A Slice and Dice Approach to Accelerate Compound Sparse Attention on GPU

Hailong Li, Jaewan Choi, Jung Ho Ahn

Seoul National University

Presenter: Jaewan Choi (cjw9202@snu.ac.kr)

A Slice and Dice Approach to Accelerate **Compound Sparse Attention** on GPU

What is our target workload?

- Transformers based on the compound-sparse attention
- Long sequence tasks with more than 2,048 tokens

A Slice and Dice Approach to Accelerate Compound Sparse Attention on GPU

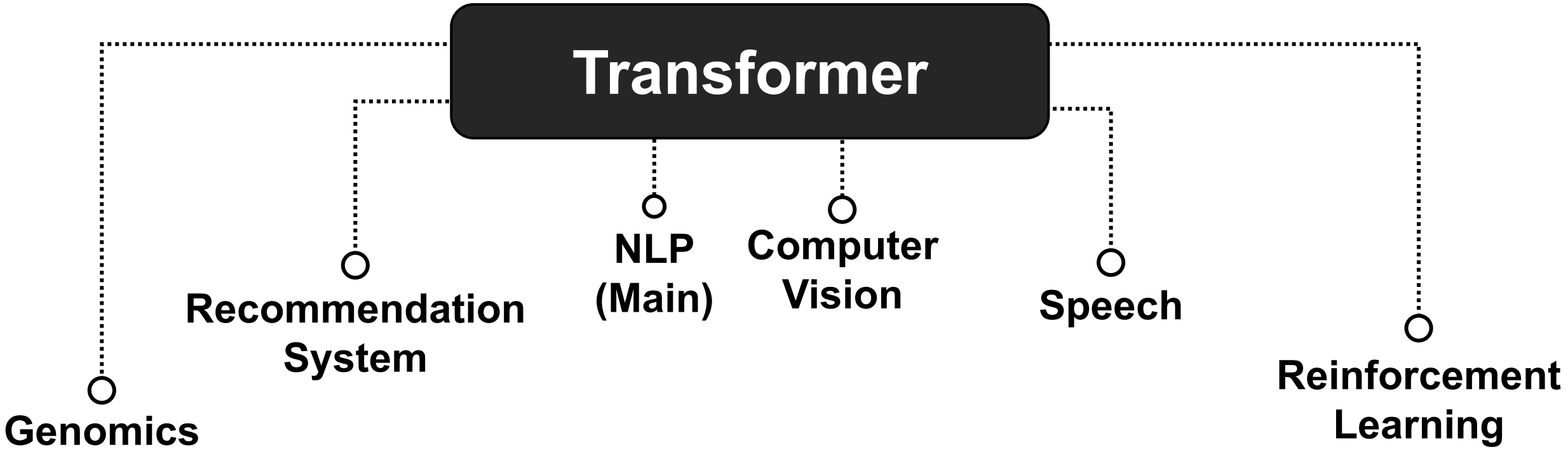
What is our target workload?

- Transformers based on the compound-sparse attention
- Long sequence tasks with more than 2,048 tokens

How to accelerate the target workload?

- Multigrain applying a slice and dice approach

“Transformer” is all you need!



NLP with a long sequence

- Why long sequence NLP?

NLP with a long sequence

- Why long sequence NLP?
  - Many practical NLP problems require processing long sequences.
    - = Scientific literature (typical document is 1K-10K words or longer.)
    - = Digital humanities (books can have 100,000 words or more: *Harry Potter and the Deathly Hallows* is about 200K words.)

NLP with a long sequence

- Why long sequence NLP?
  - Many practical NLP problems require processing long sequences.
    - = Scientific literature (typical document is 1K-10K words or longer.)
    - = Digital humanities (books can have 100,000 words or more: *Harry Potter and the Deathly Hallows* is about 200K words.)
- What are the key challenges for long sequence NLP?

NLP with a long sequence

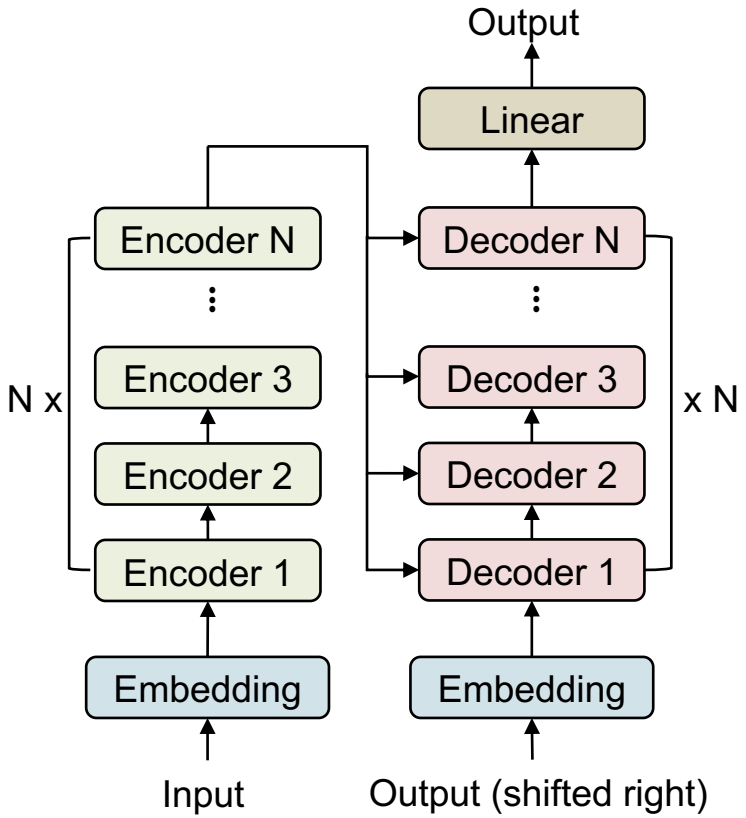
- Why long sequence NLP?
  - Many practical NLP problems require processing long sequences.
    - = Scientific literature (typical document is 1K-10K words or longer.)
    - = Digital humanities (books can have 100,000 words or more: *Harry Potter and the Deathly Hallows* is about 200K words.)
- What are the key challenges for long sequence NLP?
  - Tasks often require combining information spread over long distances, either in document, or among many documents.

NLP with a long sequence

- Why long sequence NLP?
  - Many practical NLP problems require processing long sequences.
    - = Scientific literature (typical document is 1K-10K words or longer.)
    - = Digital humanities (books can have 100,000 words or more: *Harry Potter and the Deathly Hallows* is about 200K words.)
- What are the key challenges for long sequence NLP?
  - Tasks often require combining information spread over long distances, either in document, or among many documents.
  - Early-stage transformer models have limitations in long sequence setting. (e.g., latency and memory issues)
    - = Multi-head attention has $O(L^2) \rightarrow$ cannot process long input with current hardware.
      (e.g., if the sequence length L is 4096, BERT-large requires a memory size of 64GB.)

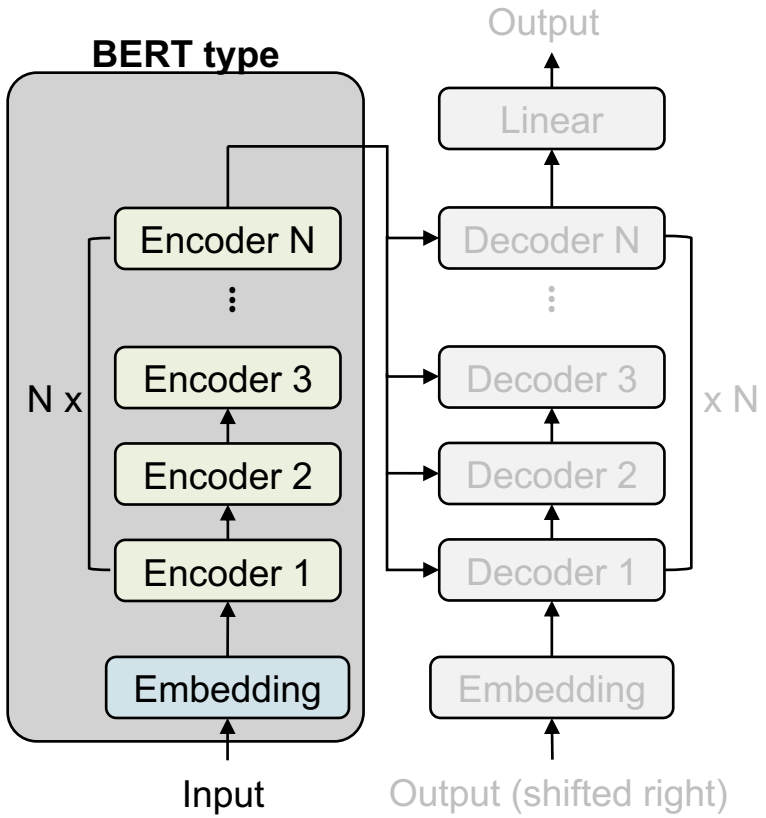
Transformer architecture

- Transformer consists of multiple encoders and decoders.



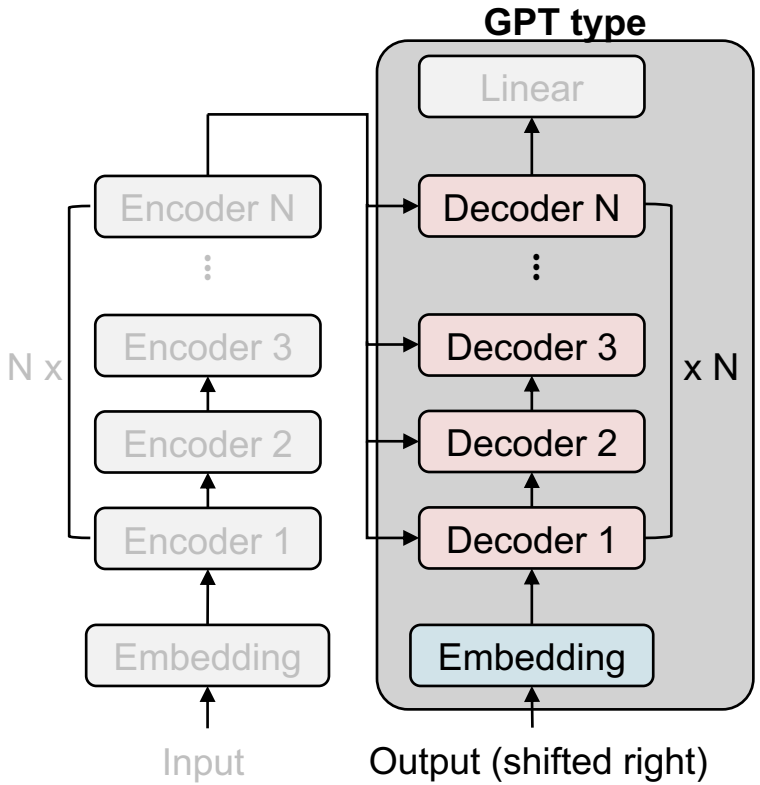
Transformer architecture

- Transformer consists of multiple stacked encoders and decoders.
  - BERT type stacks only encoders. (use case: Natural Language Understanding)



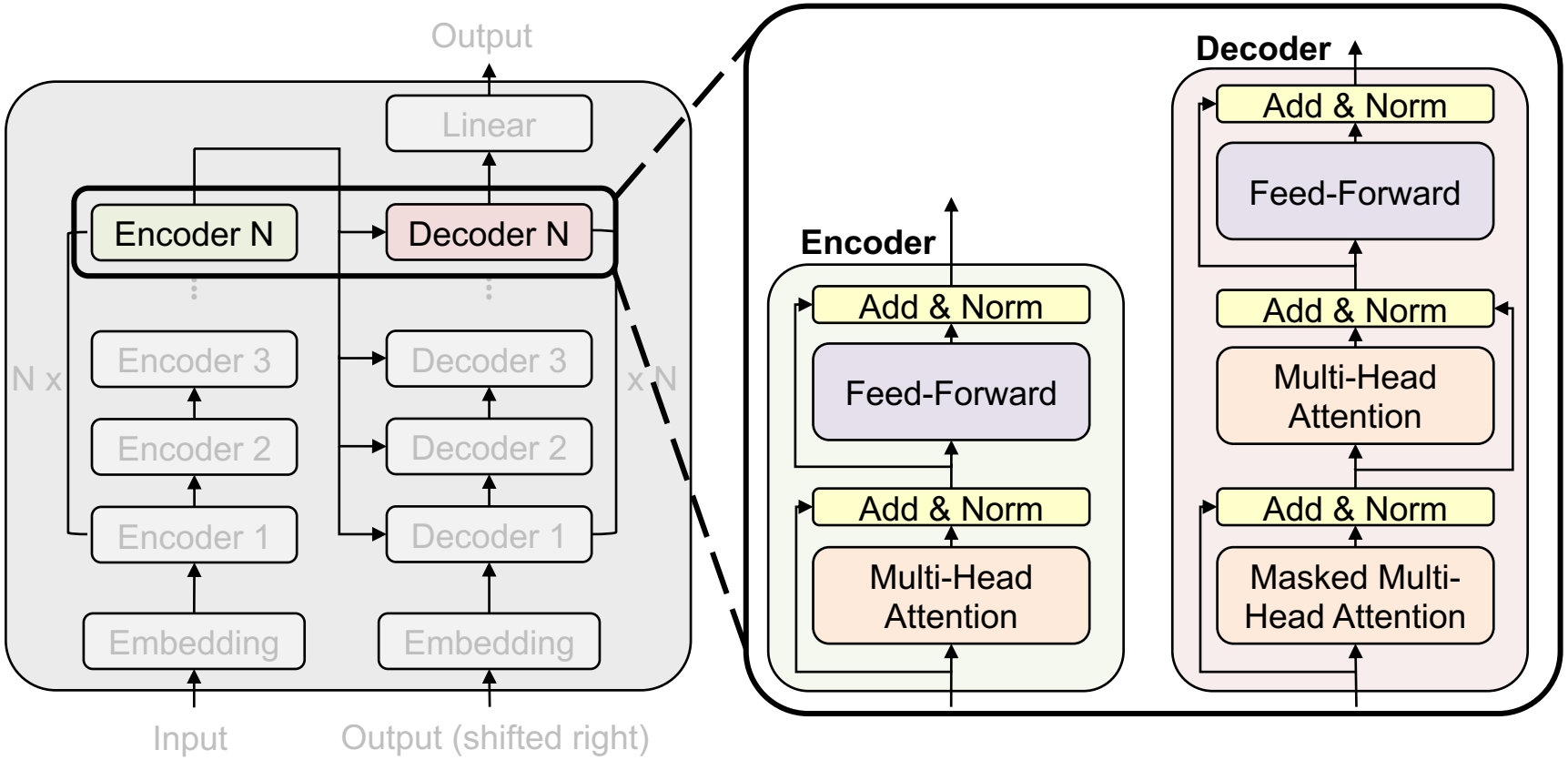
Transformer architecture

- Transformer consists of multiple stacked encoders and decoders.
  - BERT type stacks only encoders. (use case: Natural Language Understanding)
  - GPT type stacks only decoders. (use case: Natural Language Generation)



Transformer architecture

- Transformer consists of multiple stacked encoders and decoders.
- Both mainly perform Multi-Head Attention (MHA) and Feed-Forward (FF).

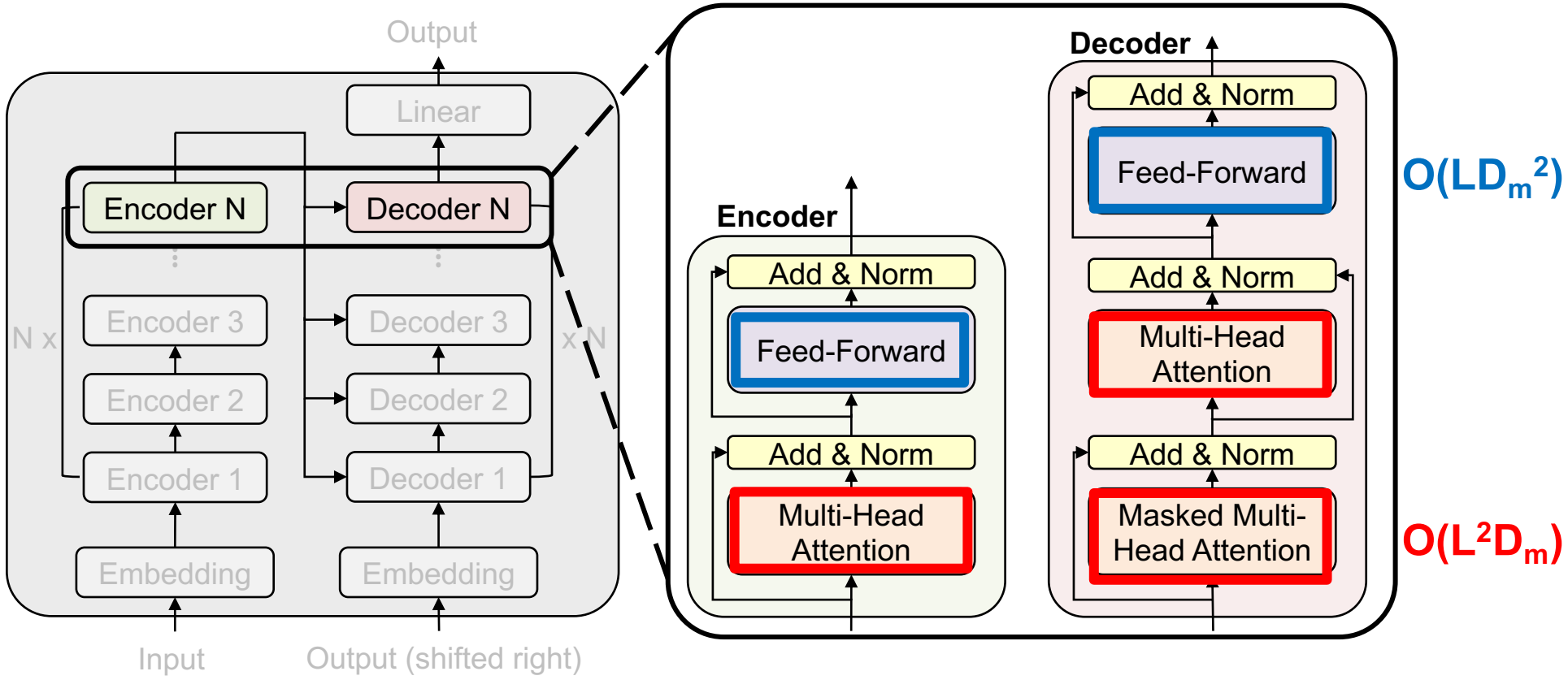


Transformer architecture

- Transformer consists of multiple stacked encoders and decoders.
- Both mainly perform Multi-Head Attention (MHA) and Feed-Forward (FF).

