ByteTransformer: A High-Performance Transformer Boosted for Variable-Length Inputs

Yujia Zhai, Chengquan Jiang, Leyuan Wang, Xiaoying Jia, Shang Zhang,
Zizhong Chen, Xin Liu, Yibo Zhu

Yanzhuo Chen

Important metrices for LLM Inference

- Throughput = query / s – maximize for batch job speed to allow more users

- Latency = s / token – minimize for user experience

LLM Inference Is Actually Slow

model id	tok out	sec	model name	ms/token
gpt-3.5-turbo-1106	3772	24.3	**latest gpt-3.5**	**6.5**
gpt-4-1106-preview	4096	74.7	**gpt-4 turbo**	**18.2**
gpt-3.5-turbo-0613	3800	79.5	**old gpt-3.5**	**20.9**
gpt-4-0613	4141	400.0	**old gpt-4**	**96.6**

Table Credit: https://www.taivo.ai/__a-wild-speed-up-from-openai-dev-day/

LLM Inference Optimization

Tensorflow XLA, PyTorch JIT
- Leverage the domain-specific just-in-time compilation technique to boost performance

- Does not support kernel fusion or variable length input

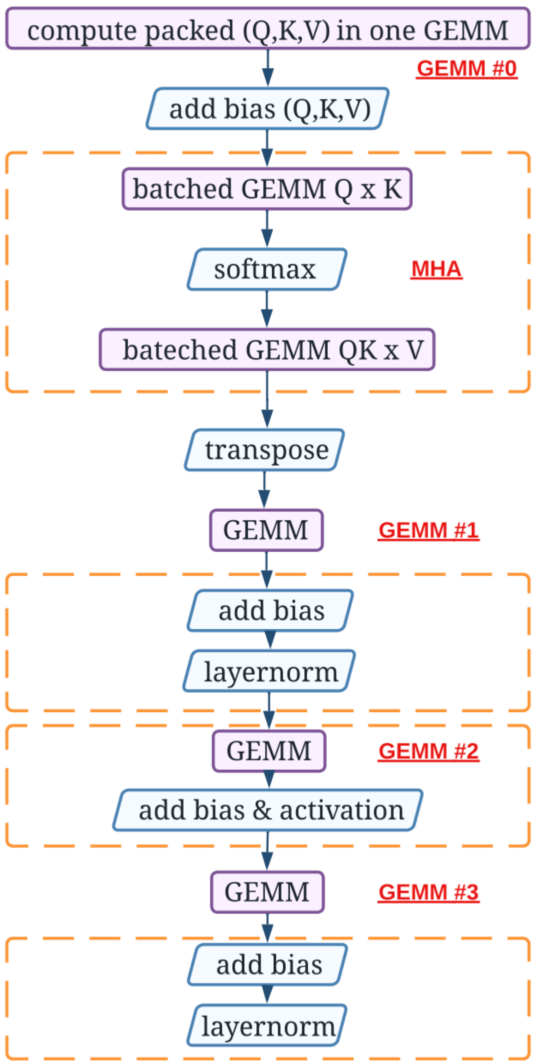
FasterTransformer
- Support variable length input by batching requests with similar sequence lengths
- Partially support fused MHA (≤ 512)

TurboTransformer
- Support variable length input by batching requests with similar sequence lengths

A lot of redundant memory & computation for batching requests with different sequence length!

ByteTransformer
- Memory-Bound Kernel Fusion
- Variable Length Support
- Fused Multi-Head Attention (MHA)

Bert Transformer Architecture

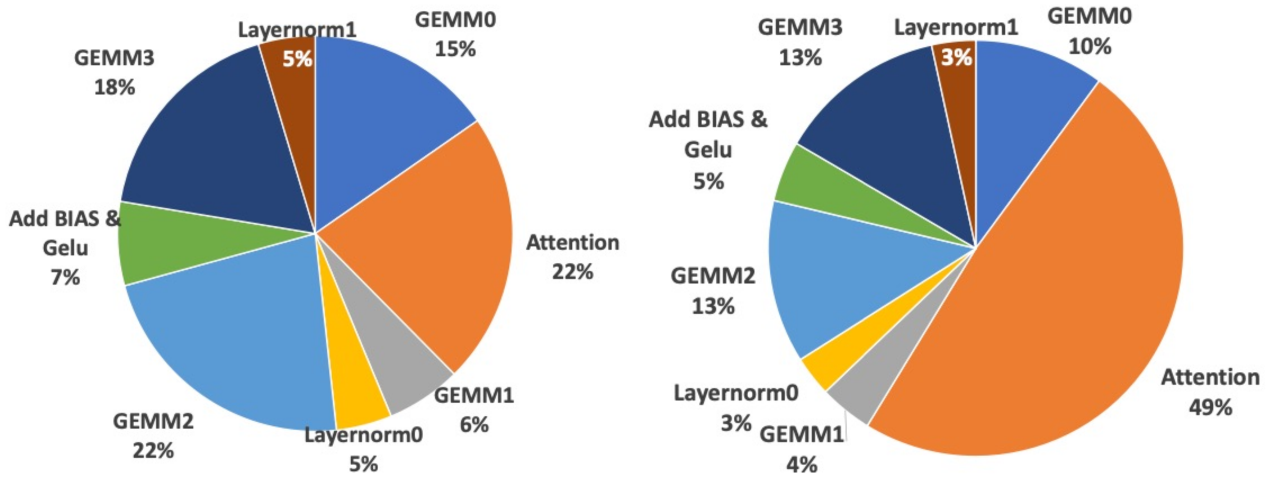


- Batch Size (bs): 16
- Head Number: 12
- Head Size: 64

$k = 12 \times 64 = 768$

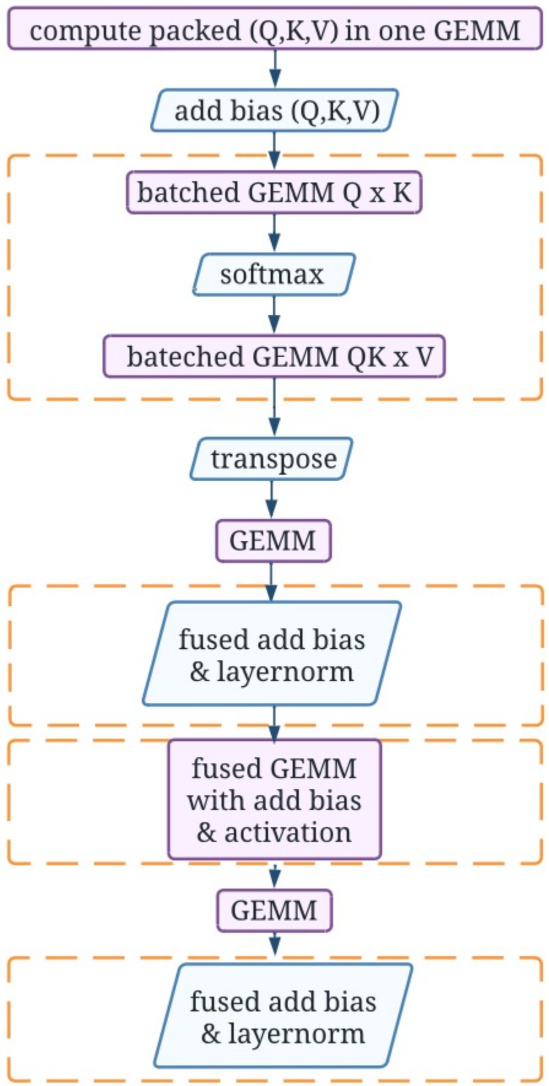
m: sequence length

	Number of Computation
Gemm #0	$6mk^2$
MHA	$4\frac{m^2}{bs}k$
Gemm #1	$2mk^2$
Gemm #2	$8mk^2$
Gemm #3	$8mk^2$

