SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning

Hanrui Wang, Zhekai Zhang, Song Han
MIT HAN Lab
Massachusetts Institute of Technology
NLP is Ubiquitous

- NLP techniques are widely used

Chat Bots
Grammar Checking
Recommender System
Machine Translation
Efficient NLP is Important

- NLP model size and computation are increasing exponentially
Efficient NLP is Important

- End of Moore’s Law
- Need specialized efficient NLP algorithm-hardware co-design

![Processor Performance Graph](image)

John Hennessy and David Patterson, Computer Architecture: A Quantitative Approach
Outline

• Quick Overview
• Background
• Algorithmic Optimizations
• Hardware Architecture
• Evaluation
• Conclusion
Outline

• Quick Overview
• Background
• Algorithmic Optimizations
• Hardware Architecture
• Evaluation
• Conclusion
Quick Overview —
Cascade Token/Head Pruning

As a visual treat, the film is almost perfect.

11 Tokens ↓ 8 Heads

• Intuition: human language contains high redundancy, remove unimportant tokens and heads
Quick Overview — Cascade Token/Head Pruning

As a visual treat, the film is almost perfect.

11 Tokens → 8 Heads

BERT Layer 1 (100% Computation & Memory Access)

- Intuition: human language contains high redundancy, remove unimportant tokens and heads
Quick Overview — Cascade Token/Head Pruning

As a visual treat, the film is almost perfect.
11 Tokens ↓ 8 Heads
BERT Layer 1 (100% Computation & Memory Access)

As treat, film perfect.
5 Tokens ↓ 6 Heads

• Intuition: human language contains high redundancy, remove unimportant tokens and heads
Quick Overview —
Cascade Token/Head Pruning

As a visual treat, the film is almost perfect.

11 Tokens → 8 Heads
BERT Layer 1 (100% Computation & Memory Access)

As treat, film perfect.
5 Tokens → 6 Heads
Layer 2 (34%)

- Intuition: human language contains high redundancy, remove unimportant tokens and heads
Quick Overview — Cascade Token/Head Pruning

As a visual treat, the film is almost perfect.
11 Tokens ↓ 8 Heads
BERT Layer 1 (100% Computation & Memory Access)

As a visual treat, the film is almost perfect.
5 Tokens ↓ 6 Heads
Layer 2 (34%)

• Intuition: human language contains high redundancy, remove unimportant tokens and heads

As a visual treat, the film is almost perfect.
2 Tokens ↓ 4 Heads
film perfect
Quick Overview — Cascade Token/Head Pruning

As a visual treat, the film is almost perfect.

BERT Layer 1 (100% Computation & Memory Access)

As a visual treat, film perfect.

Layer 2 (34%)

film perfect

Layer 3 (9%)

Sentiment Classification: Positive

Intuition: human language contains high redundancy, remove unimportant tokens and heads.
Quick Overview — Top-k Engine

- To support the fast selection of which tokens and heads to prune

```
[7, 2, 6, 4, 9, 1, 3, 5]

Quick Select

[7, 0, 6, 0, 9, 0, 0, 0]

Zero Eliminator

[7, 6, 9, 5]

Top-k Results
```

```
[7, 2, 6, 4, 9, 1, 3, 5]

FIFO

Quick Select

k_th_largest: 5

num_eq_k_th_largest: 3

[0, 0, 0, 0, 0; 0, 0, 5]

Zero Eliminator

[7, 6, 9, 5]

Top-k Results
```
Quick Overview — Progressive Quantization

- Reduce the DRAM access: eagerly fetch MSB; lazily fetch LSB
- An intuitive analogy:

<table>
<thead>
<tr>
<th>Information in DRAM</th>
<th>MSB fetched to On-chip</th>
<th>Confidence</th>
<th>Need to fetch LSB?</th>
</tr>
</thead>
<tbody>
<tr>
<td>“This is my favorite computer program”</td>
<td>“Ths is my favrit cmptr prog”</td>
<td>High</td>
<td>No</td>
</tr>
<tr>
<td>“I like the great visual treat”</td>
<td>“I lk te gt vl trt”</td>
<td>Low</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Outline

- Quick Overview
- Background
- Algorithmic Optimizations
- Hardware Architecture
- Evaluation
- Conclusion
Attention-Based NLP Models

- Discriminative Model
  - BERT
    - Summarization Stage
- Generative Model
  - GPT-2
    - Summarization Stage
    - Generation Stage
Summarization Stage

