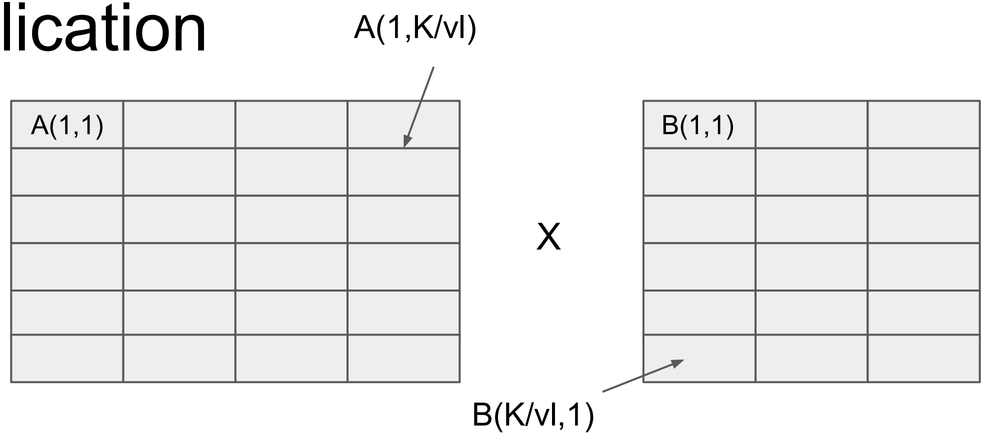
Week 2

Matrix Multiplication



Matrix A (M * K)

Matrix B (K * N)