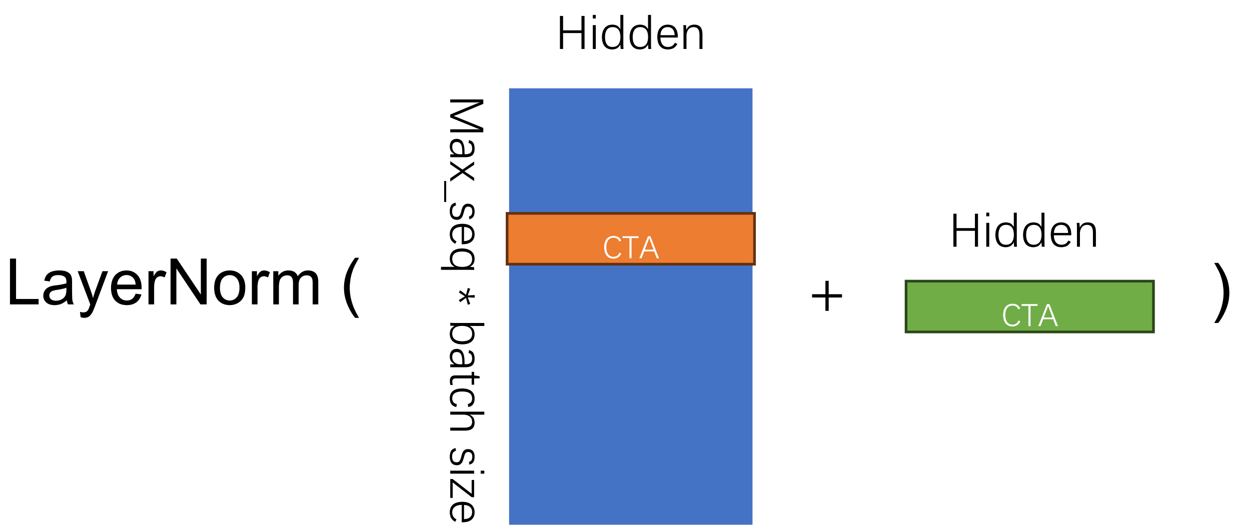


(a) Sequence lengths 256 (b) Sequence lengths 1024

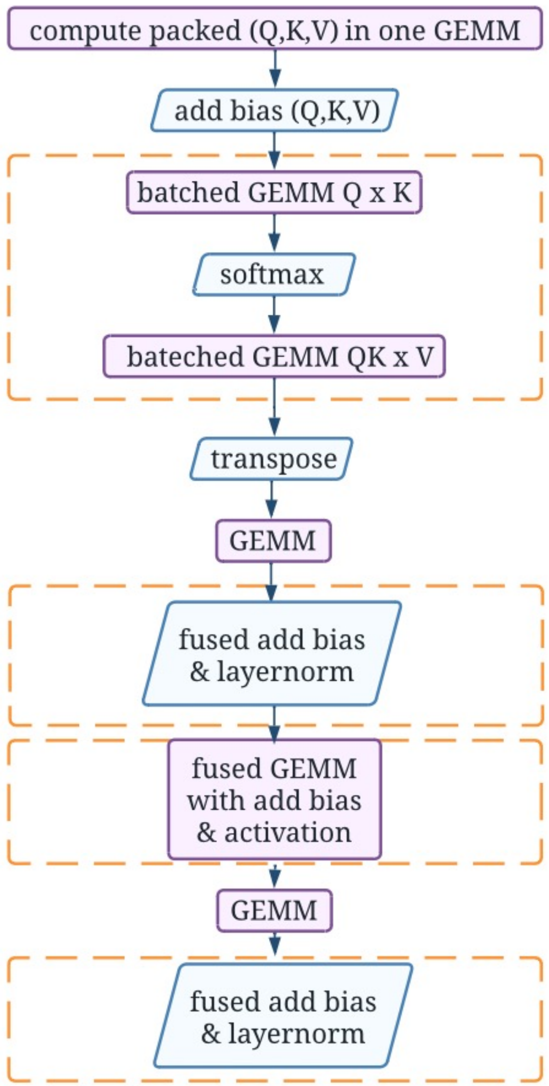
Fusing Memory-bound Operations



Fused Add bias & layernorm



Fusing Memory-bound Operations

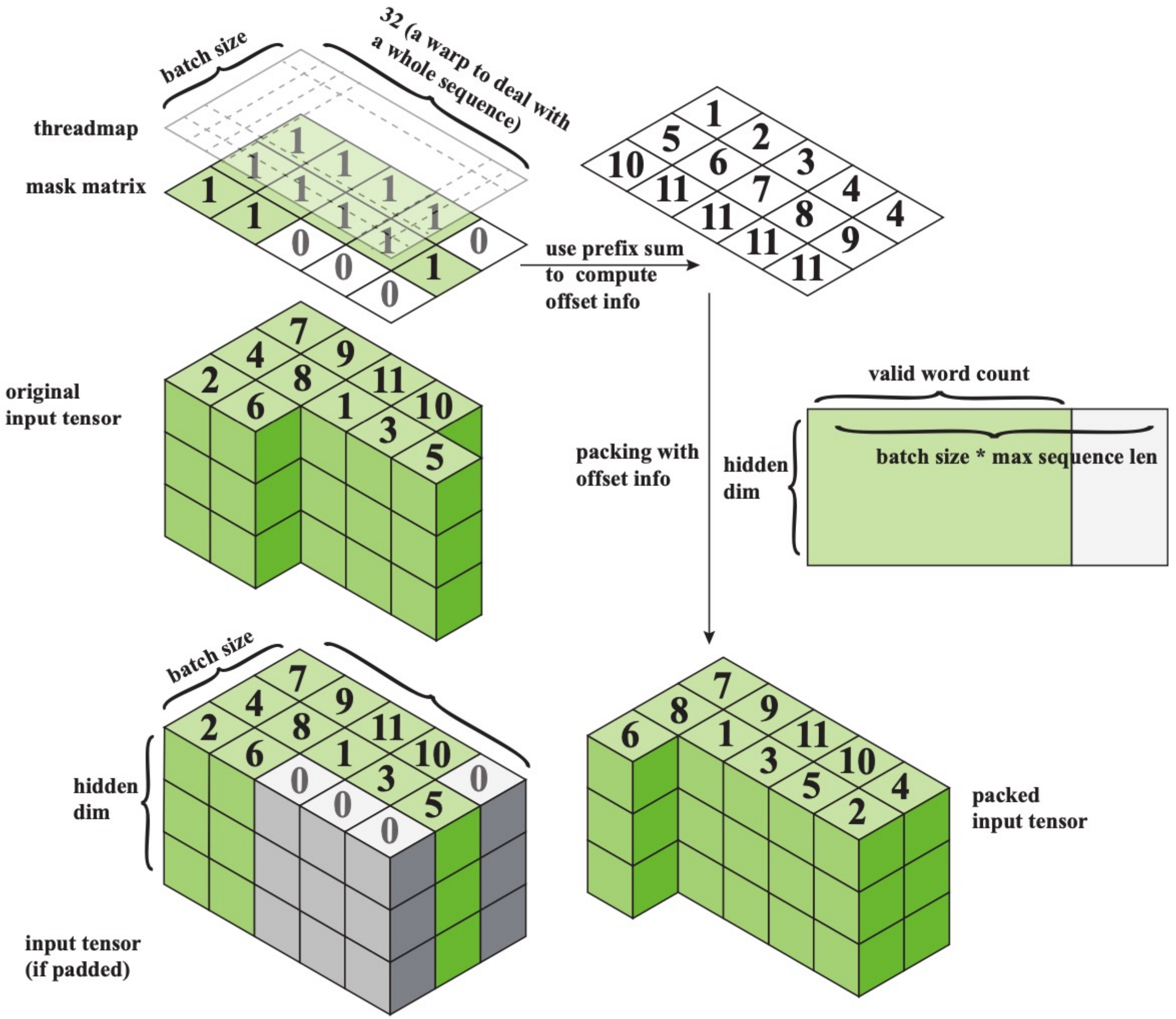


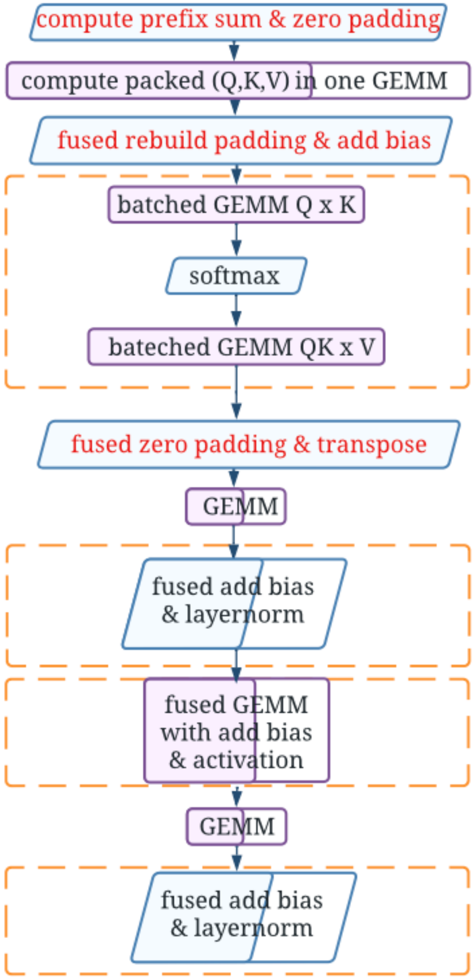
Fused Gemm with add bias & activation

```
#define GEMM_TYPE(BLOCK_M, BLOCK_N, BLOCK_K, WARP_M, WARP_N, WARP_K, INST_M, INST_N, INST_K,                    \
                  NUM_STAGES)                                                                                     \
  using ShapeMMAThreadBlock_##BLOCK_M##_##BLOCK_N##_##BLOCK_K =                                                  \
      cutlass::gemm::GemmShape<BLOCK_M, BLOCK_N, BLOCK_K>;                                                       \
  using ShapeMMAWarp_##WARP_M##_##WARP_N##_##WARP_K =                                                            \
      cutlass::gemm::GemmShape<WARP_M, WARP_N, WARP_K>;                                                          \
