ELSA: Hardware-Software Co-Design for Efficient, Lightweight Self-Attention Mechanism in Neural Networks

Tae Jun Ham*  
✉ taejunham@snu.ac.kr

Yejin Lee*  
✉ yejinlee@snu.ac.kr

Seong Hoon Seo  
✉ andyseo247@snu.ac.kr

Soosung Kim  
✉ soosungkim@snu.ac.kr

Hyunji Choi  
✉ hyunjicho@snu.ac.kr

Sung Jun Jung  
✉ miguel92@snu.ac.kr

Jae W. Lee  
✉ jaewlee@snu.ac.kr

* These authors contributed equally to this work

The 48th ACM/IEEE International Symposium on Computer Architecture (ISCA)  
ARC Lab @ Seoul National University
Self-Attention: A Key Primitive of Transformer-based Models

- Transformer-based models achieve **state-of-the-art performance** on NLP
  - Natural Language Processing: QA, Translation, Language Modeling
  - CV: Image Classification, Object Detection
- Self-attention is the key primitive for the Transformer-based models
- Self-attention identifies relations within entities and enables models to effectively handle sequential data.
Self-Attention: A Key Primitive of Transformer-based Models

- Transformer-based models often come with large computational cost
- Self-attention often accounts for *more than 30% of the runtime* on Transformers
  - Even larger portion with longer input sequences

![Diagram showing time distribution between self-attention and others for different models and sequence lengths.]

**Models:**
- BERT
- RoBERTa
- ALBERT
- SASRec
- BERT4Rec

**Sequence Lengths:**
- Default Sequences
- 4x Larger Sequences
**Self-Attention Mechanism**

**Q. What is Self-Attention?**

**A.** Find key vectors relevant to a query vector, then return the *weighted sum of value vectors* corresponding to relevant key vectors.

**Dot Product Computation**

- **Query Matrix (n x d)**
  
  - [0.4, 0.1, -0.7]  
  - [0.5, -0.1, 0.9]

- **Key Matrix (n x d)**
  
  - [-1.6, -0.6, -0.7]  
  - [-1.2, 0.7, 0.3]  
  - [2.0, 0.5, 1.1]

- **Attention Score (n x n)**
  
  - [2.21, -0.17, -2.81]  
  - [-0.21, -0.62, 0.08]  
  - [-1.37, -0.40, 1.94]

- **Value Matrix (n x d)**
  
  - [-1.6, -0.6, -0.5]  
  - [-1.1, 0.5, 0.3]  
  - [2.1, -0.5, 1.1]

**Step 1. Attention Score:** Compute dot product of each row in the key matrix and the query.
## Self-Attention Mechanism

**Q. What is Self-Attention?**

A. Find key vectors relevant to a query vector, then return the *weighted sum of value vectors* corresponding to *relevant key vectors*.

### Step 1. Attention Score

Compute the *dot product* of each row in the key matrix and the query matrix.

### Step 2. Softmax-normalized Attention Score

**Softmax** normalization on the Attention Score.

### Diagram

- **Query Matrix** $(n \times d)$
- **Key Matrix** $(n \times d)$
- **Attention Score** $(n \times n)$
- **Normalized Score** $(n \times n)$
- **Value Matrix** $(n \times d)$

**Softmax Computation**

$$\exp^\text{Score} \sum \text{Score}$$
**Self-Attention Mechanism**

**Q. What is Self-Attention?**

**A.** Find key vectors relevant to a query vector, then return *the weighted sum of value vectors* corresponding to *relevant key vectors*

---

**Weighted Sum Computation**

**Step 1. Attention Score:** Compute *dot product* of each row in the key matrix and the query

**Step 2. Softmax-normalized Attention Score:** *Softmax* normalization on Attention Score

**Step 3. Weighted Sum of Value vectors:** *Weighted sum* of value matrix row vectors using Normalized Score as weights
Opportunities for Approximation in Self-Attention Mechanism

• Attention mechanism looks like a series of dense matrix operations, but it is also a **content-based search**

• Softmax operation **filters out data that are NOT relevant** to the query based on **attention scores**

**Key Idea**

What if we can find out the **set of candidates** that are likely to be relevant to the query without much computation?

