EL-Attention: Memory Efficient Lossless Attention for Generation

Yu Yan [1] **Jiusheng Chen** [1] **Weizhen Qi** [* 2] **Nikhil Bhendawade** [* 1] **Yeyun Gong** [* 3] **Nan Duan** [3] **Ruofei Zhang** [4]

Abstract

Transformer model with multi-head attention requires caching intermediate results for efficient inference in generation tasks. However, cache brings new memory-related costs and prevents leveraging larger batch size for faster speed. We propose memory-efficient lossless attention (called EL-attention) to address this issue. It avoids heavy operations for building multi-head keys and values, cache for them is not needed. EL-attention constructs an ensemble of attention results by expanding query while keeping key and value shared. It produces the same result as multi-head attention with less GPU memory and faster inference speed. We conduct extensive experiments on Transformer, BART, and GPT-2 for summarization and question generation tasks. The results show EL-attention speeds up existing models by 1.6x to 5.3x without accuracy loss.

1. Introduction

Transformer model with multi-head attention achieves success in various generation tasks, such as text generation (Raffel et al., 2019; Radford et al., 2019; Lewis et al., 2020; Brown et al., 2020), image generation (Parmar et al., 2018; Cho et al., 2020), and music generation (Huang et al., 2018). However, inference speed is a serious problem in generation models. Recently, a variety of methods have been proposed for the speed up of Transformer and variant models. Many methods focus on reducing complexity on sequence length, like restricting tokens which can be looked at (Zaheer et al., 2020; Beltagy et al., 2020), using sort (Tay et al., 2020) or hash technology (Kitaev et al., 2020), keeping cumulative states (Katharopoulos et al., 2020), and compressing dimension (Goyal et al., 2020; Wang et al., 2020a). Others study reducing model size to accelerate inference by pruning layer (Fan et al., 2019) or training a smaller student model (Shleifer & Rush, 2020). Another way is non-autoregressive generation (Gu et al., 2018; Lee et al., 2018; Qi et al., 2020) which generates all tokens at once, instead of predicting the next token step by step. While these excellent methods can effectively speed up the models, they require users to train a new model, and it is hard to apply them to an existing model directly. Moreover, most of them suffer more or less accuracy loss (Tay et al., 2021).

In this paper, we explore generation speedup by optimizing cache size and memory movement rather than reducing computational complexity. This choice is based on an insight from studying the generation process. Multi-head attention needs to convert query, key, and value to their multi-head format for assembling attention results. Under the context of incremental generation, a query is a feature vector belonging to $\mathbb{R}^{d_m}$, while key and value are sequences of n feature vectors with shape $\mathbb{R}^{n \times d_m}$. Changes in key and value are minimal during an incremental decoding step, and they are usually cached to avoid complex and duplicated computation. With cache, overall speed is much faster than without cache, but cache maintenance cost is still non-trivial. There is a significant room to achieve quicker speed by reducing memory complexity.

We present a new memory-efficient lossless attention, called *EL-attention*. It can speed up inference by reducing cache size and memory movement complexity. Memory used for caching input related model states is reduced from $\mathcal{O}(Ld_m)$ to $\mathcal{O}(d_m)$ where L is number of decoder layers and d_m is model dimension. EL-attention further reduces memory movement when using beam search or other search methods to produce many candidates for each input. We achieve this by only expanding query when constructing an ensemble of attention results. Our method avoids converting key and value to multi-head format. All queries from different heads and beams share the same key and value. Despite the change of attention, our method can be compatible with multi-head attention via a particular converted form of query. By taking advantage of the saved memory movement, our method can accelerate inference under the same setting as the baseline. Also, our method saves a massive amount of memory. This saved memory can be harvested in multiple ways, like fitting a bigger model in GPU, or using a larger batch size for further speed up.

[*]Equal contribution [1]Microsoft, Redmond, WA, USA [2]University of Science and Technology of China [3]Microsoft Research Asia [4]Microsoft, Sunnyvale, CA, USA. Correspondence to: Yu Yan <yyua@microsoft.com>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

To summarize our contributions:

1. We propose a new attention method called EL-attention, which can replace multi-head attention at the inference stage to generate the same results with smaller cache size and less memory movement.