L: Seqeunce length

D_m: hidden dimension



Transformer architecture

- Transformer consists of multiple stacked encoders and decoders
- Both main operations are Multi-Head Attention (MHA) and Feed-Forward (FF)

L: Seqeunce length

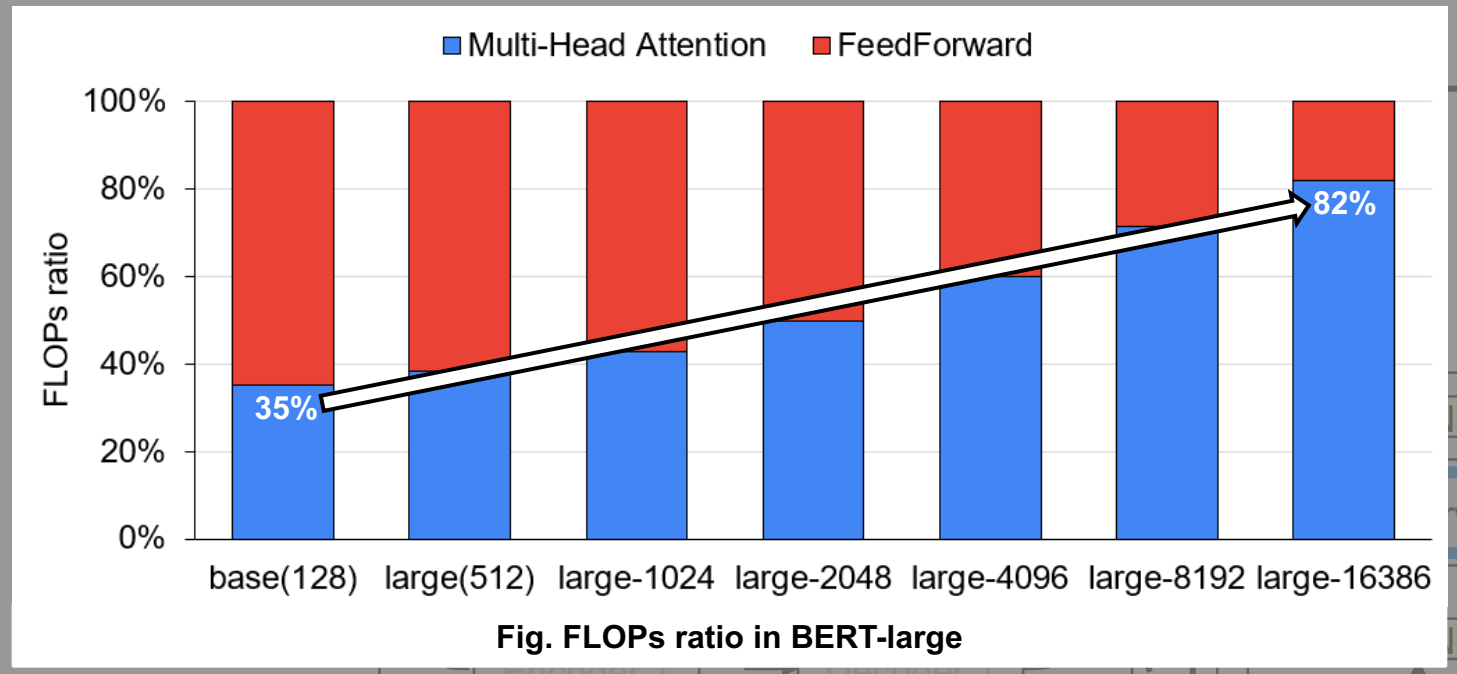


Fig. FLOPs ratio in BERT-large

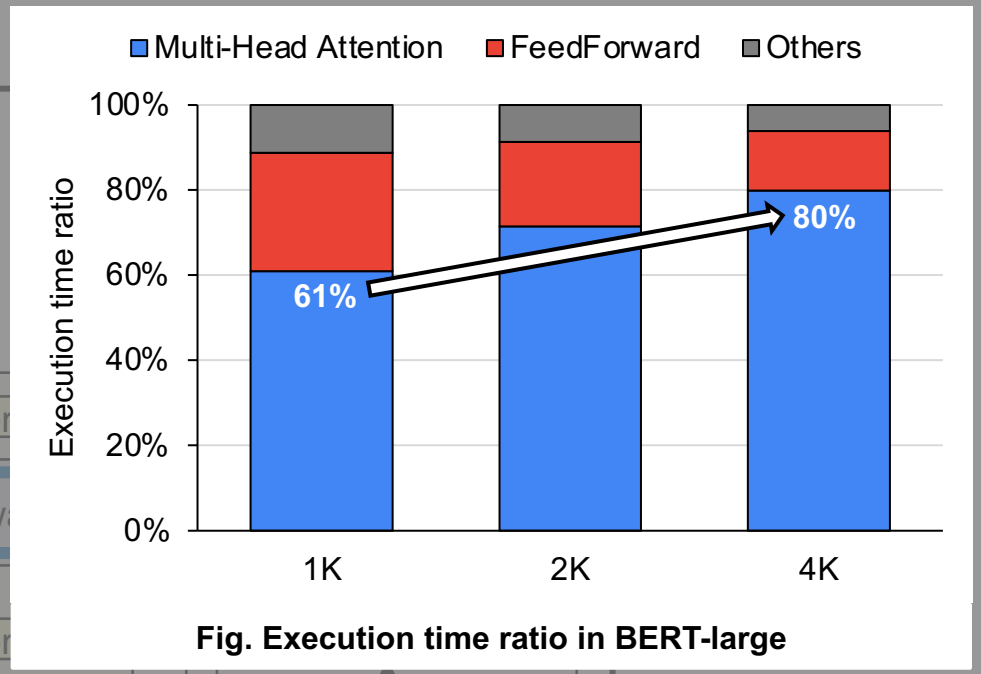
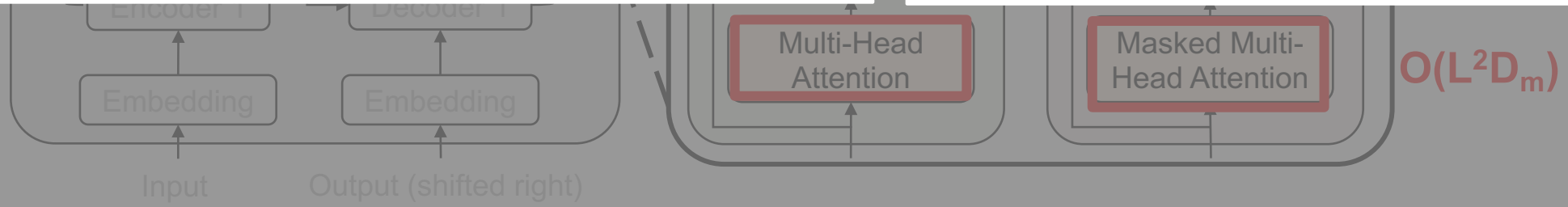


Fig. Execution time ratio in BERT-large



Transformer architecture

- Transformer consists of multiple stacked encoders and decoders
- Both main operations are Multi-Head Attention (MHA) and Feed-Forward (FF)

L: Seqeunce length

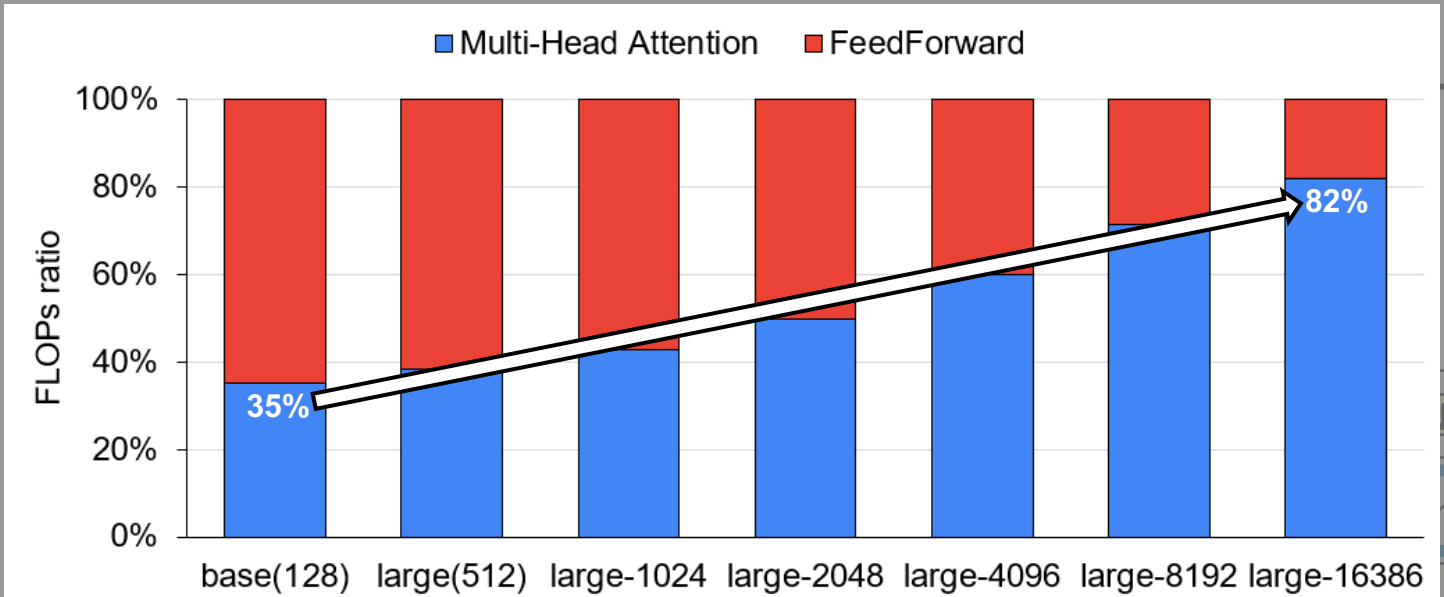


Fig. FLOPs ratio in BERT-large

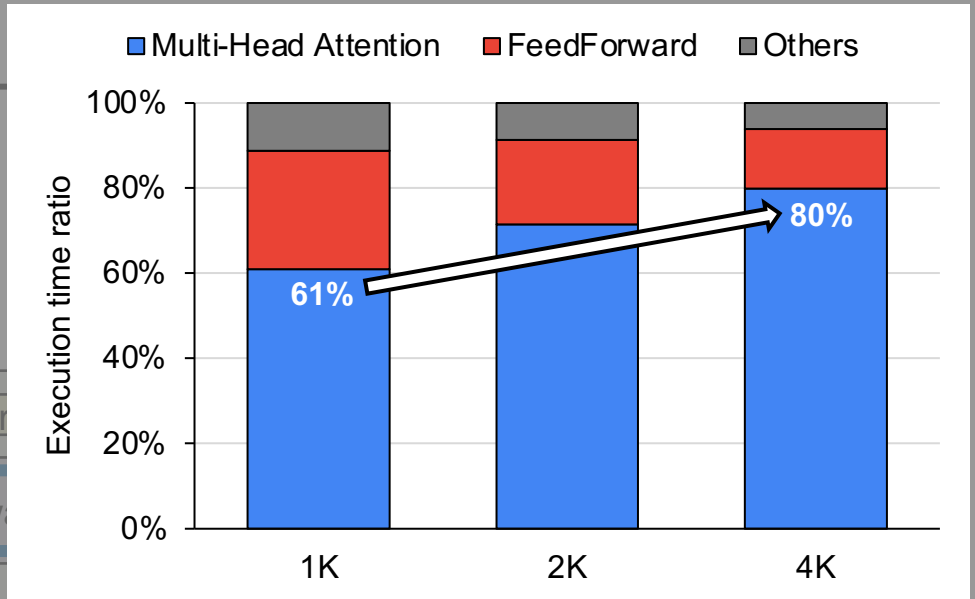
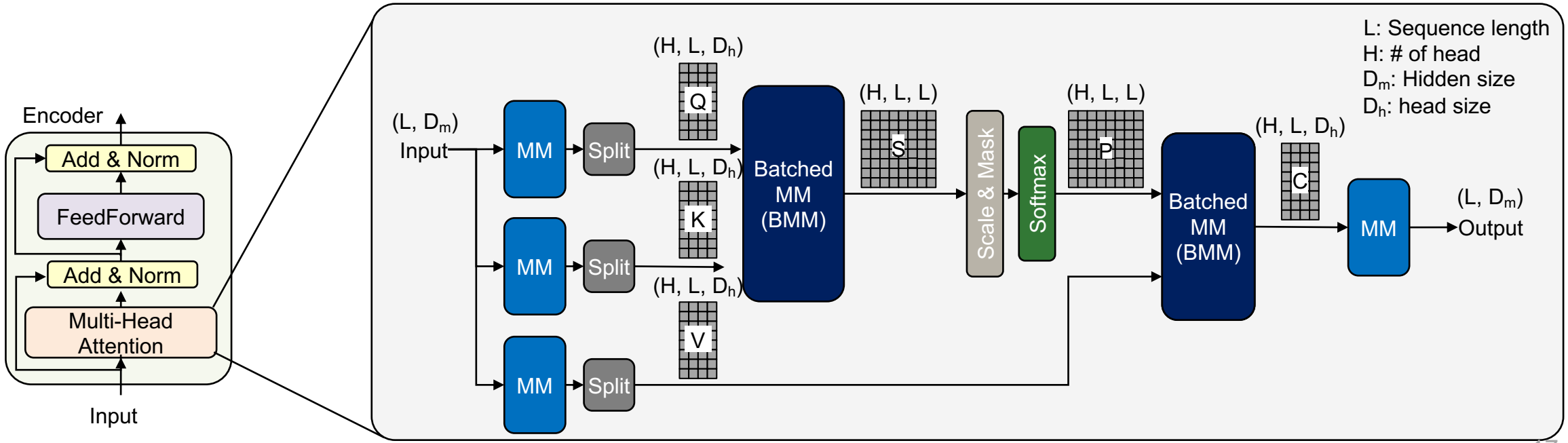


Fig. Execution time ratio in BERT-large

MHA becomes expensive when $L >> D_m$, computation and memory footprint are proportional to L^2!

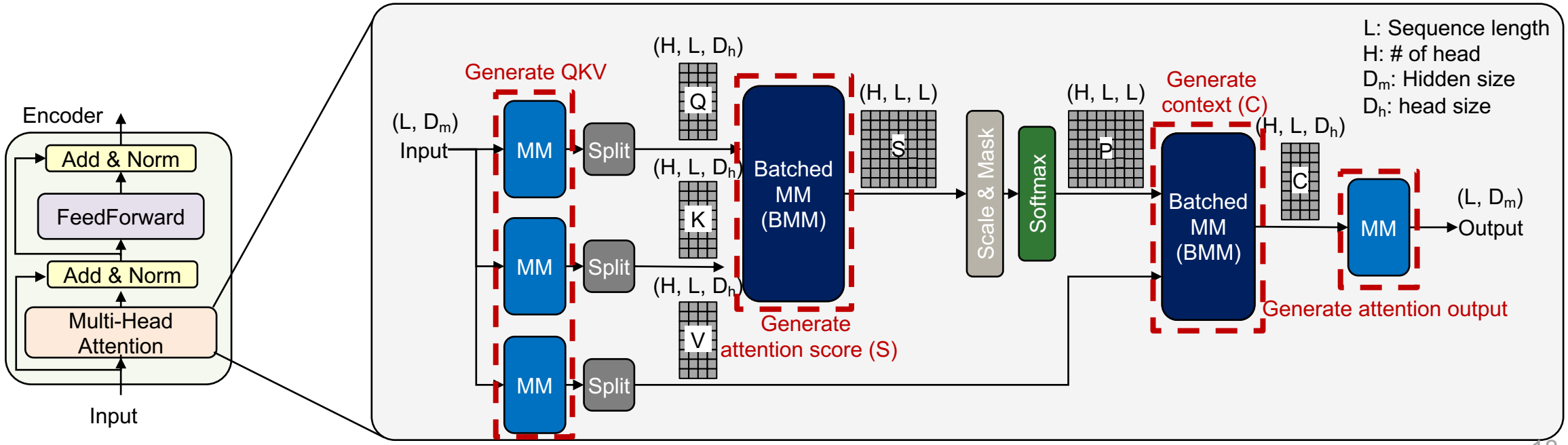
Multi-Head Attention

- MHA can be divided into MatMul (Matrix Multiplication) and Non-MatMul.