- Avoid processing irrelevant keys
- Reduce the amount of computation
Why Specialized Hardware?

Observation  Conventional HW such as GPUs do not benefit from such approximation

- GPUs are better optimized for full similarity computation (dot product) than presented approximate similarity computation!

Specialized HWs can fully exploit the potential benefits of the presented approximate self-attention mechanism
ELSA: Hardware-Software Co-design for Efficient, Lightweight Self-Attention Mechanism

Approximate Self-Attention Algorithm

1. Efficient Key Hash Computation
2. Compute Hamming Distance
3. Hamming Distance To Angle
4. Cosine
5. Key Norm Multiplication
6. Candidate Selection

Key Matrix \( K \):

<table>
<thead>
<tr>
<th>( K_1 )</th>
<th>( K_2 )</th>
<th>( K_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>-0.2</td>
<td>1.1</td>
</tr>
<tr>
<td>1.4</td>
<td>-0.7</td>
<td>0.9</td>
</tr>
<tr>
<td>0.4</td>
<td>0.3</td>
<td>-0.2</td>
</tr>
<tr>
<td>0.9</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>0.9</td>
<td>0.8</td>
<td>1.2</td>
</tr>
<tr>
<td>0.1</td>
<td>0.0</td>
<td>-0.1</td>
</tr>
<tr>
<td>0.9</td>
<td>-0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>0.1</td>
<td>-0.2</td>
<td>0.7</td>
</tr>
<tr>
<td>0.1</td>
<td>0.7</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Query Matrix \( Q \):

<table>
<thead>
<tr>
<th>( Q_x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9</td>
</tr>
<tr>
<td>0.8</td>
</tr>
<tr>
<td>1.2</td>
</tr>
<tr>
<td>-0.3</td>
</tr>
<tr>
<td>0.1</td>
</tr>
<tr>
<td>0.0</td>
</tr>
<tr>
<td>-0.1</td>
</tr>
<tr>
<td>0.7</td>
</tr>
<tr>
<td>0.6</td>
</tr>
<tr>
<td>0.1</td>
</tr>
<tr>
<td>0.1</td>
</tr>
<tr>
<td>0.7</td>
</tr>
<tr>
<td>0.6</td>
</tr>
</tbody>
</table>

With novel approximate self-attention algorithm and specialized hardware, ELSA achieves significant reduction in the runtime as well as energy spent on the self-attention mechanism.
Estimating Angular Distance

Attention Score between a given query vector $Q$ and a key vector $K$,

$$\text{AttentionScore} (K, Q) = K \cdot Q$$

$d$ multiply-and-add (MAC) operations

Assuming $n$ entities, there are total $n^2$ pairs $\rightarrow n^2d$ MAC operations

Key Idea
ELSA maps query vector $Q$ and a key vector $K$ to binary vectors and approximates attention scores.

Query $Q$

Key Matrix $K$

Hash for $Q$

Hash for $K$
Estimating Angular Distance

Attention Score between a given query vector $Q$ and a key vector $K$,

$$AttentionScore (K, Q) = K \cdot Q$$

$Q$: it
$K$: animal, walks, into

$$= ||Q|| ||K|| \cos(\theta_{K,Q})$$
$$\propto ||K|| \cos(\theta_{K,Q})$$

same $||Q||$

Key Idea: Approximate $\cos(\theta_{K,Q})$ to efficiently approximate the attention score using binary hashing.

Self-attention is finding relevant keys to a given query $Q$, thus we can ignore $||Q||$ since it is the same for all keys.
Estimating Angular Distance

Attention Score between a given query vector $Q$ and a key vector $K$,

$$\text{AttentionScore} (K, Q) = K \cdot Q$$

$$= ||Q|| ||K|| \cos(\theta_{K,Q})$$

$$\propto ||K|| \cos(\theta_{K,Q})$$

**Key Idea** Approximate $\cos(\theta_{K,Q})$ to efficiently approximate the attention score using binary hashing

**Self-attention is finding relevant keys to a given query $Q$, thus we can ignore $||Q||$ since it is the same for all keys**