- Discriminative Model
  - BERT
    - Summarization Stage
- Generative Model
  - GPT-2
    - Summarization Stage
    - Generation Stage

Token Embedding

"It" "is" "nice"

Block 1

Block n

Classification positive ✓
Summarization Stage

- Discriminative Model
  - BERT
    - Summarization Stage
- Generative Model
  - GPT-2
    - Summarization Stage
    - Generation Stage
Summarization Stage

- Discriminative Model
  - BERT
    - Summarization Stage
- Generative Model
  - GPT-2
    - Summarization Stage
    - Generation Stage
Summarization Stage

- Discriminative Model
  - BERT
    - Summarization Stage
- Generative Model
  - GPT-2
    - Summarization Stage
    - Generation Stage
Summarization Stage

• Discriminative Model
  • BERT
    • Summarization Stage
• Generative Model
  • GPT-2
    • Summarization Stage
  • Generation Stage

Q, K, V are all matrices in summarization stage
Summarization Stage

- Discriminative Model
  - BERT
- Summarization Stage
- Generative Model
  - GPT-2
  - Summarization Stage
  - Generation Stage
Summarization Stage

- Discriminative Model
  - BERT
- Generative Model
  - GPT-2

In addition:
- Summarization Stage
- Generation Stage
• Discriminative Model
  • BERT
  • Summarization Stage
• Generative Model
  • GPT-2
  • Summarization Stage
  • Generation Stage
SpAtten: Efficient Sparse Attention Architecture

**Generation Stage**

- Discriminative Model
  - BERT
- Summarization Stage
- Generative Model
  - GPT-2
- Summarization Stage
  - Generation Stage

Q is a single vector
K, V are matrices in generation stage
Generation Stage

- Discriminative Model
  - BERT
- Summarization Stage
- Generative Model
  - GPT-2
  - Summarization Stage
  - Generation Stage
Generation Stage

- Discriminative Model
  - BERT
  - Summarization Stage
- Generative Model
  - GPT-2
  - Summarization Stage
    - Generation Stage
- Block = Layer
Attention Layer - in Summarization Stage

Query

11 tokens

Key

11 tokens

Value

11 tokens

SpAtten: Efficient Sparse Attention Architecture
Attention Layer - in Summarization Stage

8 heads

Query
11 tokens

Key
11 tokens

Value
11 tokens
SpAtten: Efficient Sparse Attention Architecture

**Attention Layer - in Summarization Stage**

- Dimension of one head is typically 64
Attention Layer - in Summarization Stage

Intuition: each token is trying to assess the relevance/importance of other tokens to itself.
Attention Layer - in Summarization Stage

8 heads

Query
11 tokens

Key
11 tokens

Value
11 tokens

Head 1

Q * K^T

Attention Score

Row-wise Softmax

Attention Probability

11 tokens

11 tokens

11 tokens

11 tokens
Attention Layer - in Summarization Stage

- Query: 11 tokens
- Key: 11 tokens
- Value: 11 tokens

8 heads

Head 1

\[ Q \cdot K^T \]

Row-wise Softmax

Attention Score

Attention Probability

Attention Probability

8 heads
Attention Layer - in Summarization Stage

8 heads

Query
11 tokens

Key
11 tokens

Value
11 tokens

Attention Score
11 tokens

Row-wise
Softmax

Attention Probability
11 tokens

Attention Probability

8 heads

Head 2

Q * K^T

V
Attention Layer - in Summarization Stage

- **Query**: 11 tokens
- **Key**: 11 tokens
- **Value**: 11 tokens

- **Head 3**: 8 heads

- **Attention Score**: $Q \times K^T$
- **Attention Probability**: Row-wise Softmax
- **Attention Probability**: $V$

- **SpAtten: Efficient Sparse Attention Architecture**
Attention Layer - in Summarization Stage

8 heads

Query

11 tokens

Key

11 tokens

Value

11 tokens

11 tokens

11 tokens

11 tokens

Attention Score

Attention Probability

Attention Probability

Attention Probability

Row-wise Softmax

Head 4

Q \times K^T

11

11

11

11

Attention Layer - in Summarization Stage
Attention Layer - in Summarization Stage