2. We evaluate EL-attention on the Transformer, BART, GPT-2 model for summarization tasks (CNN/DailyMail and XSum) and question generation task (SQuAD 1.1). It shows 1.6x to 5.3x speedup across all models on these tasks in beam search, diverse beam search, and greedy search.

3. We show that EL-attention uses 96x lesser memory for caching input related states in BART model, which supports running with 10x larger batch size and 5x speed. It is potentially valuable for memory limited devices, like mobile and IoT devices.

2. Background

We first introduce Transformer (Vaswani et al., 2017) under generation context, then describe speed analysis.

2.1. Scaled Dot-Product Attention

Attention allows one position to grab information from other positions. Given key matrix and value matrix K, V $\in \mathbb{R}^{n\times d}$, and a query vector Q $\in \mathbb{R}^{d}$, the attention is calculated as:

$$\text{Att}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d}})V \quad (1)$$

2.2. Multi-Head Attention

In multi-head attention, independent attentions are applied in parallel to obtain the final result[1]:

$$\text{MultiHead}(Q, K, V) = \sum_{i=1}^{h} \text{Att}(Q_i, K_i, V_i)W_i^O$$
$$\text{where } Q_i = QW_i^Q,\ K_i = KW_i^K,\ V_i = VW_i^V \quad (2)$$

Here, W_i^Q, W_i^K, $W_i^V \in \mathbb{R}^{d_m\times d_k}$ and $W_i^O \in \mathbb{R}^{d_k\times d_m}$. The h is the number of heads, and Q_i, K_i, V_i represent the query, key, and value for the i-th head respectively. The d_k is typically set to $\frac{d_m}{h}$.

2.3. Incremental Decoding

For Transformer inference, prediction of the next token depends on the input sequence and previously generated sequence. The last token attends to its previous output and input at each step. States (key and value) are cached and reused to avoid redundant calculation. See Figure 1, although each decoder layer attends to the same encoder output, it needs to build its own cache for storing key and value due to each layer having different weight parameters W^K and W^V.

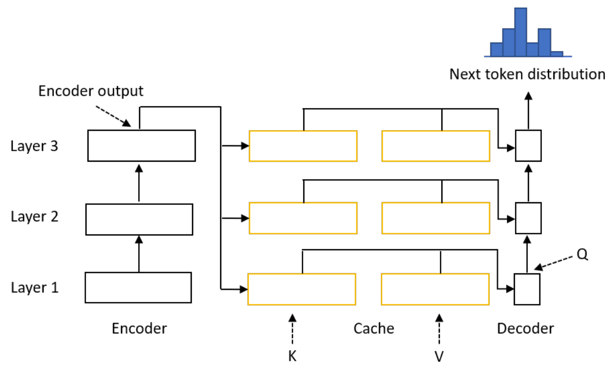

Figure 1. The cache used in encoder-decoder attention during incremental decoding.

2.4. Speed Analysis

Arithmetic intensity (AI) is a ratio of floating-point operations performed to data movement (FLOPs/Bytes). Speed performance (GFLOP/s) is:

$$\text{GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{Peak GB/s} \times \text{AI} \end{cases} \quad (3)$$

A function is bounded by memory when AI is small, and its threshold ranges from ten to a hundred for V100 GPU on single and half precision (Yang et al., 2020; Wang et al., 2020b).

Decoding is largely bounded by memory bandwidth due to low arithmetic intensity (Shazeer, 2019; Tay et al., 2020). Its speed is also affected by latency because of the small computational scale per instruction.

3. Method

In this section, we present memory-efficient lossless attention (EL-attention), which is designed for high inference speed and low memory requirement.

We start by introducing EL-attention technical detail in § 3.1. In § 3.2, we present how to use EL-attention to speed up existing vanilla Transformer-based models. Finally, we present theoretical speed analysis in § 3.3.

[1]This notation, equivalent to the "concatenation" formulation from (Vaswani et al., 2017) is used to ease exposition in the following sections.

