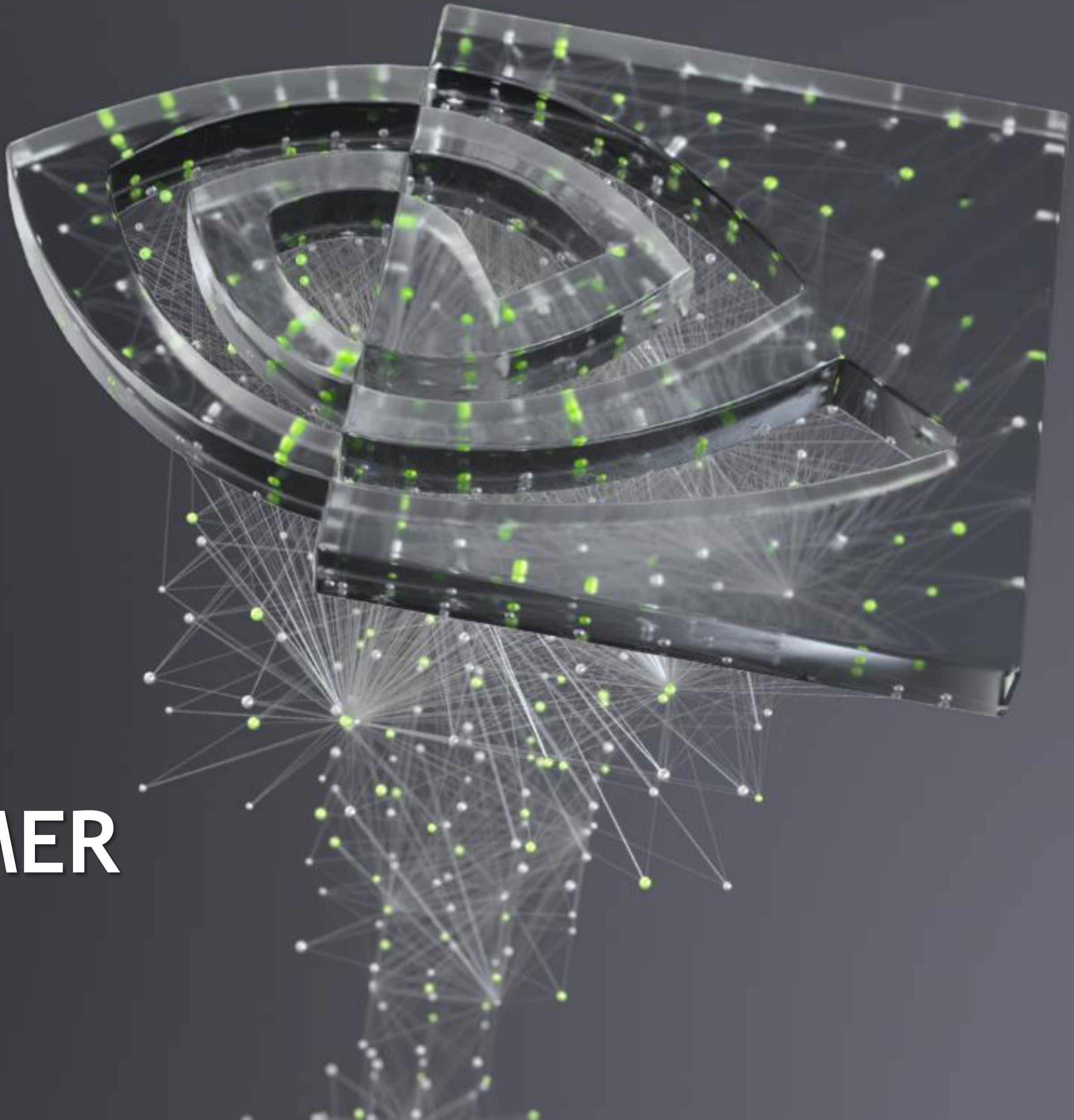


FASTER TRANSFORMER

Bo Yang Hsueh, 2019/12/18

AGENDA

What is Faster Transformer

Introduce the Transformer and Faster Transformer 1.0

New Features in Faster Transformer 2.0

Introduce the Faster Transformer 2.0

Faster Transformer 2.0 performance

Demonstrate the performance of Faster

Network Pruning

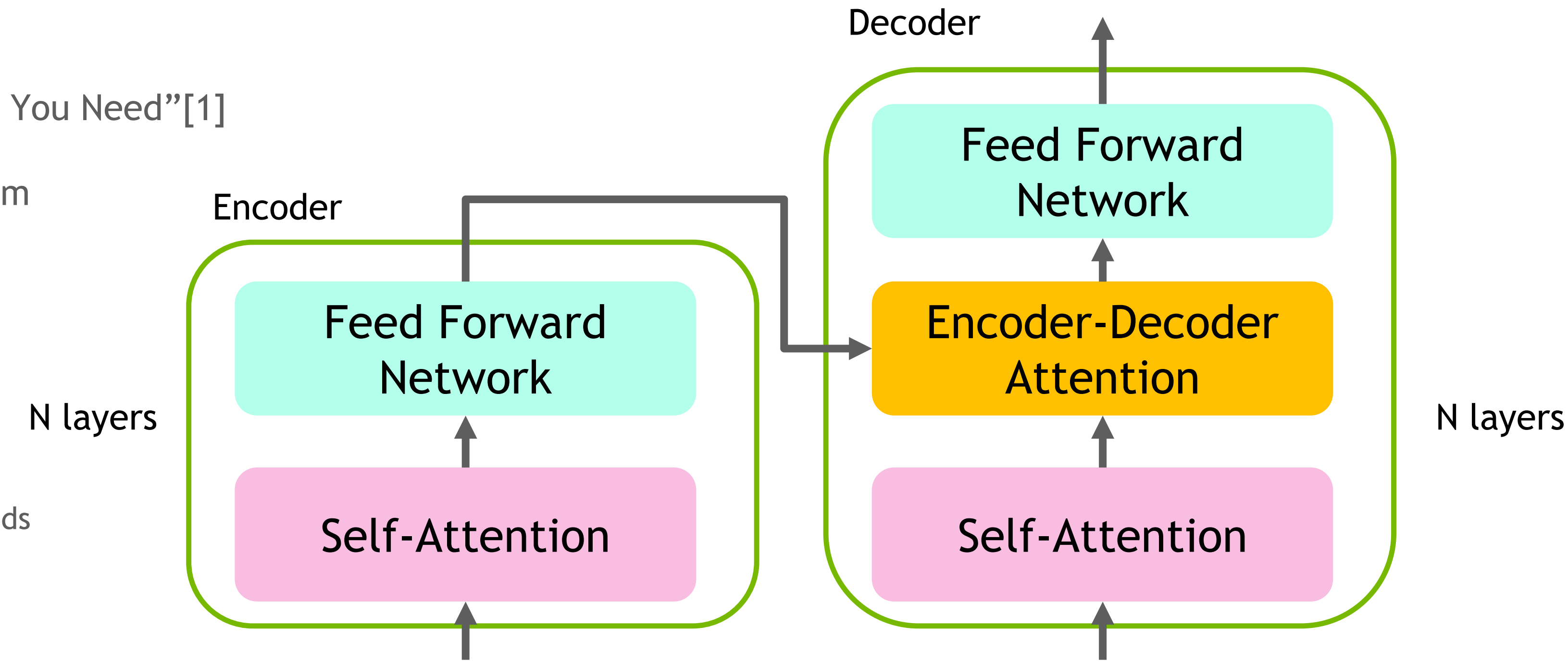
Q&A time

WHAT IS FASTER TRANSFORMER

WHAT IS FASTER TRANSFORMER

What is Transformer

- Proposed in “Attention Is All You Need”[1]
- Only use attention mechanism
- Application:
    - QA
    - Online classification
    - Search: Relationship of ads

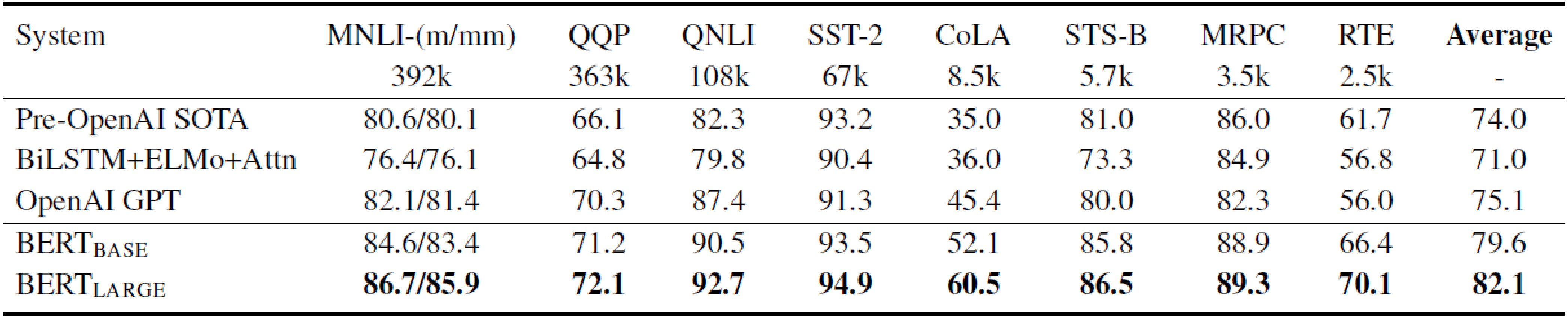


[1] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

WHAT IS FASTER TRANSFORMER

What is Transformer

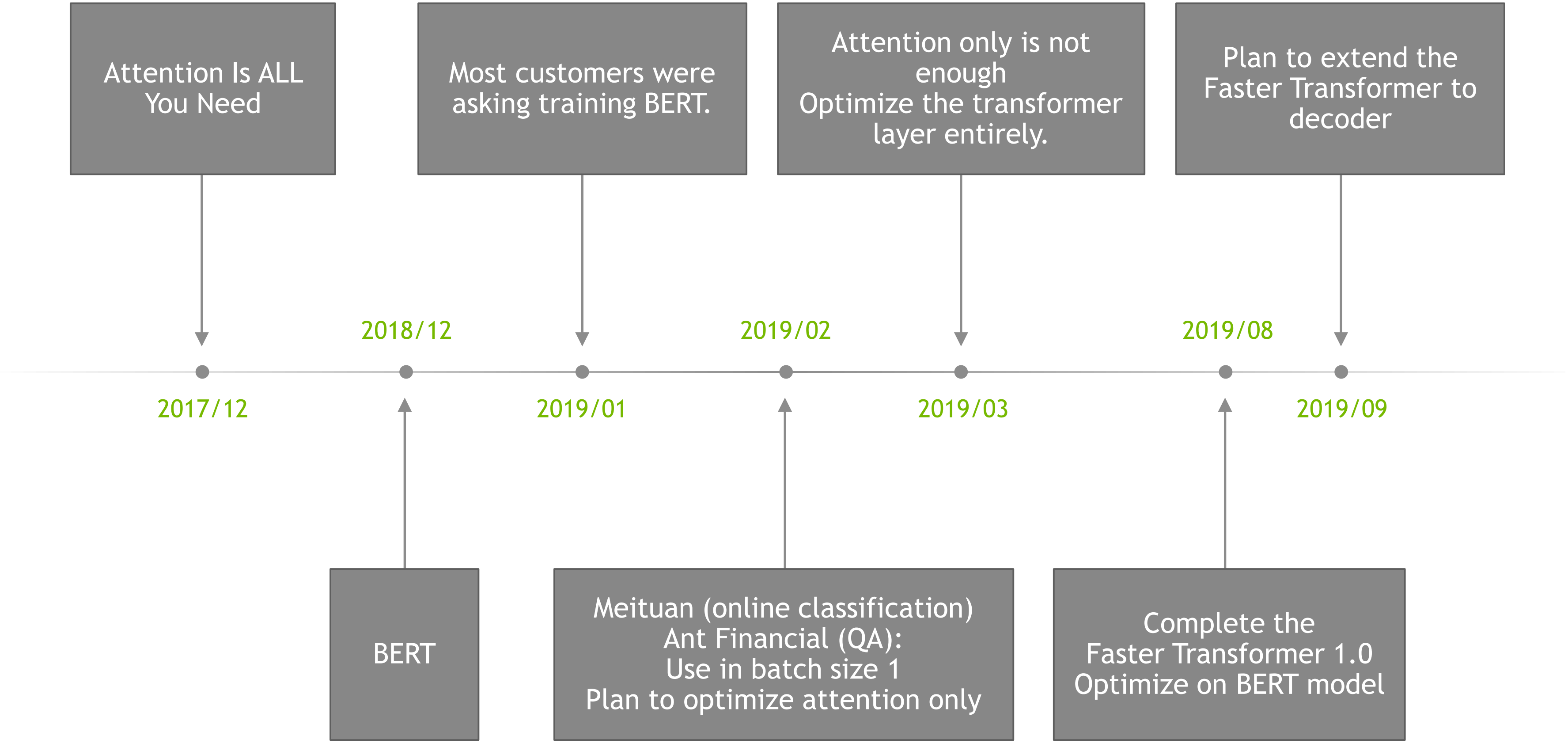
- Transformer is the major component in BERT
- BERT is proposed in 2018, and become the state-of-the-art method in the time
- However, the model is too large, and is hard to satisfy the latency requirement in real application



System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT$_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT$_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

[1] Devlin, J., Chang, M.W., Lee, K. and Toutanova, K., 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

LONG STORY OF FASTER TRANSFORMER



2017/12

Attention Is ALL You Need

2018/12

BERT

2019/01

Most customers were asking training BERT.

2019/02

Meituan (online classification)
Ant Financial (QA):
Use in batch size 1
Plan to optimize attention only

2019/03

Attention only is not enough
Optimize the transformer layer entirely.

