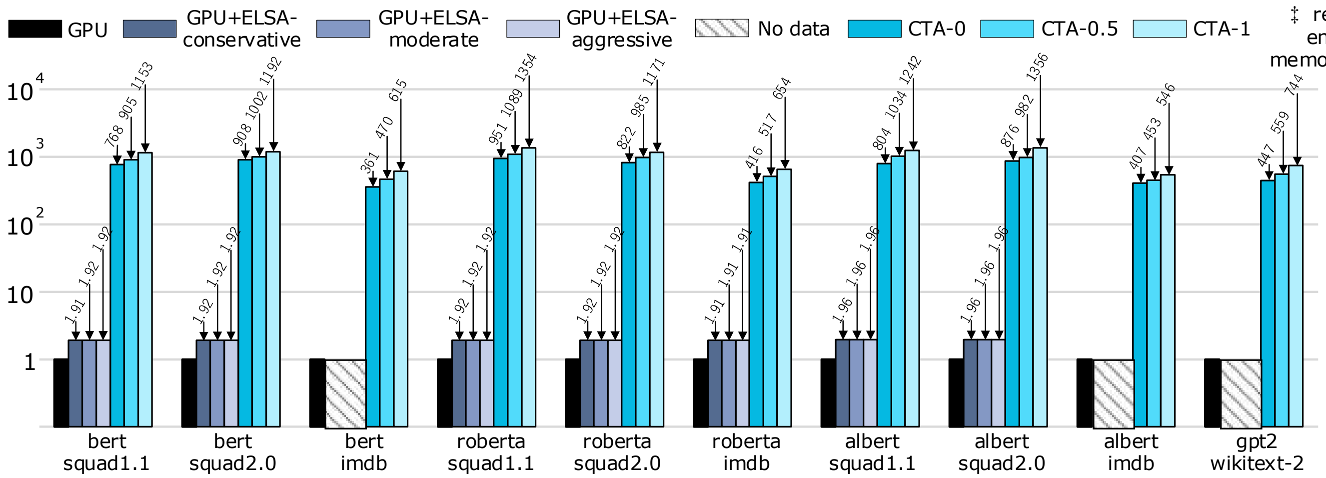
Average results

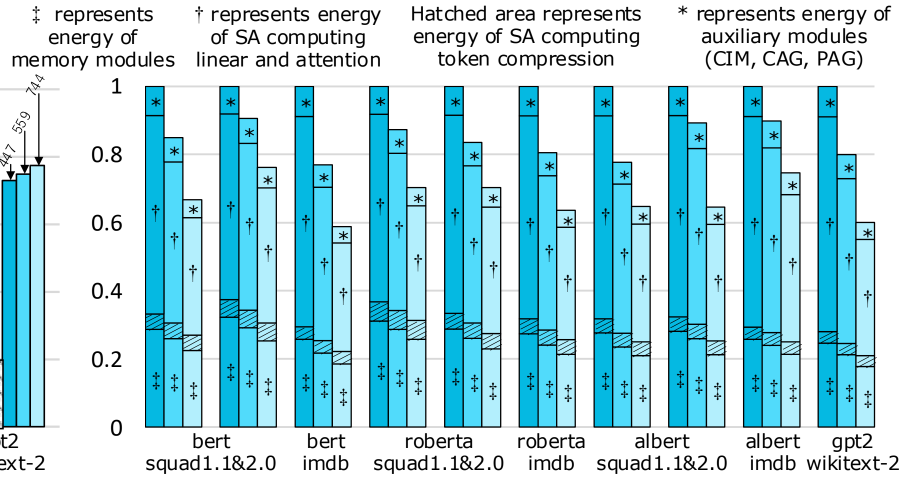
GPU: V100-SXM2 32GB

ELSA: previous state-of-the-art attention accelerator

speedup over GPU		speedup over ELSA-aggressive+GPU		latency breakdown	
CTA-0:	27.7×	CTA-0:	18.3×	attention computation:	59%
CTA-0.5:	33.8×	CTA-0.5:	22.1×	linear computation:	34%
CTA-1:	44.2×	CTA-1:	28.7×	token compression:	7%

Evaluation: energy efficiency and breakdown





Average results

GPU: V100-SXM2 32GB

ELSA: previous state-of-the-art attention accelerator

efficiency over GPU

CTA-0: 634×
CTA-0.5: 756×
CTA-1: 950×

efficiency over ELSA-aggressive+GPU

CTA-0: 399×
CTA-0.5: 471×
CTA-1: 587×

energy breakdown

Memory modules: 29%
SA module: 62%
Others: 9%

Energy spent on hashing for token compression: 5%

Conclusion

- CTA efficiently removes token semantic feature repetition
  - Token compression and attention operations on compressed tokens
- CTA keeps matrix multiplication formulation of major computation stage
  - Preserve inherent parallelism of attention mechanism
  - Support acceleration with more general architecture

Thank you!

- CTA architecture
  - Reconfigurable systolic array-based computation engine
  - Light-weight auxiliary modules