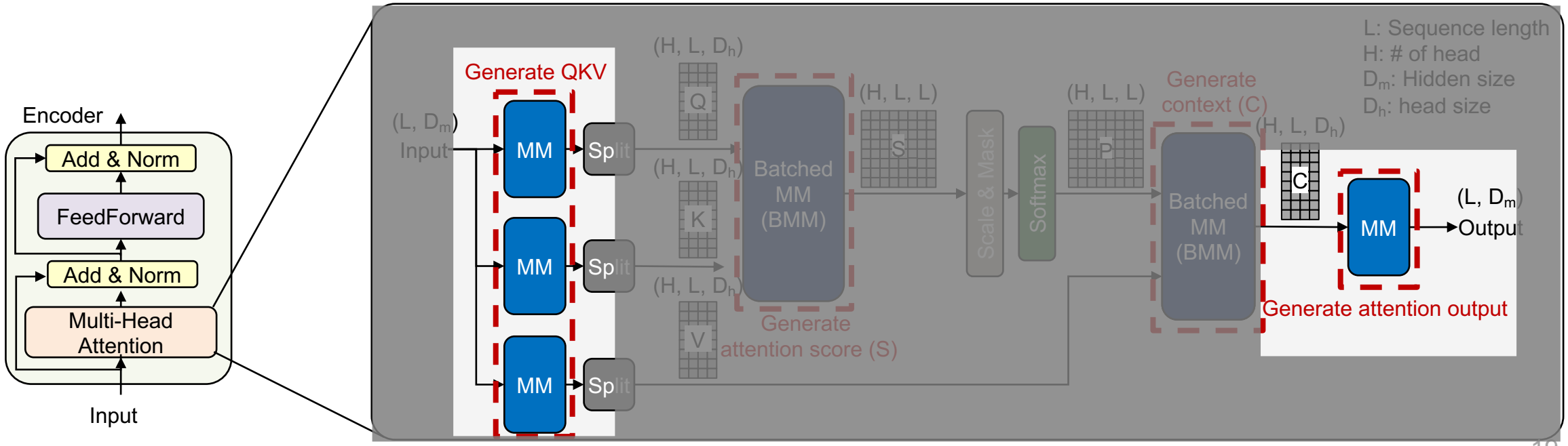
Multi-Head Attention

- MHA can be divided into MatMul (Matrix Multiplication) and Non-MatMul.
  - MatMul includes MM and batched MM (BMM).



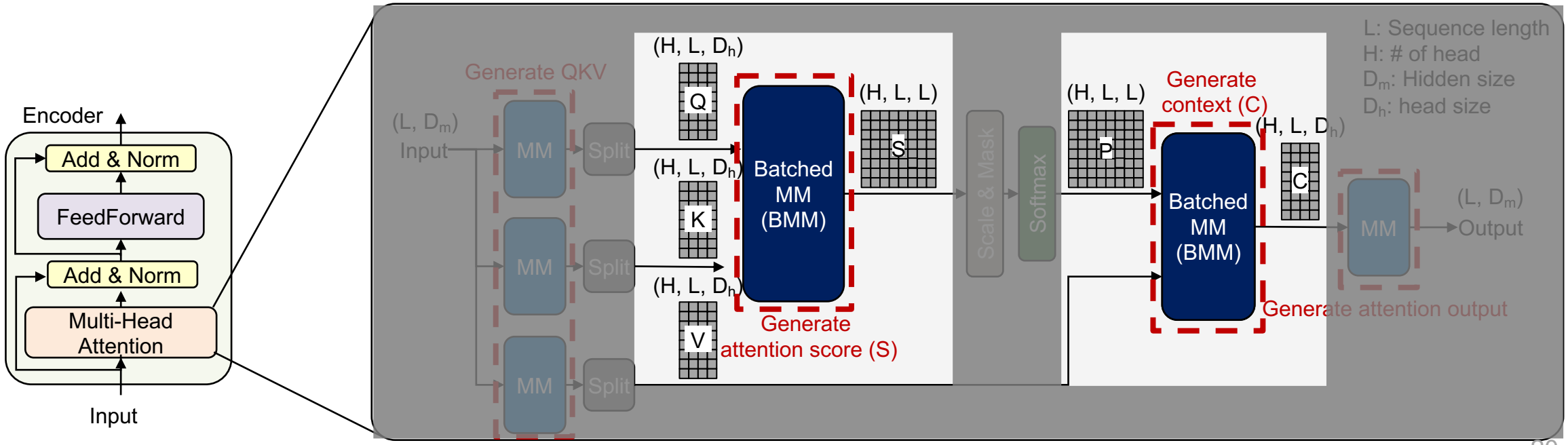
Multi-Head Attention

- MHA can be divided into MatMul (Matrix Multiplication) and Non-MatMul.
  - MatMul includes MM and batched MM (BMM).



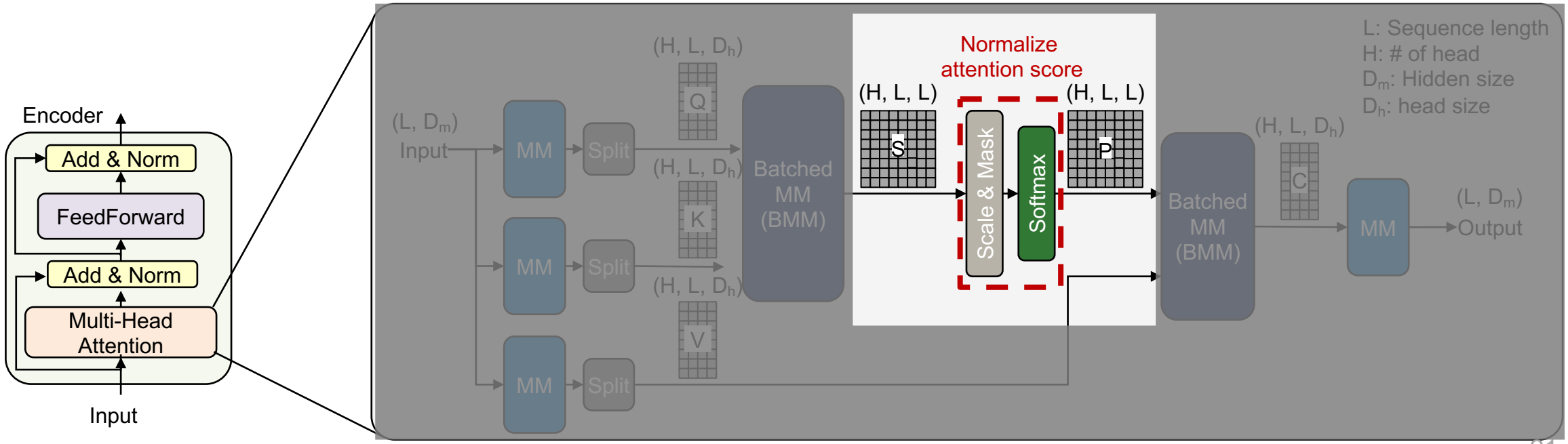
Multi-Head Attention

- MHA can be divided into MatMul (Matrix Multiplication) and Non-MatMul.
  - MatMul includes MM and batched MM (BMM).



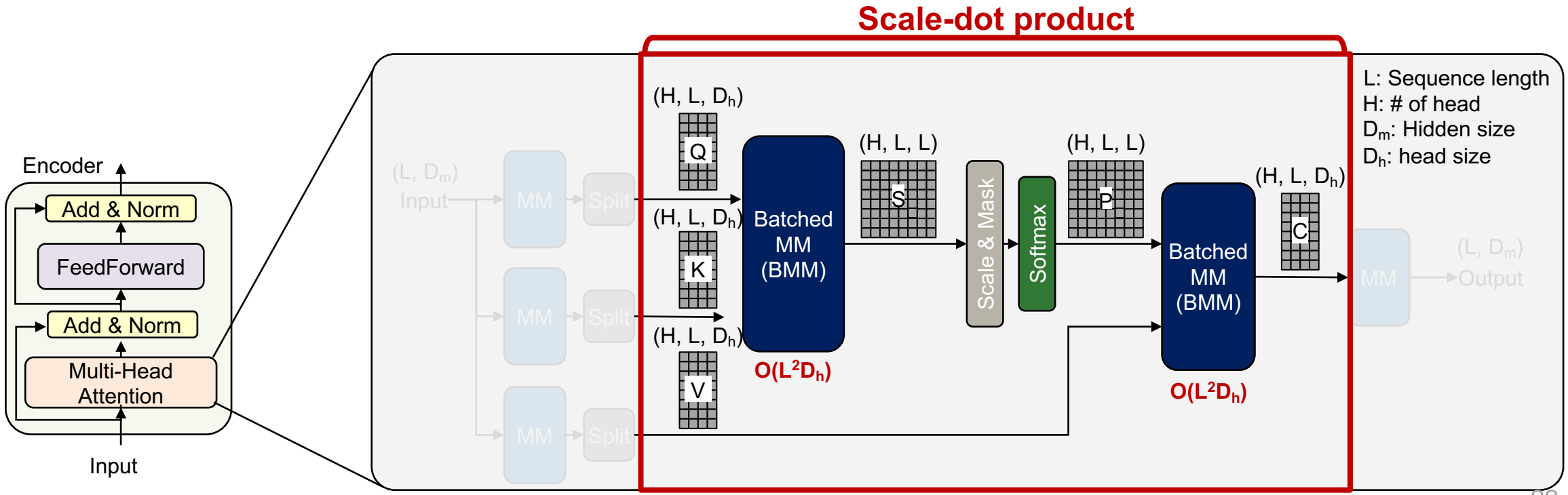
Multi-Head Attention

- MHA can be divided into MatMul (Matrix Multiplication) and Non-MatMul.
  - MatMul includes MM and batched MM (BMM).
  - Non-MatMul refers to scale, mask, and softmax.



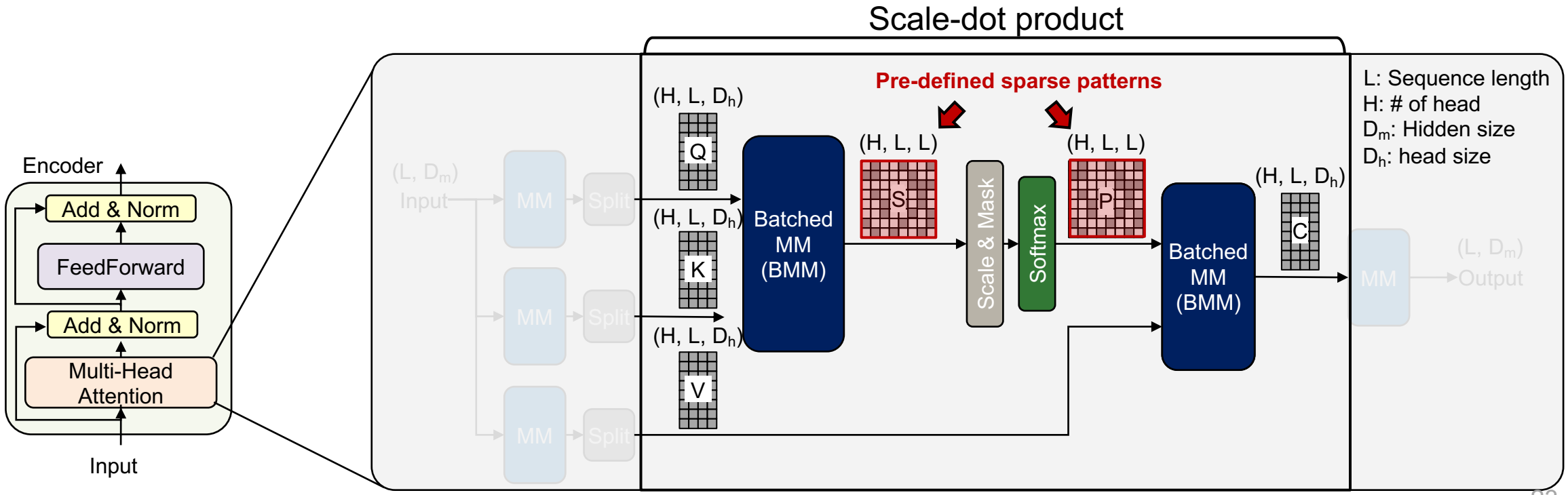
Multi-Head Attention

- MHA can be divided into MatMul (Matrix Multiplication) and Non-MatMul.
  - MatMul includes MM and batched MM (BMM).
  - Non-MatMul refers to scale, mask, and softmax.
- BMM and Non-MatMul are expensive as L increases.
  - e.g., S and P are 512MB (FP16), when L=4096, BERT-large.



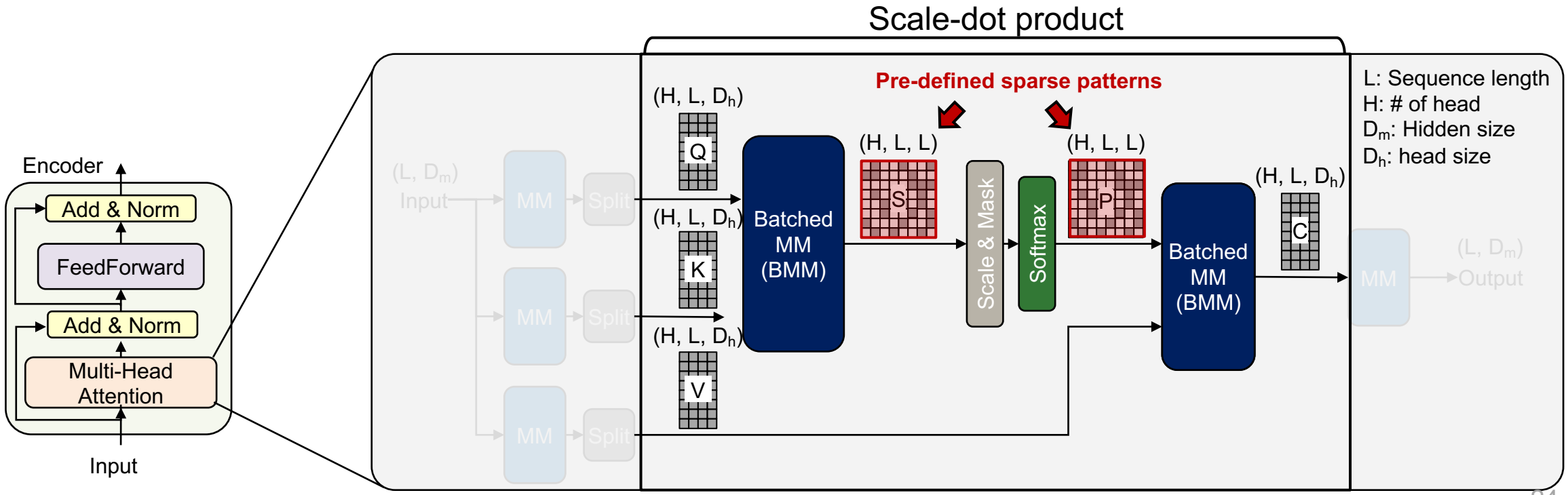
Sparse attention

- Sparse attention (SA) reduces computation time and the memory requirements of the scale-dot product operation by performing the sparse operations using a pre-defined sparse pattern.



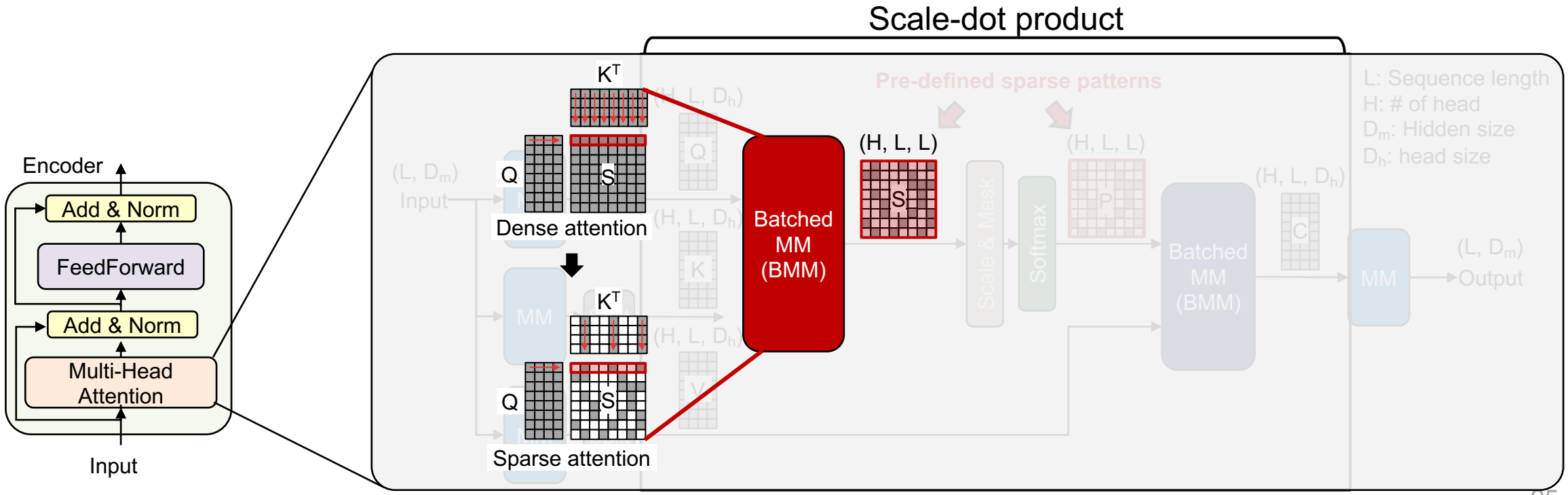
Sparse attention

- Sparse attention (SA) reduces computation time and the memory requirements of the scale-dot product operation by performing the sparse operations using a pre-defined sparse pattern.
  - The sparse pattern represents inductive biases, which reflect the additional assumption of language- and image-data features for accurate prediction.