1. Initialize $k (=4)$ random vectors $(v_1, v_2, v_3, v_4)$

Normal vectors defined by them $(n_{v_1}, n_{v_2}, n_{v_3}, n_{v_4})$ are illustrated
Estimating Angular Distance

Attention Score between a given query vector \( Q \) and a key vector \( K \),

\[
AttentionScore (K, Q) = K \cdot Q = \|Q\|\|K\|\cos(\theta_{K,Q}) \\
\propto \|K\|\cos(\theta_{K,Q})
\]

*Self-attention is finding relevant keys to a given query \( Q \), thus we can ignore \( \|Q\| \) since it is the same for all keys*

- **Key Idea** Approximate \( \cos(\theta_{K,Q}) \) to efficiently approximate the attention score using binary hashing

2. \( h(x_i) = \{h_{v_1}(x_i), h_{v_2}(x_i), h_{v_3}(x_i), h_{v_4}(x_i)\} \)

where \( h_{v_j}(x_i) = \text{sgn}(v_i \cdot x_i) \)

\( h(x_1) = [\text{0110}] \)
\( h(x_2) = [\text{1110}] \)
\( h(x_3) = [\text{0001}] \)

*Jth hash bit is*

1. if \( x_i \) is on the \( + \) side with respect to \( n_{v_j} \)
2. if \( x_i \) is on the \( - \) side with respect to \( n_{v_j} \)
1) \( n \times r \) vector space

\[
V_1 = \begin{bmatrix}
0.5 & -0.7 & 0.3 & 0.9 & 1.2 & \cdots
\end{bmatrix}
\]
1) vector space

2) hyperplane

- random vector $v_i \in \mathbb{R}^n$
- normal vector $n_{v_i}$
- $x \cdot n_{v_i} > 0$
- $y \cdot n_{v_i} < 0$
1) $n \times \frac{m}{k}$ vector space

2) hyperplane 생성

random vector $V_{i} \in \mathbb{R}^{n}$

hyperplane

$X \cdot n_{v_{i}} > 0$

$Y \cdot n_{v_{i}} < 0$

3) binary hashing (ex: $k=3$)

$h(K_{1}) = [1 \ 0 \ 0]$

$h(Q) = [1 \ 1 \ 1]$

$\therefore k \neq \text{random vector의 } 4$
4) hamming distance

\[ h(K_i) = [1 \ 0 \ 0] \]

\[ h(Q) = [1 \ 1 \ 1] \]

\[ \text{hamming} \ (K_i, Q) = \sum (0 \ 1 \ 1) = 2 \]
4) hamming distance

\[ h(K_i) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \]
\[ h(Q) = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \]

\[ \text{hamming}(K_i, Q) = \sum (0 \ 1 \ 1) = 2 \]

5) compute \( \Theta_{QK_i} \) (\( k=3 \))

\[ \Theta_{QK_i} = \frac{\pi}{k} \text{hamming}(K_i, Q) = \frac{\pi}{3} \cdot 2 = \frac{2}{3} \pi \]

Attent score \( (Q, K_i) = \|K\| \cos \left( \frac{2}{3} \pi \right) \)

\( \downarrow \) Lookup table
Estimating Angular Distance

Attention Score between a given query vector $Q$ and a key vector $K$,

$$AttentionScore (K, Q) = K \cdot Q$$

$$= \|Q\|\|K\|\cos(\theta_{K,Q})$$

$$\propto \|K\|\cos(\theta_{K,Q})$$

Key Idea Approximate $\cos(\theta_{K,Q})$ to efficiently approximate the attention score using binary hashing

1. Can obtain angular distance just from binary embeddings

$$\|K\|\cos(\text{hamming}(h(Q), h(K)) \cdot \pi / k)$$

2. Example calculations:
   - $h(x_1) = [0110]$
     $$\theta_{x_1,x_2} \approx \frac{\pi}{k} \cdot \text{hamming}(0110, 1110) = \frac{1}{4}\pi$$
   - $h(x_2) = [1110]$
     $$\theta_{x_2,x_3} \approx \frac{\pi}{k} \cdot \text{hamming}(1110, 0001) = \pi$$
   - $h(x_3) = [0001]$
     $$\theta_{x_3,x_1} \approx \frac{\pi}{k} \cdot \text{hamming}(0001, 0110) = \frac{3}{4}\pi$$
Approximate Attention Score with Binary Hashing