- Query: 11 tokens
- Key: 11 tokens
- Value: 11 tokens

8 heads

Head 5

Attention Score

Row-wise Softmax

Attention Probability

Attention Probability

\[ Q \cdot K^T \]
Attention Layer - in Summarization Stage

- **Query**: 11 tokens
- **Key**: 11 tokens
- **Value**: 11 tokens

- **8 heads**
- **Head 6**

- **Attention Score**
- **Row-wise Softmax**
- **Attention Probability**

- **Q \cdot K^T**
- **11**

- **11**

- **Attention Probability**

- **V**
Attention Layer - in Summarization Stage

8 heads

Head 7

Query
11 tokens

Key
11 tokens

Value
11 tokens

Attention Score

Row-wise Softmax

Attention Probability

Attention Probability * V
Attention Layer - in Summarization Stage

- Query:
  - 11 tokens
  - 8 heads

- Key:
  - 11 tokens

- Value:
  - 11 tokens

- Attention Score:
  - 11 tokens

- Attention Probability:
  - 11 tokens

- Attention Probability V:
  - 11 tokens

Head 8

Q \cdot K^T

Row-wise Softmax

Attention Probability V
Attention Layer - in Generation Stage

8 heads

Query
1 tokens

Key
11 tokens

Value
11 tokens

Head 1
11
11

Attention Score

Row-wise Softmax

Attention Probability

Attention Probability

\[ Q \times K \]

\[ \text{Attention Probability} \]

\[ V \]
Attention Layer - in Generation Stage

- **Query**: 1 token
- **Key**: 11 tokens
- **Value**: 11 tokens

- **8 heads**

  - **Attention Score**
  - **Softmax**
  - **Attention Probability**

- **Head 2**: 11 tokens

- **Attention Probability**: V
Attention Layer - in Generation Stage

Query: 1 tokens
Key: 11 tokens
Value: 11 tokens

8 heads

Head 3

Attention Score
Softmax
Attention Probability

Q * K

Attention Probability

V

8 heads

11 tokens

11 tokens

11 tokens
Attention Layer - in Generation Stage

- Query: 1 token
- Key: 11 tokens
- Value: 11 tokens

8 heads

Attention Layer - in Generation Stage

Attention Score

Row-wise Softmax

Attention Probability

8 heads

Head 4

11 tokens

11 tokens

11 tokens

Attention Probability
Attention Layer - in Generation Stage

Query
1 tokens

Key
11 tokens

Value
11 tokens

8 heads

Head 5

11

11

\( Q \times K T \)

Attention Score

Row-wise Softmax

Attention Probability

Attention Probability

8 heads

Head 5

11

11

\( Q \times K T \)

Attention Score

Row-wise Softmax

Attention Probability

Attention Probability

V
Attention Layer - in Generation Stage

Query
1 tokens

Key
11 tokens

Value
11 tokens

8 heads

Head 6
11 token

1

Attention Score

Row-wise

Softmax

Attention Probability

Attention Probability

V

8 heads

Head 6
11 token

1

Attention Score

Row-wise

Softmax

Attention Probability

Attention Probability

V
Attention Layer - in Generation Stage

Query 1 tokens

Key 11 tokens

Value 11 tokens

8 heads

Head 7

11

11

Attention Score

Row-wise Softmax

Attention Probability

Attention Probability * V

11 tokens

8 heads

Head 7

11

11

Attention Score

Row-wise Softmax

Attention Probability

Attention Probability * V
• Different from Convolution or FC:
  • Query, Key, Value are activations
Attention Runs Slow

- End-to-End GPT-2 latency breakdown

![Bar chart showing latency breakdown for different hardware: TitanXp GPU (49%), Xeon CPU (50%), Nano GPU (61%).]
Attention Runs Slow

• End-to-End GPT-2 latency breakdown

![Latency Breakdown Chart]

- Attention runs slow
- Attention takes over 50% latency
Attention Runs Slow

- End-to-End GPT-2 latency breakdown
- Attention latency breakdown

Attention takes over 50% latency

MatMul only accounts for 27% latency

Q × K

16.4%

10.6%
Attention Runs Slow

- End-to-End GPT-2 latency breakdown

![Latency Breakdown Graph]

- Attention latency breakdown

- Attention takes over 50% latency

- Memory operations take over 70% latency

MatMul only accounts for 27% latency
Q × K 16.4%
Transpose & Softmax 33.3%
Split Heads & Concat & Reshape 39.6%

Q × K
Attention
Prob × V
10.6%
Our Solution: SpAtten

- Cascade Token Pruning
- Cascade Head Pruning
- Local Value Pruning
- Progressive Quantization