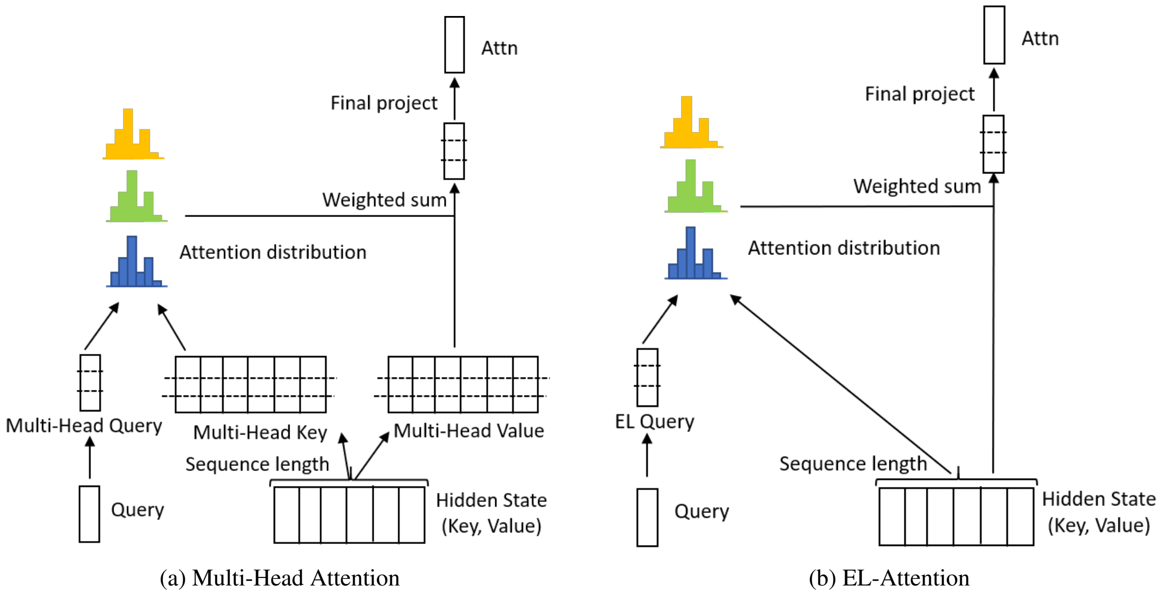

Figure 2. (a) Multi-head attention: query, key, and value are converted to their multi-head formats respectively, then the attention probability and attention result for each head are calculated; at last, results are aggregated to get the final output. (b) EL-attention: it only applies linear conversion to query. The hidden state, which is encoder output or previous layer output, is directly used as key and value without conversion.

3.1. EL-Attention

Our method EL-attention constructs an ensemble of attention results by expanding query only and sharing key and value for all heads. The overall workflow of EL-attention is illustrated in Figure 2b. We first convert query to multi-head format via expanding. Then it directly multiplies with hidden state, which is either encoder output or previous layer output, to calculate attention probability. Intermediate attention result is computed by multiplying attention probability with the same hidden state. Finally, a linear conversion is applied to produce the output. The multi-head attention (see Figure 2a), which is used in the vanilla Transformer, has linear transformation for all query, key, and value to get their multi-head formats.

EL-attention introduces two feed-forward networks (FFN) for calculating an ensemble of attention results:

$$\text{EL}(Q, K, V) = \sum_{i=1}^{h} \text{FFN}_i^O(\text{Att}(\text{FFN}_i^Q(Q), K, V)) \quad (4)$$

Let FFN_i^Q and FFN_i^O: $\mathbb{R}^{d_m} \to \mathbb{R}^{d_m}$ be conversion functions for query and individual attention results, the choice of this functions can be arbitrary, like one or multiple linear conversions, or an activation function. In this work, the following forms are adopted to be completely compatible with vanilla transformer:

$$\begin{aligned} &\text{EL-Q}_i = \text{FFN}_i^Q(Q) = QW_i^Q(W_i^K)^T \\ &\text{and } \text{FFN}_i^O(X) = XW_i^V W_i^O \end{aligned} \quad (5)$$

where W_i^Q, W_i^K, $W_i^V \in \mathbb{R}^{d_m \times d_k}$ and $W_i^O \in \mathbb{R}^{d_k \times d_m}$, these weight parameters are same as the ones in multi-head attention. The normalization factor d in attention softmax Equation 1 is d_k to keep same as multi-head attention, d_k is the rank of EL-Q$_i$ while d_m is its dimension.