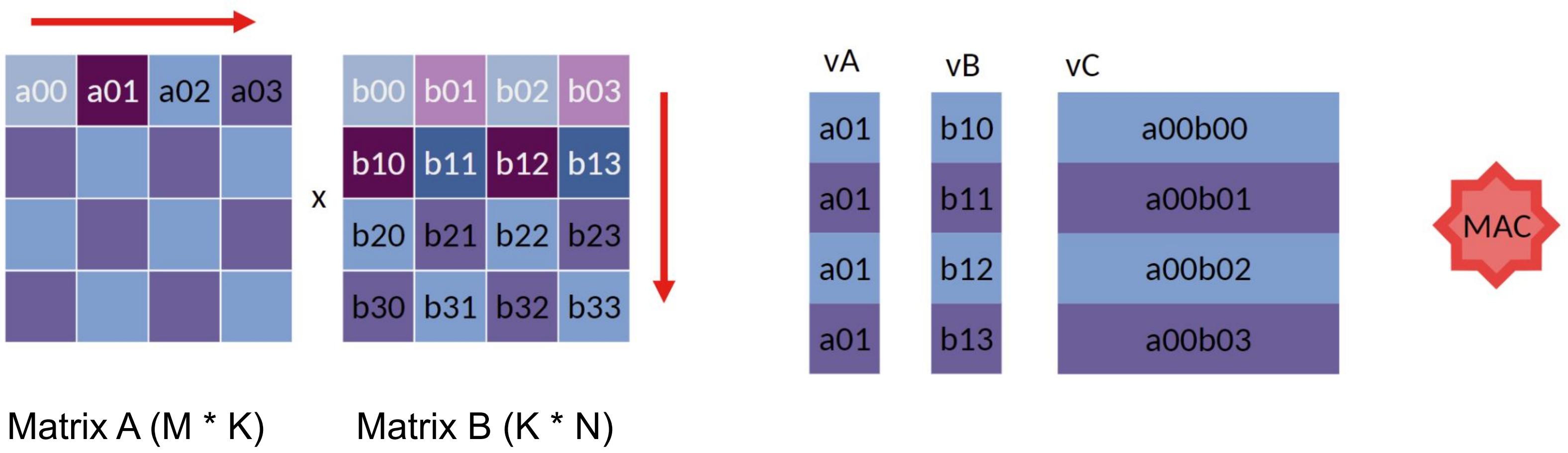
A x B = C

A(1,1), B(1,1) are vectors with length vl

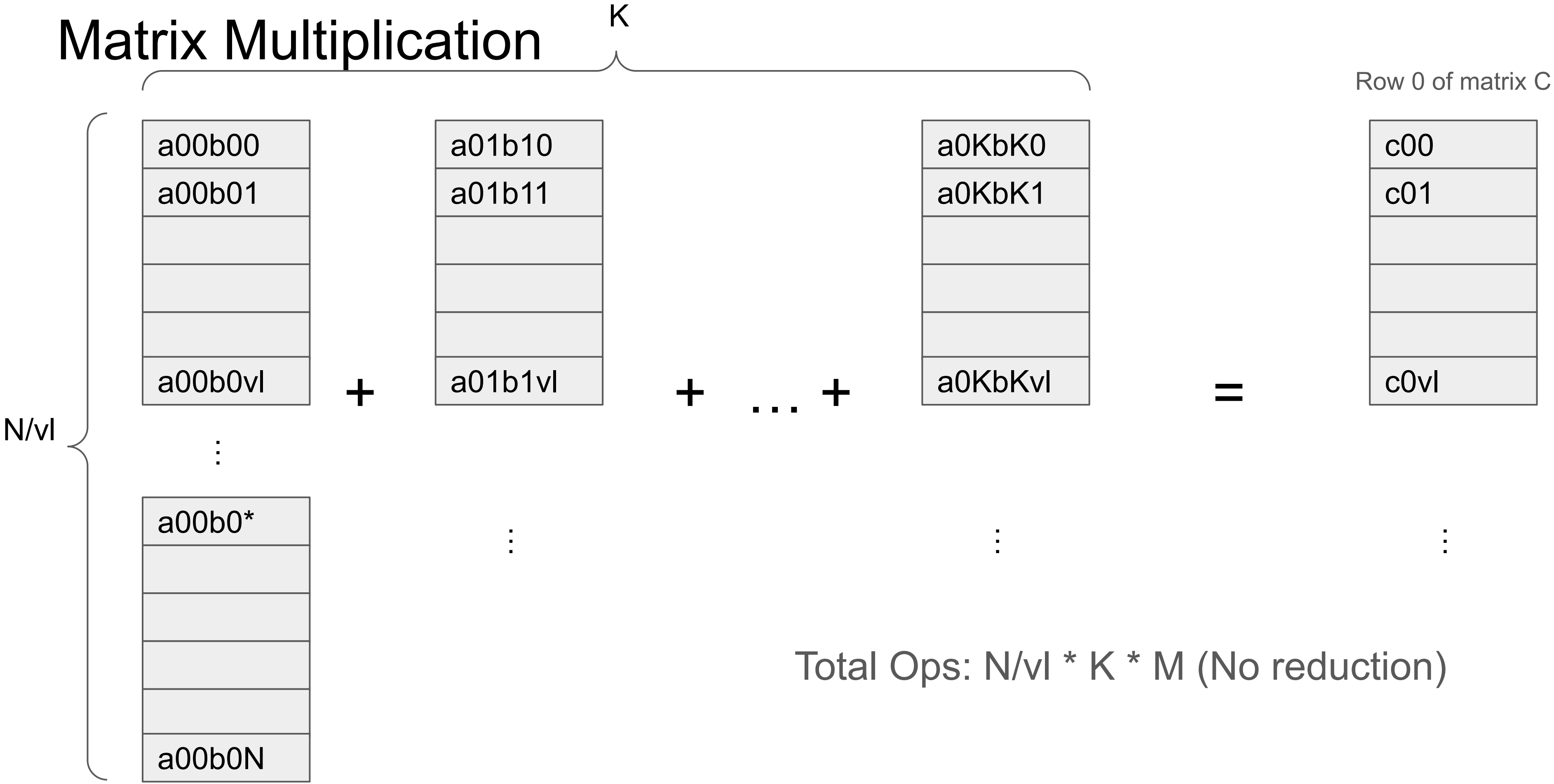
Element C(1,1) = reduction sum (A(1,1) * B(1,1) + … + A(1,K/vl) * B(K/vl, 1))

Total Ops: (K/vl + 1) * M * N

Matrix Multiplication



Matrix Multiplication

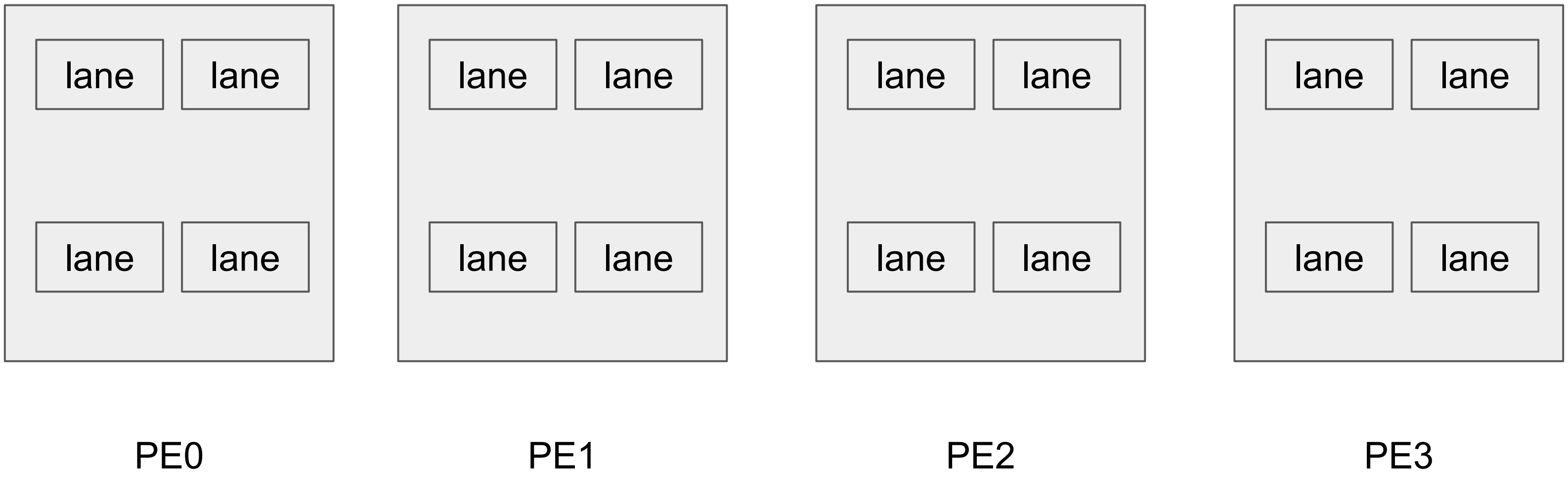


Total Ops: N/vl * K * M (No reduction)