Top-K Engine

Dedicated Accelerator
Outline

- Quick Overview
- Background
- Algorithmic Optimizations
- Hardware Architecture
- Evaluation
- Conclusion
Our Solution: SpAtten

- Cascade Token Pruning
- Cascade Head Pruning
- Local Value Pruning
- Progressive Quantization
- Top-K Engine
- Dedicated Accelerator
Cascade Token/Head Pruning

- Cascade Token/Head Pruning
- 8 heads
- Head 8
- Query: 11 tokens
  - Q
  - K
  - T
  - Attention Score
  - Attention Probability
  - V
- Key: 11 tokens
  - Row-wise Softmax
- Value: 11 tokens
  - Attention Probability
- Compute next layer’s Query, Key and Value
Cascade Token/Head Pruning

Layer 1
11 tokens
8 heads
Q
K
V

Layer 2
11 tokens
8 heads
Q
K
V

Query
Key
Value

Attention
FFN
QKV FC

8 heads

11 tokens

SpAtten: Efficient Sparse Attention Architecture
Cascade Token/Head Pruning

- Not all tokens/heads are created equal
- Find unimportant tokens and heads in front layers
- Remove them in latter layers
Cascade Token/Head Pruning

As a visual treat, the film is almost perfect.

"As a visual treat, the film is almost perfect."

"As treat, film perfect."
Cascade Token/Head Pruning

- Pruned tokens/heads will never be used in all following layers: "Cascade"
Cascade Token/Head Pruning

- Pruned tokens/heads will never be used in all following layers: “Cascade”
As a visual treat, the film is almost perfect.

As treat, film perfect.

"As a visual treat, the film is almost perfect."

"As treat, film perfect."

Pruned tokens/heads will never be used in all following layers: "Cascade"
Cascade Token/Head Pruning

- Fundamentally **different** from weight pruning:
  - Query, Key, Value are **activations**
  - Pruned tokens/heads are **input-dependent**
Find Unimportant Tokens with Attention Probabilities

Intuition: each token is trying to assess the relevance/importance of other tokens to itself.

Compute next layer's Query, Key and Value.
Find Unimportant Tokens with Attention Probabilities

- If one column in attention probability is **small**: the token is **unimportant** to all other tokens.
Find Unimportant Tokens with Attention Probabilities

- If one column in attention probability is small: the token is unimportant to all other tokens.
- Maintain an importance score for each token.

Row-wise Softmax

Attention Probability

<table>
<thead>
<tr>
<th>bet</th>
<th>the</th>
<th>video</th>
<th>game</th>
<th>is</th>
<th>a</th>
<th>lot</th>
<th>more</th>
<th>fun</th>
<th>than</th>
<th>the</th>
<th>film</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>1.0</td>
<td>0.3</td>
<td>1.2</td>
<td>1.7</td>
<td>1.0</td>
<td>0.4</td>
<td>1.8</td>
<td>0.6</td>
<td>1.9</td>
<td>1.4</td>
<td>0.3</td>
</tr>
</tbody>
</table>
Find Unimportant Tokens with Attention Probabilities

- If one column in attention probability is small: the token is unimportant to all other tokens.
- Maintain an importance score for each token.
- Accumulate attention probs to the importance scores.
Find Unimportant Tokens with Attention Probabilities

- If one column in attention probability is **small**: the token is **unimportant** to all other tokens
- Maintain an **importance score** for each token
- **Accumulate** attention probs to the importance scores

SpAtten: Efficient Sparse Attention Architecture
Find Unimportant Tokens with Attention Probabilities

- If one column in attention probability is **small**: the token is **unimportant** to all other tokens
- Maintain an **importance score** for each token
- **Accumulate** attention probs to the importance scores
- **Top-k** scores indicate top-k important tokens
  - Pruning ratio is a hyperparameter
Find Unimportant Tokens with Attention Probabilities

- If one column in attention probability is **small**: the token is **unimportant** to all other tokens.
- Maintain an **importance score** for each token.
- **Accumulate** attention probs to the importance scores.
- **Top-k** scores indicate top-k important tokens.
  - Pruning ratio is a hyperparameter.
- Importance scores can be accumulated **across** heads and layers.