  using ShapeMMAOp_##INST_M##_##INST_N##_##INST_K =                                                              \
      cutlass::gemm::GemmShape<INST_M, INST_N, INST_K>;                                                          \
  using Gemm_##BLOCK_M##_##BLOCK_N##_##BLOCK_K##_##WARP_M##_##WARP_N##_##WARP_K##_##INST_M##_##INST_N##_##INST_K = \
      cutlass::gemm::device::Gemm<ElementInputA, LayoutInputA, ElementInputB, LayoutInputB,                     \
                                  ElementOutput, LayoutOutput, ElementAccumulator, MMAOp, SmArch,               \
                                  ShapeMMAThreadBlock_##BLOCK_M##_##BLOCK_N##_##BLOCK_K,                        \
                                  ShapeMMAWarp_##WARP_M##_##WARP_N##_##WARP_K,                                  \
                                  ShapeMMAOp_##INST_M##_##INST_N##_##INST_K, EpilogueOp,                        \
                                  SwizzleThreadBlock, NUM_STAGES>;
```

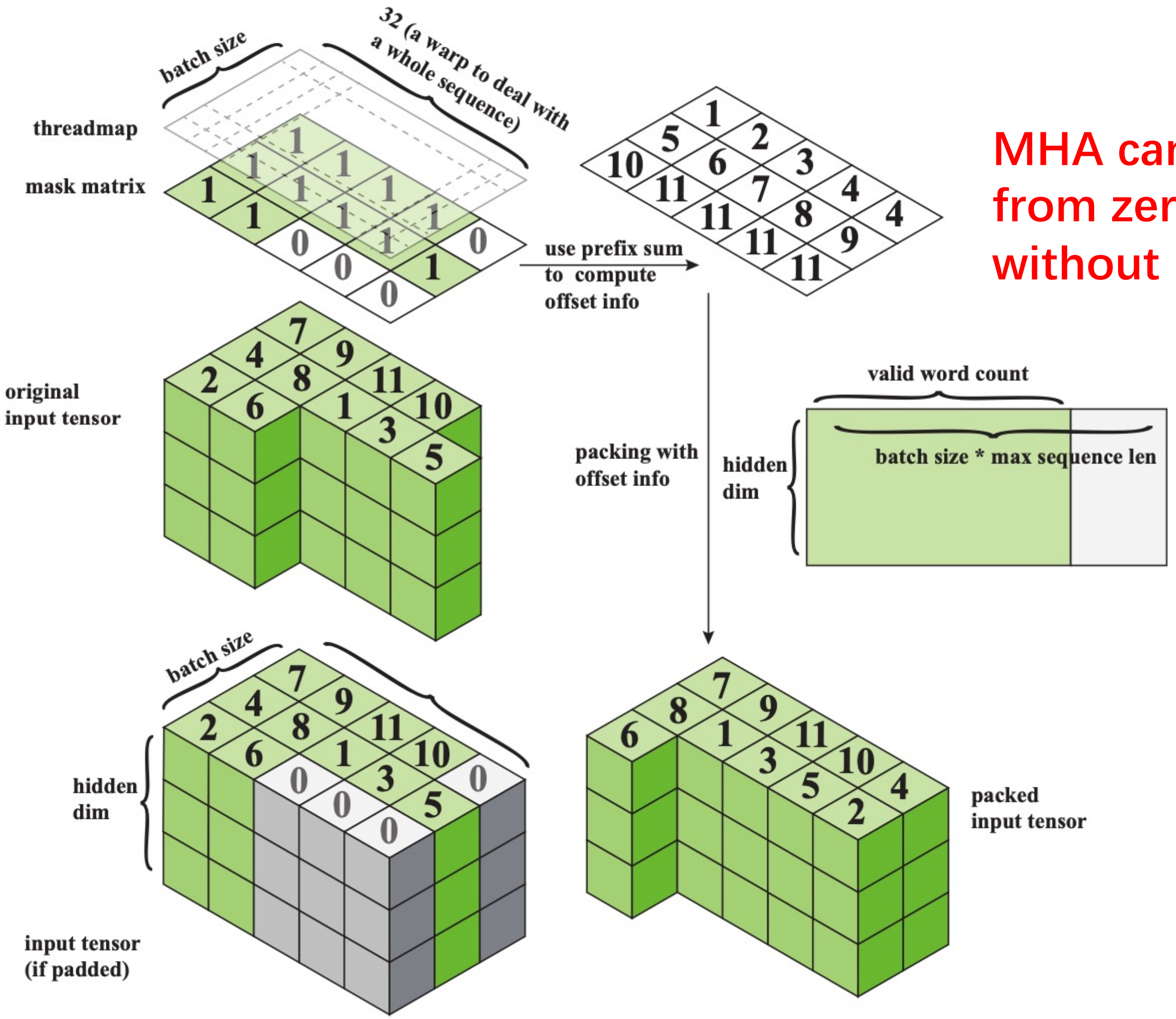
However, there is still computation redundancy!