2019/08

Complete the
Faster Transformer 1.0
Optimize on BERT model

2019/09

Plan to extend the Faster Transformer to decoder

FASTER TRANSFORMER 1.0 FEATURES

Optimize the encoder

- An equivalent forward implementation of the BERT transformer layer
    - Single layer, forward only
    - Based on top of CUDA + cuBLAS
    - Support FP32/FP16 on NVIDIA Tesla P4/V100/T4
    - Arbitrary batch size, sequence length 32/64/128
    - Basic model (12 * 64 heads) or smaller (4 * 32 heads)
    - Provide C++/TensorRT plugin/TensorFlow OP API

FASTER TRANSFORMER 1.0 DETAIL

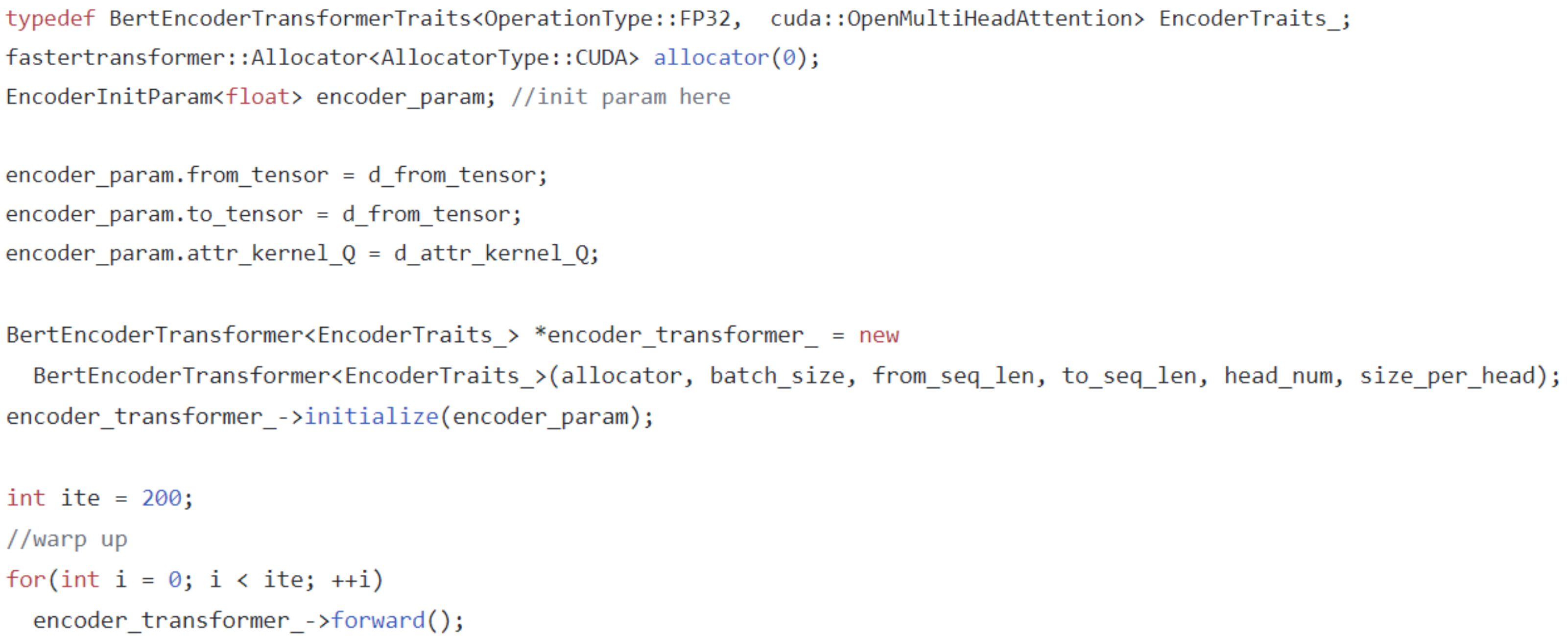
What we do in Faster Transformer 1.0?

- TensorFlow will split operation into many basic operation
  - E.g. split layer norm into add, sub, mean, sqrt, …
  - Kernel launch overhead
- Fuse the operations except GEMM as much as possible
  - add bias + layer norm
  - add bias + activation
  - Transpose 3 matrices together in attention
  - …

FASTER TRANSFORMER 1.0 DETAIL

How to use Faster Transformer?

- Provide C, Tensorflow and TensorRT API
- Provide sample codes to demonstrate how to use
- In C:



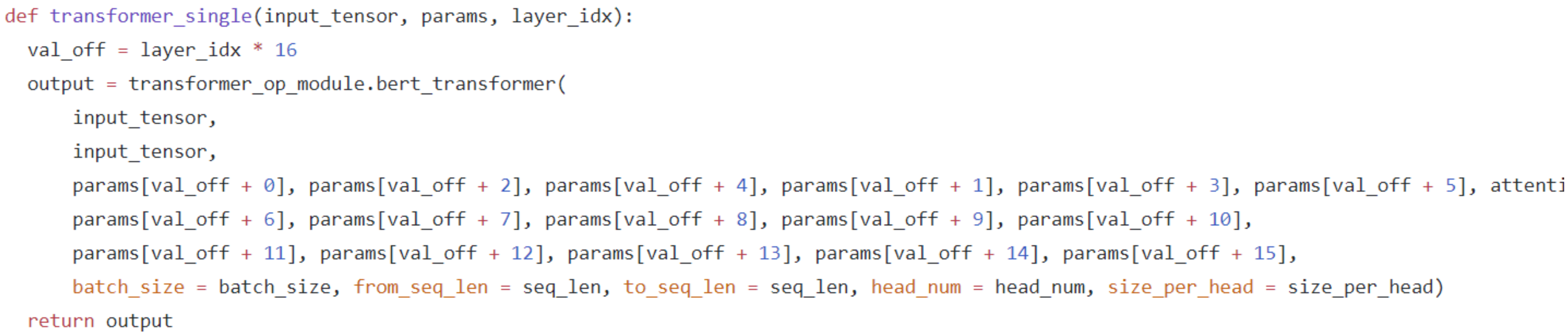
```
typedef BertEncoderTransformerTraits<OperationType::FP32,  cuda::OpenMultiHeadAttention> EncoderTraits_;
fastertransformer::Allocator<AllocatorType::CUDA> allocator(0);
EncoderInitParam<float> encoder_param; //init param here

encoder_param.from_tensor = d_from_tensor;
encoder_param.to_tensor = d_from_tensor;
encoder_param.attr_kernel_Q = d_attr_kernel_Q;

BertEncoderTransformer<EncoderTraits_> *encoder_transformer_ = new
  BertEncoderTransformer<EncoderTraits_>(allocator, batch_size, from_seq_len, to_seq_len, head_num, size_per_head);
encoder_transformer_->initialize(encoder_param);

int ite = 200;
//warp up
for(int i = 0; i < ite; ++i)
  encoder_transformer_->forward();
```

FASTER TRANSFORMER 1.0 DETAIL

How to use Faster Transformer?

- Provide C, Tensorflow and TensorRT API
- Provide sample codes to demonstrate how to use
- In TensorFlow:

```
transformer_op_module = tf.load_op_library(os.path.join('./lib/libtf_fastertransformer.so'))
```



```
def transformer_single(input_tensor, params, layer_idx):
  val_off = layer_idx * 16
  output = transformer_op_module.bert_transformer(
      input_tensor,
      input_tensor,
      params[val_off + 0], params[val_off + 2], params[val_off + 4], params[val_off + 1], params[val_off + 3], params[val_off + 5], attenti
      params[val_off + 6], params[val_off + 7], params[val_off + 8], params[val_off + 9], params[val_off + 10],
      params[val_off + 11], params[val_off + 12], params[val_off + 13], params[val_off + 14], params[val_off + 15],
      batch_size = batch_size, from_seq_len = seq_len, to_seq_len = seq_len, head_num = head_num, size_per_head = size_per_head)
  return output
```

FASTER TRANSFORMER 1.0 SUMMARY

- Faster Transformer 1.0 speedup about 1.5x compare to TensorFlow with XLA on FP16
- Faster Transformer 1.0 is released in https://github.com/NVIDIA/DeepLearningExamples/tree/master/FasterTransformer
- Currently, we only optimize the encoder, what about decoder?

WHY WE NEED TO OPTIMIZE DECODER

Encoder v.s. Decoder

- Encoder: Compute entire sentence in one time
  - Few large matrix multiplication
  - E.g., one time for a length 128 sentence
- Decoder: Compute word by word, sequence length times
  - Many small matrix multiplication
  - E.g., 128 times for a length 128 sentence

WHY WE NEED TO OPTIMIZE DECODER

Translating Progress

I

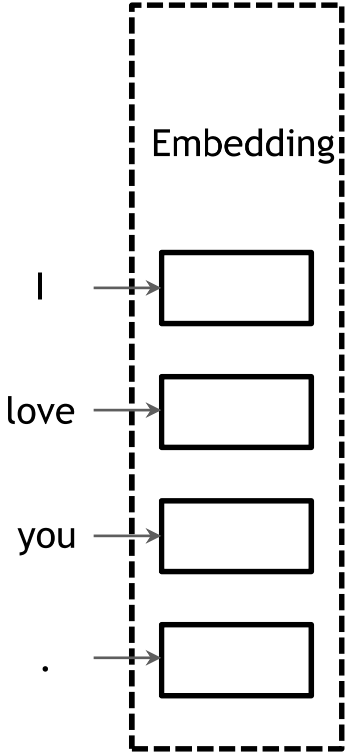
love

you

.