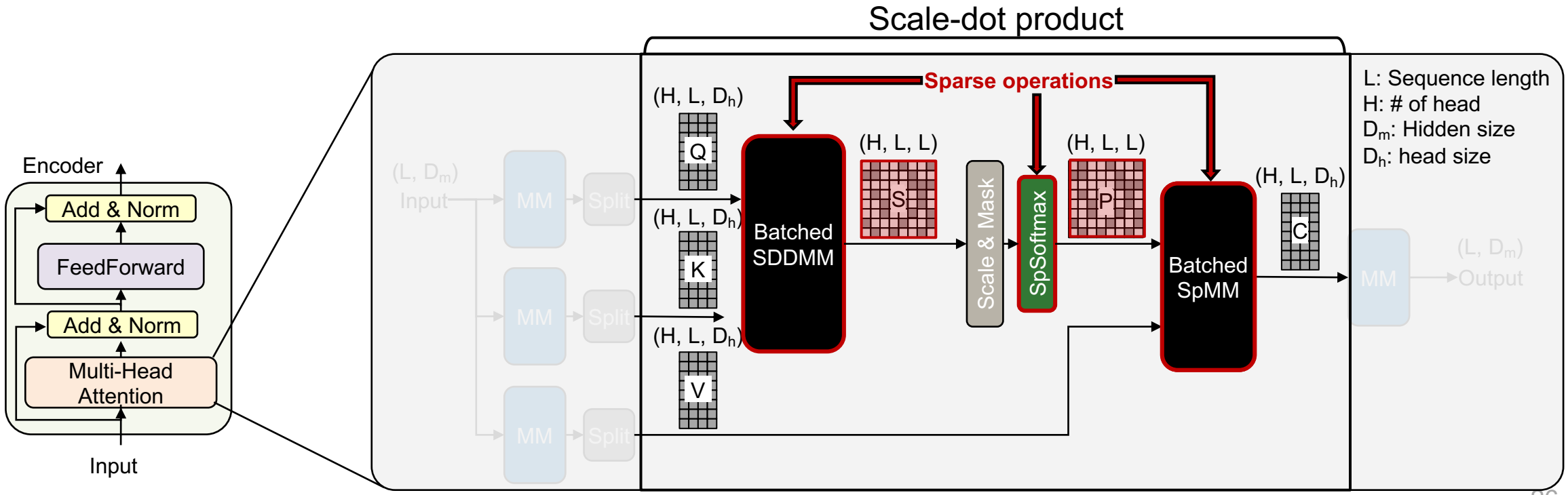
Sparse attention

- Sparse attention (SA) reduces computation time and the memory requirements of the scale-dot product operation by performing the sparse operations using a pre-defined sparse pattern.
  - The sparse pattern represents inductive biases, which reflect the additional assumption of language- and image-data features for accurate prediction.
  - Each sparse operation computes a limited selection of the sparse pattern, resulting in a sparse matrix rather than a full matrix.



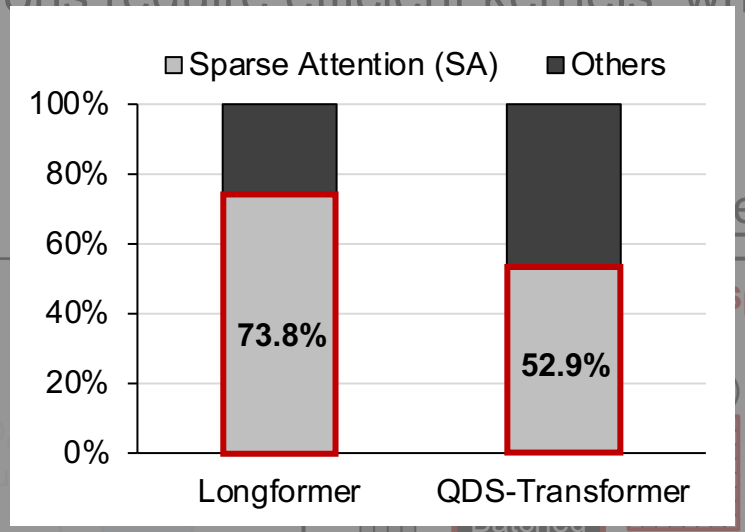
Sparse attention

- Sparse attention (SA) reduces computation time and the memory requirements of the scale-dot product operation by performing the sparse operations using a pre-defined sparse pattern.
  - The sparse pattern represents inductive biases, which reflect the additional assumption of language- and image-data features for accurate prediction.
  - Each sparse operation computes a limited selection of the sparse pattern, resulting in a sparse matrix rather than a full matrix.



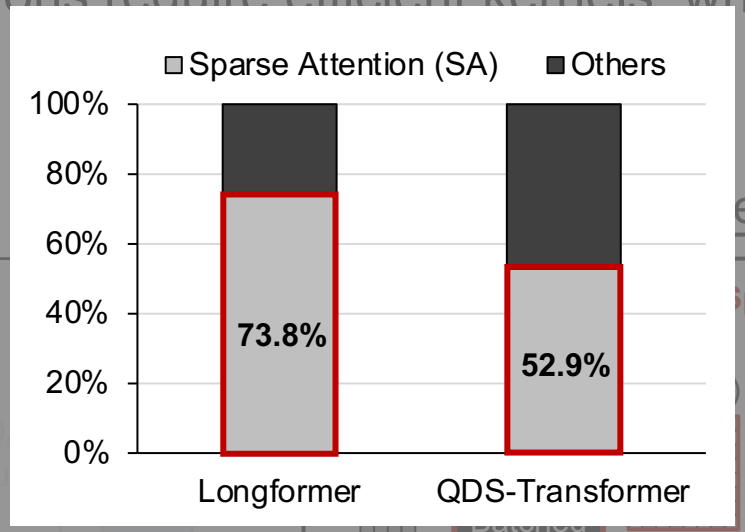
Sparse attention

- Sparse attention (SA) reduces computation time and the memory requirements of the scale-dot product operation by performing the sparse operations using a pre-defined sparse pattern.
- In the SA, the sparse operations require efficient kernels, which are available on the GPUs.



Sparse attention

- Sparse attention (SA) reduces computation time and the memory requirements of the scale-dot product operation by performing the sparse operations using a pre-defined sparse pattern.
- In the SA, the sparse operations require efficient kernels, which are available on the GPUs.



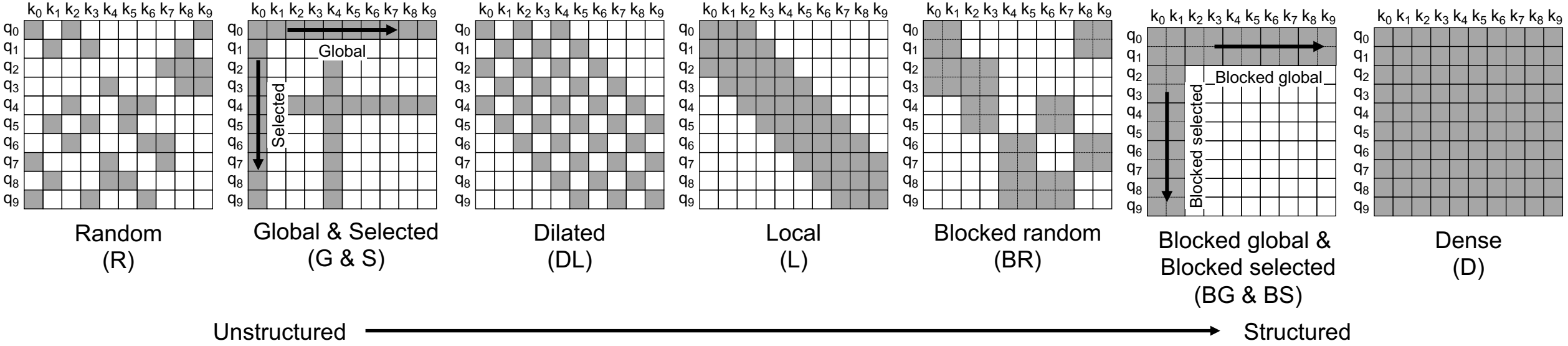
The SA is processed inefficiently on existing GPUs during inference because the locality of sparse patterns are not considered.

Sparse pattern

- Most of SA-based transformers show compound sparse patterns.

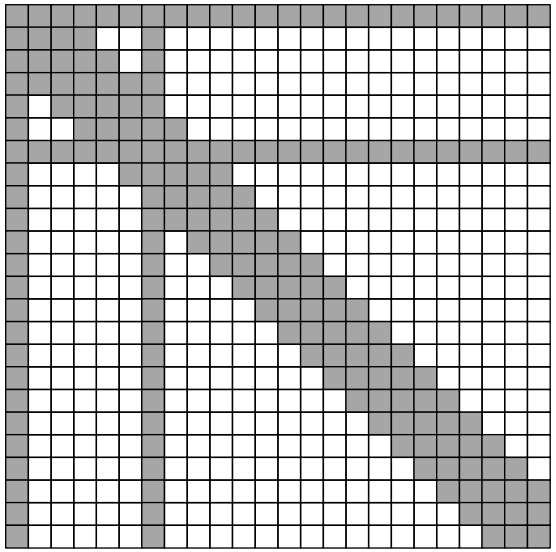
Sparse pattern

- Most of SA-based transformers show compound sparse patterns.
  - A compound sparse pattern combines multiple atomic sparse patterns.

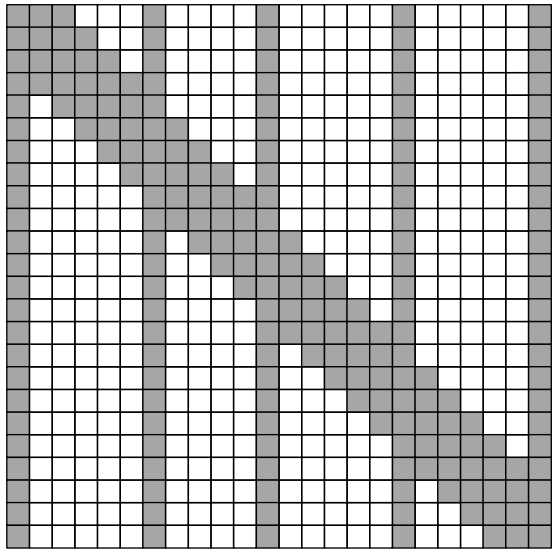


Sparse pattern

- Most of SA-based transformers show compound sparse patterns.
  - A compound sparse pattern combines multiple atomic sparse patterns.
  - e.g., Longformer (**local (L),** global (G) & selected (S))
    QDS-Transformer (**local (L),** selected (S))



Longformer (L + S + G)



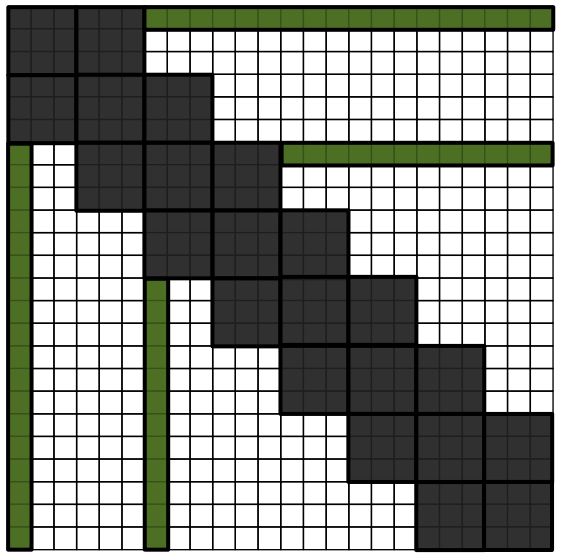
QDS-Transformer (L + S)

Existing methods

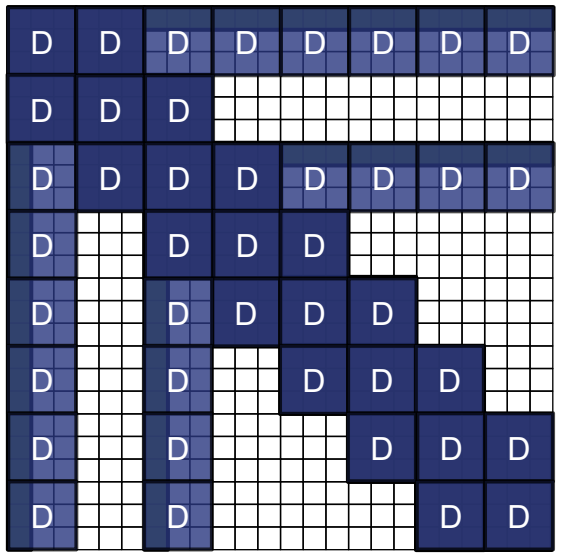
- Sparse operations are processed by either coarse-grained methods or fine-grained methods.
  - The coarse-grained methods use blocked sparse format.
    (e.g., block coordinate format (BCOO) and block compressed row format (BSR))
  - The fine-grained methods use element-wise sparse format.
    (e.g., compressed sparse row format (CSR), coordinate format (COO))

Coarse-grained method

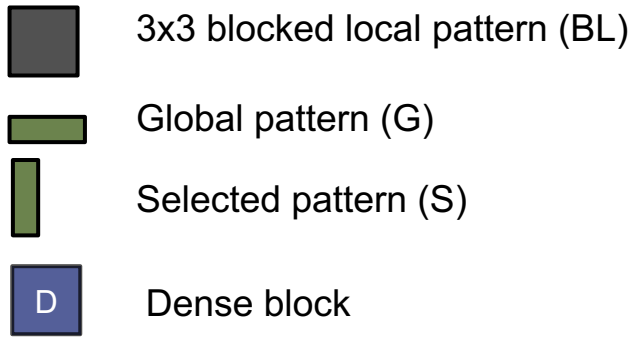
- Coarse-grained methods use blocked sparse format treating each nonzero block as a dense block.



Sparse pattern (BL + S + G)

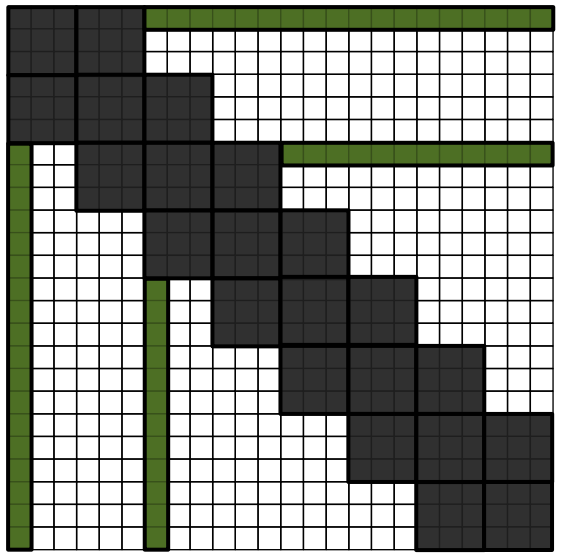


Coarse-grained kernel

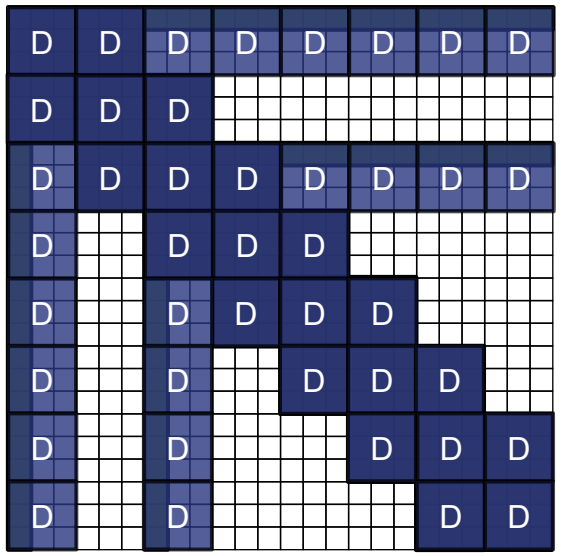


Coarse-grained method

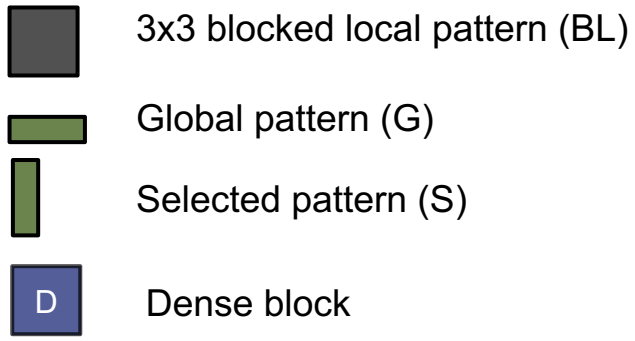
- Coarse-grained methods use blocked sparse format treating each nonzero block as a dense block.
  - Increasing data reuse and maximizing computational throughput by utilizing the tensor cores



Sparse pattern (BL + S + G)

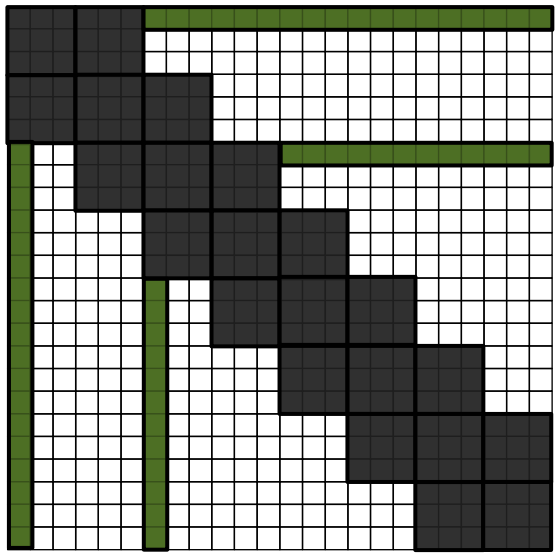


Coarse-grained kernel

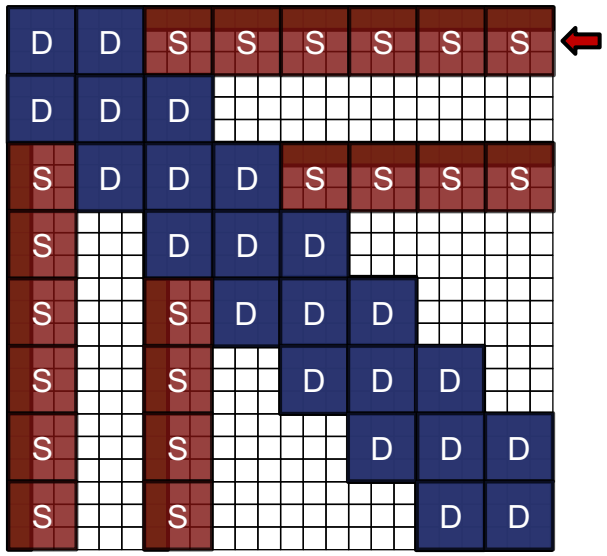


Coarse-grained method

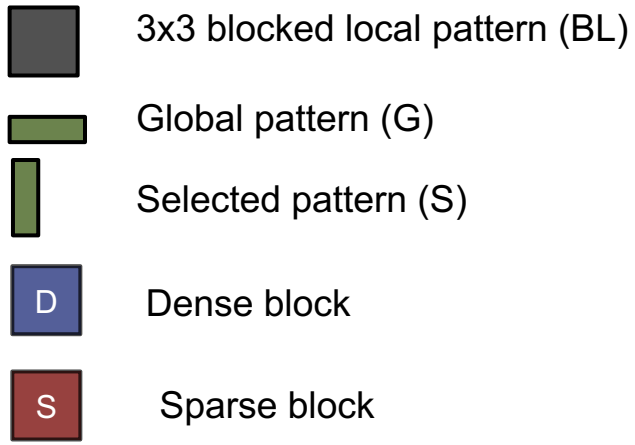
- Coarse-grained methods use blocked sparse format treating each nonzero block as a dense block
  - Increasing data reuse and maximizing computational throughput by utilizing the tensor cores
  - Causing unnecessary computation and memory access in the unstructured patterns with low locality