A critical difference between EL-attention and multi-head attention is how to assemble attention results. EL-attention can provide an ensemble result without projecting key and value to sub-spaces, only query is projected to low-rank format while keeping dimension size unchanged. In multi-head attention, all query, key, and value are projected to sub-spaces, requiring high computational complexity or large memory size for caching.

3.1.1. Efficient Implementation

Several changes are made on the generation implementation to reduce the cache size and memory movement. We present pseudocode to show the key difference between EL-attention and multi-head attention in Appendix A.

First, as EL-attention does not depend on multi-head key and value, their cache can be entirely removed for all encoder-decoder attention sub-layers. Only encoder output is saved, and it is shared for all decoder layers. EL-attention reduces cache size by $2L$ times for encoder-decoder attention, where L is the number of layers in a decoder. In self-attention, we only store output from the previous layer, instead of both key and value, thereby reducing cache size by half.

Second, since multiple query heads are mapped to the same key in EL-attention, we can reduce memory movement during attention calculation. To recall batch matrix multiplication (BMM) background, BMM of two 3D matrices is equivalent to computing multiple 2D matrix multiplications in parallel. However, this good property requires two 3D matrices to have their first dimension size equal. The first dimensions for query and key need to be adjusted to meet the requirement. Key generally uses a bigger memory size than query because sequence length for query is always one in incremental decoding. In contrast, key's length is either input length or previous output length. Repetitive loading of key and value linearly slows down the speed in this case. Our implementation involves reshaping query to match the first dimension of key. We decrease query's first dimension and increase its second dimension, and then one matrix multiplication can generate scores for all heads, which avoids loading key repetitively.

In beam search or other searching methods, which generate many outputs per input, we can map multiple queries to one key and then the memory movement can be further reduced.

3.1.2. Exchangeable with Multi-Head Attention

Our method EL-attention can provide the same functional ability as multi-head attention. By leveraging the associative property of matrix multiplication, the multi-head attention formula can be derived from EL-attention, see the proof in Appendix B. We will present more generation result analyses in § 4.5.

3.2. EL-Attention Applications

In this work, we explore using EL-attention to replace multi-head attention at the inference stage for speed up. We show how to use EL-attention in transformer model and language model.

In text generation tasks, input length is generally longer than output length. Query-input attention is much heavier than query-previous output attention in terms of computational and memory complexity because at each step query attends to the whole input but only attends to the previously generated output, which gradually grows from one to max output length. With this in mind, we only apply EL-attention to calculate attention for input related part.

Encoder-Decoder Attention For a model having encoder-decoder architecture, we apply EL-attention for encoder-decoder attention while keeping self-attention unchanged.

Self-Attention For a decoder-only model, input (prefix text) and output sequences are concatenated to feed into the decoder. Both input and previous output tokens are attended to in the self-attention calculation. In this case, we apply EL-attention to calculate attention for the input part, while multi-head attention is still used to handle the previous output part. Attention weights for the input and the previous output are computed separately, then concatenated before applying softmax for computing attention probability, then split for building attention result from input and output, and followed by element-wise addition.

3.3. Theoretical Analysis

Here we present analytical comparisons between EL-attention and multi-head attention with/without caching key and value, from the perspective of computational and memory complexity. Attention related operations are divided into three groups based on arithmetic intensities, see Table 1.