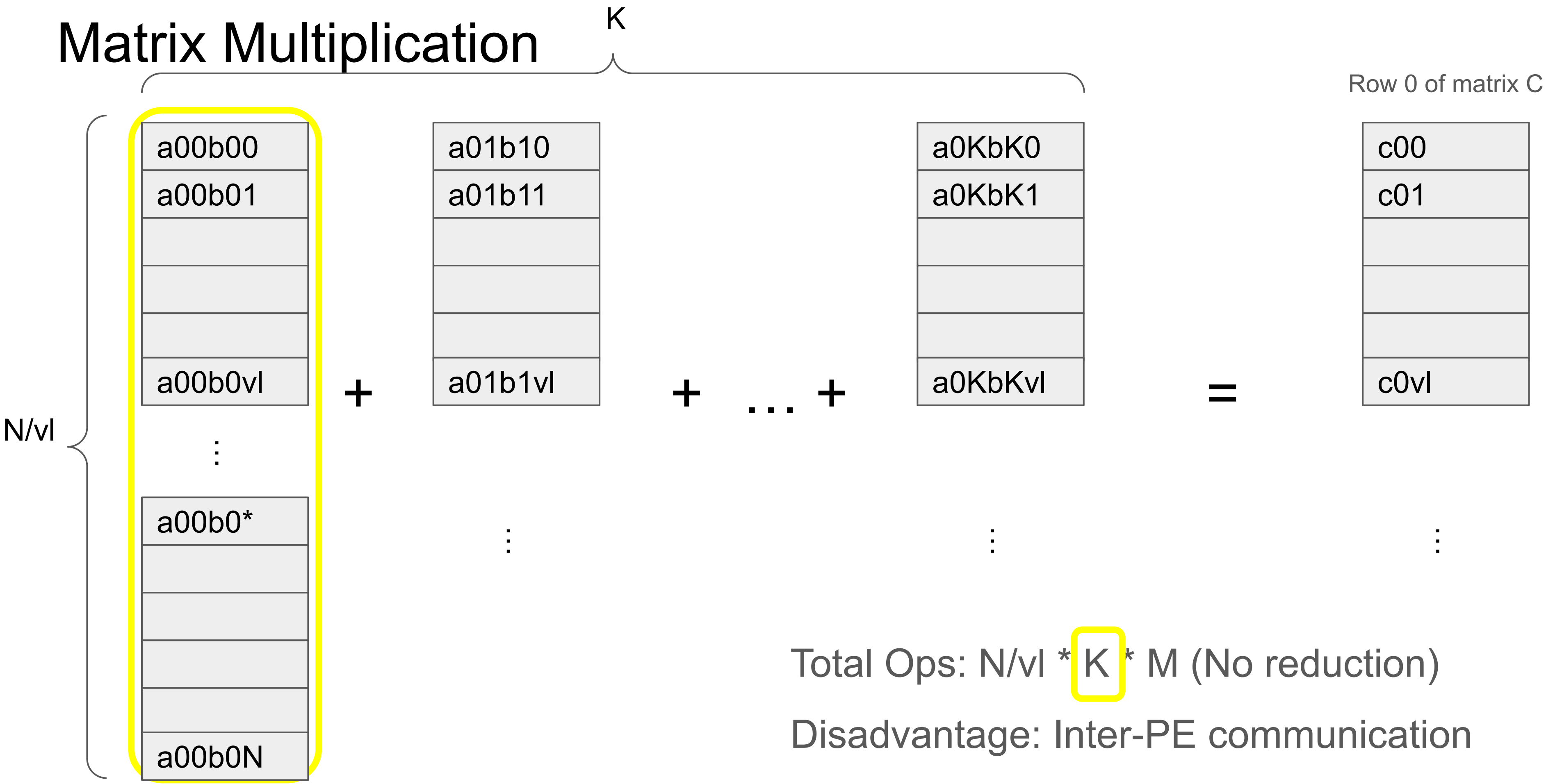
Matrix Multiplication (Ara)

1. Duplicate a00 to v1
2. Load b0* to v2
3. MAC(v1, v2)
4. Duplicate a01 to v1
5. Load b1* to v2
6. MAC(v1, v2)
7. …
8. Duplicate a10 to v1 (the second row of A)
9. …