Zero Padding Algorithm

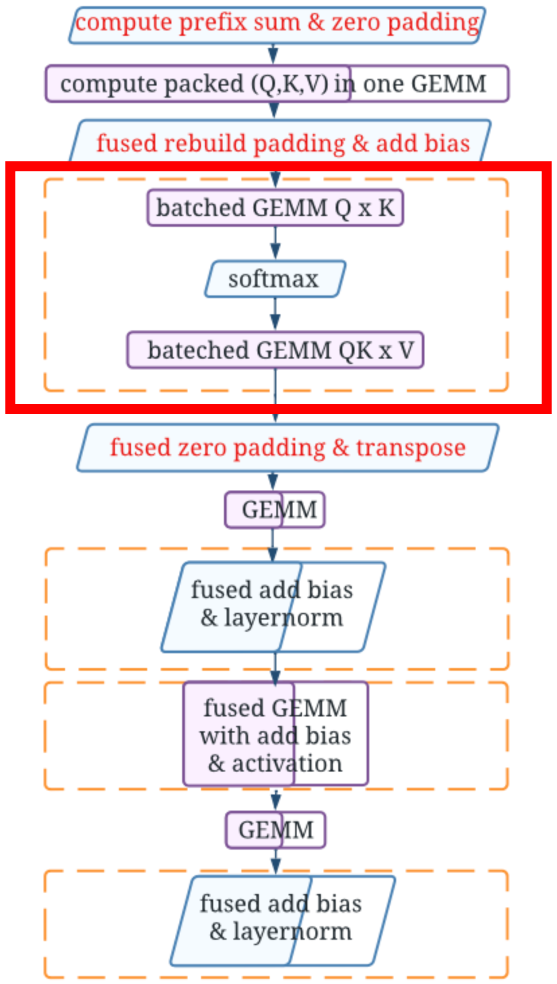




Zero Padding Algorithm

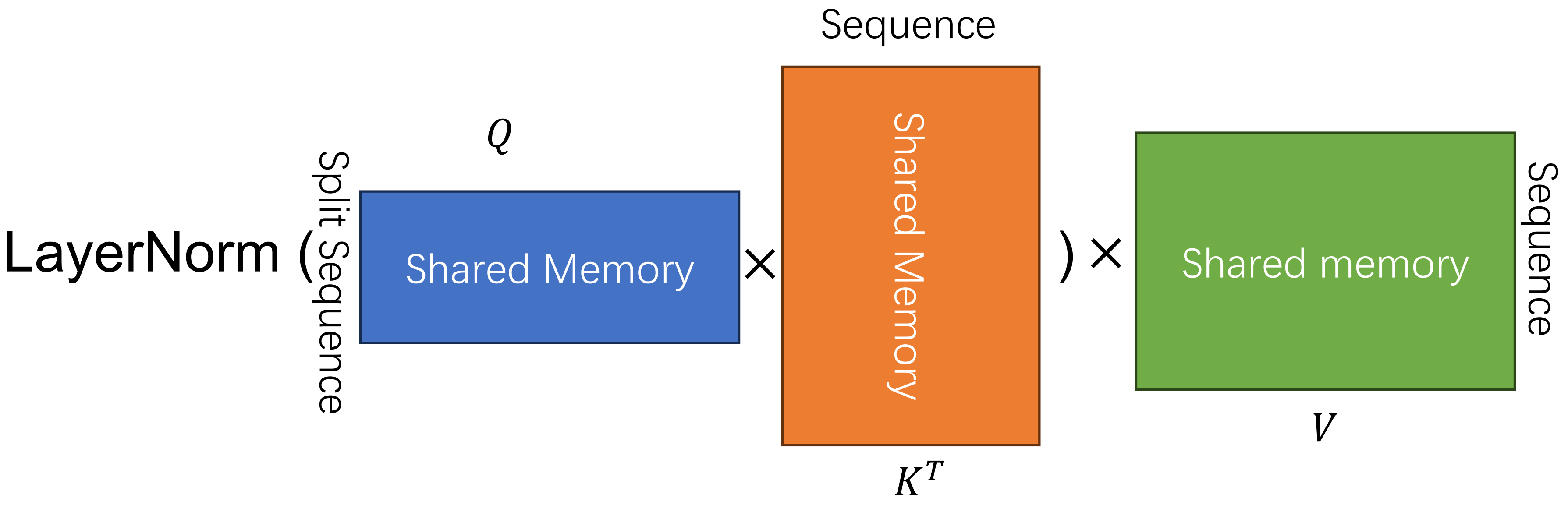


MHA cannot benefit from zero padding without modification



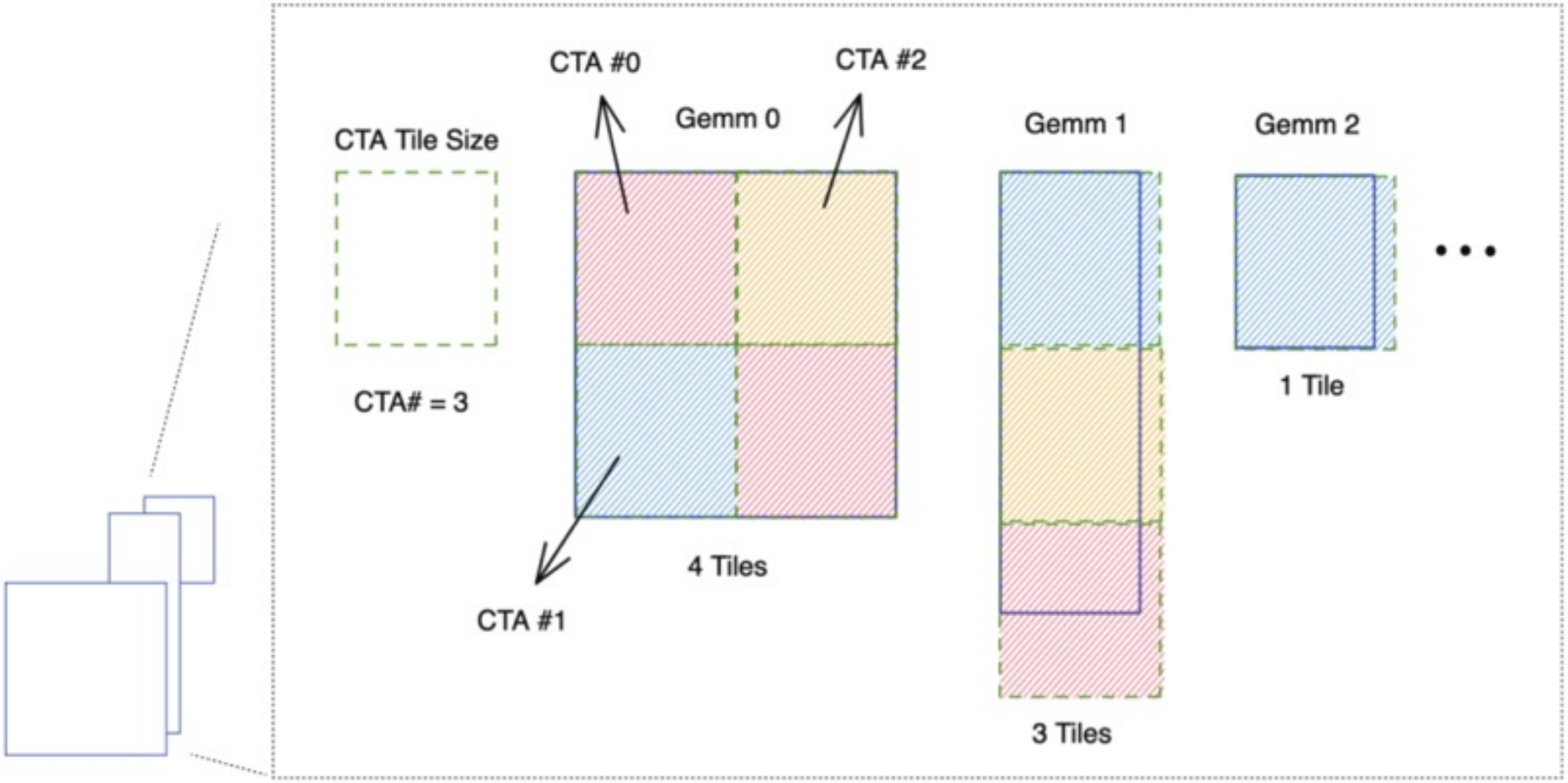
Fused Multi Head Attention: Short Sequence