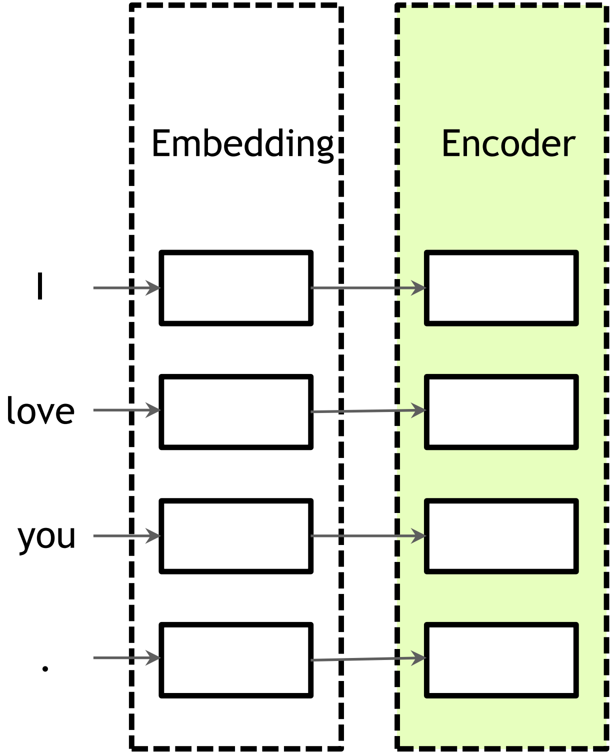
WHY WE NEED TO OPTIMIZE DECODER

Translating Progress



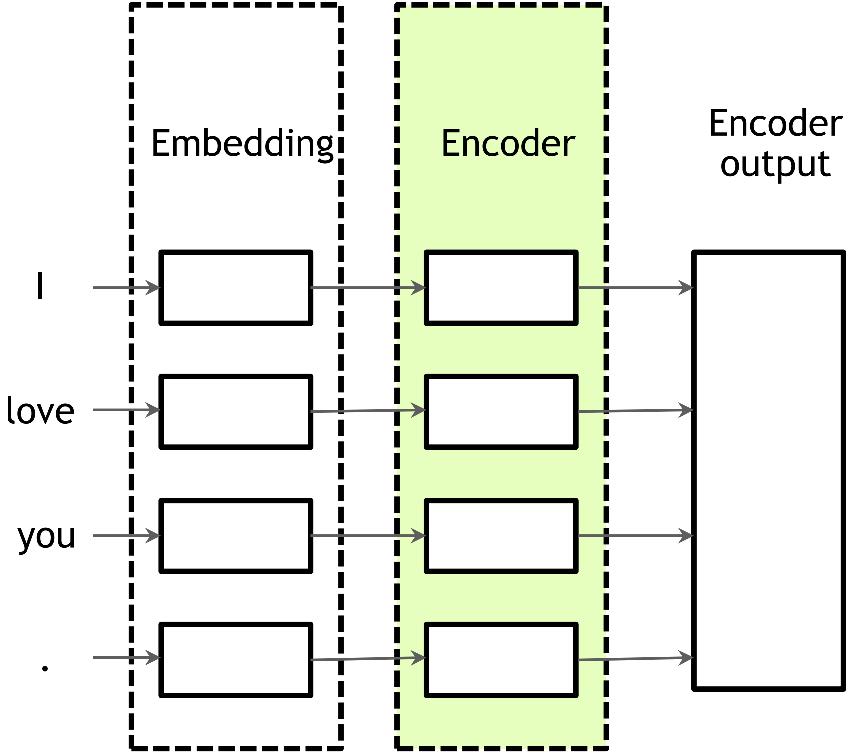
WHY WE NEED TO OPTIMIZE DECODER

Translating Progress



WHY WE NEED TO OPTIMIZE DECODER

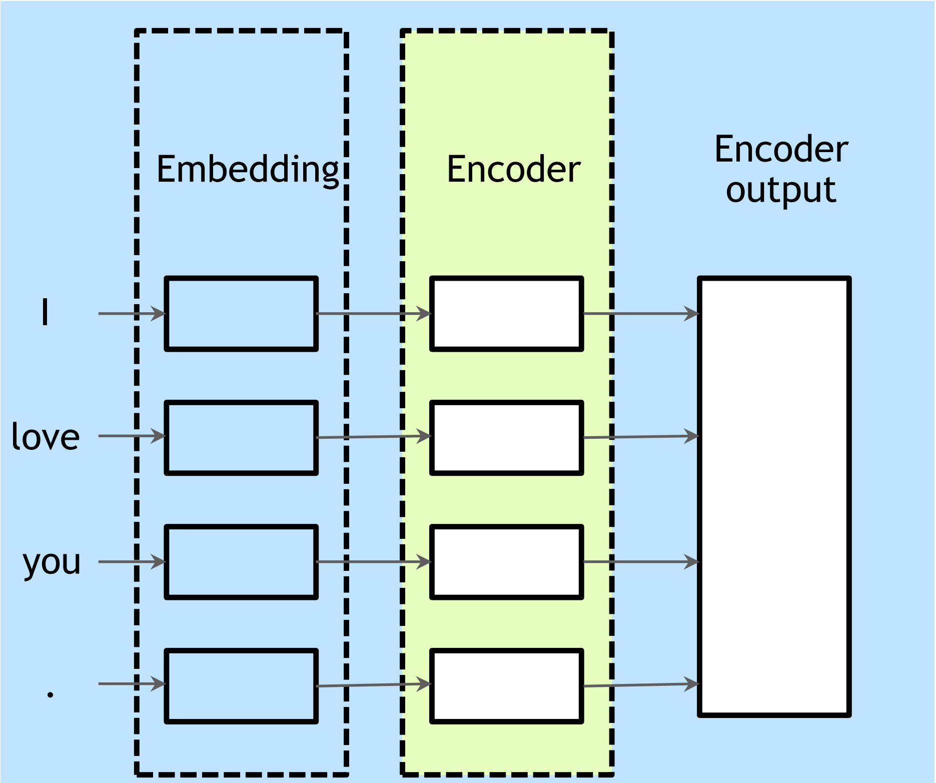
Translating Progress



WHY WE NEED TO OPTIMIZE DECODER

Translating Progress

Encoder progress

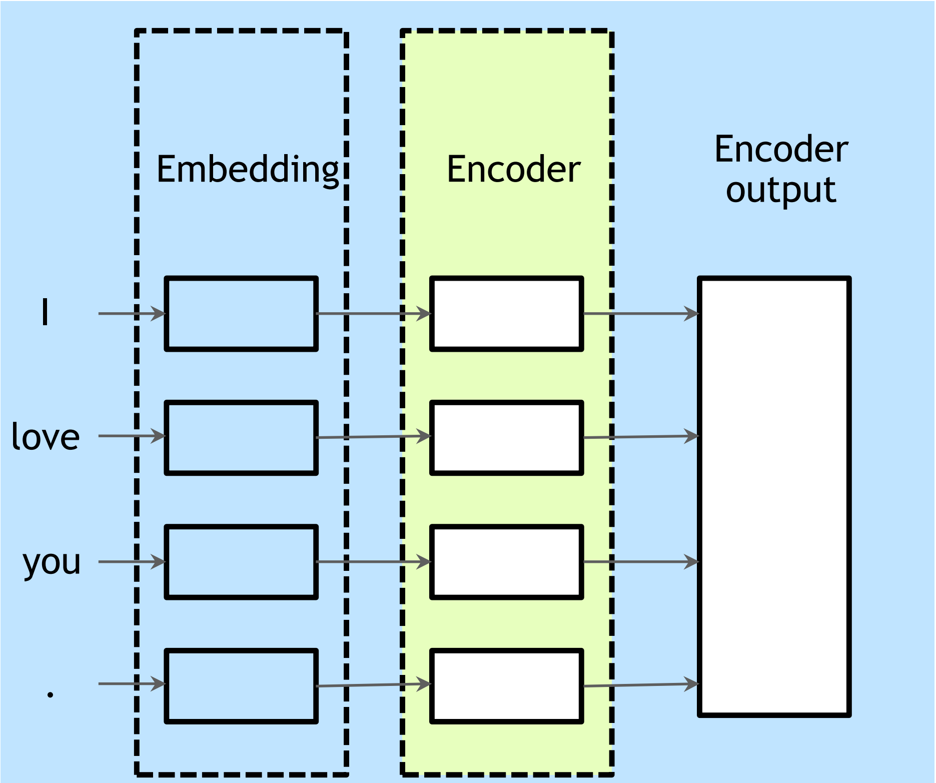


WHY WE NEED TO OPTIMIZE DECODER

Translating Progress

Encoder progress

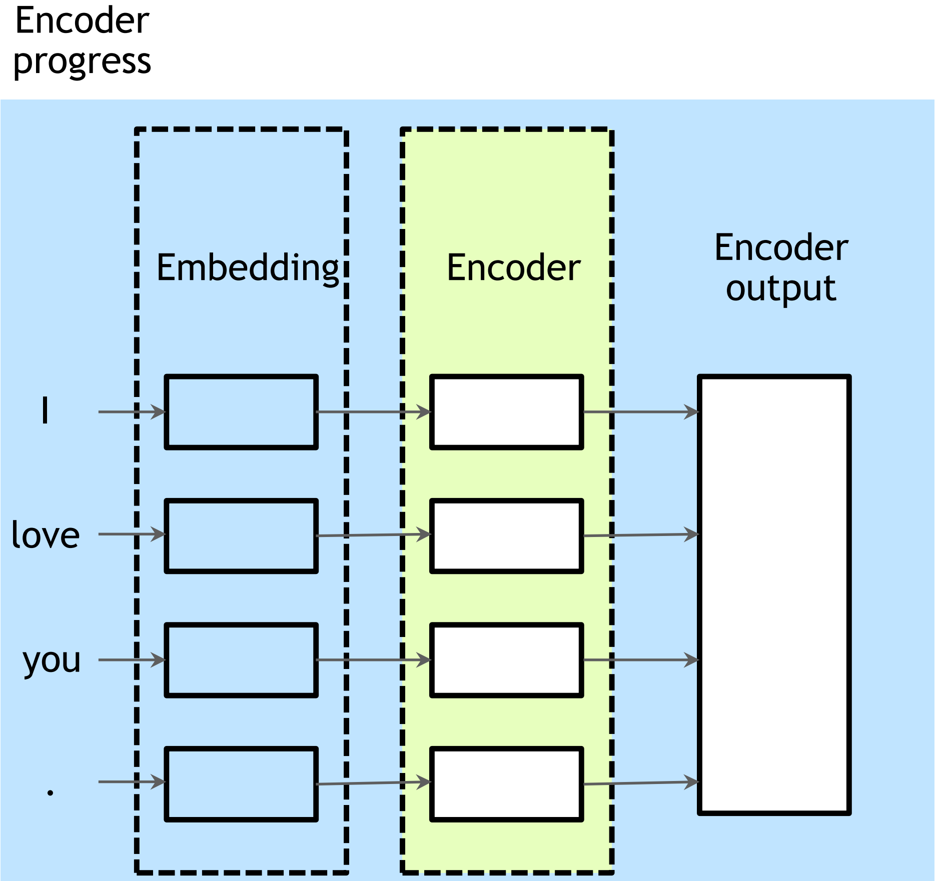
Decoder progress

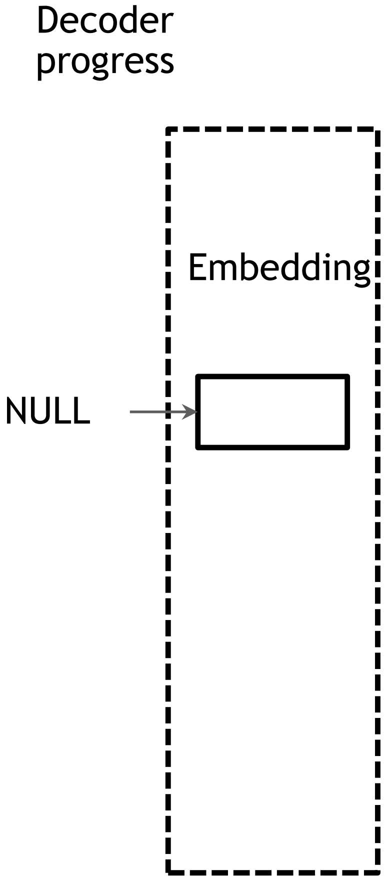


NULL

WHY WE NEED TO OPTIMIZE DECODER

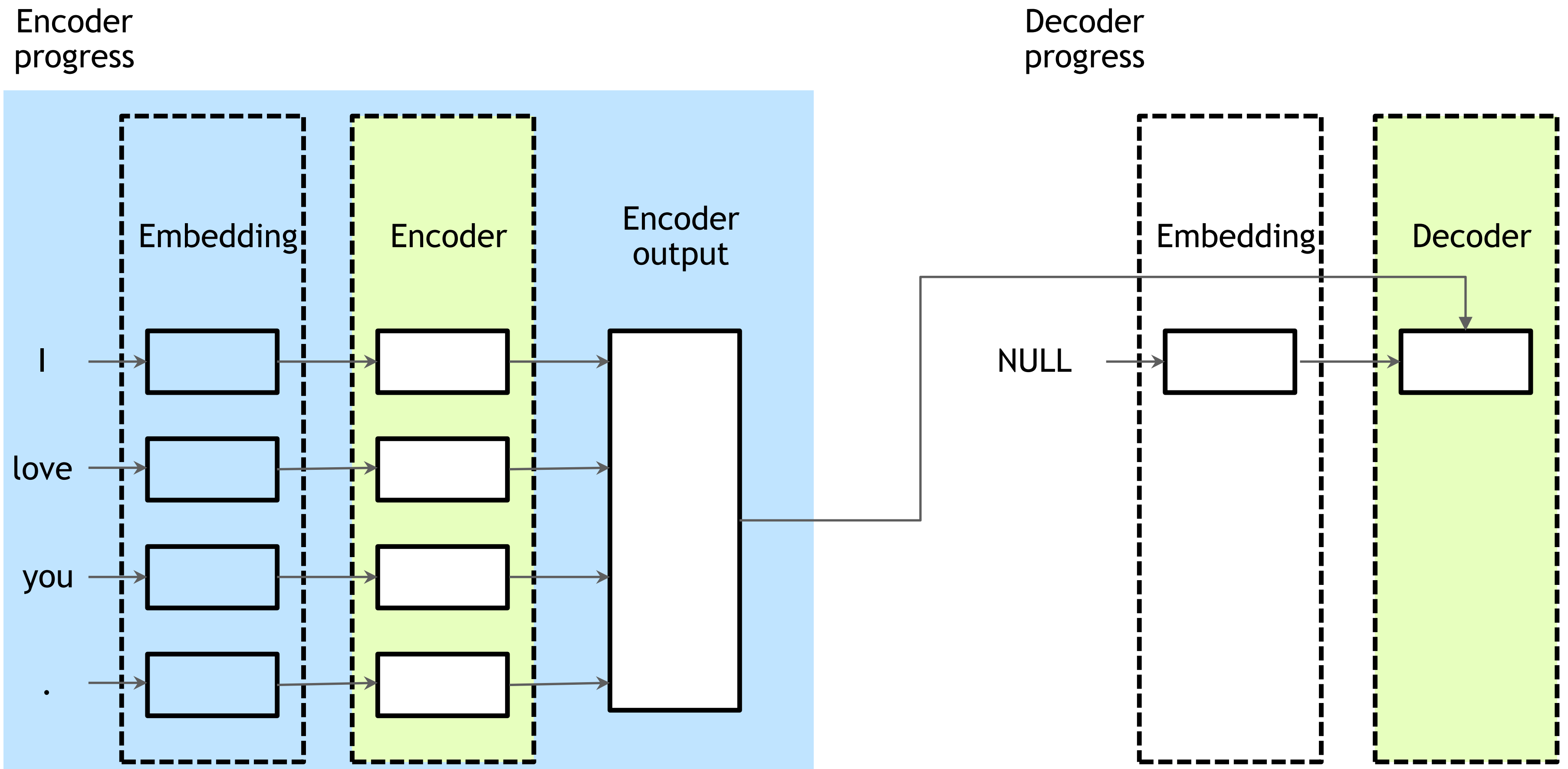