A More Parallel Architecture



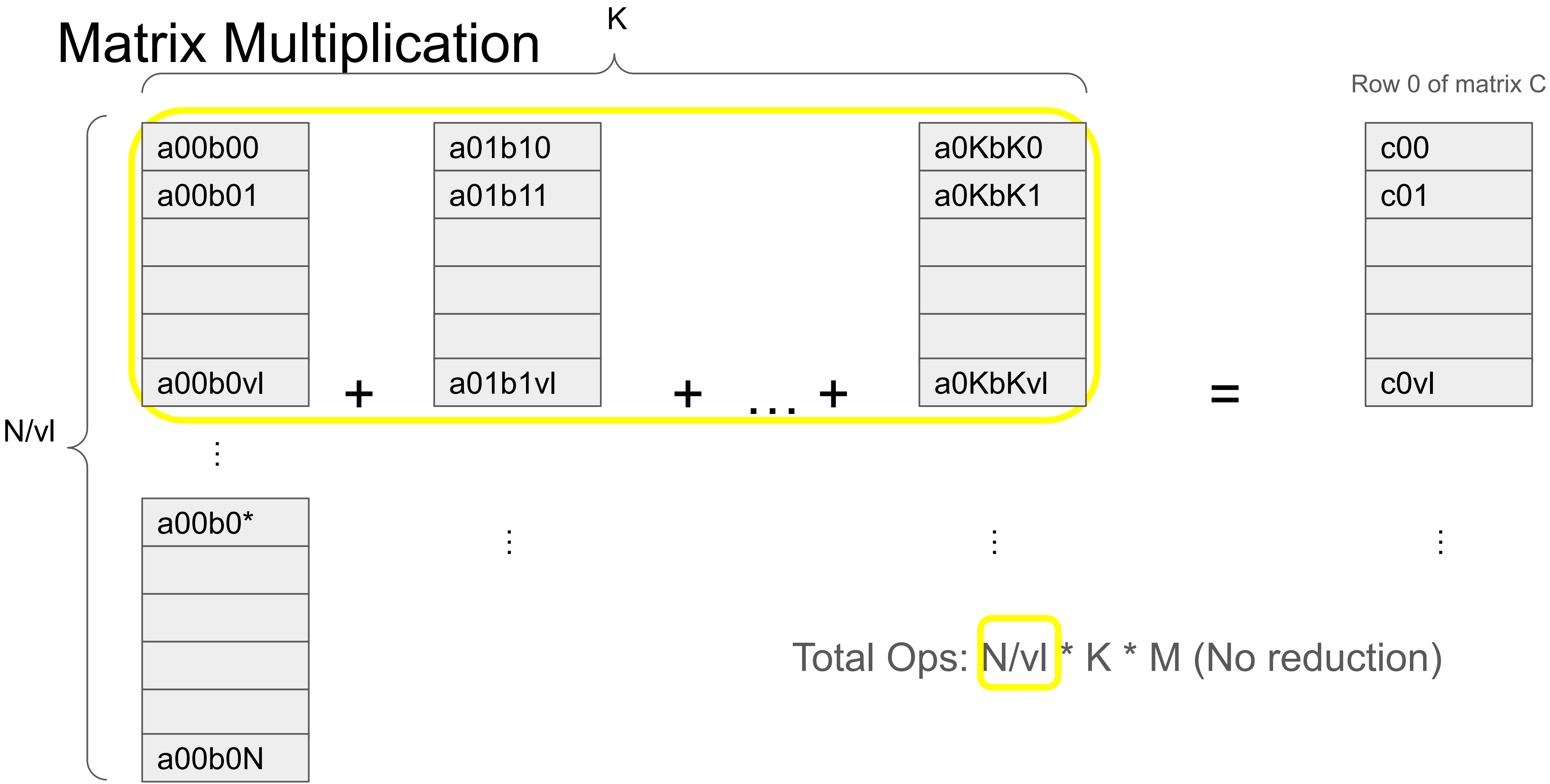
Matrix Multiplication



Total Ops: N/vl * K * M (No reduction)

Disadvantage: Inter-PE communication

Matrix Multiplication

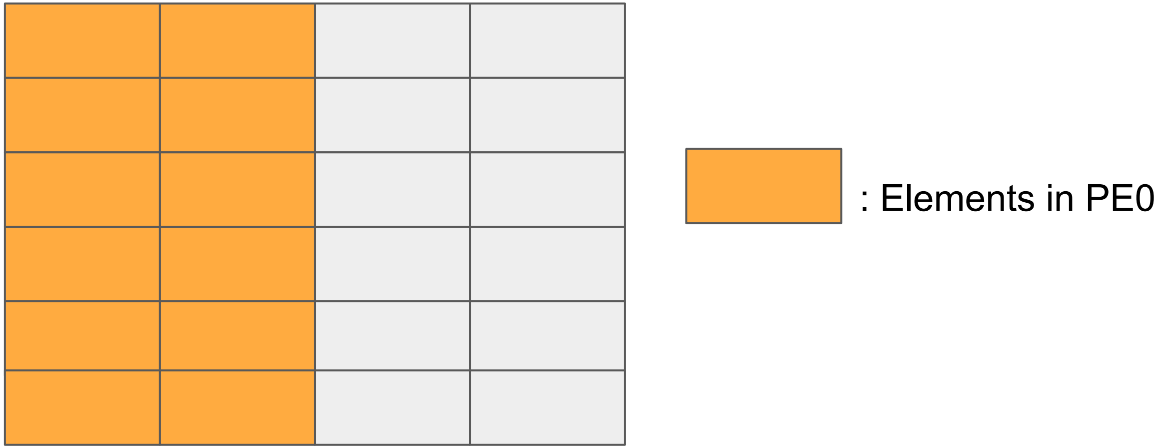


Total Ops: N/vl * K * M (No reduction)