Launch grid={head_num, seq_len / split_seq_len, batch_size}



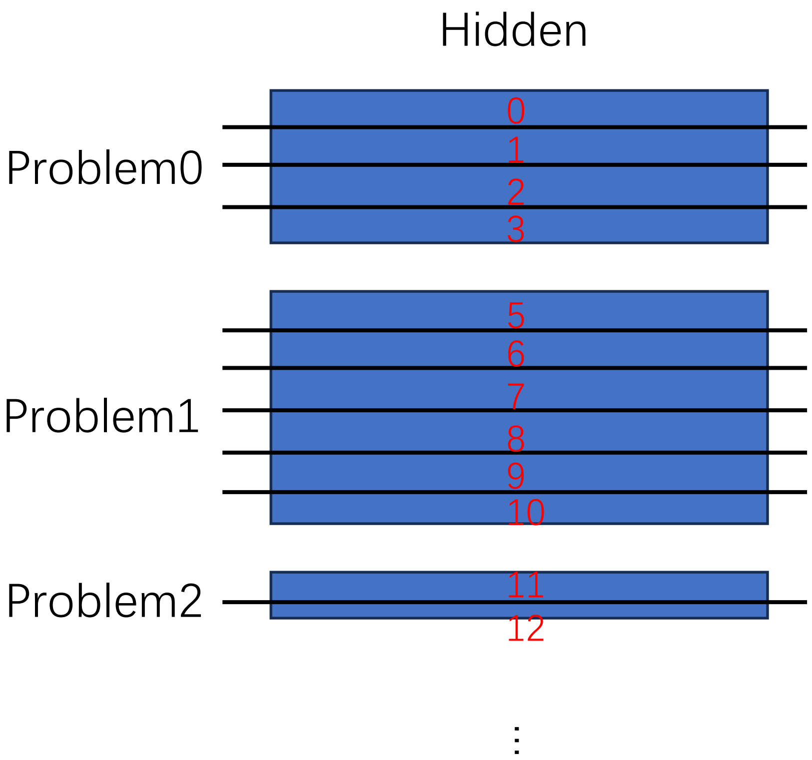
Fused Multi Head Attention: Long Sequence