Translating Progress





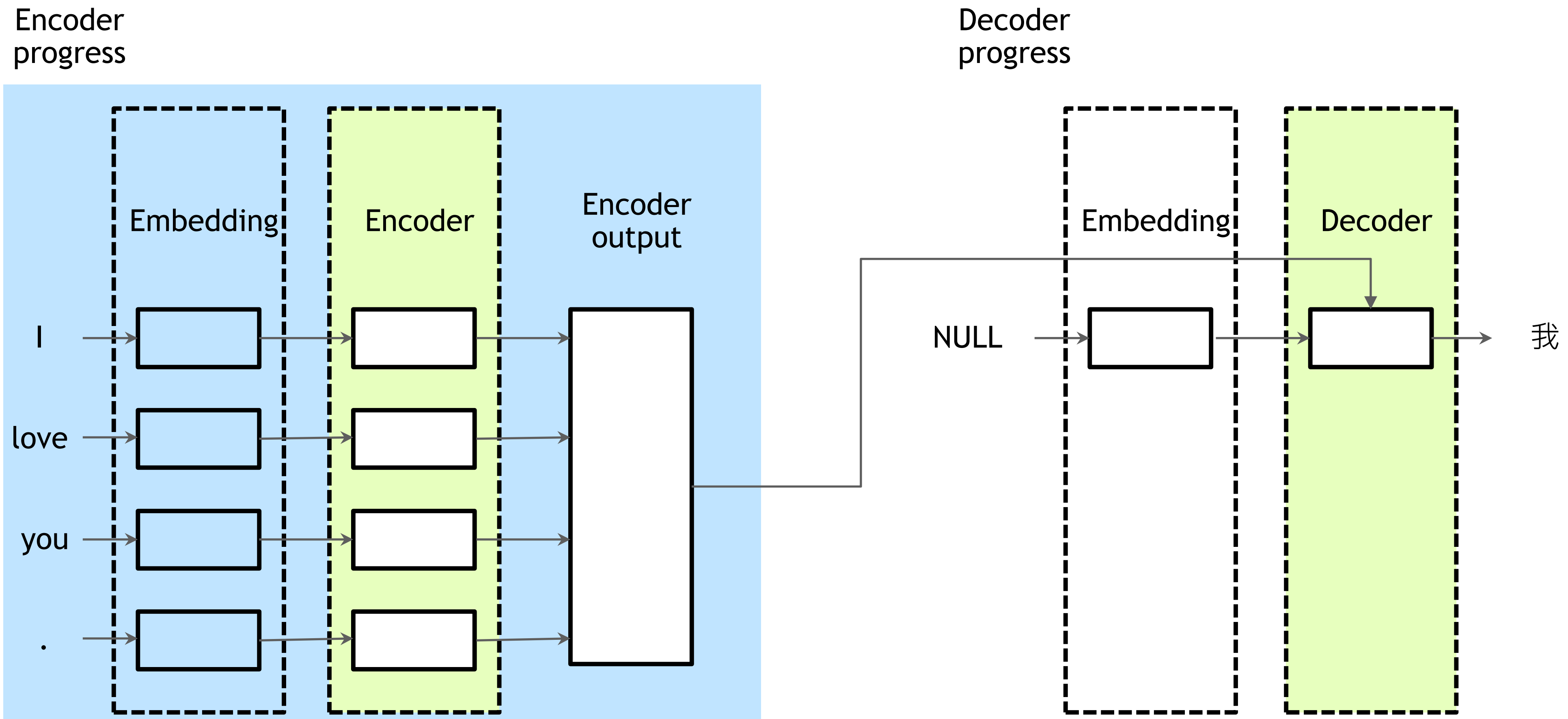
WHY WE NEED TO OPTIMIZE DECODER

Translating Progress



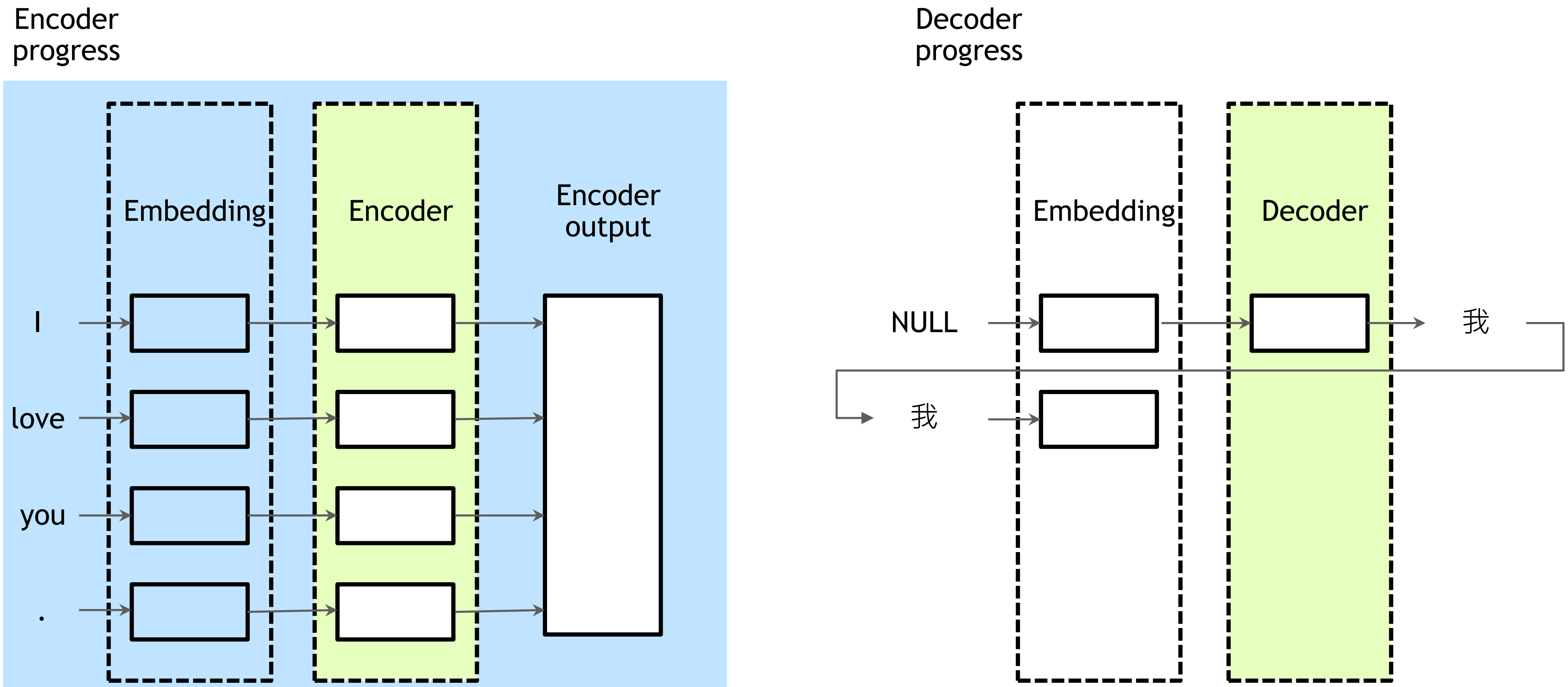
WHY WE NEED TO OPTIMIZE DECODER

Translating Progress



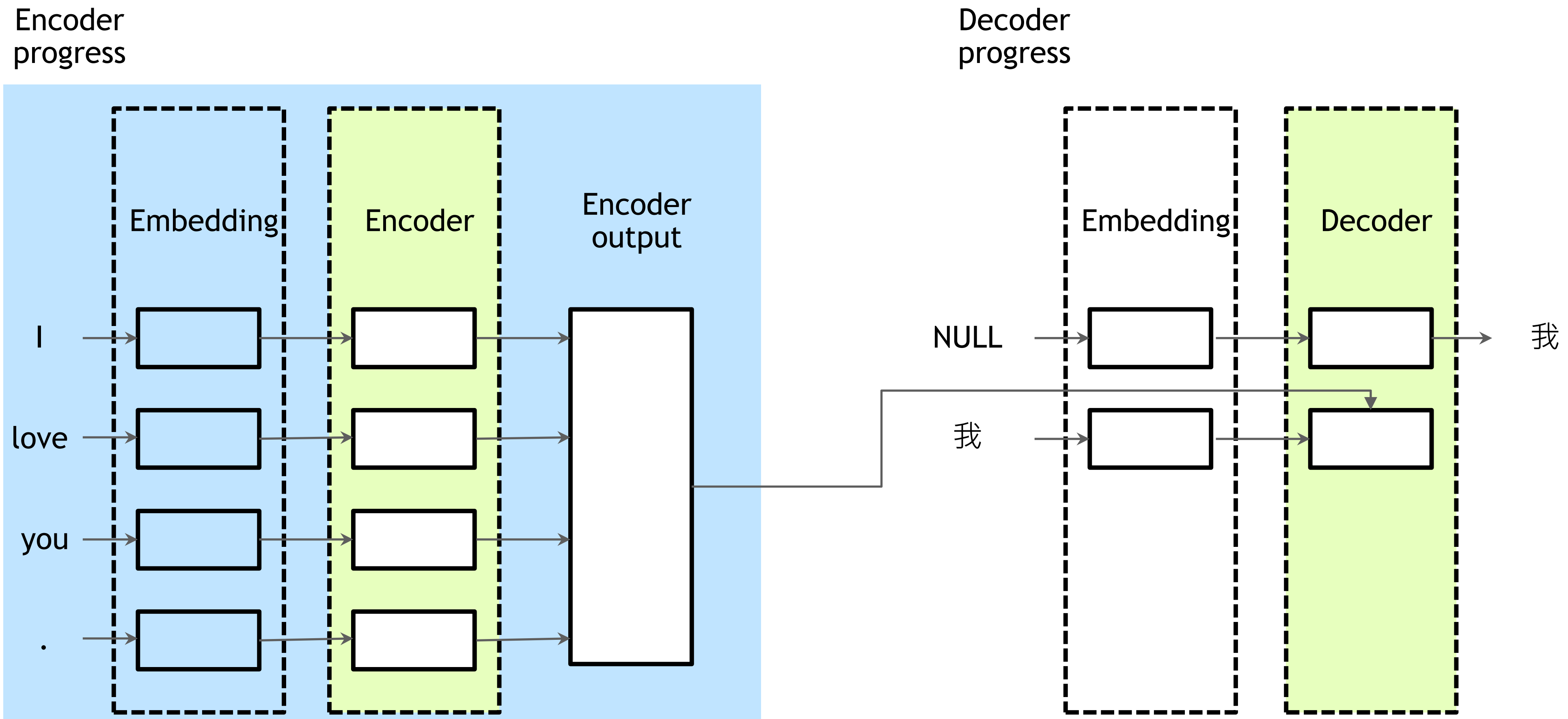
WHY WE NEED TO OPTIMIZE DECODER

Translating Progress



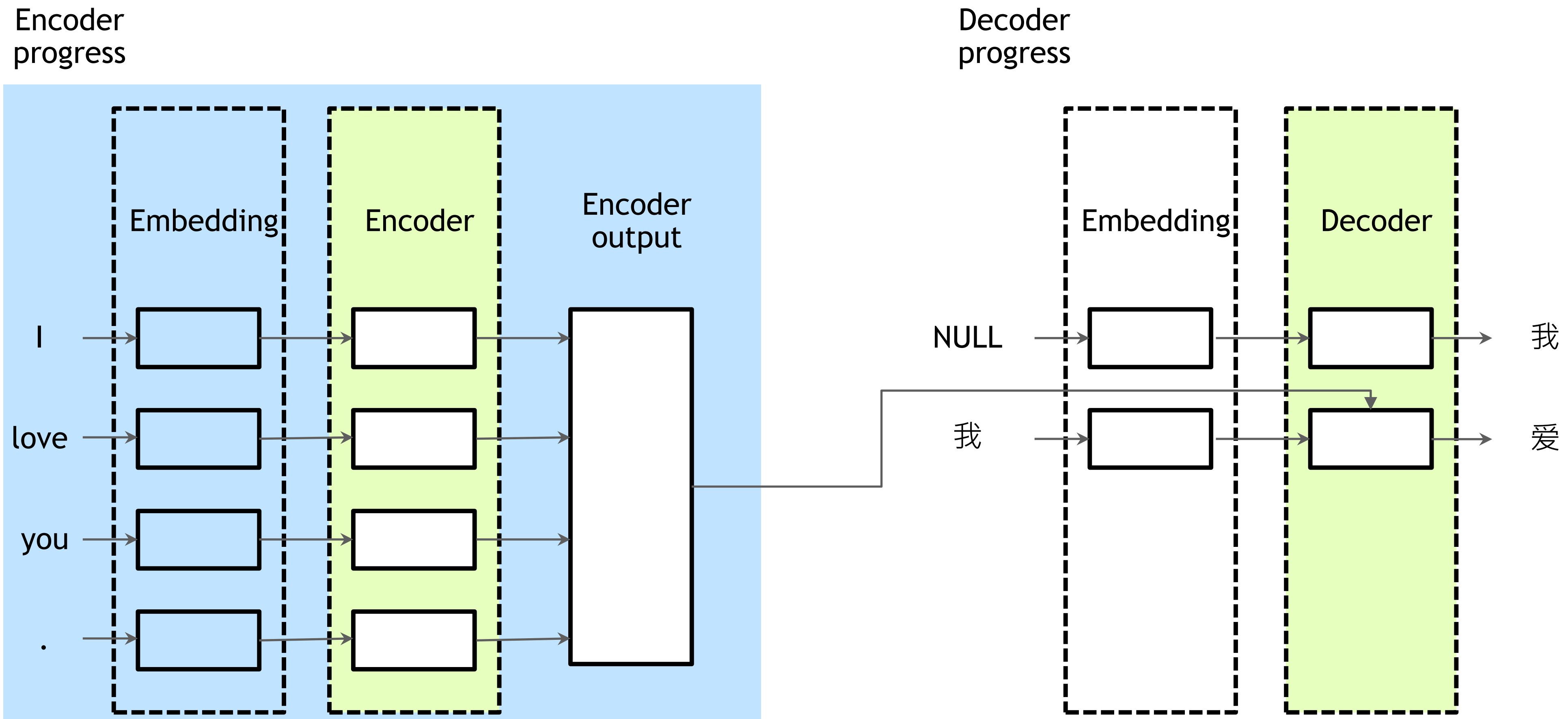
WHY WE NEED TO OPTIMIZE DECODER

Translating Progress



WHY WE NEED TO OPTIMIZE DECODER

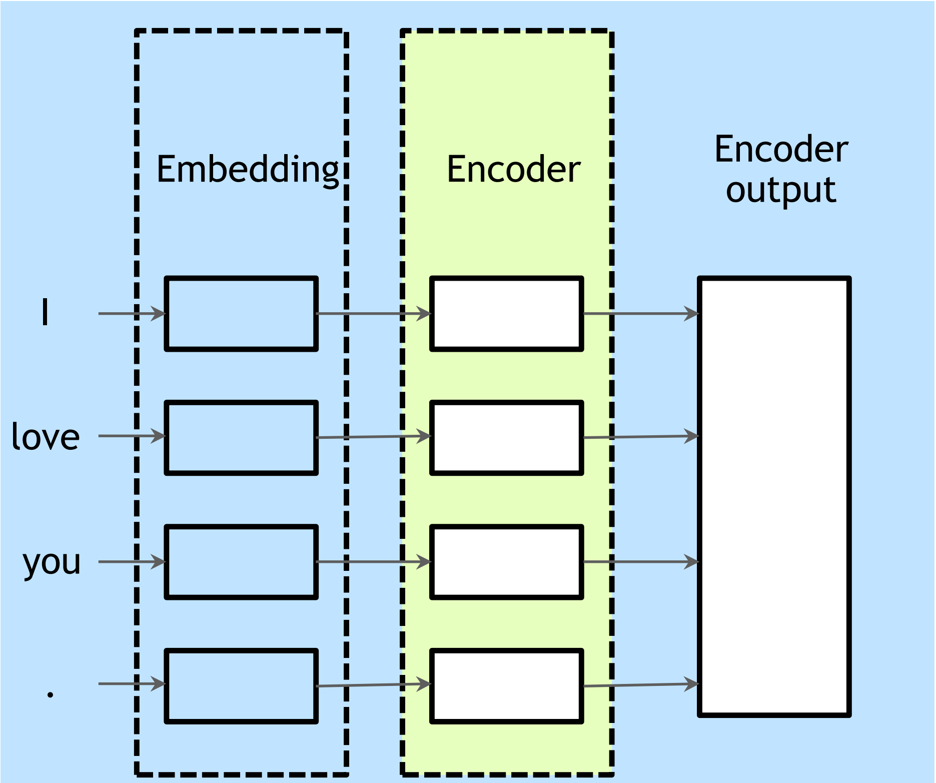
Translating Progress



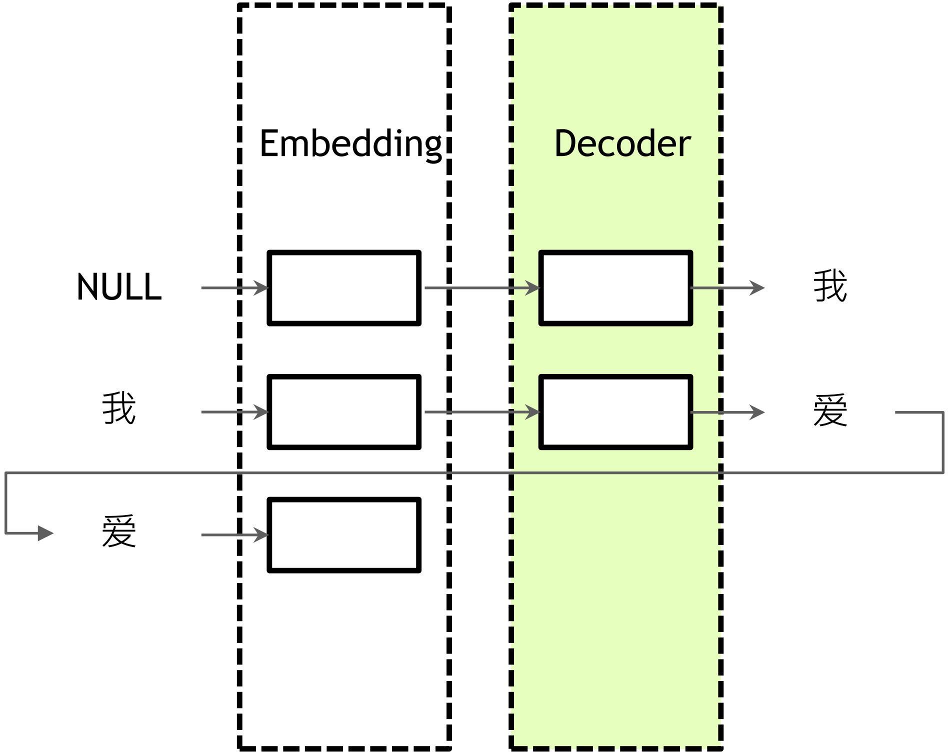
WHY WE NEED TO OPTIMIZE DECODER

Translating Progress

Encoder progress