Matrix Multiplication

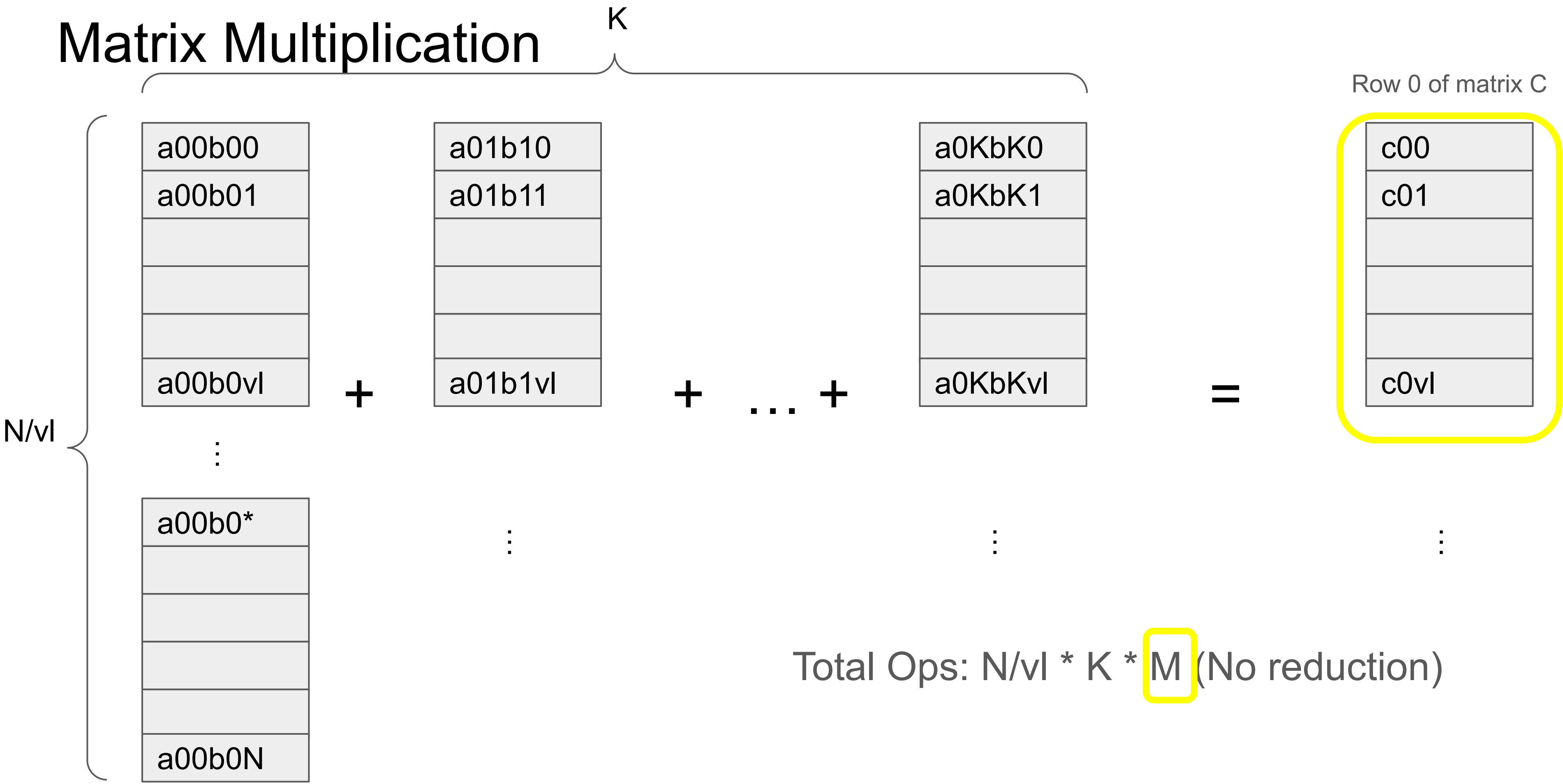
Total Ops: $\boxed{N/vl}$ * K * M (No reduction)

- Each PE allocates
  - Entire matrix A
  - Part of matrix B: b*vl
- After computation



Result matrix C

Matrix Multiplication



Total Ops: N/vl * K * M (No reduction)