Grouped GEMM



Fused Multi Head Attention: Long Sequence

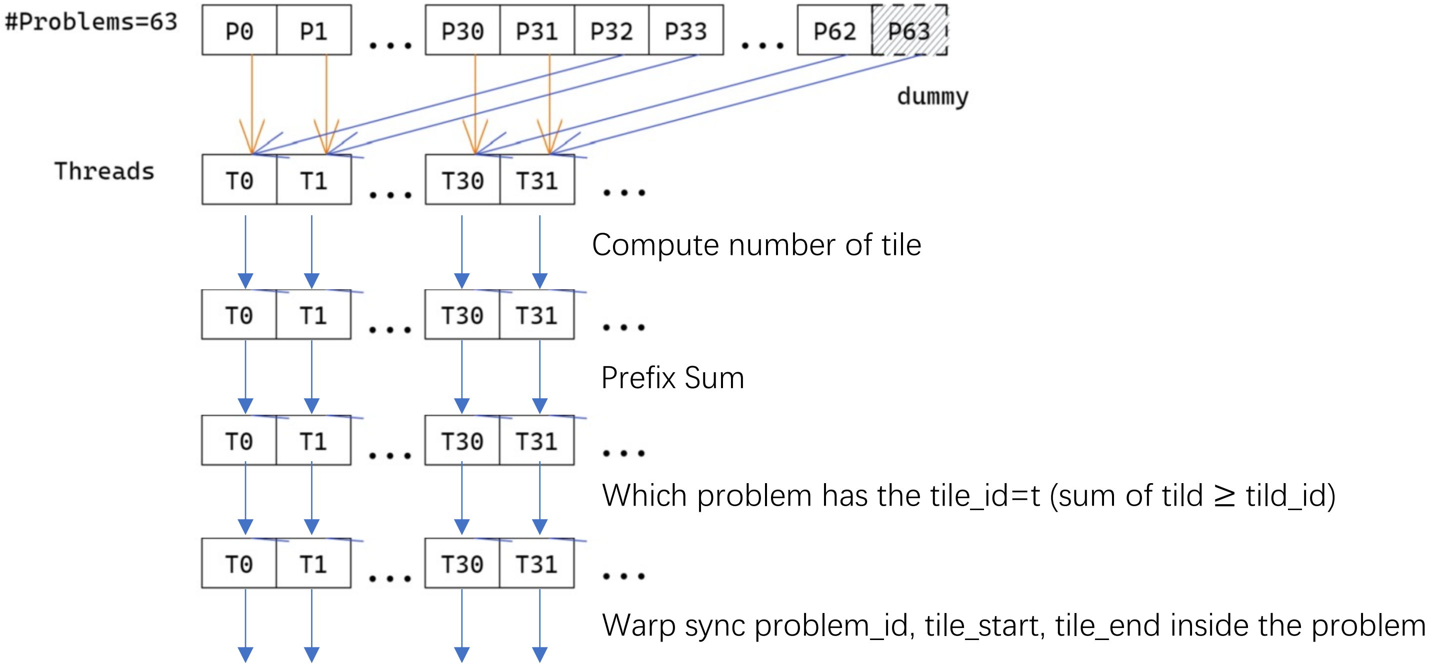
Tile Scheduling



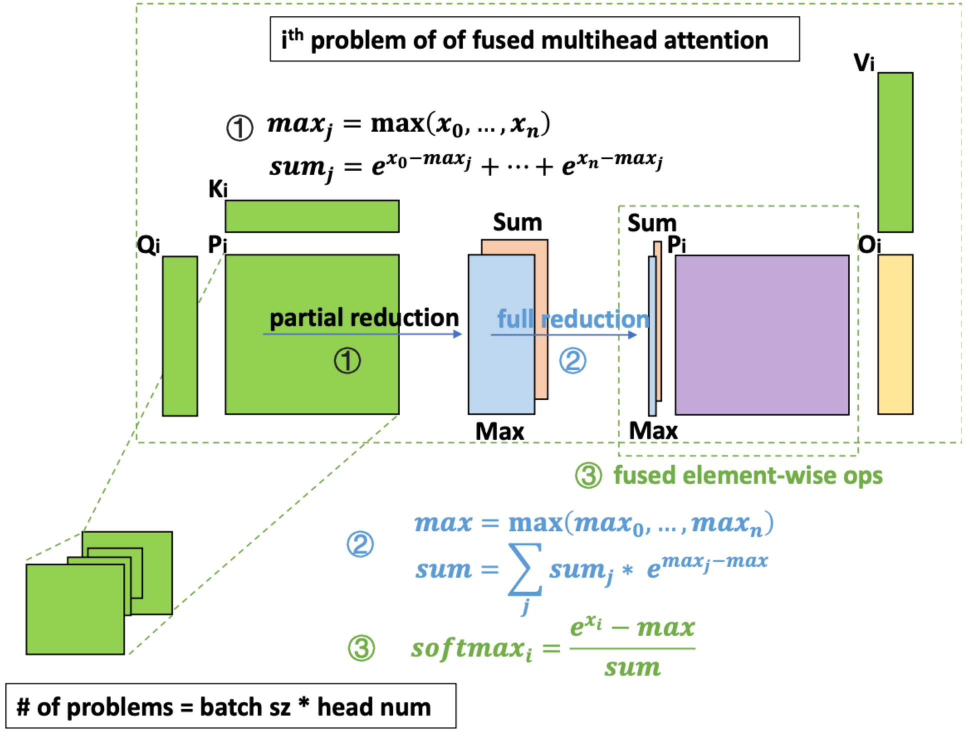
Every block handles one tile, suppose we have a tile_id=t, we need to know

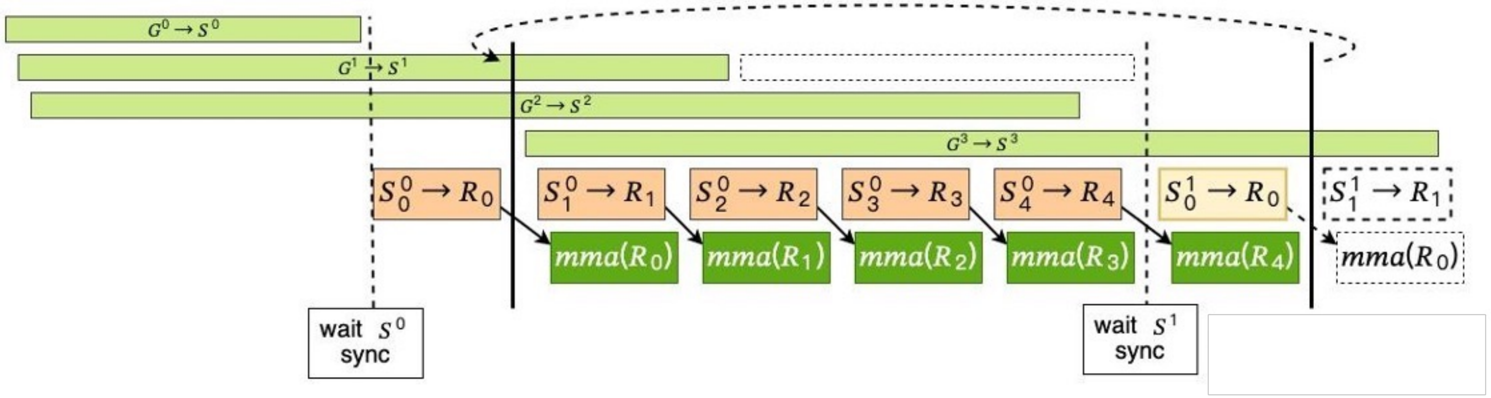
1. problem_id
2. Tile_offset inside that problem