**Step 1. Preprocessing** Compute hash values for keys and a query and norms for each key.

**Step 2. Approximate Attention Score Computation** Compute approximate attention scores for each key.

\[
AttentionScore(K, Q) = AttentionScore(K, Q)/\|Q\| \\
= K \cdot Q /\|Q\| = \|K\| \cos(\theta_{K,Q}) \\
\approx \|K\| \cos(\text{hamming}(h(Q), h(K)) \cdot \pi/k), \text{ where } k=\text{number of hash bits}
\]

**Step 3. Candidate Selection** Filter out potentially irrelevant keys whose approximate attention score is below a pretrained threshold (t)

\[
t \cdot \|K_{\max}\| \geq \|K\| \cos(\text{hamming}(h(Q), h(K)) \cdot \pi/k)
\]
Candidate Selection

One should sort all approximate attention scores and select top-scoring keys

- **Issue #1** Sorting is expensive in hardware
- **Issue #2** Sorting can only happen after computing all key’s estimated score is done

Select keys whose estimated score exceeds certain threshold

- **Pros #1** Do not need expensive sort
- **Pros #2** Filtering can happen during computing each key’s estimated score

Q. How do we find the right value for all of NN layers (and sub-layers)?

A. Learn from training data!
Candidate Selection

Step 1. Set Hyperparameter $p$ User enters hyperparameter $p$ for accuracy-performance tradeoff

Assume $n$ is the number of input entities (keys), we identify the set of keys whose softmax-normalized attention score ($s'$) exceeds $p/n$.

If $p = 2, n = 200$, user considers entities whose $s' > 0.01$ to be relevant

**Higher $p$: aggressive approximation, Lower $p$: conservative approximation**

Step 2. Learn Raw Attention Score whose Softmax-Normalized Attention Score $> p/n$

<table>
<thead>
<tr>
<th>Attention Score</th>
<th>Softmax Normalized Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>eScore/ΣeScore</td>
<td>0.48 0.06 0.30 0.16</td>
</tr>
</tbody>
</table>

Assume $n = 4, p = 1$. Filter out keys with $s' < 0.25$

Step 3. Repeat for All Cases In Training Data

User gets different thresholds for each layer (sub-layer)
ELSA Hardware Implementation

1. Preprocessing Phase (Key hash computation) → binary hash

2. Execution Phase
   2-1. Compute Query Hash
   2-2. Candidate Selection & Attention Computation
   2-3. Output Division
ELSA Hardware Implementation

1. **Preprocessing Phase** (Key hash computation)

2. **Execution Phase**
   - 2-1 Compute Query Hash
   - 2-2 Candidate Selection & Attention Computation
   - 2-3 Output Division

Compute hash for the query

\[ h(Q) = AQ_x \]
**ELSA Hardware Implementation**

1. **Preprocessing Phase** (Key hash computation)

2. **Execution Phase**
   - 2-1 Compute Query Hash
   - 2-2 Candidate Selection & Attention Computation
   - 2-3 Output Division

Compute approximate similarity & filter out keys with low similarity

**Approximate Similarity**

\[ ||K|| \cos(\text{hamming}(h(Q), h(K)) \cdot \pi/k) \]
ELSA Hardware Implementation

1. Preprocessing Phase (Key hash computation)
2. Execution Phase
   2-1 Compute Query Hash
   2-2 Candidate Selection & Attention Computation
   2-3 Output Division

Compute partial attention for selected keys and accumulate it in the buffer

\[ \sum e^{K_i \cdot Q_i} \cdot V_i \]
ELSA Hardware Implementation

1. **Preprocessing Phase** (Key hash computation)

2. **Execution Phase**
   - 2-1 Compute Query Hash
   - 2-2 Candidate Selection & Attention Computation
   - 2-3 Output Division

Normalize the outcome by dividing it by the sum of exponents

\[
\frac{\sum e^{K_i \cdot Q_i} \cdot V_i}{\sum e^{K_i \cdot Q_i}}
\]
ELSA Hardware Pipelining