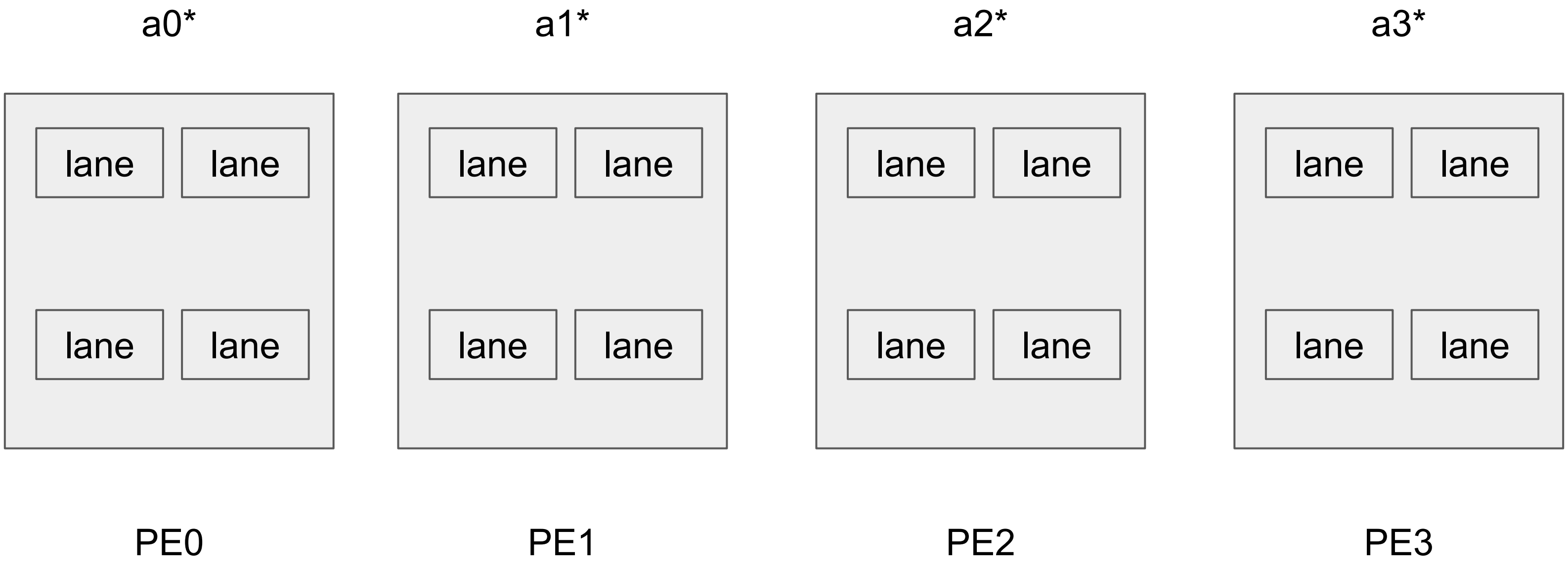
Matrix Multiplication

Total Ops: N/vl * K * $\boxed{M}$

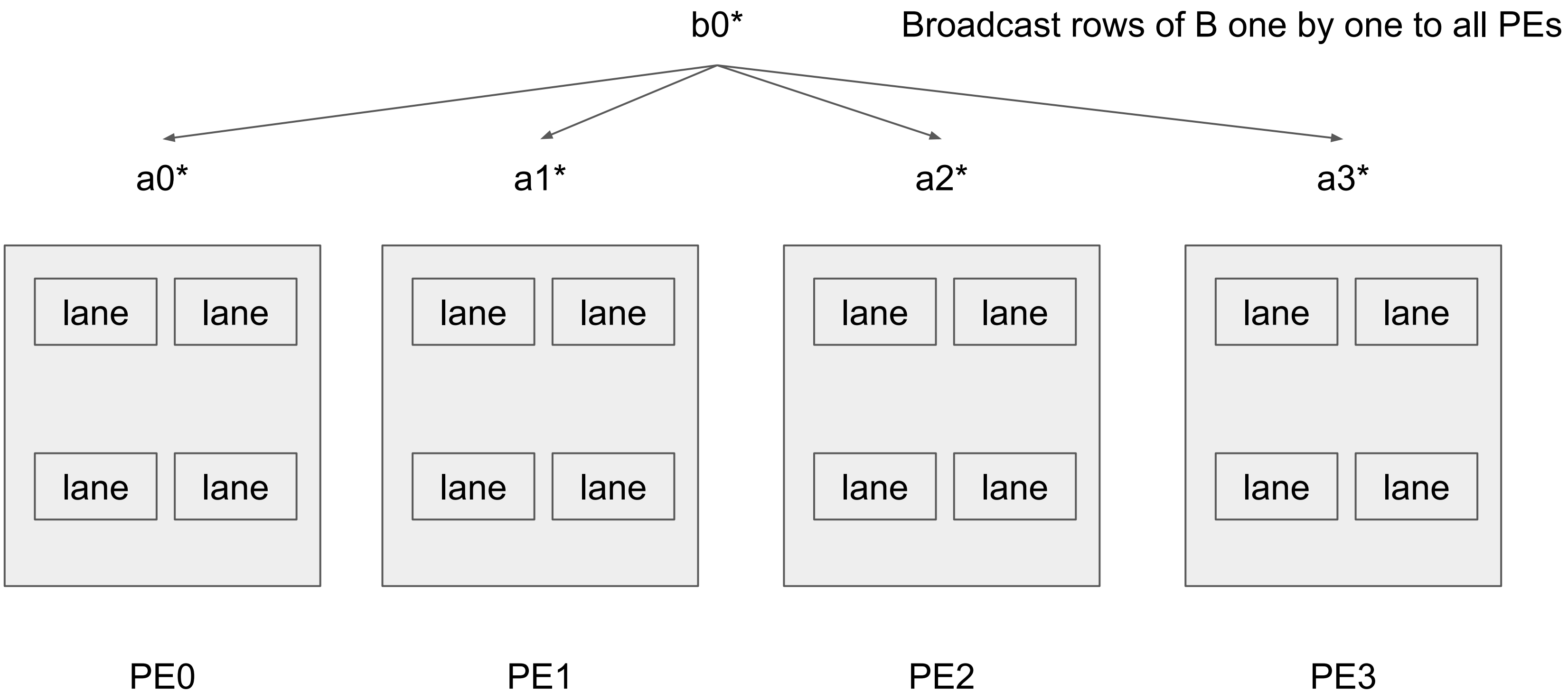
- Each PE allocates: entire matrix B, one row of matrix A
- After the computation, each PE holds one row of matrix C (or multiple rows, if M > #PEs, but complete)
- If the next operation is also a matrix multiplication (new A = C), one of the operand is already in the PE
- **A-row-parallel**

A More Parallel Architecture

Assign different rows of A to different PEs



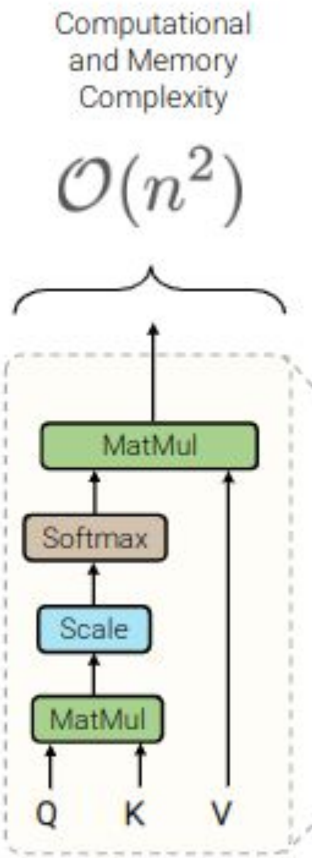
A More Parallel Architecture



Self Attention (SA)