Small score tokens can be pruned away.
Find Unimportant Heads with Attention Outputs

Query

11 tokens

Key

11 tokens

Value

11 tokens

Head 8

8 heads

Intuition: each token is trying to assess the relevance/importance of other tokens to itself

Attention Score

Attention Probability

Attention Output

Compute next layer’s Query, Key and Value
Find Unimportant Heads with Attention Outputs

- If one head output is **small**: the head is **unimportant** to latter layers
- Maintain an **importance score** for each head
- **Accumulate** attention output magnitude to the importance scores
- Top-k scores indicate top-k important heads
- Importance scores can be accumulated **across** layers
Our Solution: SpAtten

- Cascade Token Pruning
- Cascade Head Pruning
- Local Value Pruning
- Progressive Quantization

Top-K Engine

Dedicated Accelerator
Local Value Pruning

- Cascade token/head pruning find unimportant token/head in **current** layer, prune in **next** layer
- Local value pruning find unimportant Value in **current** layer, prune in **current** layer
Local Value Pruning

- Cascade token/head pruning find unimportant token/head in **current** layer, prune in **next** layer
- Local value pruning find unimportant Value in **current** layer, prune in **current** layer
- **Directly** rank top-k attention probabilities and only fetch their Value vectors

![Diagram of local value pruning](image)

8 heads

1 tokens

1 tokens

1 tokens

Query

Key

Value

\[ Q \cdot K^T \]

Attention Score

Row-wise Softmax

Attention Probability

Small Magnitude

Attention Probability

**V**
Local Value Pruning

- Cascade token/head pruning find unimportant token/head in **current** layer, prune in **next** layer
- Local value pruning find unimportant Value in **current** layer, prune in **current** layer
- **Directly** rank top-k attention probabilities and only fetch their value vectors

![Diagram showing SpAtten: Efficient Sparse Attention Architecture](image)
Local Value Pruning

- Cascade token/head pruning find unimportant token/head in **current** layer, prune in **next** layer
- Local value pruning find unimportant Value in **current** layer, prune in **current** layer
- **Directly** rank top-k attention probabilities and only fetch their value vectors
- Only for **generation** stage
  - Because in summarization, small attention probs have different locations in different rows
Our Solution: SpAtten

- Cascade Token Pruning
- Cascade Head Pruning
- Local Value Pruning
- Progressive Quantization
- Top-K Engine
- Dedicated Accelerator
Progressive Quantization

- Generation stage is **memory-bounded** because of matrix vector multiplication
- Need to fetch Key and Value from DRAM
- **Static** quantization: quantizes Key, Value to reduce DRAM access
Progressive Quantization

- Generation stage is **memory-bounded** because of matrix vector multiplication
- Need to fetch Key and Value from DRAM
- **Static** quantization: quantizes Key, Value to reduce DRAM access
- **Progressive** quantization on Key to further *trade more computation to less DRAM access*

![Diagram of Query and Key tokens]
Progressive Quantization

- **Separate** Key to LSB part and MSB part
**Progressive Quantization**

- **Separate** Key to LSB part and MSB part
- **Only fetch MSB** from DRAM to on-chip, and compute attention probability

![Diagram of Progressive Quantization]

**Diagram Details:**
- Query and Key have 11 tokens each.
- LSB and MSB parts are separated.
- Only the MSB is fetched from DRAM to on-chip.
- The attention score is computed row-wise.
- Softmax is applied to obtain attention probability.
Progressive Quantization

- **Separate** Key to LSB part and MSB part
- **Only fetch MSB** from DRAM to on-chip, and compute attention probability
- If attention probability has **low error**:

No need to fetch LSB
**Progressive Quantization**

- **Separate** Key to LSB part and MSB part
- **Only fetch MSB** from DRAM to on-chip, and compute attention probability
- If attention probability has **high error:**
  
  ![Diagram showing attention computation]

Need to fetch LSB and recompute attention probability

- **Eagerly** fetch MSB, **lazily** fetch LSB: reduce the average bitwidth
Determine Attention Error with Attention Probability

- How to check whether error is high?
Determine Attention Error with Attention Probability

- How to check whether error is high?
- The larger the max attention probability, the smaller the attention error
Determine Attention Error with Attention Probability

- How to check whether error is high?
- The larger the max attention probability, the smaller the attention error

- Therefore, check the max attention prob, if large, not need fetch LSB.
Outline

• Quick Overview
• Background
• Algorithmic Optimizations
• Hardware Architecture
• Evaluation
• Conclusion
Our Solution: SpAtten

- Cascade Token Pruning
- Cascade Head Pruning
- Local Value Pruning
- Progressive Quantization