Sparse pattern (BL + S + G)

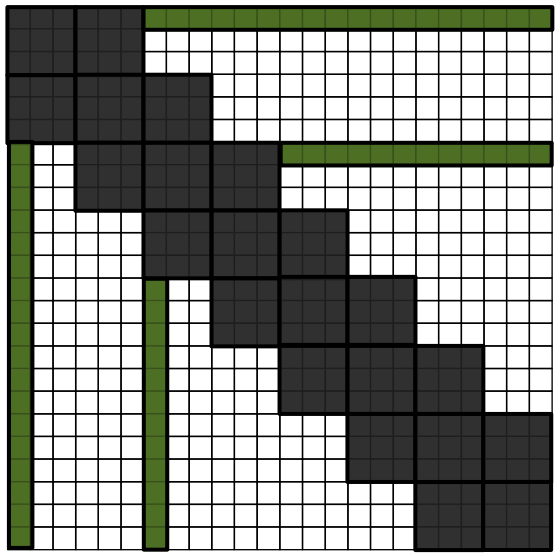


Coarse-grained kernel

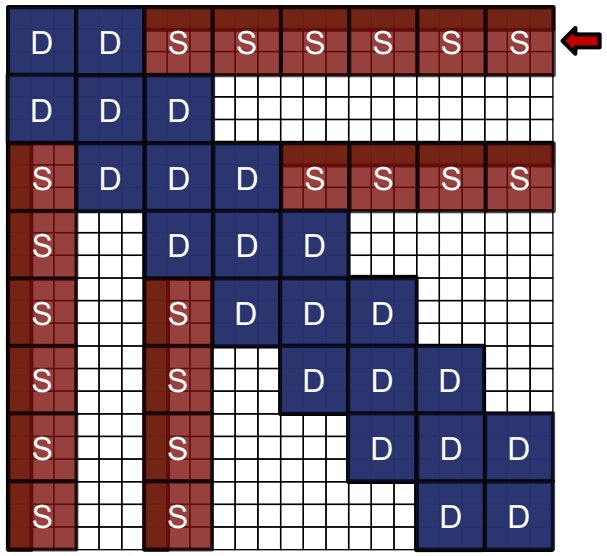


Coarse-grained method

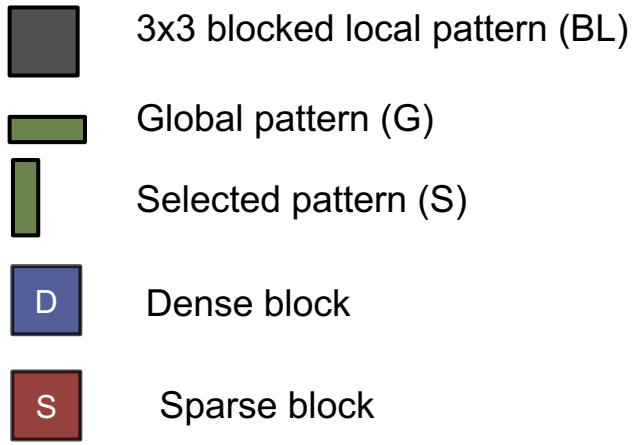
- Coarse-grained methods use blocked sparse format treating each nonzero block as a dense block
  - Increasing data reuse and maximizing computational throughput by utilizing the tensor cores
  - Causing unnecessary computation and memory access in the unstructured patterns with low locality
- Hence, coarse-grained methods are good for structured sparse patterns.



Sparse pattern (BL + S + G)

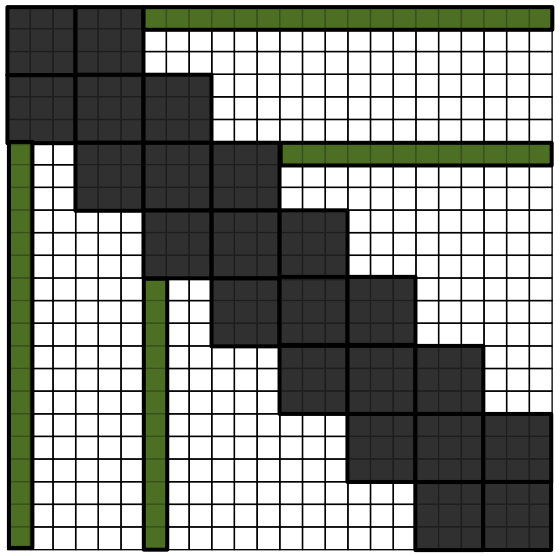


Coarse-grained kernel

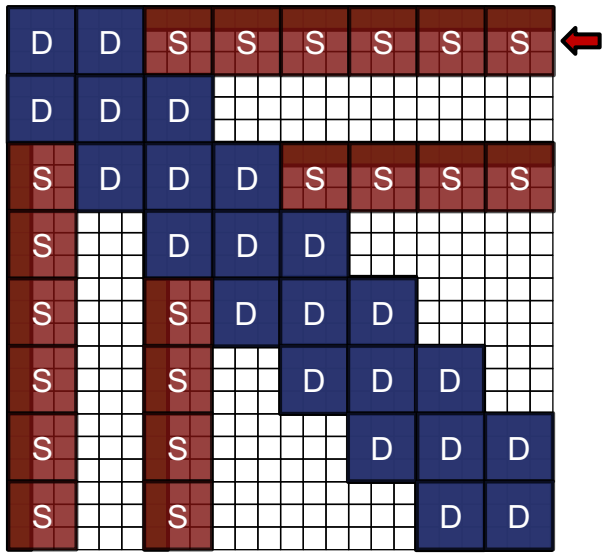


Coarse-grained method

- Coarse-grained methods use blocked sparse format treating each nonzero block as a dense block
  - Increasing data reuse and maximizing computational throughput by utilizing the tensor cores
  - Causing unnecessary computation and memory access in the unstructured patterns with low locality
- Hence, coarse-grained methods are good for structured sparse patterns.
- e.g., DeepSpeed (Triton), cuSPARSE-blockedELL (NVIDIA)

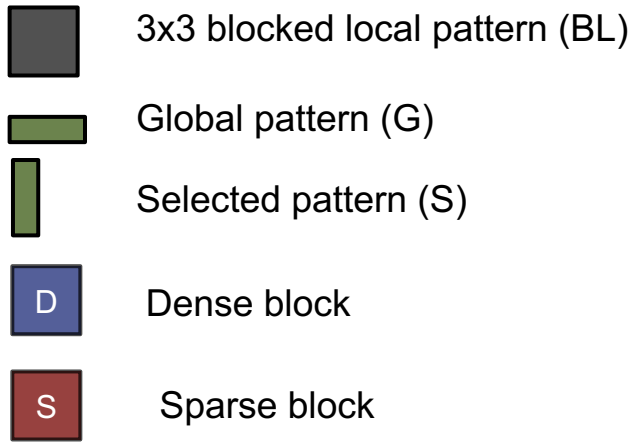


Sparse pattern (BL + S + G)



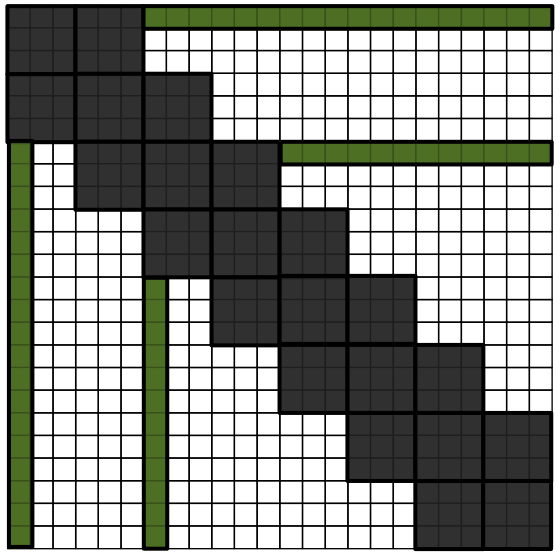
Coarse-grained kernel

Sparse blocks have unnecessary computation and memory access!

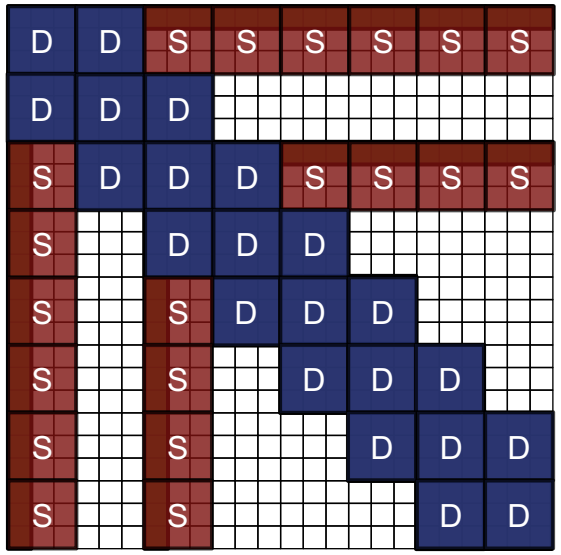


Fine-grained method

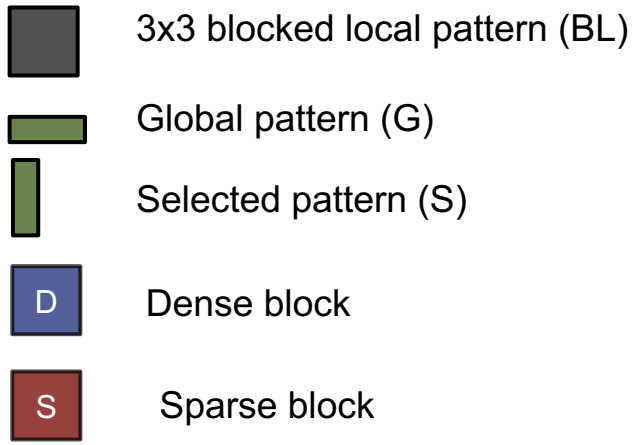
- Fine-grained methods use an element-wise sparse format treating each nonzero elements as valid elements.



Sparse pattern (BL + S + G)

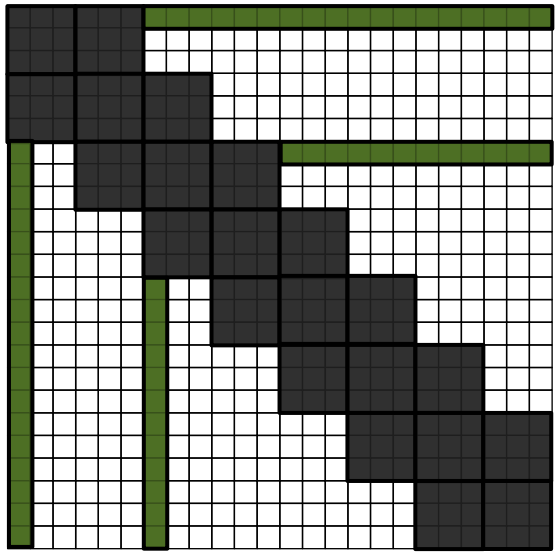


Coarse-grained kernel

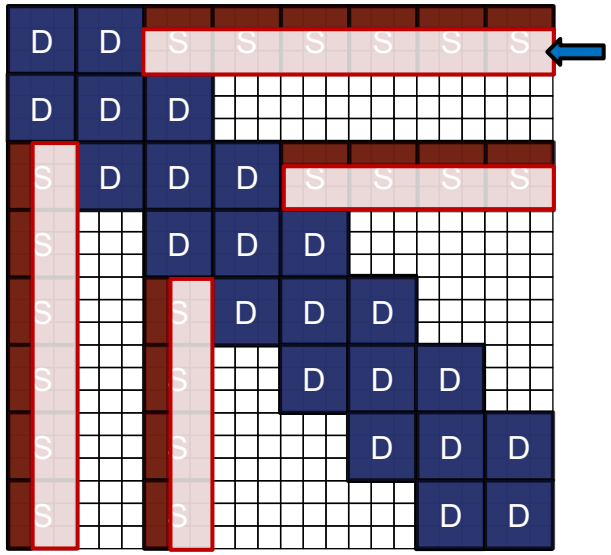


Fine-grained method

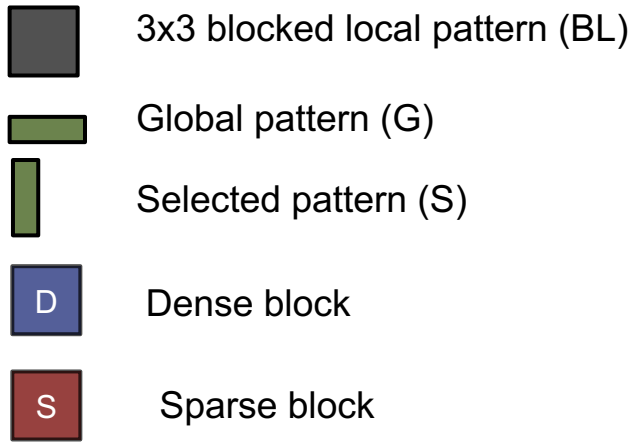
- Fine-grained methods use an element-wise sparse format treating each nonzero elements as valid elements.
  - There is no unnecessary data access or computation.



Sparse pattern (BL + S + G)

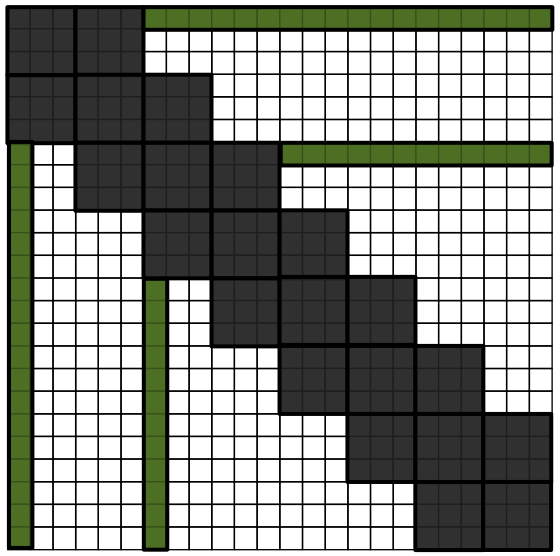


Coarse-grained kernel

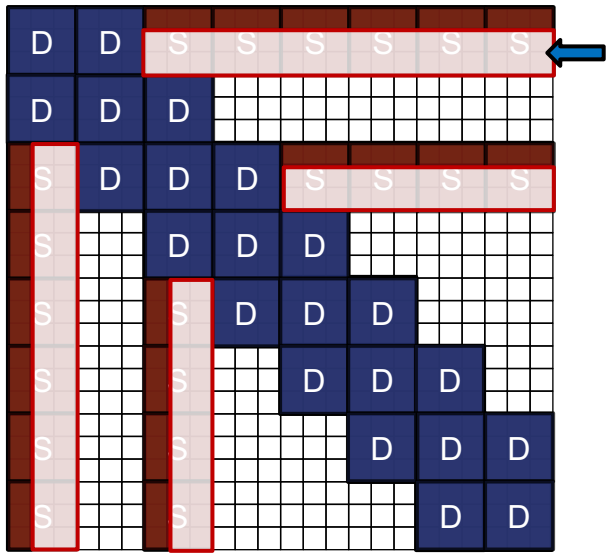


Fine-grained method

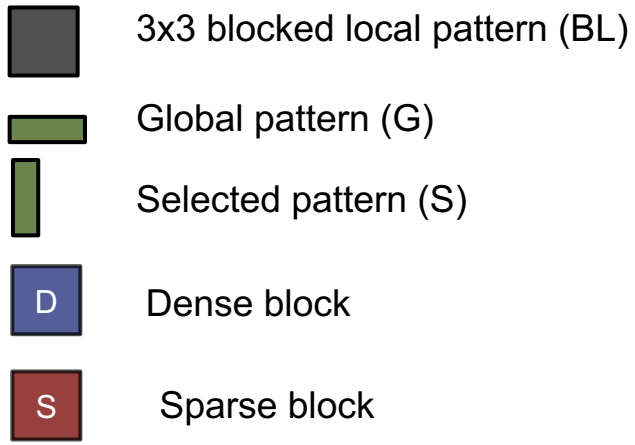
- Fine-grained methods use an element-wise sparse format treating each nonzero elements as valid elements.
  - There is no unnecessary data access or computation.
  - It neither benefits from data reuse nor utilizes high-performance tensor cores.