The first group of operations is to build multi-head key and value, it has the highest computational complexity $\mathcal{O}(nd_m^2)$ and is bounded by computing, where n is sequence length and d_m is model dimension. Our method EL-attention only depends on previous output. Hence it does not need this part, we mark it as 0 in Table 1. To mitigate the computational cost, many sequence-to-sequence libraries (Ott et al., 2019; Wolf et al., 2020; Vaswani et al., 2018) support incremental decoding which caches multi-head key and value in each layer. However, a calculation is still needed at the first decode step. Caching key and value consumes lots of memory, see Table 4, which prevents using bigger batch size for faster inference speed. It also introduces new maintenance costs. For example, beam candidates' order needs to be changed at each step, and beam search needs to adjust cache order accordingly. Another case happens when part of a batch is finished, the finished sentences need to be removed, it also involves cache adjustment effort.

The second group is to build multi-head query, and project attention results to final output, which is also bounded by computing. EL-attention and multi-head Attention have same computational complexity O(d_m^2) which is only $1/n$ of the first group. It has the shortest execution time among all three groups.

The third group is to calculate attention weights and weighted sum. This group has n times higher memory complexity compared to group 2, hence bounded by memory. Although EL-attention has higher computational complexity than multi-head attention, it has comparable or faster speed due to lower memory complexity. In beam search or diverse beam search, EL-attention only uses 1/x (beam size) memory movement compared to multi-head attention as presented in § 3.1.1.

4. Experiments

First, EL-attention and multi-head attention are compared on synthetic data in § 4.1. Then we integrate EL-attention into existing models and compare performance at model level. Experiment setup is introduced in § 4.2 and main

Table 1. Computational and memory complexity for three groups of operations in attention. When caching key and value, multi-head attention only calculates key and value for the first time and re-uses them in the following steps. We mark its computational complexity as 0 for simplicity. Our EL-attention does not depend on multi-head key and value from group 1. Assuming sequence length is bigger than the number of heads, it has lower memory complexity for group 3 when using beam search. For notations, n is sequence length, d_m is model dimension, h is number of heads and x is beam size.

	① Build Key and Value Compute bound			② Build Query Compute bound		③ Calculate Attention Memory bound	
	Multi-Head Attention		EL Attention	Multi-Head Attention	EL Attention	Multi-Head Attention	EL Attention
Cache Key/Value	✗	✓	✗	✗ or ✓	✗	✗ or ✓	✗
Compute Complexity	$\mathcal{O}(nd_m^2)$	0	0	$\mathcal{O}(d_m^2)$	$\mathcal{O}(d_m^2)$	$\mathcal{O}(xnd_m)$	$\mathcal{O}(hxnd_m)$
Memory Complexity	$\mathcal{O}(nd_m)$	$\mathcal{O}(nd_m)$	0	$\mathcal{O}(d_m)$	$\mathcal{O}(hd_m)$	$\mathcal{O}(xnd_m)$	$\mathcal{O}(nd_m + hxd_m)$

results are presented in § 4.3. Speed for various batch sizes is in § 4.4. Finally, inference accuracy is verified in § 4.5. Our code is open sourced in `https://github.com/microsoft/fastseq`.

4.1. Analysis on Synthetic Data

4.1.1. Complexity Requirement

We collect performance data for each operation in the attention function. When profiling, parameters are set to batch size 32, beam size 4, input sequence length 1024, head number 16, and model dimension 1024.

In Figure 3, x-axis is memory data movement, y-axis is floating-point operations, both are log scaled, bubble area size represents execution time. Our method EL-attention only has two groups of operations, and it is significantly faster than multi-head attention. In multi-head attention without cache, the first group of operations require more than 500 Giga floating-point operations (GFLOPs) and is undoubtedly bounded by computation. Arithmetic intensity for group 3 operations is close to 1 in baseline. Recall the threshold for memory-computation even point is from ten to hundred, see 2.4. So group 3 of baseline is significantly bounded by memory. Here baseline uses about 0.5 GB memory movement, while our method only costs 0.15 GB.