1. Preprocessing Phase (Key hash computation)
2. Execution Phase
   - 2-1 Compute Query Hash
   - 2-2 Candidate Selection & Attention Computation
   - 2-3 Output Division

- Query-level pipelining (2-1, 2-2, 2-3)
- Fine-grained pipelining within each pipeline stage
Evaluation: Impact of Approximate Attention

- **Line Graph**: model accuracy metric
- **Bar Graph**: portion of selected candidates

< 1% accuracy metric degradations are observed while inspecting only ~33% of the keys
## Evaluation: Performance Evaluation

<table>
<thead>
<tr>
<th>Dataset/Model</th>
<th>Ideal Speedup</th>
<th>Ours (No Approx) Speedup</th>
<th>Ours (&lt;1%) Speedup</th>
<th>Ours (&lt;2.5%) Speedup</th>
<th>Ours (&lt;5%) Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>BERT</td>
<td>118x</td>
<td>8x</td>
<td>123x</td>
<td>20x</td>
<td>32x</td>
</tr>
<tr>
<td>SQuAD1.1v2</td>
<td>84x</td>
<td>8x</td>
<td>88x</td>
<td>8x</td>
<td>32x</td>
</tr>
<tr>
<td>RACE</td>
<td>20x</td>
<td>8x</td>
<td>19x</td>
<td>13x</td>
<td>15x</td>
</tr>
<tr>
<td>SQuAD1.1v2</td>
<td>66x</td>
<td>55x</td>
<td>58x</td>
<td>28x</td>
<td>60x</td>
</tr>
<tr>
<td>RACE</td>
<td>66x</td>
<td>55x</td>
<td>58x</td>
<td>28x</td>
<td>60x</td>
</tr>
<tr>
<td>IMDB</td>
<td>60x</td>
<td>50x</td>
<td>60x</td>
<td>50x</td>
<td>60x</td>
</tr>
<tr>
<td>SQuAD1.1v2</td>
<td>152x</td>
<td>114x</td>
<td>139x</td>
<td>43x</td>
<td>43x</td>
</tr>
<tr>
<td>RACE</td>
<td>152x</td>
<td>114x</td>
<td>139x</td>
<td>43x</td>
<td>43x</td>
</tr>
<tr>
<td>IMDB</td>
<td>152x</td>
<td>114x</td>
<td>139x</td>
<td>43x</td>
<td>43x</td>
</tr>
<tr>
<td>SASR BERT4R</td>
<td>101x</td>
<td>92x</td>
<td>101x</td>
<td>92x</td>
<td>101x</td>
</tr>
<tr>
<td>MovieLens-1M</td>
<td>168x</td>
<td>157x</td>
<td>168x</td>
<td>157x</td>
<td>168x</td>
</tr>
</tbody>
</table>

**Proposed HW Accelerator (w/o Approximation)**

- **8x – 46x** speedup over GPU
- **<1%** accuracy metric degradation

**Proposed HW Accelerator (w/ Approximation)**

- **20x – 157x** speedup over GPU
- **<2.5%** accuracy metric degradation
- **32x – 168x** speedup over GPU
Evaluation: Area / Energy Efficiency Evaluation

- **Area**: 1.26 mm² per unit / 15.06 mm² for 12 units (V100 GPU : 815 mm²)
- **Peak Power**: 0.96W per unit / 11.47 W for 12 units (V100 GPU : 250W)

ELSA achieves **2 – 3 orders of magnitude energy efficiency** over conventional GPU

Approximation provides **extra 2.2x – 5.4x additional energy efficiency**

Energy breakdown shows that approximation **significantly reduces the energy spent on attention computation** while consuming very little energy
ELSA: Hardware-Software Co-Design for Efficient, Lightweight Self-Attention Mechanism in Neural Networks

Tae Jun Ham*  
taejunham@snu.ac.kr

Yejin Lee*  
yejinlee@snu.ac.kr

Seong Hoon Seo  
andyseo247@snu.ac.kr

Soosung Kim  
soosungkim@snu.ac.kr

Hyunji Choi  
hyunjchoi@snu.ac.kr

Sung Jun Jung  
miguel92@snu.ac.kr

Jae W. Lee  
jaewlee@snu.ac.kr

* These authors contributed equally to this work