Input: x, Wq, Wk, Wv

1. Q = x*Wq, K = x*Wk, V = x*Wv
2. Q*K^T
3. (Q*K^T)/sqrt(Dim)
4. softmax((Q*K^T)/sqrt(Dim))
5. softmax((Q*K^T)/sqrt(Dim)) * V



SA

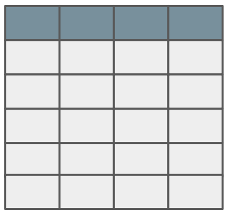
1. Q = x * Wq, K = x * Wk, V = x * Wv
    - Observation:
        - Reuse data x
        - Q & K will be used in the next operation
    - Vanilla Ara (A = x, B = Wq, Wk, Wv)
        - A-row-stationary: load one row of x, and compute one row of Q, K, V

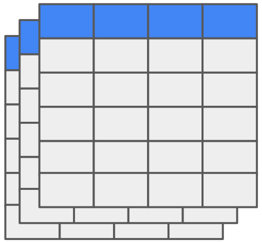
2. Q * K^T (how to fetch K columnwise?)

3. (Q * K^T) / sqrt(#columns of Q)

   For inference: scale weight Wq

1. Duplicate x00 to v1
2. Load Wq, Wk, Wv 0* to v2, v3, v4
3. MAC(v1, v2), MAC(v1, v3), MAC(v1, v4)
4. Duplicate x01 to v1
5. Load Wq, Wk, Wv 1* to v2
6. MAC(v1, v2), MAC(v1, v3), MAC(v1, v4)
7. …
8. Duplicate x10 to v1 (the second row of x)
9. …





SA

4. softmax((Q * K^T) / sqrt(dim))

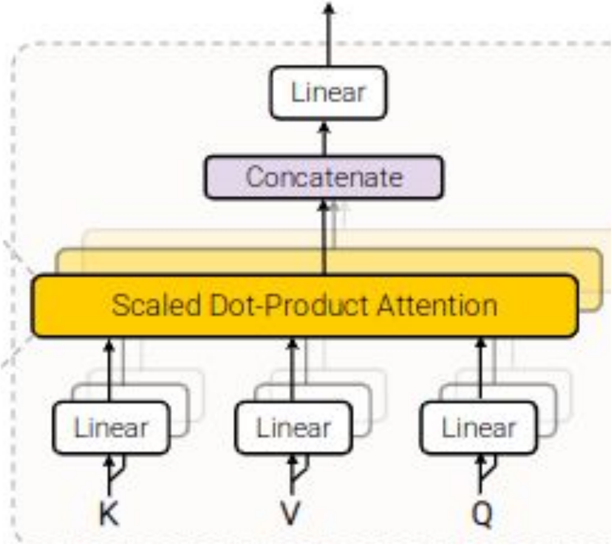
$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

- For x in each row, softmax(x) = exp(x) / sum(exp(x’ in row))
- In Ara
  - Element-wise exponentiation (not supported?)
  - Reduction sum | column-wise add (log-tree parallelism over different PEs)
  - Division

5. softmax((Q * K^T) / sqrt(dim)) * V

Multi-Head Self Attention (MHSA)

- MHSA(Q, K, V) = Concat (head_1, …, head_h) * Wo
  - Where head_i = SA(x, Wq_i, Wk_i, Wv_i)
  - Linear transformation: matrix multiplication
    - How to fetch data from different vectors in memory? (different heads)



Layer Normalization

- Average:
  - column-wise add, division
- Element minus average (x - E[x])
- Variance:
  - use (x - E[x]) from previous operation

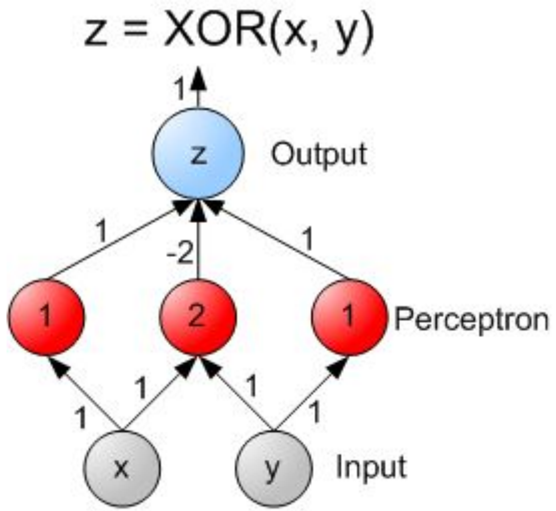
$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

$$\mu_i = \frac{1}{M} \sum x_i$$

$$\sigma_i = \sqrt{\frac{1}{M} \sum (x_i - \mu_i)^2 + \epsilon}$$

Feed Forward (FFN)

- relu(dropout(X * W1)) * W2
- relu(x) = (x > 0) ? x : 0
    - Vector mask?
    - Convert mask tensor to a compressed format?
- dropout(x) = 0 with probability p
    - Probability supported?
- Special format for sparsity



Linear Transformer (kernel)

- self-attention score = similarity score

$$\operatorname{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V \longrightarrow V_i' = \frac{\sum_{j=1}^{N} \operatorname{sim}\left(Q_i, K_j\right) V_j}{\sum_{j=1}^{N} \operatorname{sim}\left(Q_i, K_j\right)}$$

$$\operatorname{sim}(q, k) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$$