Top-K Engine

Dedicated Accelerator
Our Solution: SpAtten

- Cascade Token Pruning
- Cascade Head Pruning
- Local Value Pruning
- Progressive Quantization

- Top-K Engine
- Dedicated Accelerator
Top-k Engine

• Top-k engine supports cascade token/head pruning and local value pruning

• Example: find top 4 elements from [7, 2, 6, 4, 9, 1, 3, 5]
  • Find the 4th largest: 5
  • Filter the input, preserve those ≥ 4th largest: [7, 6, 9, 5]
Top-k Engine

- Top-k engine supports cascade token/head pruning and local value pruning
- Example: find top 4 elements from [7, 2, 6, 4, 9, 1, 3, 5]
Top-k Engine

- Top-k engine supports cascade token/head pruning and local value pruning
- Example: find top 4 elements from [7, 2, 6, 4, 9, 1, 3, 5]
Top-k Engine

- Top-k engine supports cascade token/head pruning and local value pruning
- Example: find top 4 elements from $[7, 2, 6, 4, 9, 1, 3, 5]$
Top-k Engine

- Top-k engine supports cascade token/head pruning and local value pruning
- Example: find top 4 elements from [7, 2, 6, 4, 9, 1, 3, 5]
Top-k Engine

- Top-k engine supports cascade token/head pruning and local value pruning
- Example: find top 4 elements from [7, 2, 6, 4, 9, 1, 3, 5]
Top-k Engine

- Top-k engine supports cascade token/head pruning and local value pruning
- Example: find top 4 elements from [7, 2, 6, 4, 9, 1, 3, 5]
Quick Select

- Quick Select module selects the $k^{th}$ largest element
- Example: selects the 4th largest from [7, 2, 6, 4, 9, 1, 3, 5]
Quick Select

- Top-k Engine has **high-parallelism**
- 16 '<' comparators and 16 '>' comparators in Quick Select
- Compare the elements with pivot **in parallel**
Our Solution: SpAtten

Cascade Token Pruning

Cascade Head Pruning

Local Value Pruning

Progressive Quantization

Top-K Engine

Dedicated Accelerator
Dedicated Accelerator

- Pipelined architecture to improve the throughput
Dedicated Accelerator

- The **token importance score accumulator module** stores scores across layers
- Initialized to all-zero in the first layer

Token Importance Score Accumulator

- Attention Prob
- SpAtten: Efficient Sparse Attention Architecture
- HBM (16 Channels)
- Address token importance score accumulator module
- Xbar 32 × 16
- FIFO

- Data
- Fetcher
- Bitwidth Convertor
- Key ids
- K2
- K3
- K2
- K1
- K2
- K3
- K1
- K2
- K0
- K1
- K0

- Score Accumulator
- Top-k for Cascade
- Adder
- Tree

- sentence_length
- attention_prob
- sentence_length
- attention_prob
- sentence_length
- attention_prob

- V5
- V2
- V0
- V6
- V2
- V9
- V5
- V2
- V6
- V6

- weighted sum
- V
- V
- V
- V

- Accumulate vertically
Dedicated Accelerator

- Pipelined architecture to improve the throughput
Dedicated Accelerator
Dedicated Accelerator
• The consecutive Key/Value vectors are stored in **different HBM channels** to leverage HBM bandwidth.
Dedicated Accelerator

HBM (16 Channels)

SpAtten: Efficient Sparse Attention Architecture
Dedicated Accelerator

HBM (16 Channels)

Data

FIFO × 32

Xbar 16×32

Q, K, V Fetcher

Address

FIFO × 32

Xbar 32×16

Bitwidth Converter

Key SRAM (196KB)

8×64×12b

64×12b

Token Importance

Score Accumulator

Top-k for Cascade

Token Pruning

Remained Key ids

Attention

Progressive Quantization

normailization

Attention_prob

sentence_length

Pruning

Top-k

Local

Adder

matrix

Tree

Score Accumulator

Token Importance

Attention_out
Dedicated Accelerator
Dedicated Accelerator
Dedicated Accelerator

SpAtten: Efficient Sparse Attention Architecture
Dedicated Accelerator

HBM (16 Channels)

FIFO × 32

Bitwidth Converter

Q, K, V Fetcher

Score Accumulator

Softmax

Top-k for Cascade

Need LSB?

Progressive Quantization

LSBs

Need LSB?