To get the Q, K, V pointer and offset in Q

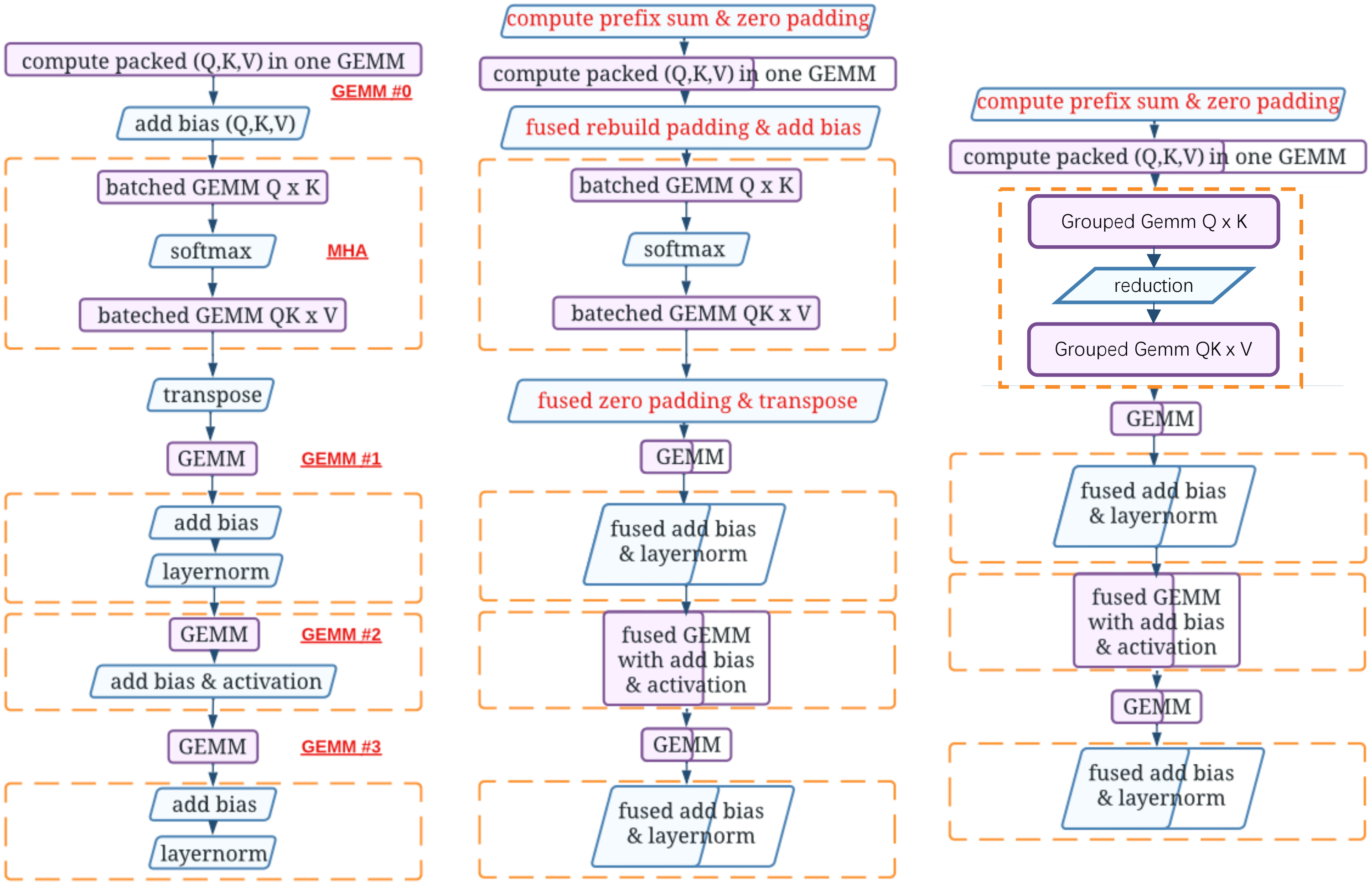


Fused Multi Head Attention: Long Sequence



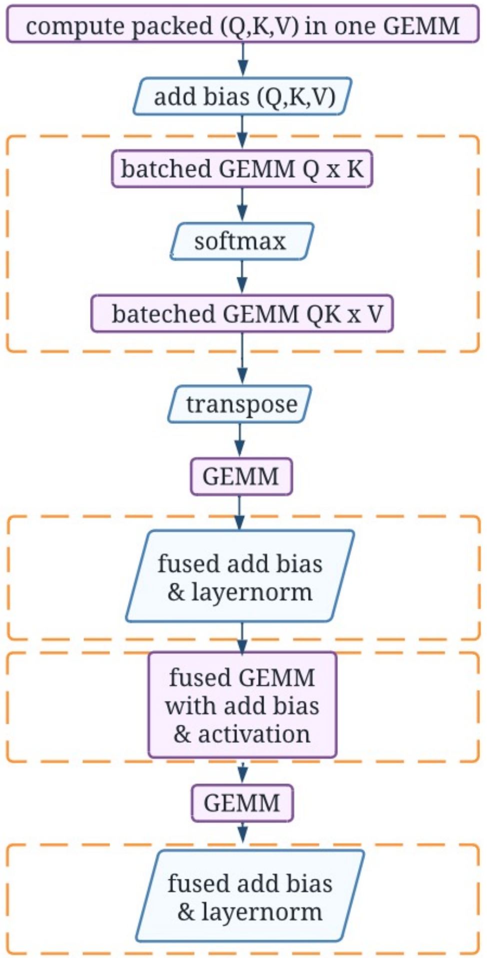


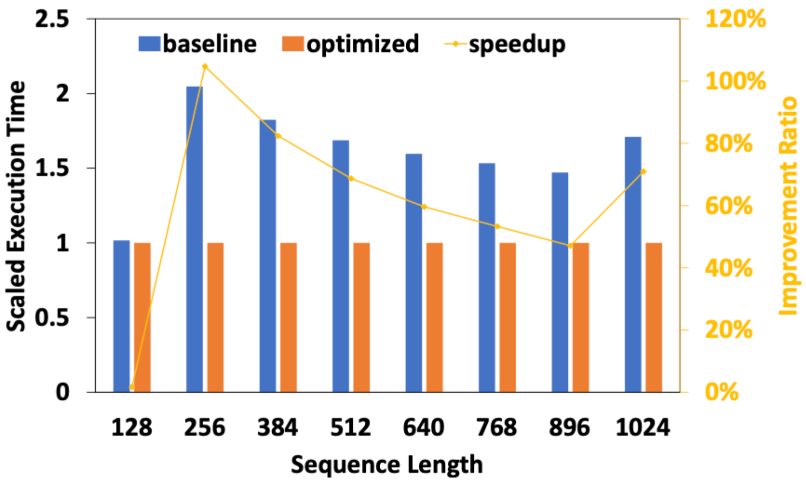
Stepwise Optimization

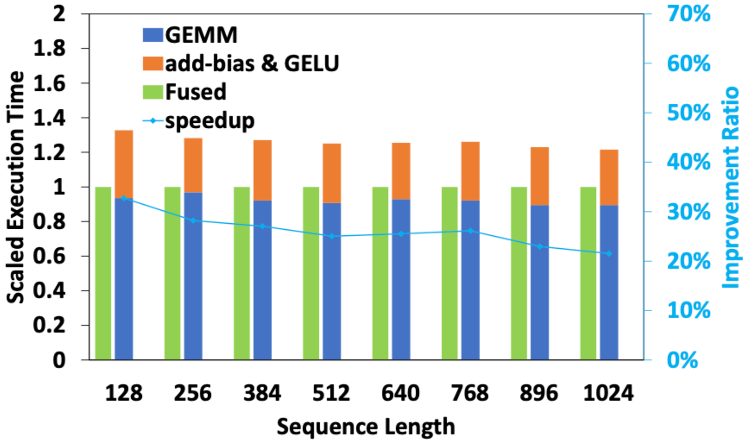


	Baseline	Zero Padding	Zero Padding + fused MHA
GEMM0	$6mk^2$	$6(\alpha \cdot m)k^2$	$6(\alpha \cdot m)k^2$
MHA	$4\frac{m^2}{bs}k$	$4\frac{m^2}{bs}k$	$4\frac{(\alpha \cdot m)^2}{bs}k$
GEMM1	$2mk^2$	$2(\alpha \cdot m)k^2$	$2(\alpha \cdot m)k^2$
GEMM2	$8mk^2$	$8(\alpha \cdot m)k^2$	$8(\alpha \cdot m)k^2$
GEMM3	$8mk^2$	$8(\alpha \cdot m)k^2$	$8(\alpha \cdot m)k^2$

Evaluation: Fusion Kernel







Evaluation: Fusion MHA

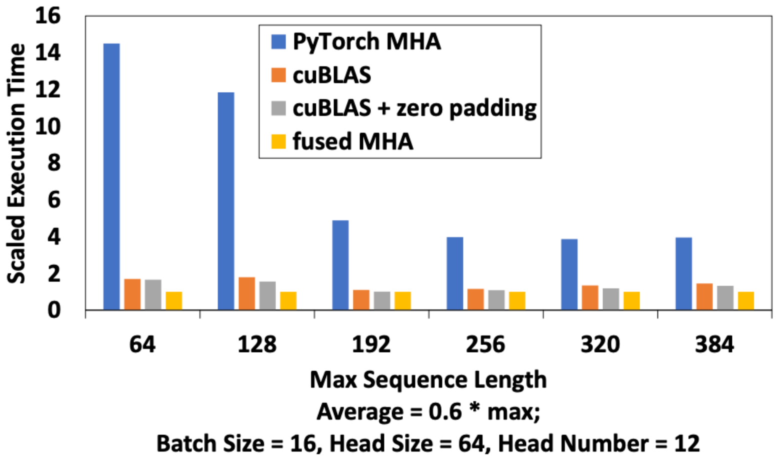


Fig. 11: Fused MHA for short sequences.

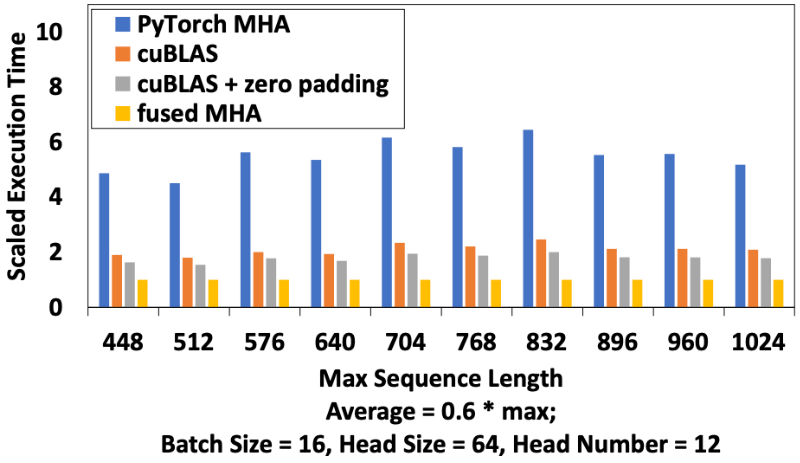


Fig. 12: Fused MHA for long sequences.