Linear Transformer (kernel)

- self-attention score = similarity score
- Similarity can be represented as kernel with feature representation $\phi(x)$
  - Q, K can be reused

$$\mathrm{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V$$

$$V_i' = \frac{\sum_{j=1}^{N} \mathrm{sim}\left(Q_i, K_j\right) V_j}{\sum_{j=1}^{N} \mathrm{sim}\left(Q_i, K_j\right)}$$

$$\mathrm{sim}\left(q, k\right) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$$

$$\frac{\phi\left(Q_i\right)^T \sum_{j=1}^{N} \phi\left(K_j\right) V_j^T}{\phi\left(Q_i\right)^T \sum_{j=1}^{N} \phi\left(K_j\right)}$$

Linear Transformer (kernel)

- self-attention score = similarity score
- Similarity can be represented as kernel with feature representation $\phi(x)$
- For matrix multiplication:

  (Q * K^T) * V = Q * (K^T * V)

$$\left(\phi(Q)\,\phi(K)^T\right)V = \phi(Q)\left(\phi(K)^T V\right)$$

{ (N | D) * (D | N)} * (N | D) = (N | D) * {(D | N) * (N | D)}

O(N^2 * D) O(N*D^2)

Linear Transformer (kernel)

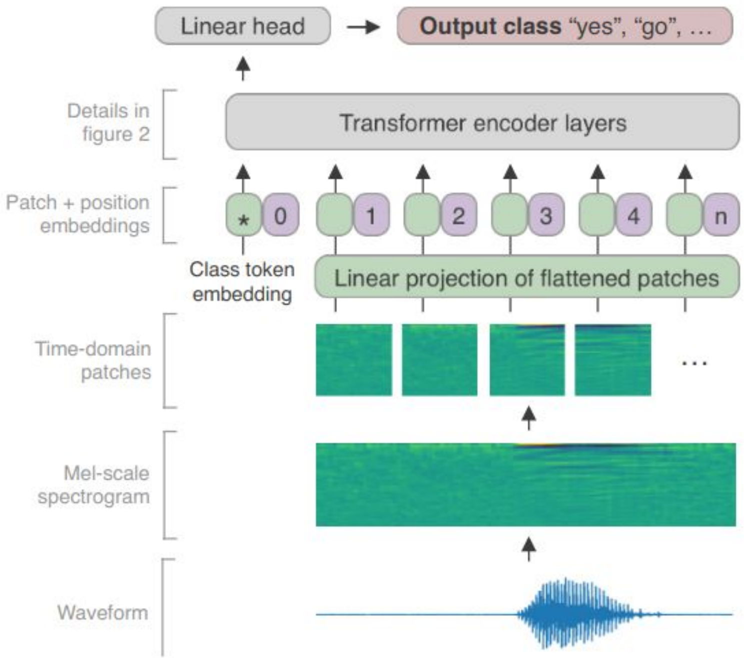
Feature function: $\phi(x) = \mathrm{elu}(x) + 1$

$$\mathrm{ELU}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha * (\exp(x) - 1), & \text{if } x \leq 0 \end{cases}$$

- Why use exponentiation?
  - Avoid zero gradient
- Can it be replaced by ReLU?

KWT

- Keyword spotting
  - Identification of keywords in audio (up, down, wake, sleep …)
- Contribution
  - Apply transformer model for KWS
- Pre-trained model: knowledge distillation



KWT

Model	V1-12	V2-12	V2-35
DS-CNN [17]	95.4		
TC-ResNet [19]	96.6		
Att-RNN [12]	95.6	96.9	93.9
MatchBoxNet [20]	97.48 ±0.11	97.6	
Embed + Head [18]		97.7	
MHAtt-RNN [13]	97.2	98.0	
Res15 [31]		98.0	96.4
MHAtt-RNN (Ours)	**97.50** ±0.29	98.36 ±0.13	97.27 ±0.02
KWT-3 (Ours)	97.24±0.24	**98.54** ±0.17	97.51 ±0.14
KWT-2 (Ours)	97.36 ±0.20	98.21 ±0.06	97.53 ±0.07
KWT-1 (Ours)	97.05 ±0.23	97.72 ±0.01	96.85 ±0.07
KWT-3 ⚗ (Ours)	**97.49** ±0.15	**98.56** ±0.07	97.69 ±0.09
KWT-2 ⚗ (Ours)	97.27 ±0.08	98.43 ±0.08	**97.74** ±0.03
KWT-1 ⚗ (Ours)	97.26 ±0.18	98.08 ±0.10	96.95 ±0.14

Model	dim	mlp-dim	heads	layers	# parameters
KWT-1	64	256	1	12	607K
KWT-2	128	512	2	12	2,394K
KWT-3	192	768	3	12	5,361K

Delta KWT

- Dense matrix to highly-sparse matrix
  - leave the first token untouched
  - If |x(t) - x‘(t-1)| < theta, x(t) = 0, else x(t) = x(t) - x’(t-1)
  - 80% operations reduction (no accuracy loss)
- MatMul: data reuse

$$\Delta X(t) = \begin{cases} X(t) - \hat{X}(t-1) & \text{if } |X(t) - \hat{X}(t-1)| > \theta \\ 0 & \text{otherwise} \end{cases}$$

$$\hat{X}(t) = \begin{cases} X(t) & \text{if } |X(t) - \hat{X}(t-1)| > \theta \\ \hat{X}(t-1) & \text{otherwise} \end{cases}$$