Attention Prob

Attention_out

Attention_out

{sentence_length} × 12b

attention_prob

Sentence_length

V11

V0

V5

V6

V9
Dedicated Accelerator
Dedicated Accelerator
Dedicated Accelerator
Dedicated Accelerator
Outline

- Quick Overview
- Background
- Algorithmic Optimizations
- Hardware Architecture
- Evaluation
- Conclusion
Evaluation

- Hardware Implementation

<table>
<thead>
<tr>
<th>Technology</th>
<th>TSMC 40nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area (w/o DRAM)</td>
<td>18.71mm²</td>
</tr>
<tr>
<td>Power (w/ DRAM)</td>
<td>8.30W</td>
</tr>
<tr>
<td>Multipliers</td>
<td>1024</td>
</tr>
<tr>
<td>SRAM</td>
<td>392KB</td>
</tr>
<tr>
<td>DRAM</td>
<td>HBM2 16 Channels, each @ 32GB/s</td>
</tr>
<tr>
<td>Performance on Summarization Stage</td>
<td>1.61TFLOPS</td>
</tr>
<tr>
<td>Performance on Generation Stage</td>
<td>0.43TFLOPS</td>
</tr>
</tbody>
</table>
Power & Area Breakdown

- On-chip power and area breakdown

(a) Area

- Attn_Prob×V: 38.6%
- Top-k: 2.7%
- Softmax: 4.2%
- Q×K: 38.1%
- QKV Fetcher: 14.2%
- Others: 2%
- Others: 2%
- Attn_Prob × V: 38.6%
- Top-k: 2.7%
- Softmax: 4.2%
- Q×K: 38.1%
- QKV Fetcher: 14.2%
- Others: 2%
- Others: 2%
Power & Area Breakdown

- On-chip power and area breakdown

(a) Area

- Q×K: 38.1%
- Softmax: 4.2%
- Top-k: 2.7%
- Attn_Prob×V: 38.6%

(b) Power

- Q×K: 43.4%
- QKV Fetcher: 9.4%
- Others: 5%
- Top-k: 3.1%
- Attn_Prob×V: 20.4%
- Others: 2%
30 Benchmarks

• Model Architecture
  • BERT-Base, BERT-Large, GPT-2-Small, GPT-2-Medium

• Task:
  • Discriminative: GLUE (9 tasks), SQuAD-V1, SQuAD-V2
  • Generative: Language modeling on Wikitext-103, Wikitext-2, Pen-tree Bank, Google One Billion Words
Pruning and Quantization Results

- Pruning ratio is **highly dependent** on sentence length
  - For long sentence (1000 in LM with GPT-2), can prune more than **70%** tokens
  - For short sentence (12 in CoLA with BERT), can prune less than **15%** tokens
- Under same performance (<2% loss on several BERT models), 30 benchmarks average:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascade Token Pruning</td>
<td>Prune away 35% (ranging 10%~75%)</td>
<td></td>
</tr>
<tr>
<td>Cascade Head Pruning</td>
<td>Prune away 10%</td>
<td></td>
</tr>
<tr>
<td>Progressive Quantization</td>
<td>Effective Bitwidth</td>
<td>7.8 bits</td>
</tr>
<tr>
<td>(on top of cascade token pruning)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Local Value Pruning</td>
<td>Prune away 63% (Value)</td>
<td></td>
</tr>
</tbody>
</table>
Performance Comparisons

- Over general-purpose CPUs/GPUs on attention layers
- SpAtten applies all algorithmic optimizations
- 30 benchmarks average

![Graph showing speedup and energy efficiency comparisons between TITAN Xp GPU and Xeon CPU.](image)
Performance Comparisons

- Over general-purpose CPUs/GPUs on attention layers
  - SpAtten applies all algorithmic optimizations
  - 30 benchmarks average

![Graph showing performance comparisons over different platforms and devices.](image)
Performance Comparisons

- Over state-of-the-art attention accelerators
  - A3 (ASIC) supports local key/value pruning
  - MNNFast (FPGA) supports local value pruning

Bar chart:
- Speedup: 1.6 (Over A3 Accelerator)
- Energy Efficiency: 1.4 (Over A3 Accelerator)
- Area Efficiency: 2.2 (Over A3 Accelerator)
Performance Comparisons

- Over state-of-the-art attention accelerators
  - A3 (ASIC) supports local key/value pruning
  - MNNFast (FPGA) supports local value pruning
Speedup Breakdown