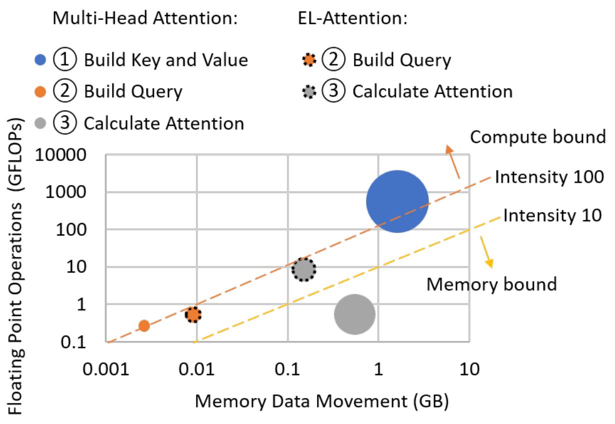

Figure 3. Performance data for operations in attention. X-axis is floating-point operations, y-axis is memory data movement, both are log scaled. Bubble area size represents execution time. EL-attention does not have group 1 operations for building key and value. See § 4.1.1 for details.

4.1.2. Speed Analysis

In this subsection, we compare execution time per sample by using synthetic input. For studying impact on sequence length n, values $\{2^6, 2^7, 2^8, 2^9, 2^{10}\}$ are used, and beam size is set to 4. For studying impact on beam size x, values $\{1, 2, 4, 8\}$ are used, and sequence length is set to 1024. We scale the batch size inversely with the sequence length and beam size. When comparing speed, parameter setting is kept the same for three attention methods. Results are shown in Figure 4.

As expected, the execution time for multi-head attention with or without cache increase linearly with respect to sequence length and beam size. Our method EL-attention is faster than multi-head attention (both with and without cache) for all configurations. Compared to multi-head attention without cache, EL-attention is up to 36x faster. Compared to multi-head attention with cache, the speed up ratio grows from 1.4x to 5x when increasing sequence length. Similarly, our method enlarges speed gain from 2x to 5x when increasing beam size from 1 (greedy search) to 8. EL-attention's execution time increases sub-linearly at the beginning when sequence length and beam size are small. Starting from certain threshold (around sequence length 256, beam size 4), arithmetic intensity is larger than the even point, it starts increasing linearly due to shift from memory bound to compute bound.

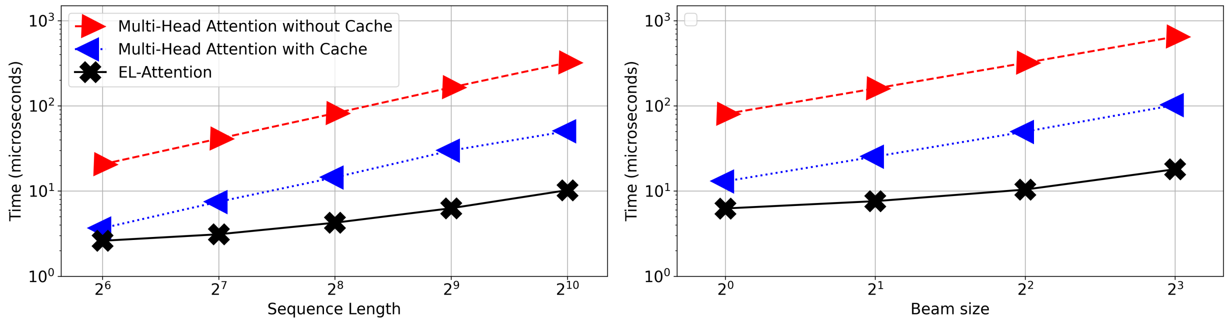

Figure 4. Comparison of attention execution time on various sequence lengths and beam sizes for EL-attention, multi-head attention without cache, and multi-head attention with cache. Axes are log-log scaled. Our method EL-attention has the fastest speed in all settings. Full details of this experiment can be found in § 4.1.2.

4.2. Setup

In this section, we introduce datasets and models used for studying inference speed in this work. We conduct experiments on a NVIDIA Tesla V100 PCIe 16GB. We choose the maximum batch size that can fit in GPU memory.

4.2.1. Datasets

Two summarization datasets and a question generation dataset are used as benchmarks.

SQuAD 1.1 (Rajpurkar et al., 2016) contains over 100K questions in 536 Wikipedia articles. The version redistributed by GLGE (Liu et al., 2020) has 75722/10570/11877 samples in training/validation/test set. The average input and output length are 149.4 and 11.5 respectively.