Sparse pattern (BL + S + G)

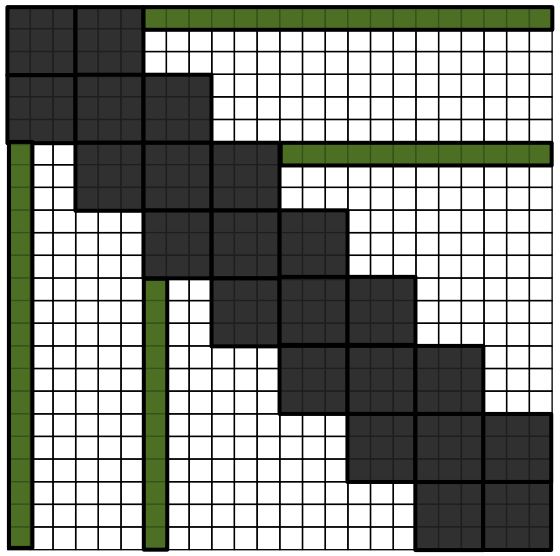


Coarse-grained kernel

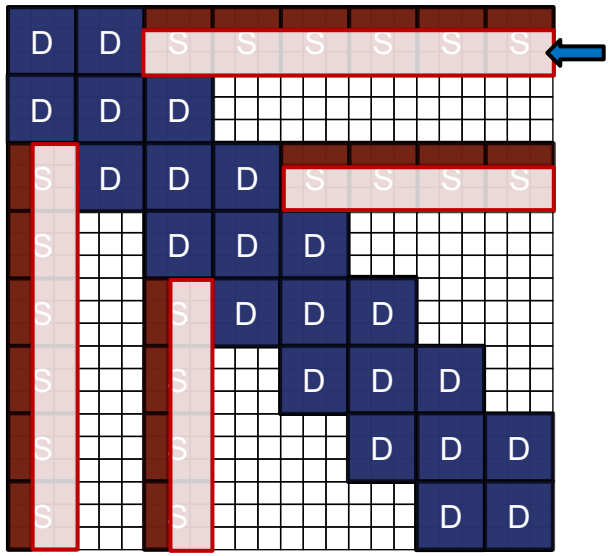


Fine-grained method

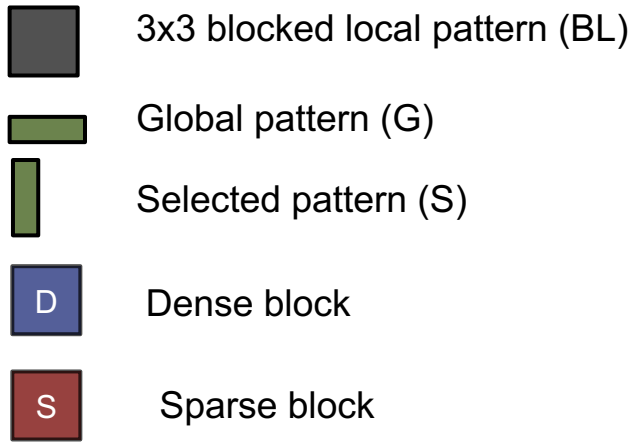
- Fine-grained methods use an element-wise sparse format treating each nonzero elements as valid elements.
  - There is no unnecessary data access or computation.
  - It neither benefits from data reuse nor utilizes high-performance tensor cores.
- Hence, the fine-grained methods target sparse operations with unstructured sparsity.



Sparse pattern (BL + S + G)

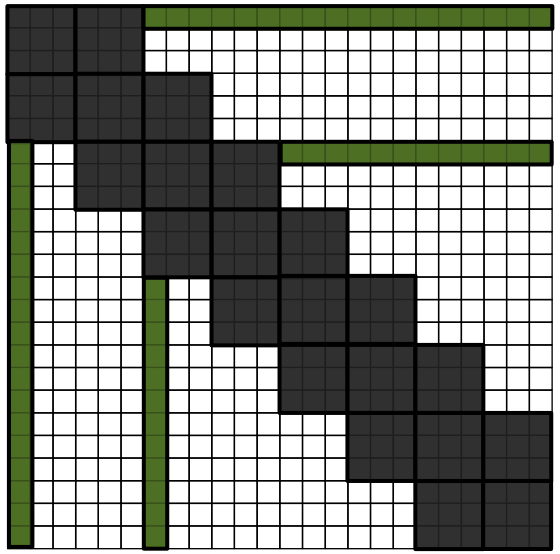


Coarse-grained kernel

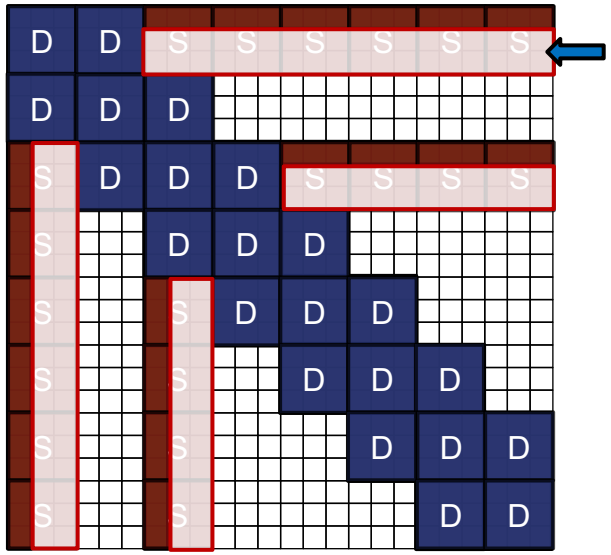


Fine-grained method

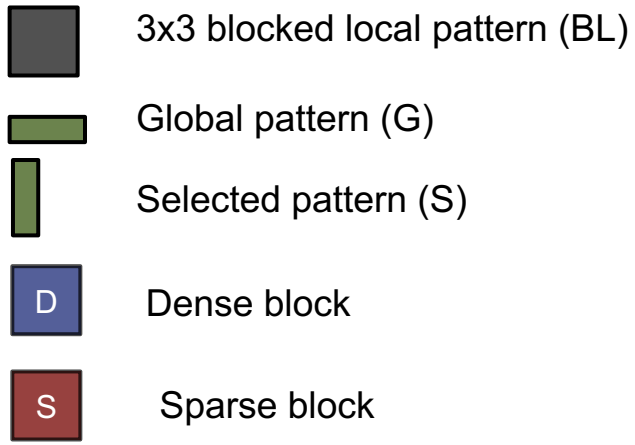
- Fine-grained methods use an element-wise sparse format treating each nonzero elements as valid elements.
  - There is no unnecessary data access or computation.
  - It neither benefits from data reuse nor utilizes high-performance tensor cores.
- Hence, the fine-grained methods target sparse operations with unstructured sparsity.
- e.g., Sputnik (Google), cuSPARSER-CSR/CSC/COO (NVIDIA)



Sparse pattern (BL + S + G)



Coarse-grained kernel



Multigrain

- Multigrain is a new transformer-specific optimization approach.

Multigrain

- Multigrain is a new transformer-specific optimization approach.
  - Accelerating sparse operations on the compound-sparse patterns

Multigrain

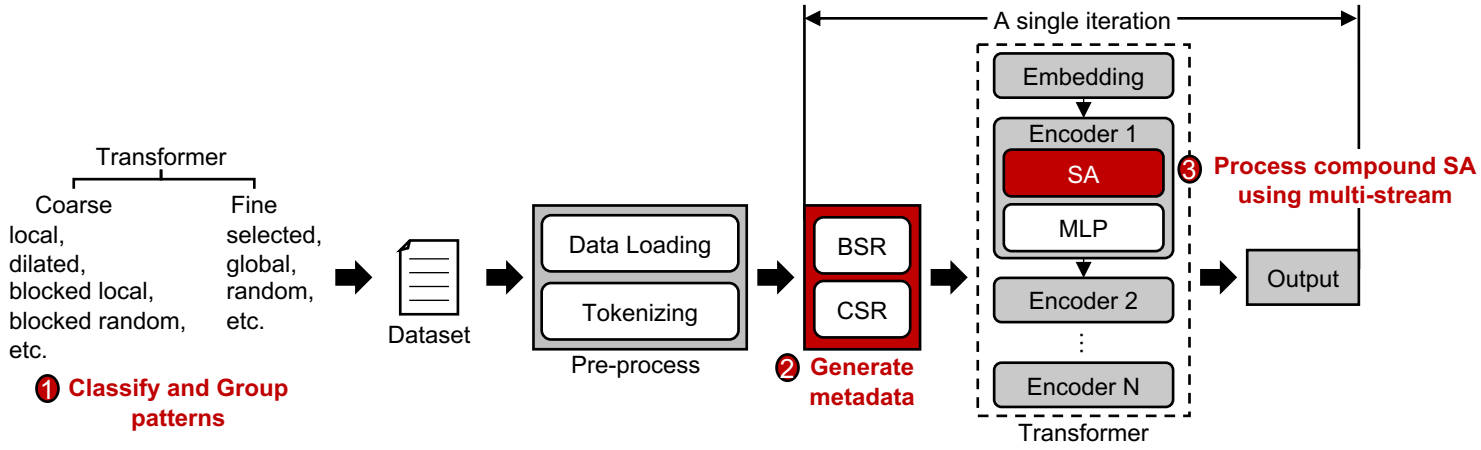
- Multigrain is a new transformer-specific optimization approach
  - Accelerating sparse operations on the compound-sparse patterns
  - Maintaining the advantages of the coarse-grained and fine-grained methods as much as possible and mitigating the disadvantages

Multigrain

- Multigrain is a new transformer-specific optimization approach
  - Accelerating sparse operations on the compound-sparse patterns
  - Maintaining the advantages of the coarse-grained and fine-grained methods as much as possible and mitigating the disadvantages
  - Using the transformer-specific characteristics that the sparse patterns are determined offline by the model, but the number and position of non-zeros are changed by the input data at every iteration

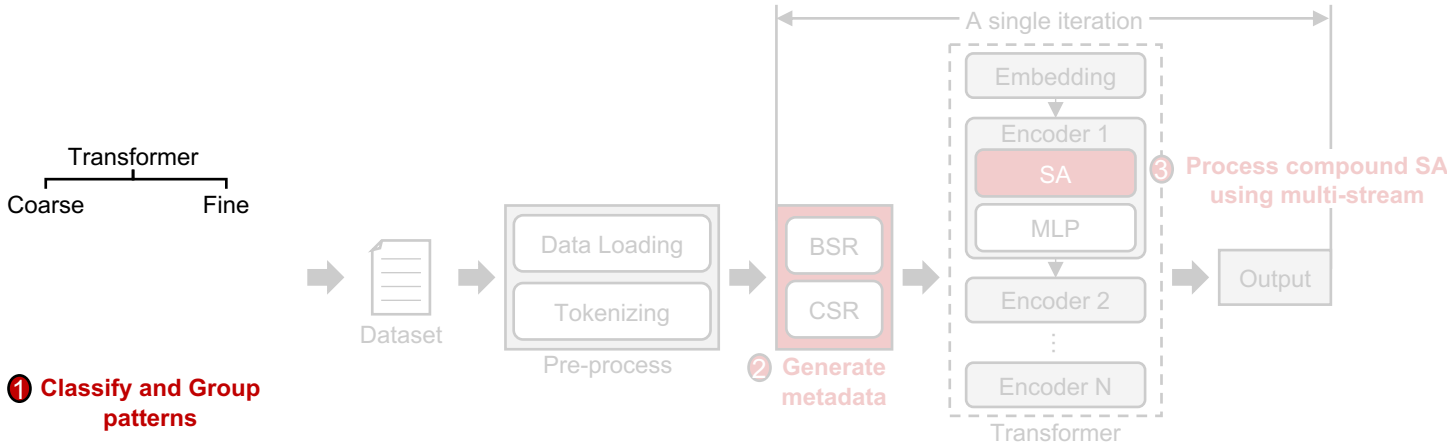
Multigrain

- Multigrain is a new transformer-specific optimization approach.
- Multigrain works with the following steps.



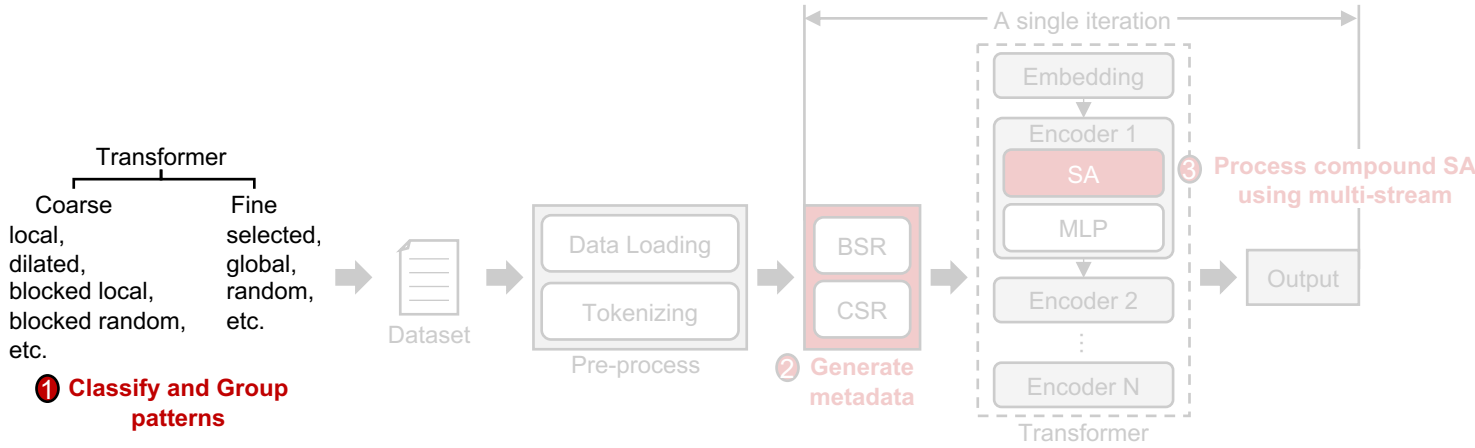
Multigrain

- Multigrain is a new transformer-specific optimization approach.
- Multigrain works with the following steps.
  - Classifying and grouping sparse patterns into the coarse-grained and fine-grained parts according to the spatial locality



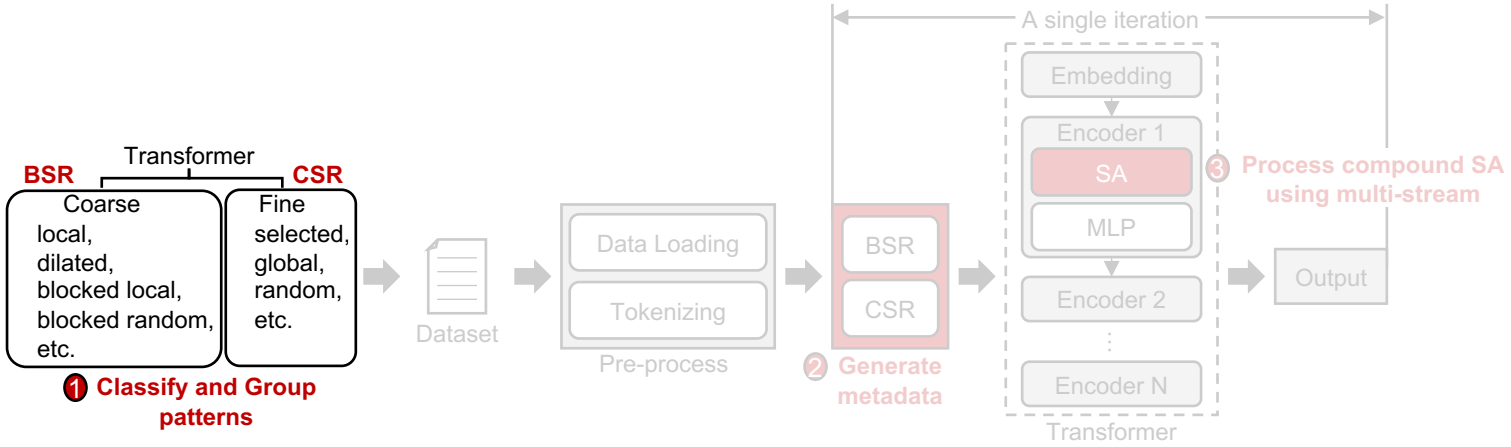
Multigrain

- Multigrain is a new transformer-specific optimization approach.
- Multigrain works with the following steps.
  - Classifying and grouping sparse patterns into the coarse-grained and fine-grained parts according to the spatial locality
    - = Regarding local, dilated, blocked local, blocked selected, blocked global, and blocked random patterns as the coarse-grained part and selected, global, and random patterns as the fine-grained part



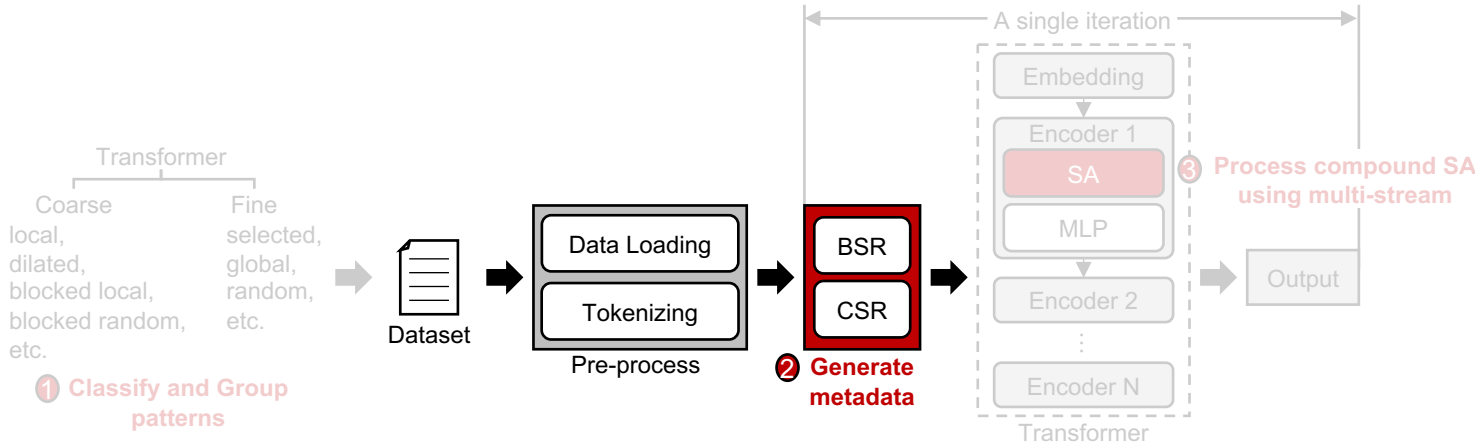
Multigrain

- Multigrain is a new transformer-specific optimization approach.
- Multigrain works with the following steps.
  - Classifying and grouping sparse patterns into the coarse-grained and fine-grained parts according to the spatial locality
    - = Regarding local, dilated, blocked local, blocked selected, blocked global, and blocked random patterns as the coarse-grained part and selected, global, and random patterns as the fine-grained part
    - = Representing the coarse-grained part where nonzeros are structured and clustered as BSR format and the fine-grained part where nonzeros are unstructured as CSR format