- Speedup breakdown on GPT-2 Models over TITAN Xp GPU
**Speedup Breakdown**

- Speedup breakdown on GPT-2 Models over TITAN Xp GPU

![Speedup Breakdown Graph]

- **22.1× speedup** with specialized datapath (ASIC)

**Table 2**

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>240</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Table 2-1**

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>240</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Table 2-2**

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>240</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Table 2-3**

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>240</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td>0.0</td>
</tr>
</tbody>
</table>
**Speedup Breakdown**

- Speedup breakdown on GPT-2 Models over TITAN Xp GPU

- **22.1× speedup** with specialized datapath (ASIC)

- **3.4× speedup** with cascade token & head & value pruning
SpAtten: Efficient Sparse Attention Architecture

Speedup Breakdown

- Speedup breakdown on GPT-2 Models over TITAN Xp GPU

Table 2

<table>
<thead>
<tr>
<th>Data Size (MB)</th>
<th>Speedup with Specialized Datapath (ASIC)</th>
<th>Speedup with Cascade Token &amp; Head &amp; Value Pruning</th>
<th>Speedup with Progressive Quantization</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>22.1×</td>
<td>3.4×</td>
<td>2.8×</td>
</tr>
<tr>
<td>120</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 1 2 3 0 60 120 180 240

- 22.1× speedup with specialized datapath (ASIC)
- 3.4× speedup with cascade token & head & value pruning
- 2.8× speedup with progressive quantization
More Examples

- BERT sentence classification: (Film sentiment classification result: positive)

A wonderful movie, I am sure that you will remember it, you admire its conception and are able to resolve some of the confusions you had while watching it.
More Examples

• BERT sentence classification: (Film sentiment classification result: positive)

A wonderful movie, I am sure that you will remember it, you admire its conception and are able to resolve some of the confusions you had while watching it.
More Examples

- GPT-2 for language modeling: (‘English’ is the generated token.)

Du Fu was a great poet of the Tang dynasty. Recently a variety of styles have been used in efforts to translate the work of Du Fu into English.
Du Fu was a great poet of the Tang dynasty. Recently a variety of styles have been used in efforts to translate the work of Du Fu into English.

Du translate into English
Specialized Model for SpAtten with HAT: Hardware-Aware Transformer NAS

- HAT: Hardware-Aware Transformer NAS is a framework to search for most suitable model for a target hardware

Specialized Model for SpAtten with HAT: Hardware-Aware Transformer NAS

- HAT: Hardware-Aware Transformer NAS is a framework to search for most suitable model for a target hardware

- The co-designed specialized model for SpAtten can be **1.9x faster** than vanilla model

Specialized Model for SpAtten with HAT: Hardware-Aware Transformer Transformer NAS

- HAT: Hardware-Aware Transformer NAS is a framework to search for most suitable model for a target hardware
- Computation breakdown


SpAtten: Efficient Sparse Attention Architecture
Specialized Model for SpAtten with HAT: Hardware-Aware Transformer NAS

- HAT: Hardware-Aware Transformer NAS is a framework to search for the most suitable model for a target hardware.
- Computation breakdown

```
<table>
<thead>
<tr>
<th>Model</th>
<th>GFLOPs</th>
<th>Attention FLOPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vanilla Transformer-Base</td>
<td>2.7G</td>
<td>28.9M</td>
</tr>
<tr>
<td>Co-designed Transformer</td>
<td>1.9G</td>
<td>30.5M</td>
</tr>
<tr>
<td>For SpAtten</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

SpAtten: Efficient Sparse Attention Architecture

**Specialized Model for SpAtten with HAT: Hardware-Aware Transformer NAS**

- HAT: Hardware-Aware Transformer NAS is a framework to search for the most suitable model for a target hardware.
- Computation breakdown

![Computation breakdown diagram](image)

- SpAtten is good at processing attention layers

Outline

• Quick Overview
• Background
• Algorithmic Optimizations
• Hardware Architecture
• Evaluation
• Conclusion
SpAtten: Sparse Attention Architecture

Pushing the frontier of Green AI and Tiny AI

- SpAtten accelerates NLP by removing human language redundancy
  - Cascade token/head pruning
  - Local value pruning
  - Progressive quantization
- Hardware accelerator
  - High-Parallelism Top-k engine
  - Specialized data path and operators

HPCA 2021 Live Q&A:
**Session 1B**
Mon. March 1, 10:40 EST

Thank you!