XSum (Narayan et al., 2018) consists online articles from BBC. There are 204017/11327/11333 samples in training/validation/test set. The average input and output length are 358.5 and 21.1 respectively.

CNN/DailyMail (Hermann et al., 2015) contains articles from CNN and Daily Mail newspapers. There are 287113/13368/11490 samples in training/validation/test set. The average input and output length are 822.3 and 57.9.

4.2.2. Models

EL-attention can be applied to a wide range of multi-head attention based models. Here we select three popular models (Transformer, BART, and GPT-2) as representatives. They cover two widely used architectures: encoder-decoder and decoder only. Experiments load their existing model checkpoints without extra training effort.

Transformer (Vaswani et al., 2017) is a widely studied encoder-decoder model with attention function. We use checkpoint shared by GLGE (Liu et al., 2020) which is trained from scratch on SQuAD 1.1.

BART (Lewis et al., 2020) is another popular encoder-decoder model which is pretrained via denoising. We use two official released checkpoints which are finetuned on CNN/DailyMail and XSum respectively.

GPT-2 (Radford et al., 2019) is a decoder only model, we load its released pretrain checkpoint and do inference on summarization task by following their paper.

For Transformer model and BART model, we use implementations in FairSeq (Ott et al., 2019) v0.9.0; for GPT-2 model, we use Huggingface Transformers (Wolf et al., 2020) v3.0.2. Based on these implementations, multi-head attention is replaced by our EL-attention.

4.2.3. Inference Parameters

First, we list parameters for beam search. In SQuAD 1.1 task, we set length penalty to 1.0, max input length is 512, and beam size is 4. In XSum task, we use parameters listed in BART[2], with length penalty 1.0, max input length 1024, max output length 60, min output length 10, and beam size 6. In CNN/DailyMail task, we conduct experiments for both BART and GPT-2. For BART model, we follow their parameters, with length penalty 2.0, max input length 1024, max output length 140, min output length 55, and beam size 4. For GPT-2 model, following their paper, we directly use pretrained model checkpoint to generate summarization, max input length is 512, max output length is set to 200.

In diverse beam search, we set diverse beam group same as beam size, and diverse beam strength is set to 0.2.

When switching from beam search to greedy search, we change the beam size to 1 and keep all other parameters unchanged.

[2]https://github.com/pytorch/fairseq/blob/master/examples/bart

Table 2. Inference speed (samples/second) comparison across decoding methods, models, tasks, and computation precision. Speed up ratio is calculated within the same precision. Single precision marked as fp32 and half precision marked as fp16. See details in § 4.3.

Model	Parameter Number	Task	Multi-Head Attention fp16 (fp32)	EL-Attention fp16 (fp32)	Speed Up Ratio fp16 (fp32)
			Beam Search		
Transformer	270M	SQuAD 1.1	170.9 (86.6)	458.1 (173.6)	**2.7x** (2.0x)
$\text{BART}_{\text{large}}$	400M	XSum	14.7 (6.8)	69.4 (26.3)	**4.7x** (3.9x)
$\text{BART}_{\text{large}}$	400M	CNN/DailyMail	5.7 (3.4)	28.6 (12.2)	**5.0x** (3.6x)
$\text{GPT-2}_{\text{small}}$	117M	CNN/DailyMail	2.1 (1.5)	3.8 (2.5)	**1.8x** (1.7x)
$\text{GPT-2}_{\text{medium}}$	345M	CNN/DailyMail	0.9 (0.6)	2.0 (1.1)	**2.2x** (1.8x)
			Diverse Beam Search		
Transformer	270M	SQuAD 1.1	162.3 (82.1)	454.1 (171.8)	**2.8x** (2.1x)
$\text{BART}_{\text{large}}$	400M	XSum	15.8 (7.2)	71.9 (27.5)	**4.6x** (3.8x)
$\text{BART}_{\text{large}}$	400M	CNN/DailyMail	5.4 (3.2)	28.5 (12.0)	**5.3x** (3.8x)
			Greedy Search		
Transformer	270M	SQuAD 1.1	436.4 (190.3)	699.7 (260.3)	**1.6x** (1.4x)
$\text{BART}_{\text{large}}$	400M	XSum	42.6 (15.0)	107.8 (44.9)	**2.5x** (3.0x)
$\text{BART}_{\text{large}}$	400M	CNN/DailyMail	13.0 (7.0)	40.0 (19.5)	**3.1x** (2.8x)
$\text{GPT-2}_{\text{small}}$	117M	CNN/DailyMail	14.7 (9.0)	26.2 (13.6)	**1.8x** (1.5x)
$\text{GPT-2}_{\text{medium}}$	345M	CNN/DailyMail	5.9 (3.6)	10.4 (5.9)	**1.8x** (1.6x)