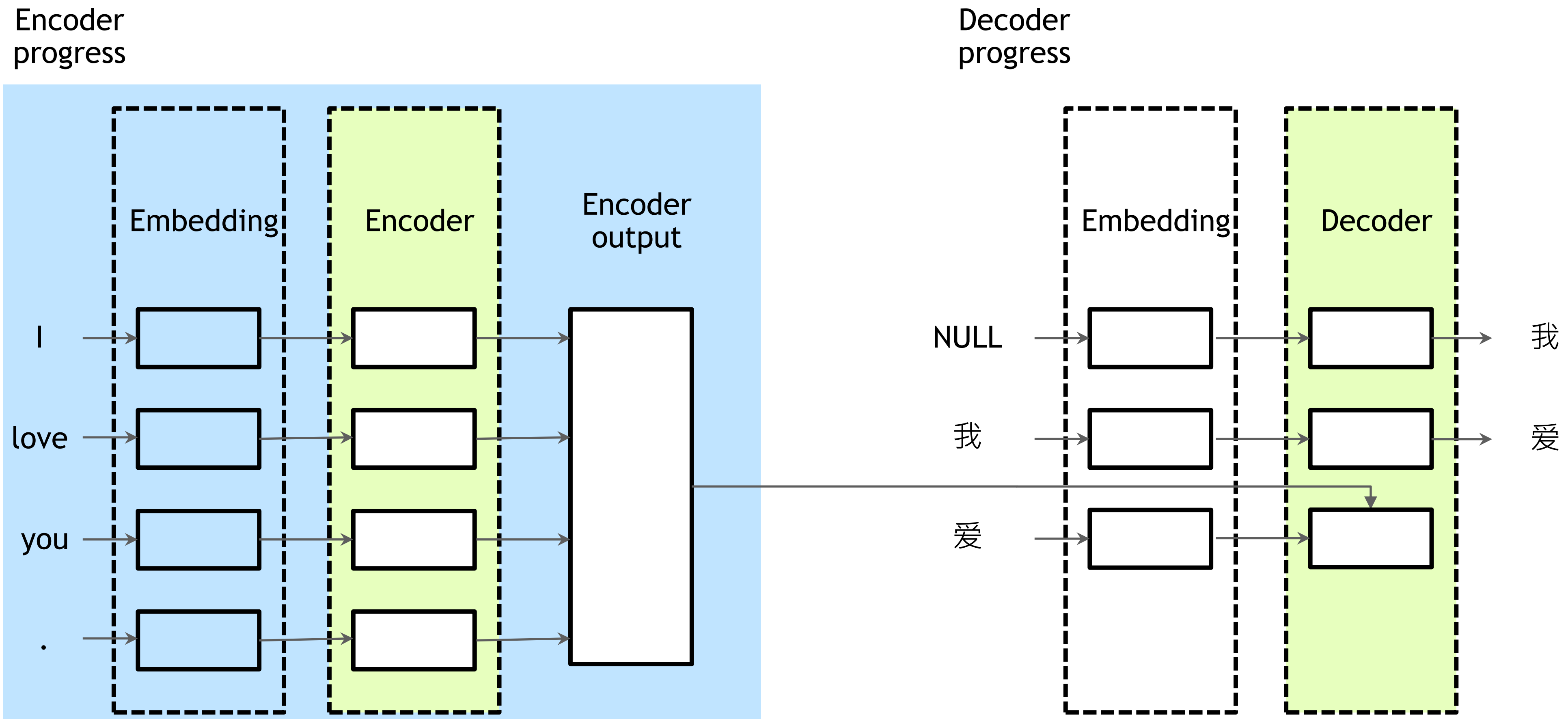


Decoder progress



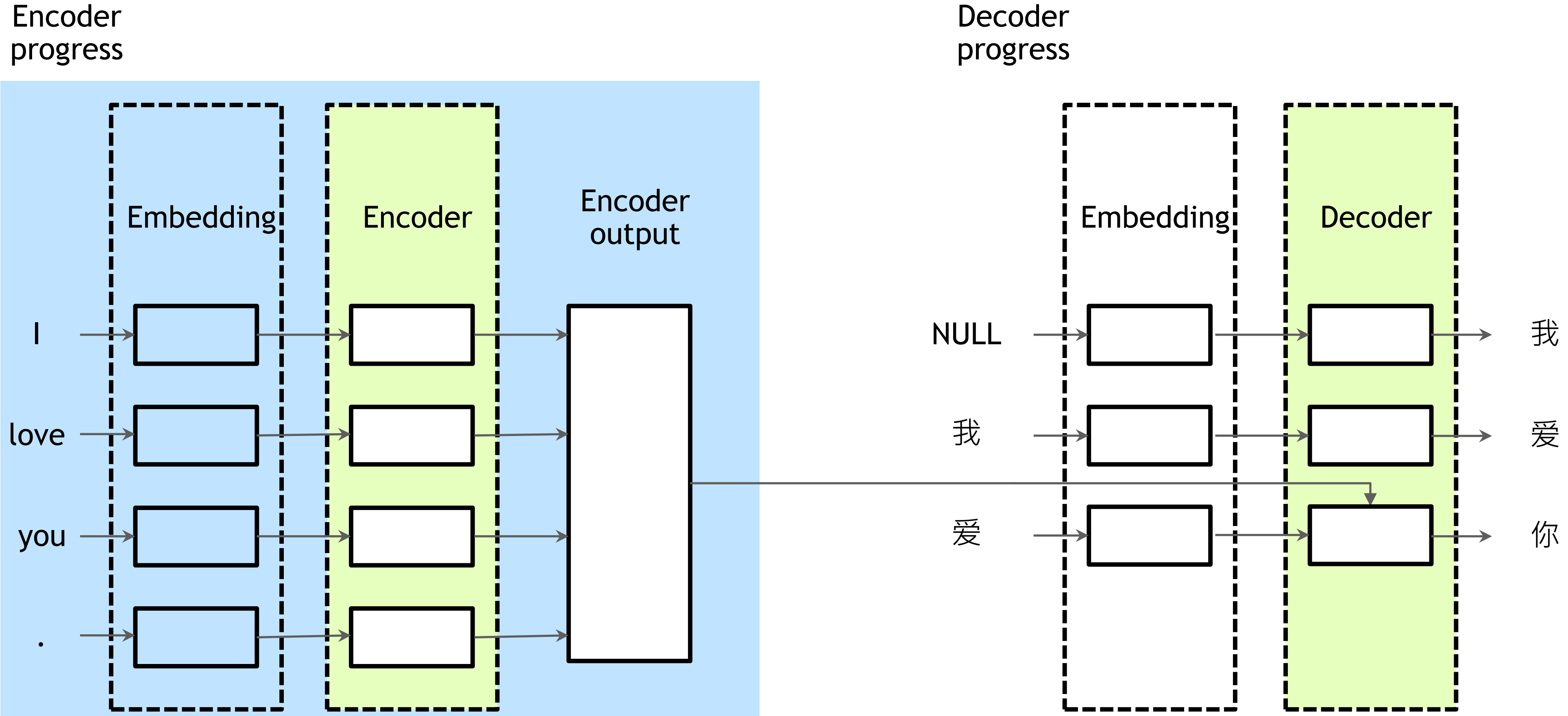
WHY WE NEED TO OPTIMIZE DECODER

Translating Progress



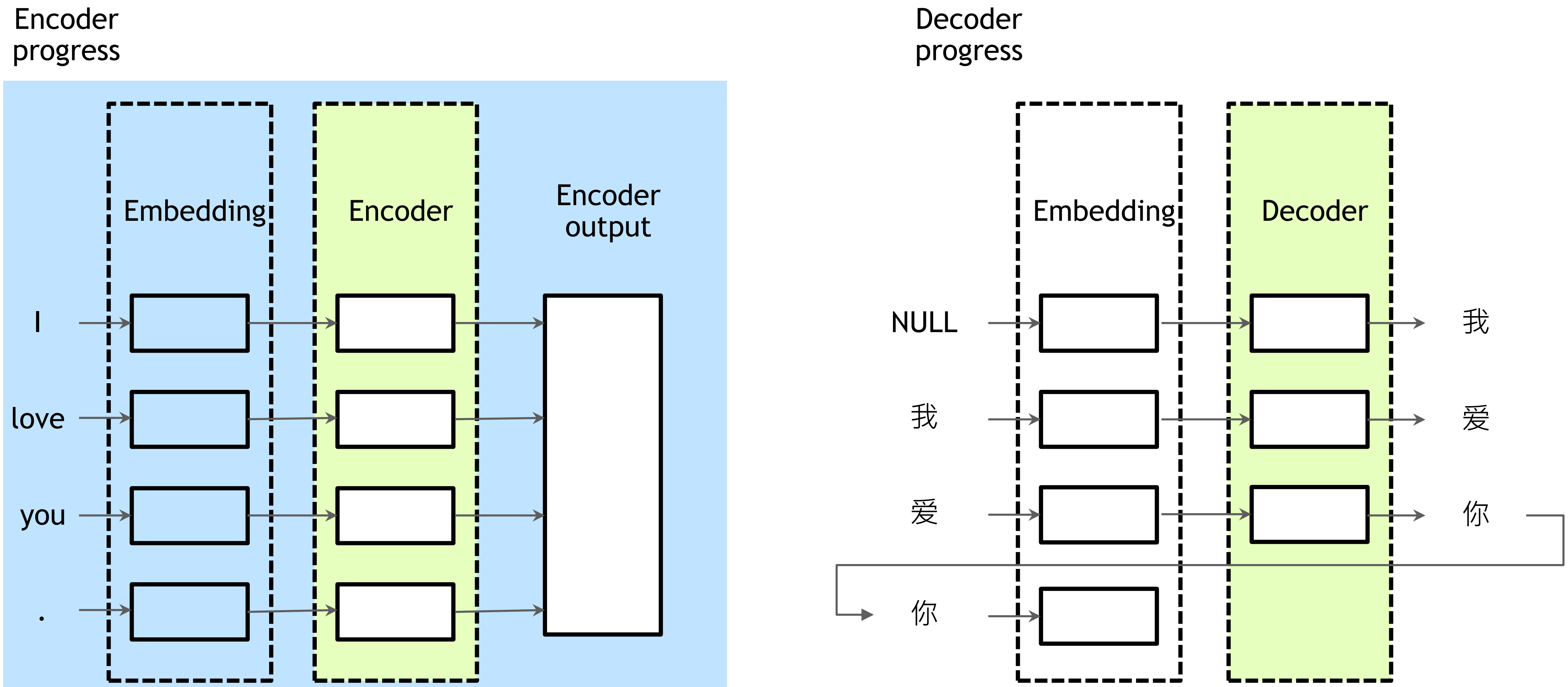
WHY WE NEED TO OPTIMIZE DECODER

Translating Progress



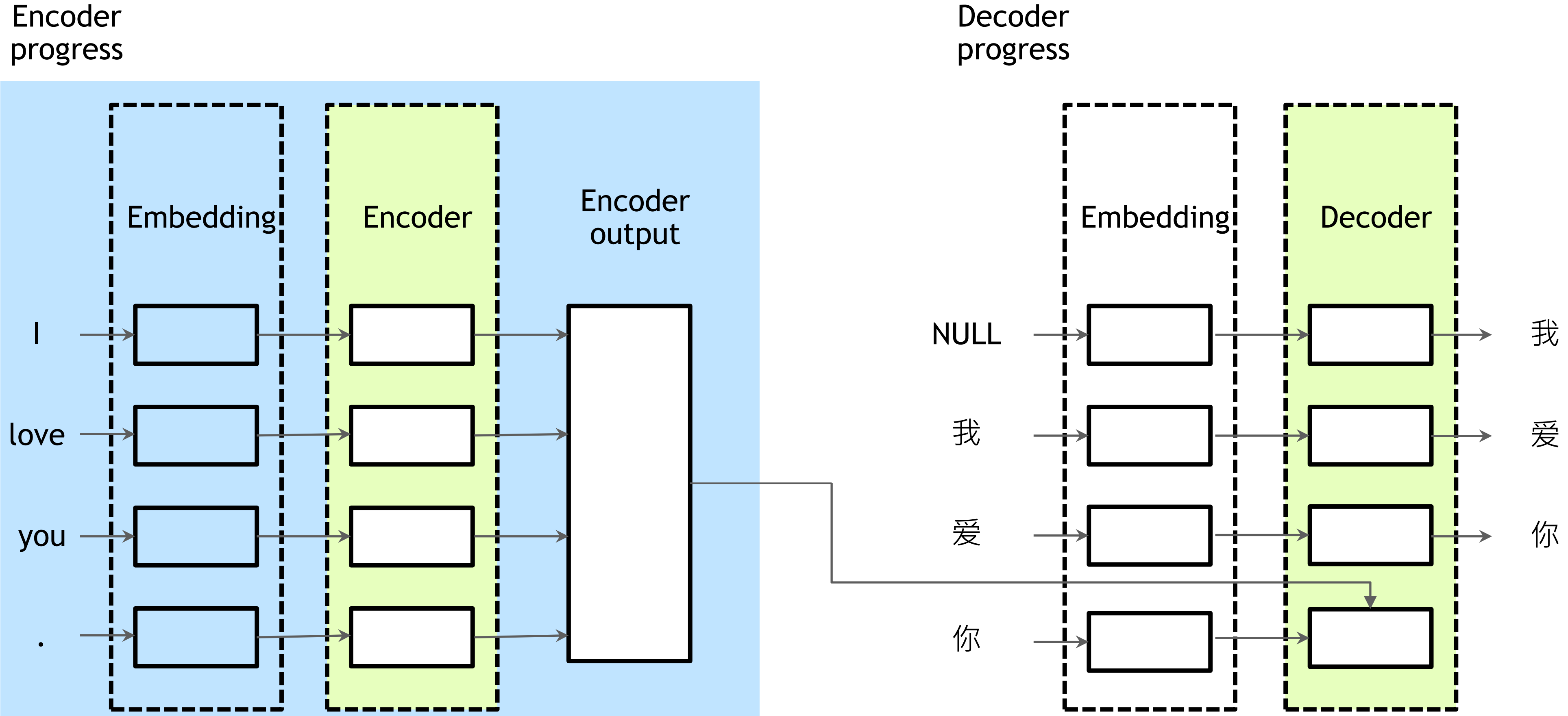
WHY WE NEED TO OPTIMIZE DECODER

Translating Progress



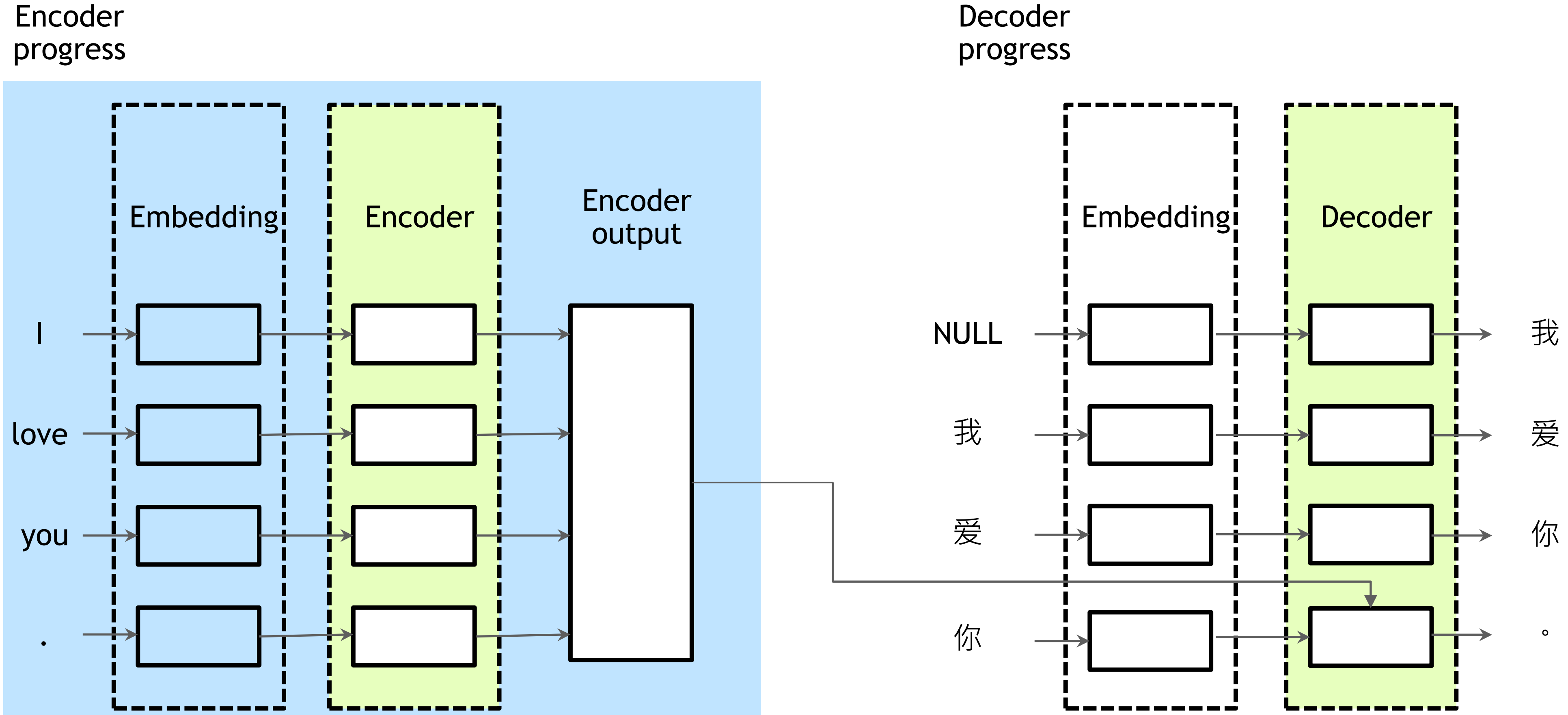
WHY WE NEED TO OPTIMIZE DECODER

Translating Progress



WHY WE NEED TO OPTIMIZE DECODER

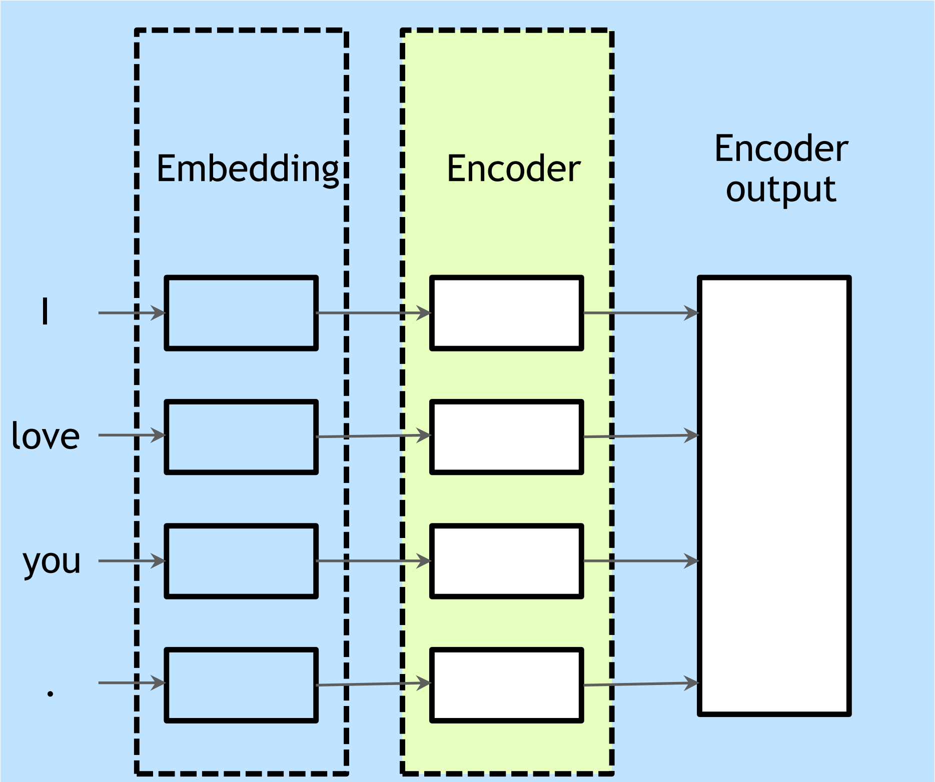
Translating Progress



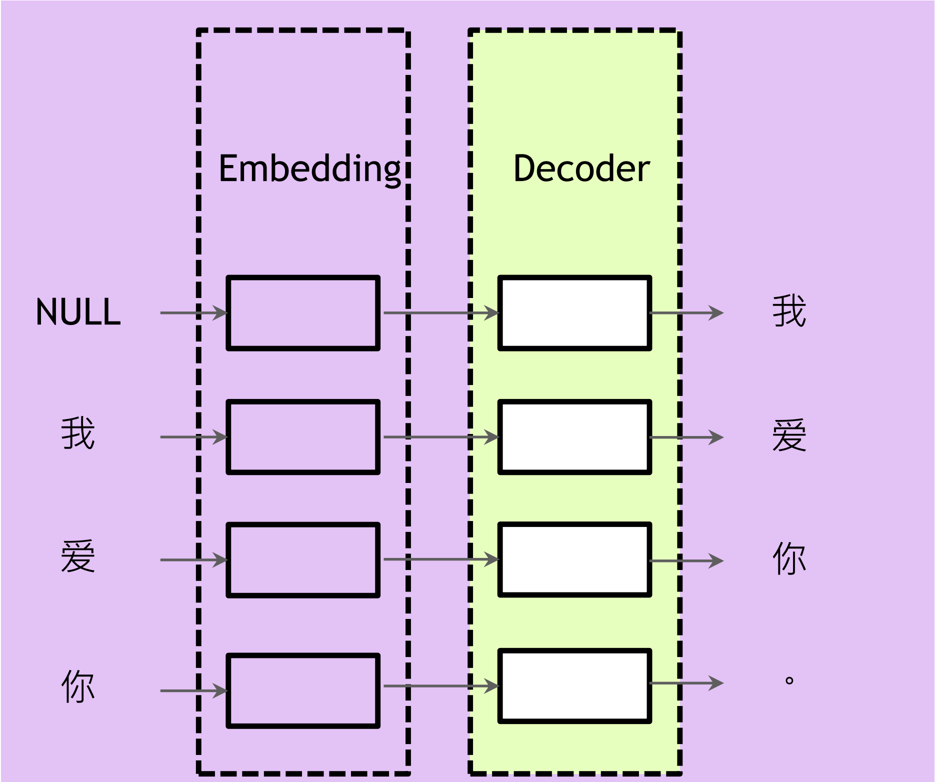
WHY WE NEED TO OPTIMIZE DECODER

Translating Progress