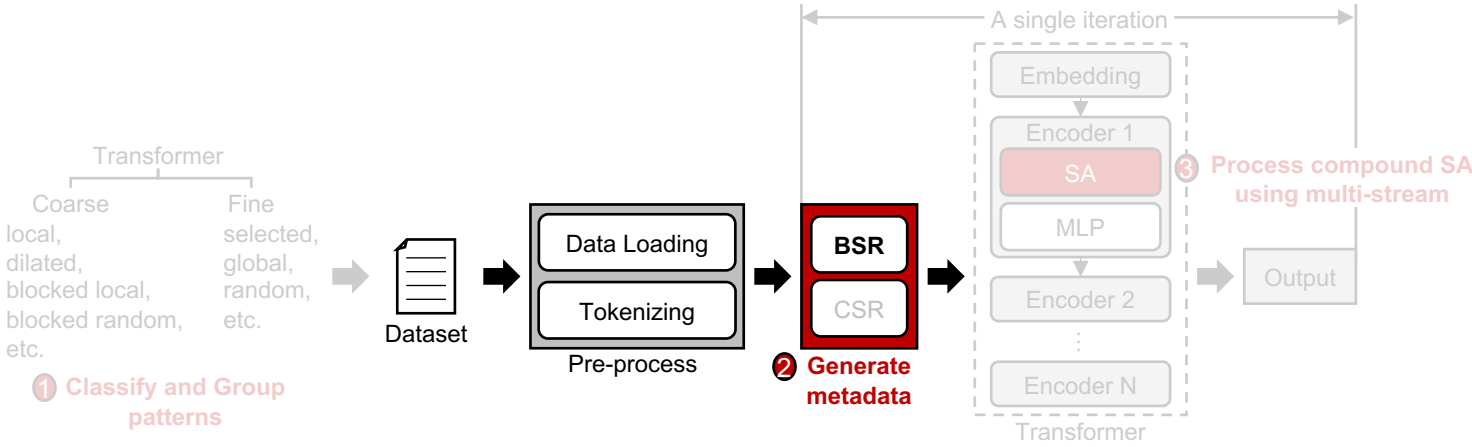
Multigrain

- Multigrain is a new transformer-specific optimization approach.
- Multigrain works with the following steps.
  - Classifying and grouping sparse patterns into the coarse-grained and fine-grained parts according to the spatial locality
  - Generating the BSR only once and CSR metadata at every iteration for the compressed sparse matrices with the model configuration and positions of the special tokens



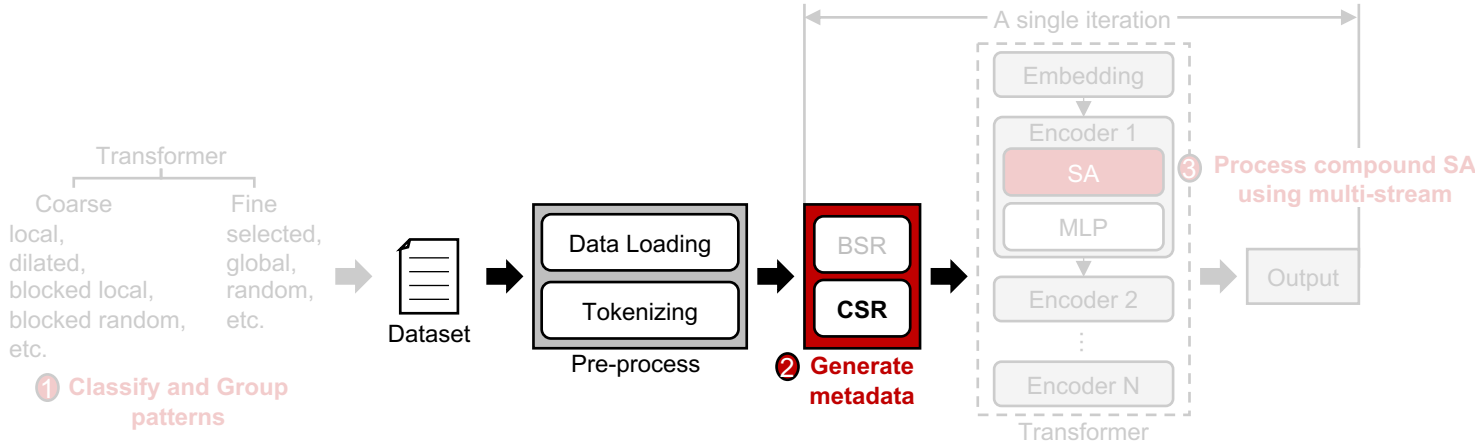
Multigrain

- Multigrain is a new transformer-specific optimization approach.
- Multigrain works with the following steps.
  - Classifying and grouping sparse patterns into the coarse-grained and fine-grained parts according to the spatial locality
  - Generating the BSR only once and CSR metadata at every iteration for the compressed sparse matrices with the model configuration and positions of the special tokens
    - = Using the window size in the model configuration to generate the BSR metadata for the coarse-grained part



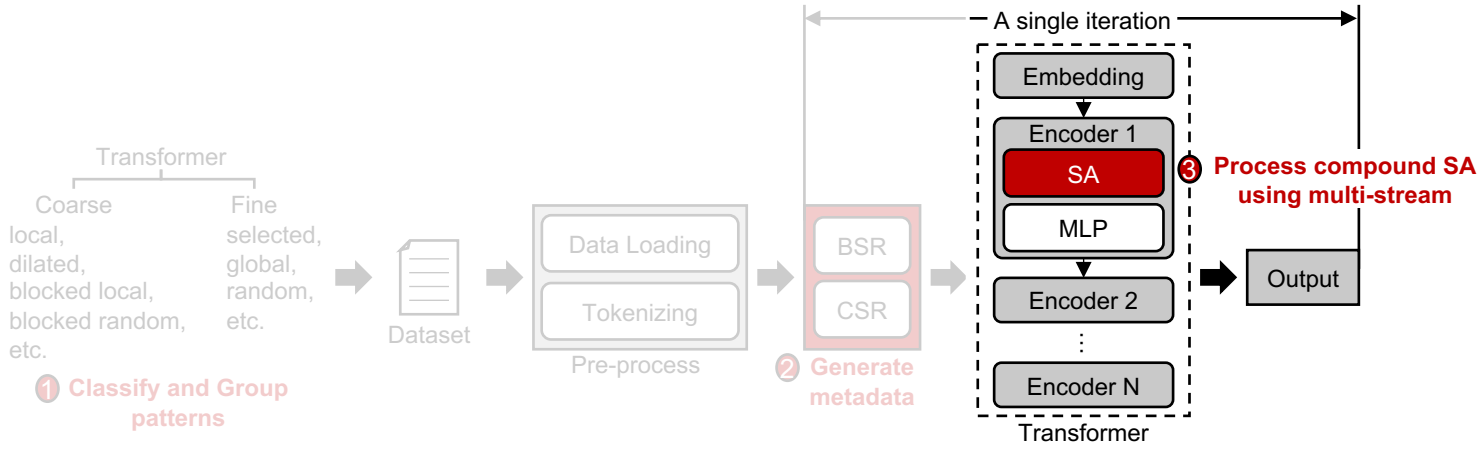
Multigrain

- Multigrain is a new transformer-specific optimization approach.
- Multigrain works with the following steps.
  - Classifying and grouping sparse patterns into the coarse-grained and fine-grained parts according to the spatial locality
  - Generating the BSR only once and CSR metadata at every iteration for the compressed sparse matrices with the model configuration and positions of the special tokens
    - = Using the window size in the model configuration to generate the BSR metadata for the coarse-grained part
    - = Using global indices (i.e., start and end tokens for a sentence, paragraph, and document) from the position of special tokens to generate the CSR metadata for the fine-grained part.



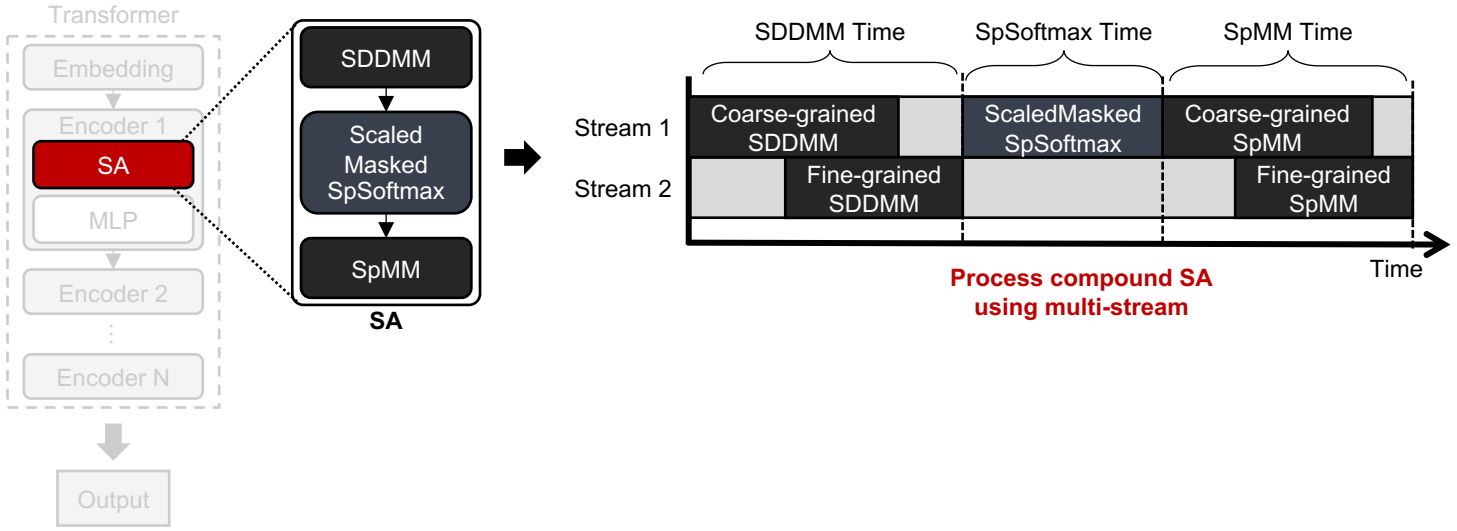
Multigrain

- Multigrain is a new transformer-specific optimization approach.
- Multigrain works with the following steps.
  - Classifying and grouping sparse patterns into the coarse-grained and fine-grained parts according to the spatial locality
  - Generating the BSR and CSR metadata (e.g., row offsets and column indices) for the compressed sparse matrices with the model configuration and positions of the special tokens
  - Processing the SA utilizing BSR and CSR formats with multi-stream



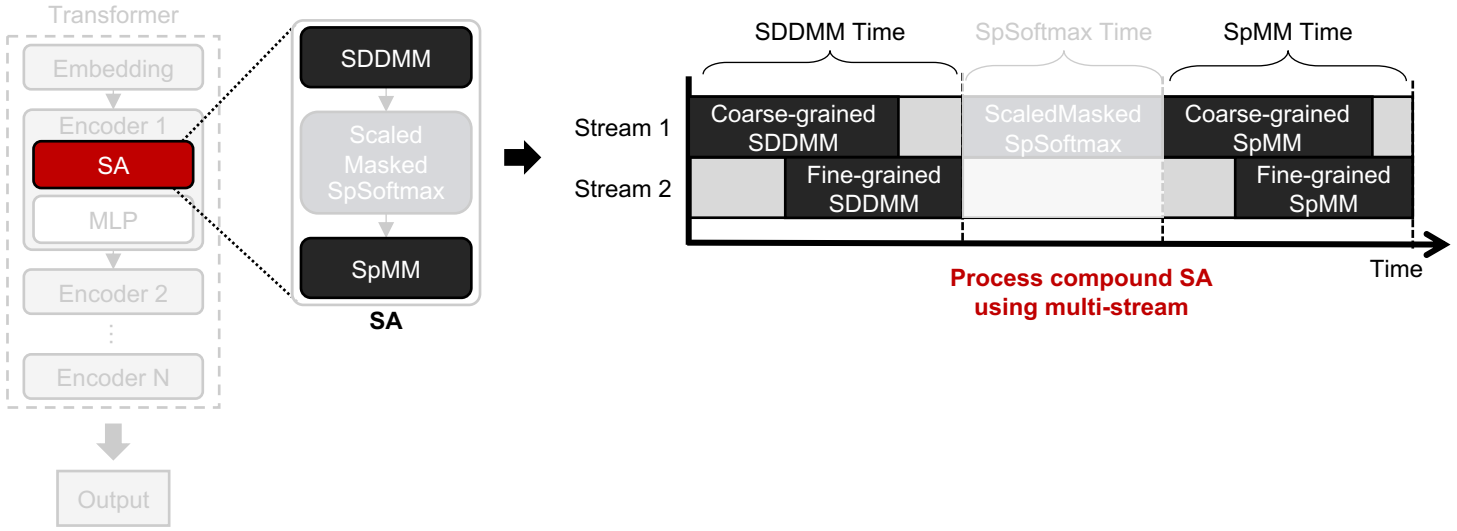
Sparse attentions in Multigrain

- Multigrain processes the SA (e.g., SDDMM, Scaled-masked SpSoftmax and SpMM) by executing multiple kernels concurrently.



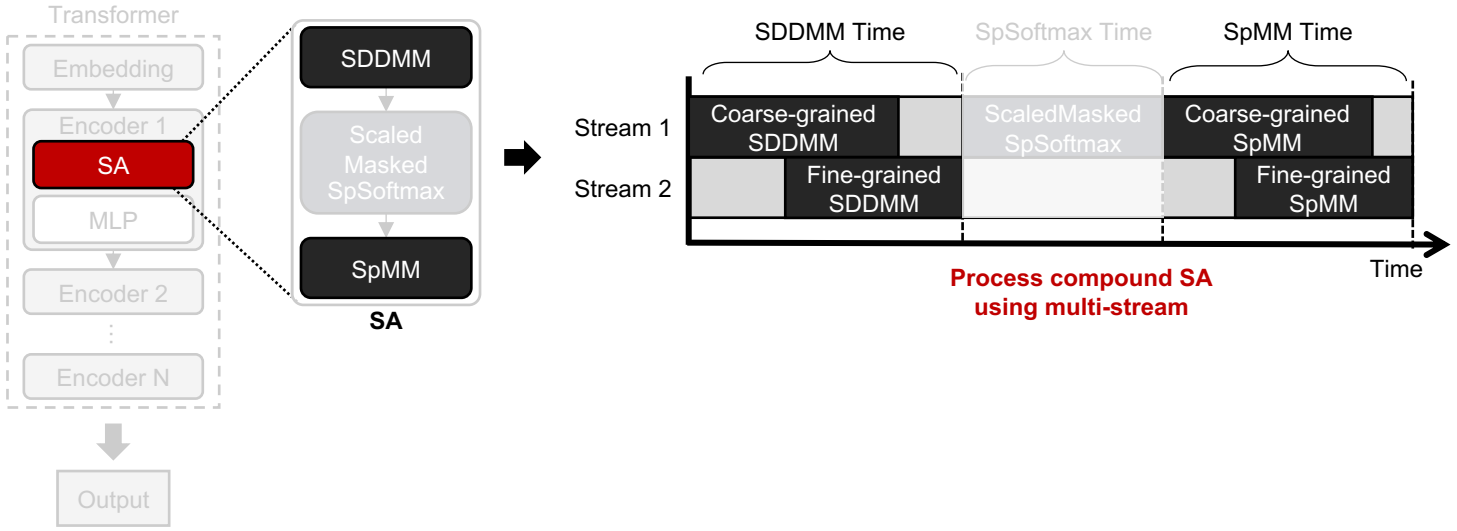
Sparse attentions in Multigrain

- Multigrain processes the SA (e.g., SDDMM, Scaled-masked SpSoftmax and SpMM) by executing multiple kernels concurrently.
  - SDDMM and SpMM are executed through both coarse-grained and fine-grained kernels, processed in parallel using multi-stream.