Evaluation: Fusion MHA

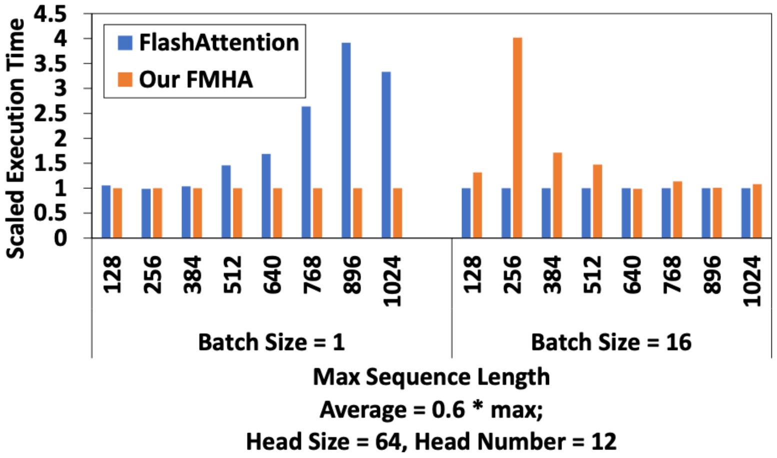
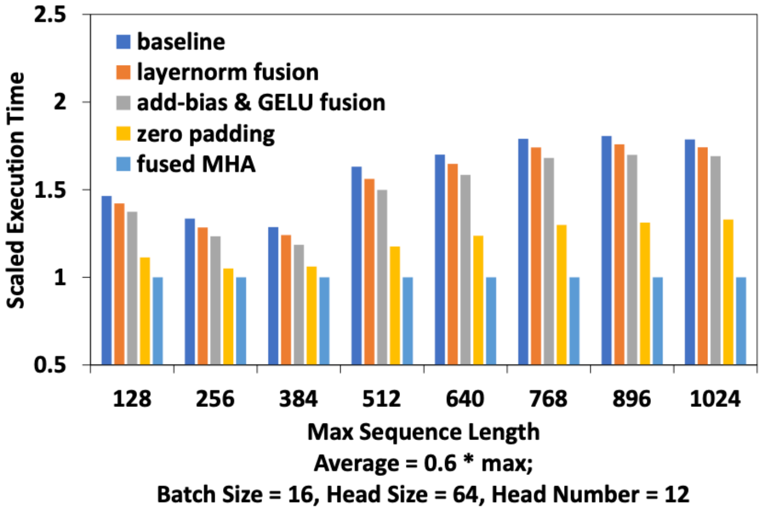
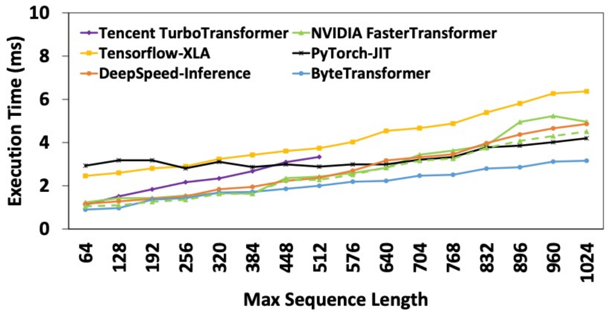


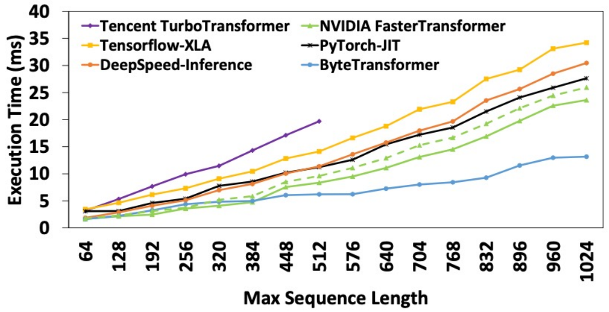
Fig. 13: Comparisons of our FMHA with FlashAttention.



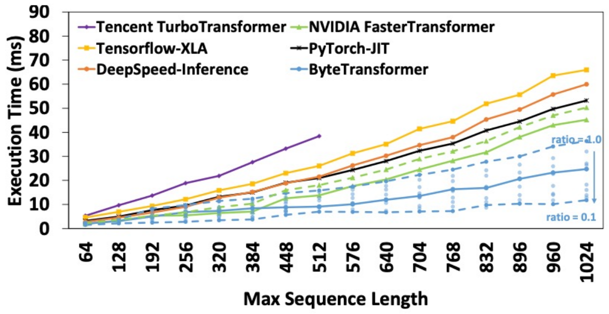
Evaluation: End-To-End



(a) Batch size = 1



(b) Batch size = 8



(c) Batch size = 16

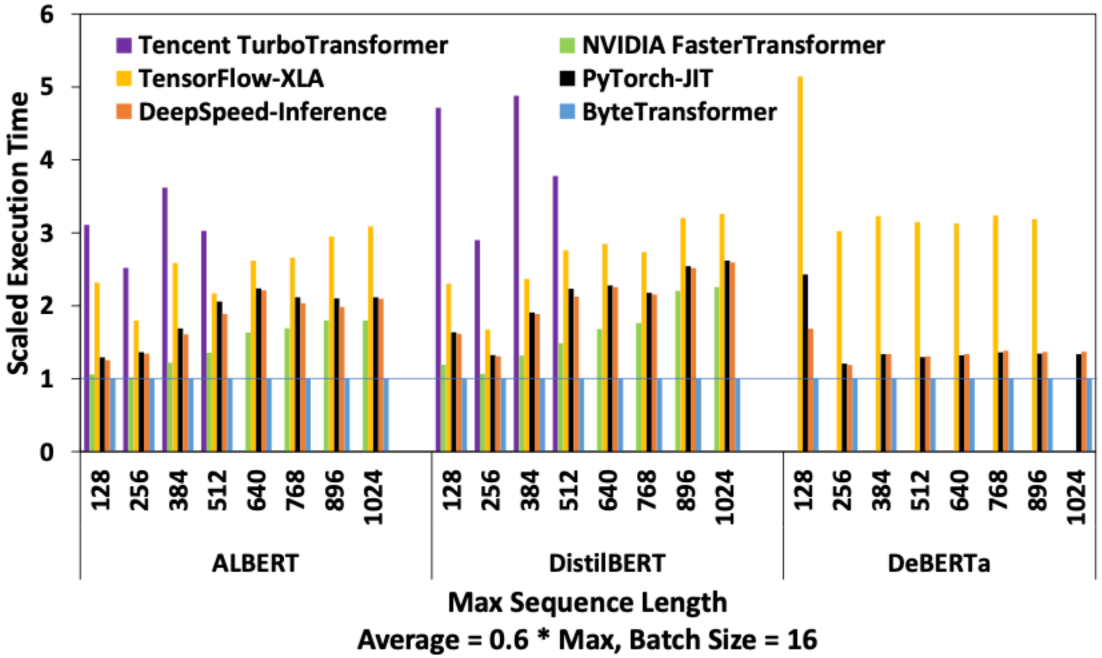


Fig. 16: End-to-end benchmark for other BERT-like models.

Discussion

ByteTransformer provides a high-performance implementation that supports variable sequence length input and achieves an average of 50% speedup end-to-end on different models.

However, there are other possible optimizations to concern…

- Tail Effect