Encoder progress



Decoder progress



WHY WE NEED TO OPTIMIZE DECODER

Decoder consumes more time

- In Faster Transformer 1.0, we implement a highly optimized transformer layer for encoder.
- However, in a whole translating progress, most time is consumed in decoder.
- Encoder v.s. Decoder
  - Encoder < 10 ms v.s. decoder > 100 ms in most time
  - E.g., batch 1, sequence length 32 on NVIDIA Tesla T4 with FP32

    Encoder: 12 layers, hidden units 768: 2.74 ms

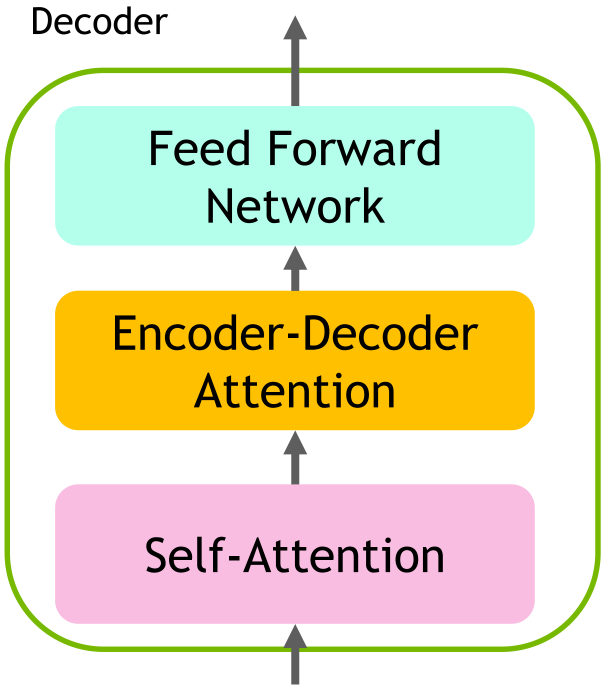
    Decoding: Beam width 4, 6 layers, hidden units 512: 64.16 ms
- So, we optimize the decoder in the Faster Transformer 2.0