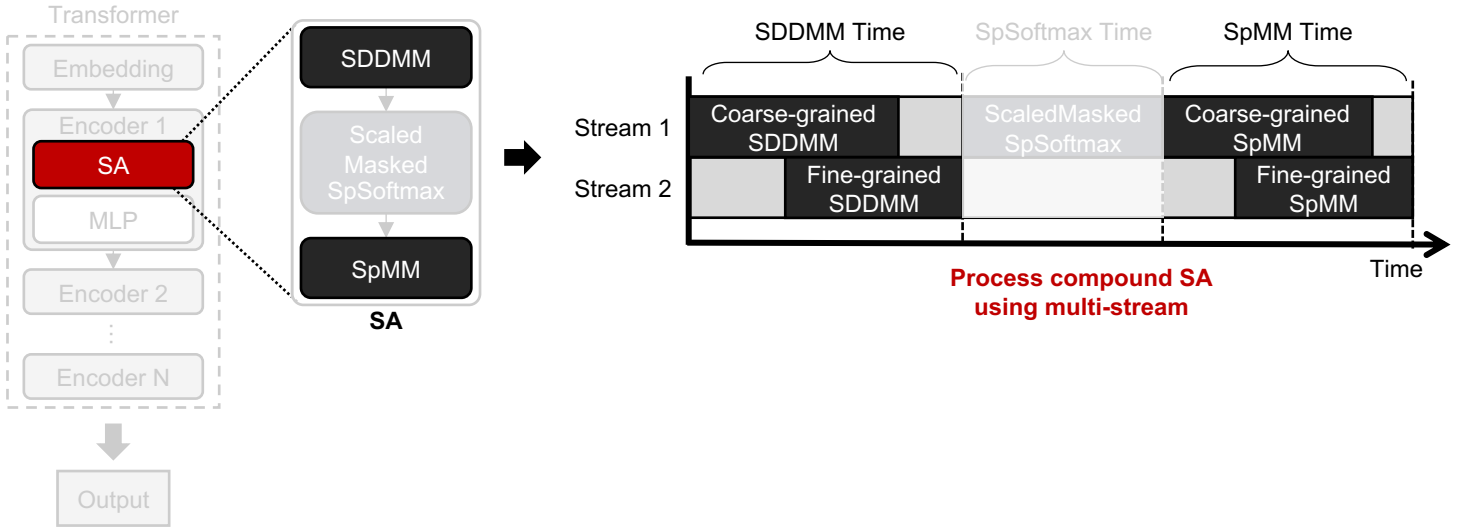
Sparse attentions in Multigrain

- Multigrain processes the SA (e.g., SDDMM, Scaled-masked SpSoftmax and SpMM) by executing multiple kernels concurrently.
  - SDDMM and SpMM are executed through both coarse-grained and fine-grained kernels, processed in parallel using multi-stream.
    - = For the coarse-grained parts, we designed customized coarse-grained kernels using BSR. (see Sec 3.2)



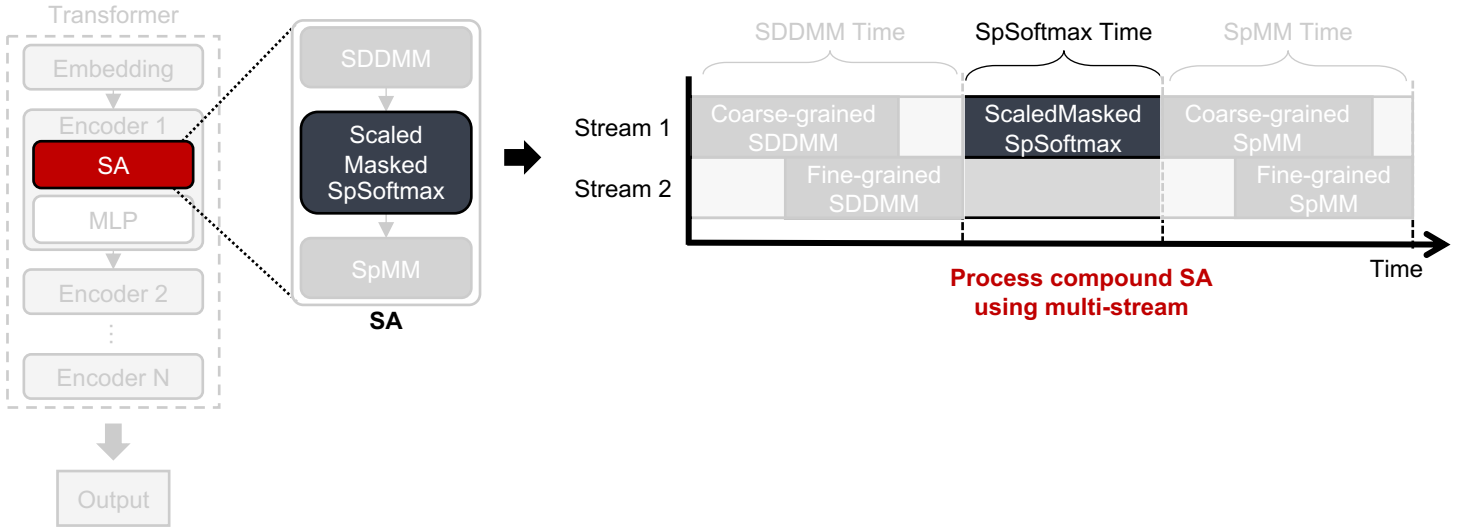
Sparse attentions in Multigrain

- Multigrain processes the SA (e.g., SDDMM, Scaled-masked SpSoftmax and SpMM) by executing multiple kernels concurrently.
  - SDDMM and SpMM are executed through both coarse-grained and fine-grained kernels, processed in parallel using multi-stream.
    - = For the coarse-grained parts, we designed customized coarse-grained kernels using BSR. (see Sec 3.2)
    - = For the fine-grained parts, we adopted Sputnik which has fine-grained kernels using CSR, and extended it to support half-precision (FP16) operations in SDDMM.



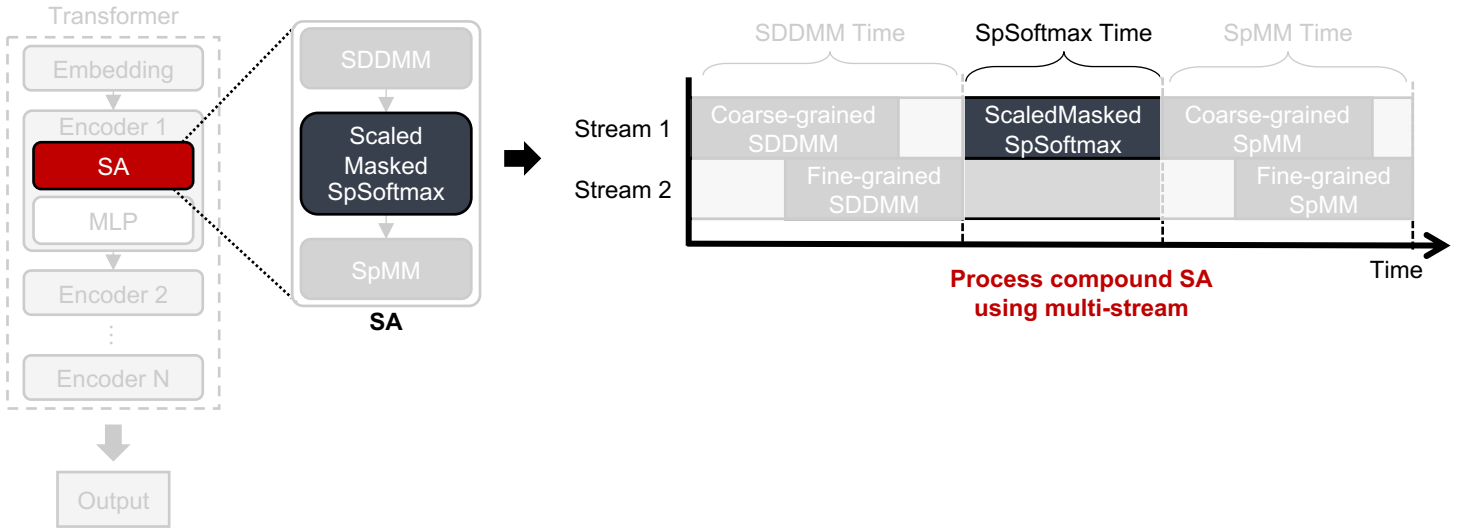
Sparse attentions in Multigrain

- Multigrain processes the SA (e.g., SDDMM, Scaled-masked SpSoftmax and SpMM) by executing multiple kernels concurrently.
  - SDDMM and SpMM are executed through both coarse-grained and fine-grained kernels, processed in parallel using multi-stream.
  - Scaled-masked sparse softmax is executed through a single kernel utilizing the BSR and CSR formats.



Sparse attentions in Multigrain

- Multigrain processes the SA (e.g., SDDMM, Scaled-masked SpSoftmax and SpMM) by executing multiple kernels concurrently.
  - SDDMM and SpMM are executed through both coarse-grained and fine-grained kernels, processed in parallel using multi-stream.
  - Scaled-masked sparse softmax is executed through a single kernel utilizing the BSR and CSR formats.
    - = We fused the scaling and masking operations for further optimization. (see Sec 3.3)



Experimental Setup

- We evaluated inference time and memory traffic for Longformer and QDS-Transformer using FP16 in the Hotpot-QA and Microsoft MAchine Reading Compensation (MSMARCO) datasets.
  - Triton[1)]: a coarse-grained library from OpenAI used by DeepSpeed for performing SA
  - Sputnik[2)]: a fine-grained library from Google
- We evaluated them on two GPUs with different computational characteristics (A100 and GeForce RTX 3090) and on the various batch sizes.

	A100	RTX3090
Memory Bandwidth (GB/s)	1,555	936.2
TFLOPS (FP16 CUDA core)*	43.3	29.3
TFLOPS (FP16 Tensor core)*	169	58
L1 D$ per SM (KB)	192	128
L2 (MB) $	40	6

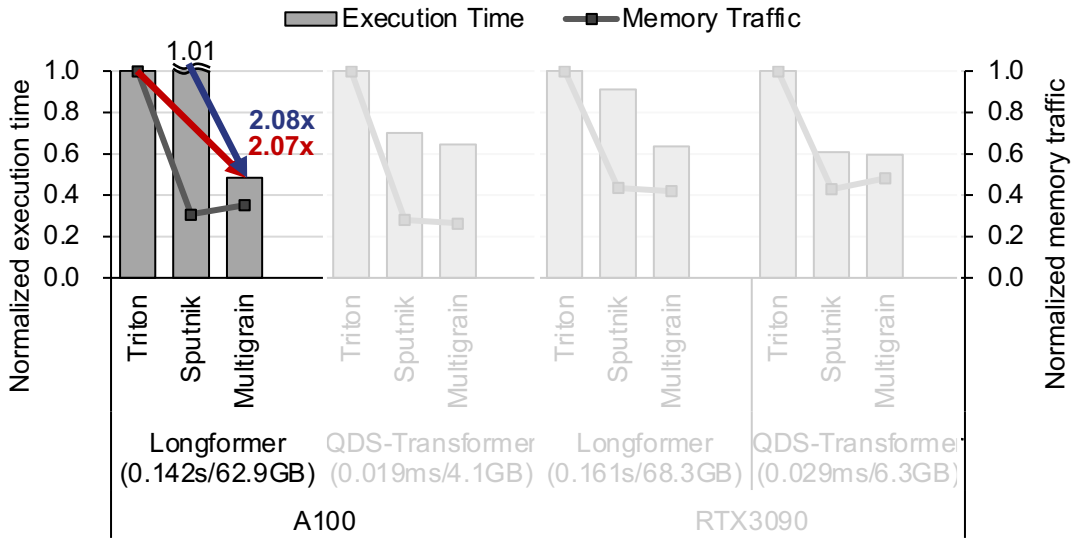
* Peak rates are based on the GPU's base clock

1) We optimized the SDDMM in Triton. Our optimized Triton was 6.24×, 6.23×, and 6.73× faster than the original version at the local, blocked local, and blocked random patterns, respectively. (configuration: a single batch, four multi-head, and 64 head dimensions)

2) We extended Sputnik for supporting FP16 and batched operations. In addition, We optimized the SDDMM kernel with the row-splitting scheme, reducing execution time by 3.3× to 6.2× over the original version. (1D tiling scheme)

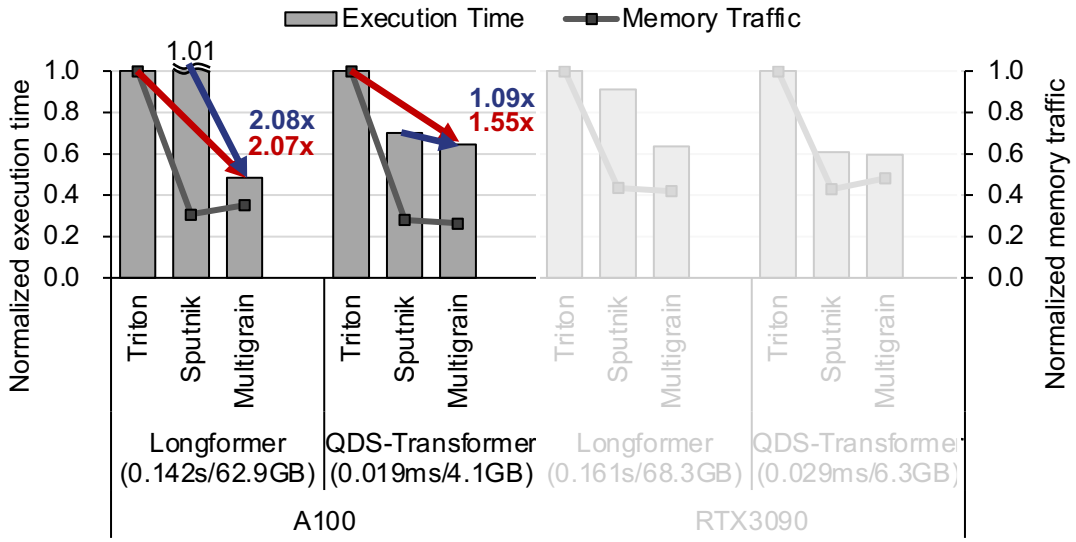
End-to-end

- In the Longformer, Multigrain reduced
  - Execution time by 2.07× and 2.08× compared to Triton and Sputnik, respectively.
  - Memory traffic by 2.84× compared to Triton.



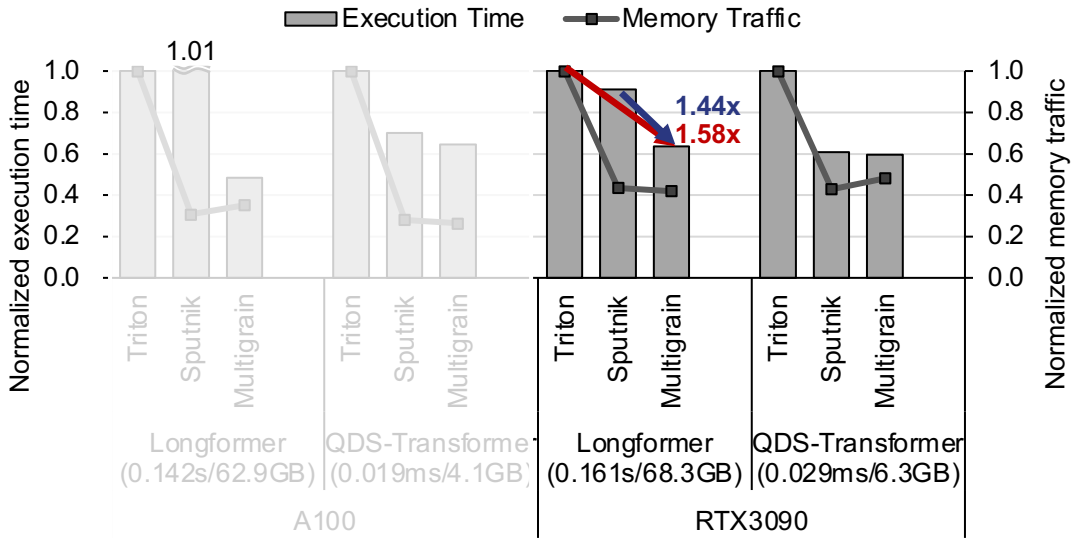
End-to-end

- In the Longformer, Multigrain reduced
  - Execution time by 2.07× and 2.08× compared to Triton and Sputnik, respectively.
  - Memory traffic by 2.84× compared to Triton.
- In the QDS-Transformer, Multigrain reduced
  - Execution time by 1.55× and 1.09× compared to Triton and Sputnik, respectively.
  - Memory traffic by 3.78x compared to Triton.



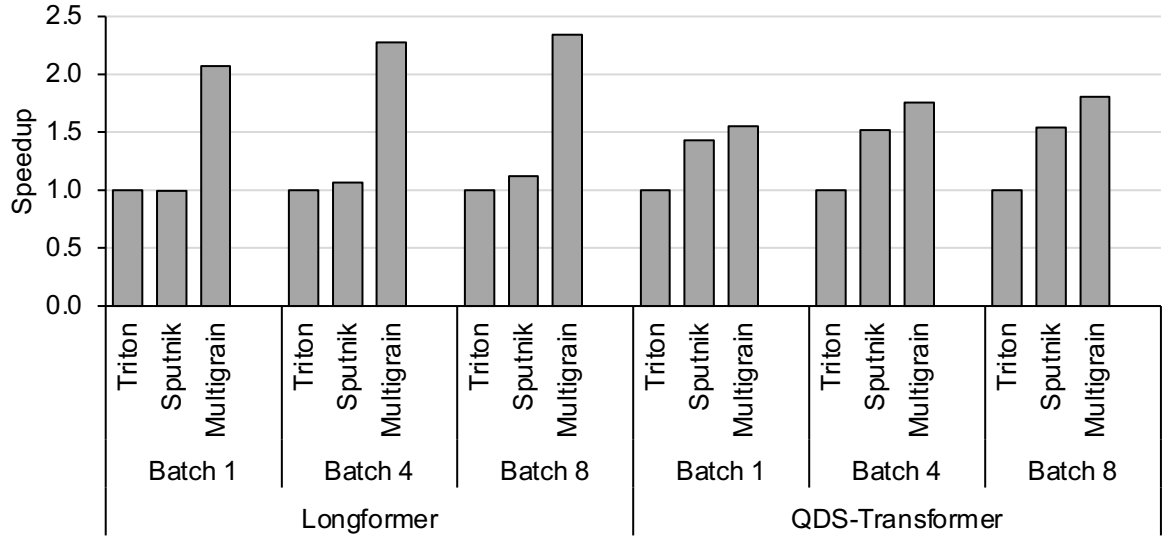
End-to-end in the different GPU

- On the RTX3090, the performance improvement compared to fine-grained (Sputnik) was less than the performance improvement compared to coarse-grained (Triton)
  - Coarse- and find-grained kernels are processed on Tensor cores and cuda cores respectively.
  - The peak floating-point operations per second (FLOPS) of the tensor cores is reduced more than that of the CUDA cores. (Tensor core: 2.9x, CUDA core: 1.5x from A100 to RTX3090)



End-to-end in the various batch sizes

- Multigrain gains additional performance improvement as the batch size increases in Longformer and QDS-Transformer, respectively.
  - Achieving up to 2.34× and 1.82× speedups compared to Triton
  - Achieving up to 2.13× and 1.17× speedups compared to Sputnik



Conclusion

- We proposed Multigrain, a transformer-specific optimization approach, to accelerate compound-sparse attention by applying compound sparse GPU kernels with multi-stream.
- We divided the sparse attention of the compound-sparse patterns into coarse-grained and fine-grained parts, processing the two parts separately with the corresponding kernels.
  - Compared to Triton, Multigrain significantly reduced unnecessary computation and memory accesses.
  - Compared to Sputnik, Multigrain can enable efficient data reuse and exploit the power of the high-performance tensor cores.

Thank you!