NEW FEATURES IN FASTER TRANSFORMER 2.0

NEW FEATURE IN FASTER TRANSFORMER 2.0

Summary

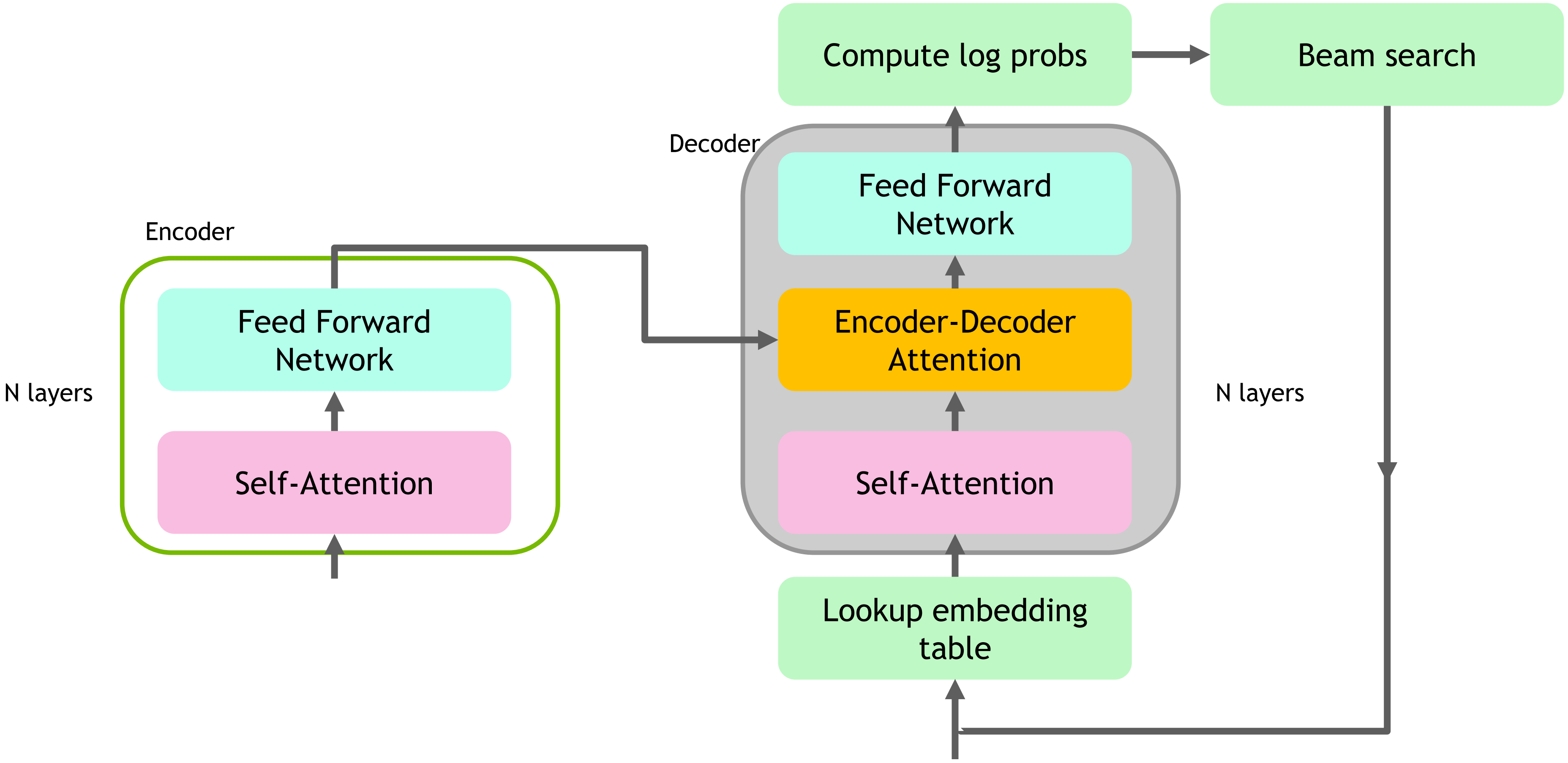
- We propose two components: Decoder and Decoding
  - Both based on OpenNMT-tf [1] model
- Decoder contains two attention layer and a FFN, providing 1.4x ~ 2x speedup
- Decoding contains whole translating process, providing 1.5x ~ 9x speedup
- The smaller batch size, the larger speedup



[1] https://github.com/OpenNMT/OpenNMT-tf

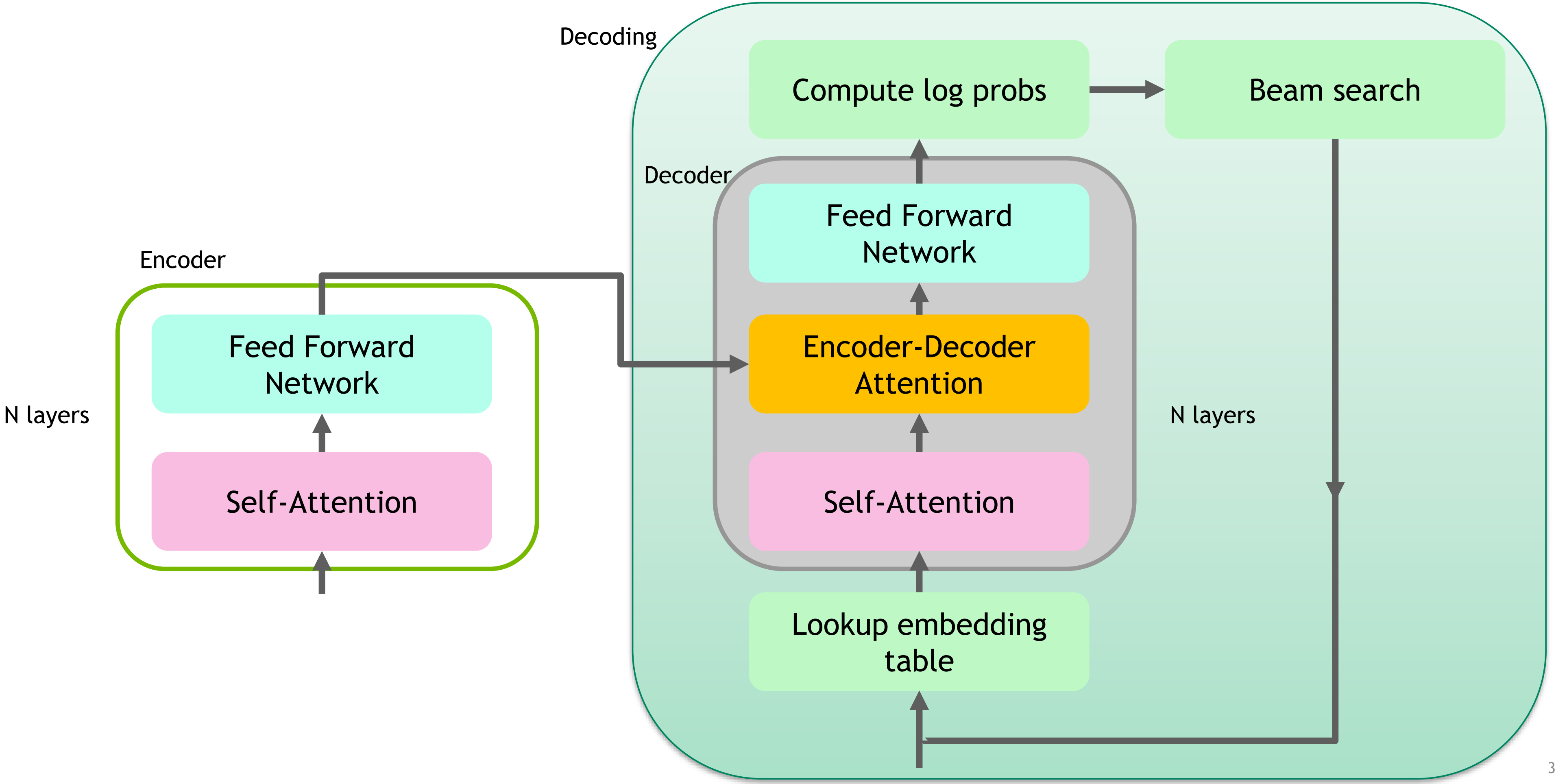
NEW FEATURE IN FASTER TRANSFORMER 2.0

Decoder and Decoding



NEW FEATURE IN FASTER TRANSFORMER 2.0

Decoder and Decoding



NEW FEATURE IN FASTER TRANSFORMER 2.0

Decoder and Decoding

```
decoding(encoder_result, start_id){
    id = start_id
    while(finished == false){
        decoder_input = lookup_embedding_table(id)
        decoder_output = decoder(decoder_input, encoder_output, num_layer)
        log_prob = dense(decoder_output)
        id = beamsearch(log_prob, candidate_number)
    }
}
```

NEW FEATURE IN FASTER TRANSFORMER 2.0

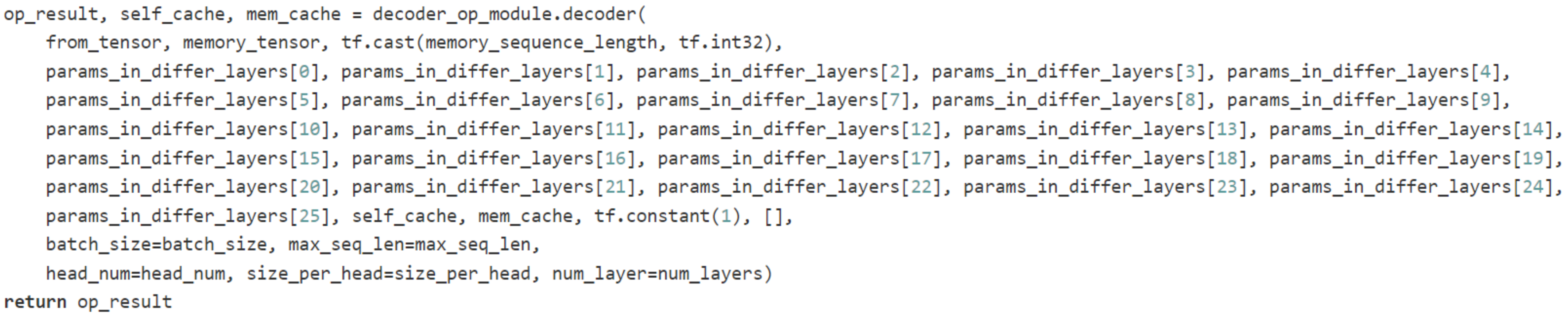
Decoder and Decoding

- Compare to Decoder, Decoding is more efficient
- If we translate a 32 words sentence
    - We need to call 32 times Decoder, and lead to 32 times of op launch overhead
    - We only need to call 1 time Decoding
- Decoding also provides an optimized naïve beamsearch

NEW FEATURE IN FASTER TRANSFORMER 2.0

How to use decoder and decoding?

- Similar to Faster Transformer 1.0
- Provide C and Tensorflow API
- Provide sample codes to demonstrate how to use
- Decoder in TensorFlow:

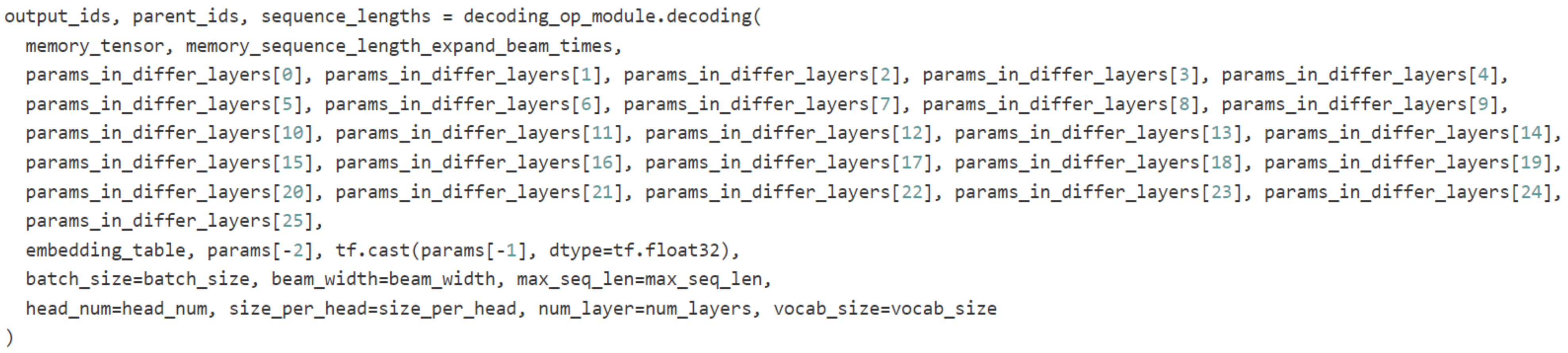


```
op_result, self_cache, mem_cache = decoder_op_module.decoder(
    from_tensor, memory_tensor, tf.cast(memory_sequence_length, tf.int32),
    params_in_differ_layers[0], params_in_differ_layers[1], params_in_differ_layers[2], params_in_differ_layers[3], params_in_differ_layers[4],
    params_in_differ_layers[5], params_in_differ_layers[6], params_in_differ_layers[7], params_in_differ_layers[8], params_in_differ_layers[9],
    params_in_differ_layers[10], params_in_differ_layers[11], params_in_differ_layers[12], params_in_differ_layers[13], params_in_differ_layers[14],
    params_in_differ_layers[15], params_in_differ_layers[16], params_in_differ_layers[17], params_in_differ_layers[18], params_in_differ_layers[19],
    params_in_differ_layers[20], params_in_differ_layers[21], params_in_differ_layers[22], params_in_differ_layers[23], params_in_differ_layers[24],
    params_in_differ_layers[25], self_cache, mem_cache, tf.constant(1), [],
    batch_size=batch_size, max_seq_len=max_seq_len,
    head_num=head_num, size_per_head=size_per_head, num_layer=num_layers)
return op_result
```

NEW FEATURE IN FASTER TRANSFORMER 2.0

How to use decoder and decoding?

- Similar to Faster Transformer 1.0
- Provide C and Tensorflow API
- Provide sample codes to demonstrate how to use
- Decoding in TensorFlow:



```
output_ids, parent_ids, sequence_lengths = decoding_op_module.decoding(
  memory_tensor, memory_sequence_length_expand_beam_times,
  params_in_differ_layers[0], params_in_differ_layers[1], params_in_differ_layers[2], params_in_differ_layers[3], params_in_differ_layers[4],
  params_in_differ_layers[5], params_in_differ_layers[6], params_in_differ_layers[7], params_in_differ_layers[8], params_in_differ_layers[9],
  params_in_differ_layers[10], params_in_differ_layers[11], params_in_differ_layers[12], params_in_differ_layers[13], params_in_differ_layers[14],
  params_in_differ_layers[15], params_in_differ_layers[16], params_in_differ_layers[17], params_in_differ_layers[18], params_in_differ_layers[19],
  params_in_differ_layers[20], params_in_differ_layers[21], params_in_differ_layers[22], params_in_differ_layers[23], params_in_differ_layers[24],
  params_in_differ_layers[25],
  embedding_table, params[-2], tf.cast(params[-1], dtype=tf.float32),
  batch_size=batch_size, beam_width=beam_width, max_seq_len=max_seq_len,
  head_num=head_num, size_per_head=size_per_head, num_layer=num_layers, vocab_size=vocab_size
)
```

FASTER TRANSFORMER 2.0 PERFORMANCE

FASTER TRANSFORMER 2.0 PERFORMANCE

Environment Setting

- Docker: nvcr.io/nvidia/tensorflow:19.07-py2
  - CUDA 10.1
  - TensorFlow 1.14
  - Python 2.7
- CPU: Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz
- NVIDIA Tesla T4 (with mclk 5000MHz, pclk 1590MHz)
- NVIDIA Tesla V100 (with mclk 877MHz, pclk 1380MHz)

FASTER TRANSFORMER 2.0 PERFORMANCE

Decoder benchmark on NVIDIA Tesla T4

- Since batch size is 1, the bottleneck is not the computing ability. So, no benefit on FP16.

< batch size, seq len>	TensorFlow FP32 (ms)	Faster Decoder FP32 (ms)	FP32 Speedup	TensorFlow FP16 (ms)	Faster Decoder FP16 (ms)
(1, 32)	441.68	146.54	3.01	508.81	165.88
(1, 64)	872.39	309.96	2.81	1038.71	326.69
(1, 128)	1714.01	660.30	2.59	2082.92	661.00

FASTER TRANSFORMER 2.0 PERFORMANCE

Decoder benchmark on NVIDIA Tesla T4

- FP16 Speedup is computed by faster TensorFlow version (sometimes is TensorFlow FP32).

< batch size, seq len>	TensorFlow FP32 (ms)	Faster Decoder FP32 (ms)	FP32 Speedup	TensorFlow FP16 (ms)	Faster Decoder FP16 (ms)	FP16 Speedup
(32, 32)	470.93	183.48	2.56	568.83	167.42	2.81
(64, 32)	503.57	232.70	2.16	579.21	183.74	2.74
(128, 32)	614.59	344.77	1.78	641.98	238.27	2.58
(256, 32)	802.18	573.25	1.40	735.67	348.74	2.11

FASTER TRANSFORMER 2.0 PERFORMANCE

Decoding benchmark on NVIDIA Tesla T4

- FP16 Speedup is computed by faster TensorFlow version (sometimes is TensorFlow FP32).
- Beam width is set to 4

< batch size, seq len>	TensorFlow FP32 (ms)	Faster Decoder FP32 (ms)	FP32 Speedup	TensorFlow FP16 (ms)	Faster Decoder FP16 (ms)	FP16 Speedup
(1, 4, 32)	430.39	64.16	6.70	537.95	49.07	8.77
(1, 4, 64)	876.24	135.42	6.47	1056.78	97.45	8.99
(1, 4, 128)	1799.16	318.65	5.64	2145.74	240.85	7.47

FASTER TRANSFORMER 2.0 PERFORMANCE

Decoding benchmark on NVIDIA Tesla T4

- FP16 Speedup is computed by faster TensorFlow version (sometimes is TensorFlow FP32).
- Beam width is set to 4

< batch size, seq len>	TensorFlow FP32 (ms)	Faster Decoder FP32 (ms)	FP32 Speedup	TensorFlow FP16 (ms)	Faster Decoder FP16 (ms)	FP16 Speedup
(32, 4, 32)	597.42	217.61	2.74	646.07	128.39	4.65
(64, 4, 32)	789.22	395.85	1.99	769.17	246.89	3.11
(128, 4, 32)	1223.72	726.43	1.68	996.03	424.53	2.34
(256, 4, 32)	2188.00	1385.60	1.58	1599.58	781.38	2.04

FASTER TRANSFORMER 2.0 PERFORMANCE

Decoding benchmark on NVIDIA Tesla V100

- FP16 Speedup is computed by faster TensorFlow version (sometimes is TensorFlow FP32).
- Beam width is set to 4

< batch size, sequence length>	TensorFlow FP32 (ms)	Faster Decoder FP32 (ms)	FP32 Speedup	TensorFlow FP16 (ms)	Faster Decoder FP16 (ms)	FP16 Speedup
(1, 4, 32)	440.46	58.70	7.50	531.70	46.18	9.53
(1, 4, 64)	888.19	122.50	7.25	1065.76	93.84	9.46
(1, 4, 128)	1821.76	293.21	6.21	2076.63	293.21	6.21

FASTER TRANSFORMER 2.0 PERFORMANCE

Decoding benchmark on NVIDIA Tesla V100

- FP16 Speedup is computed by faster TensorFlow version (sometimes is TensorFlow FP32).
- Beam width is set to 4

< batch size, seq len>	TensorFlow FP32 (ms)	Faster Decoder FP32 (ms)	FP32 Speedup	TensorFlow FP16 (ms)	Faster Decoder FP16 (ms)	FP16 Speedup
(32, 4, 32)	543.27	101.35	5.36	630.55	73.37	7.40
(64, 4, 32)	648.27	157.54	4.11	793.83	106.77	6.07
(128, 4, 32)	838.43	277.77	3.02	867.71	169.04	4.96
(256, 4, 32)	1221.30	493.85	2.47	1101.36	290.44	3.79

FASTER TRANSFORMER 2.0 PERFORMANCE

Summary

- Decoder on NVIDIA Tesla T4
  - 2.5x speedup for batch size 1 (online translating scheme)
  - 2x speedup for large batch size in FP16
- Decoding on NVIDIA Tesla T4
  - 7x speedup for batch size 1 and beam width 4 (online translating scheme)
  - 2x speedup for large batch size in FP16.
- Decoding on NVIDIA Tesla V100
  - 6x speedup for batch size 1 and beam width 4 (online translating scheme)
  - 3x speedup for large batch size in FP16.

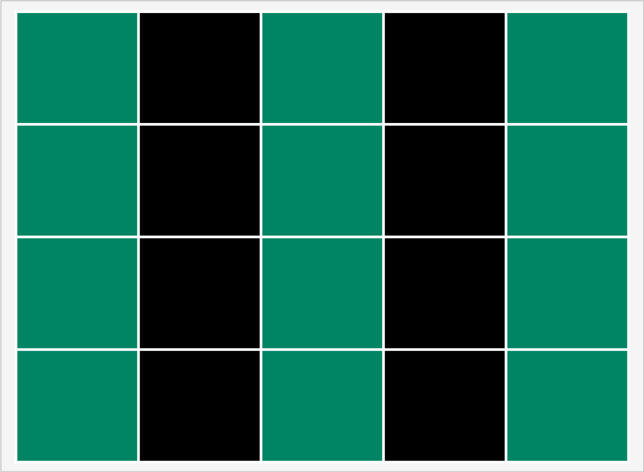
OTHER WORK

NETWORK PRUNING

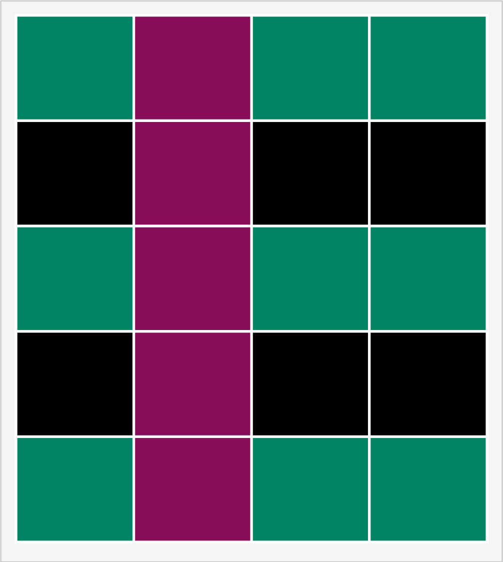
- To speedup the transformer more on large batch size case, we try to accelerate the inference by network pruning
- We choose [1] as pruning algorithm
- Prune a column or a row of the weight in one time

[1] Molchanov, P., Mallya, A., Tyree, S., Frosio, I. and Kautz, J., 2019. Importance Estimation for Neural Network Pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 11264-11272).

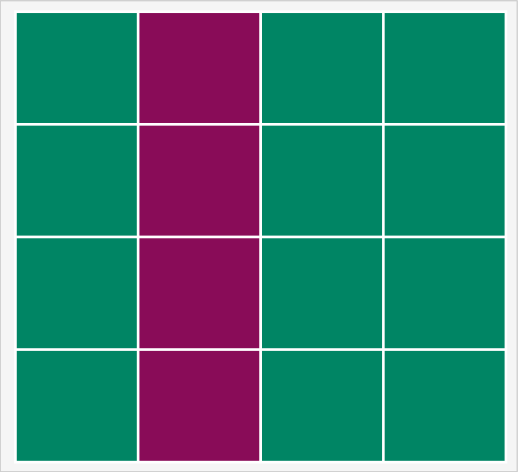
NETWORK PRUNING



Input in $\mathbb{R}^{M \times K}$



Weight in $\mathbb{R}^{K \times N}$



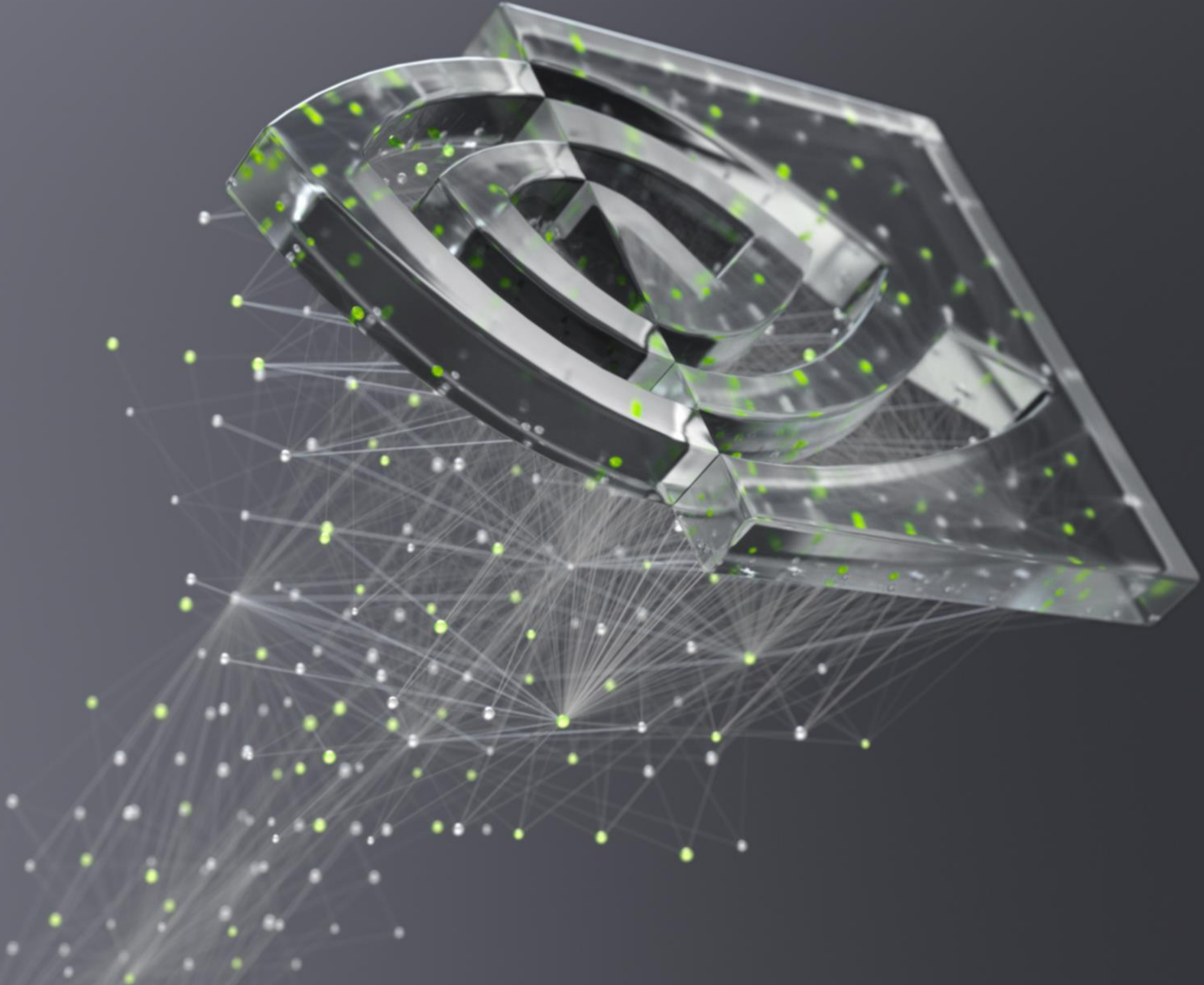
Output in $\mathbb{R}^{M \times N}$

NETWORK PRUNING

- we successfully prune 50% useless rows/columns of weights on BERT model
- Expect to get 2x speedup with about 2.8% accuracy loss

Model	Sparsity	Acc (%)	Reduced acc (%)	Total fine-tuning time
Baseline	0%	84.06	0.00	
Multiple stages 1	30%	83.23	-0.83	3 epochs
	40%	82.22	-1.84	
	50%	79.80	-4.26	
Multiple stages 2	30%	83.37	-0.69	2 epochs
	40%	82.52	-1.54	3 epochs
	50%	81.27	-2.79	4 epochs

Q&A TIME



NVIDIA