4.3. Main Results

We present inference speed for three model types (total five models), three tasks, three decoding methods (beam search, diverse beam search and greedy search) and two computation precision settings (fp16 and fp32) in Table 2. Compared with the baseline, EL-attention achieves 1.6x to 5.3x speedup for all models on these tasks and decoding methods.

In beam search, the speed-up ratio is between 1.8x and 5.0x for half precision and 1.7x to 3.9x for single precision. For example, EL-attention has 5x speed-up for $\text{BART}_{\text{large}}$ model when inference on the CNN/DailyMail. EL-attention shows more than twice speed up for Transformer model on SQuAD 1.1. For GPT-2 model, speed is 1.8x and 2.2x for small and medium model size, respectively[3].

In diverse beam search, there is a similar speedup ratio as that in beam search[4]. Transformer model on SQuAD 1.1 dataset is accelerated to 2.8x, BART model on XSum and CNN/DailyMail has speed up 4.6x and 5.3x, respectively.

In greedy search, EL-attention achieves speedup from 1.6x to 3.1x for half precision. For example, speedup of BART model is 3.1x on summarization tasks CNN/DailyMail. GPT-2 model on this task shows 1.8x speed up.

[3]The absolute speed for GPT-2 model are smaller due to a bottleneck operation in Huggingface Transformers v3.0.2, which searches for finished beams sequentially.

[4]Diverse beam search experiments for GPT-2 model are skipped because we did not find this feature in Huggingface Transformers v3.0.2.

In general, EL-attention has more significant speed gain for longer input and larger model size. The speedup ratio is higher when using half precision than single precision, due to half precision has higher arithmetic intensity threshold that balances memory bound and compute bound.

4.4. Speed on Various Batch Sizes

In this section, We study inference speed on varied batch sizes and present their cache size differences. We include another baseline multi-head attention without cache here.

In Table 3, compared with multi-head attention without cache, when using the same batch size 32, speed is 3x by using cache, speed is 4.2x by using EL-attention. Our method can further enlarge speed gain to 15.1x when batch size grows to 320. Here are two explanations why bigger batch increases speed: 1) higher arithmetic intensity from more calculations per memory movement; 2) longer execution time per instruction, which mitigates latency between CPU and GPU communication. As expected, among the three methods, multi-head attention with cache consumes the most memory, therefore, supports the smallest batch size. While EL-attention can support batch size up to 320, multi-head attention without cache will be OOM at batch size 128, because it re-computes states for all previously generated tokens at each step, which consumes more run-time memory. Multi-head attention without cache has no speed gain by increasing batch size since it already has high arithmetic intensity and long execution time per instruction even in small batch size. However, most of these calculations are duplicated efforts.

Table 3. Inference speed (samples/second) on different batch sizes. OOM means out of memory. All speedup ratios are compared to the same cell value of no cache and batch size 32.

	Multi-Head Attention		EL-Attention
Batch size	No Cache	Has Cache	
32	1.9 (1x)	5.7 (3x)	8.0 (4.2x)
64	1.9 (1x)	OOM	12.6 (6.6x)
128	OOM	OOM	21.3 (11.2x)
320	OOM	OOM	28.6 (15.1x)

Table 4. Comparison of memory sizes used for storing input related model states, see § 4.4 for detail.

Sequence Length	Multi-Head Attention	EL Attention
Batch size 32		
256	1.5 GB	0.02 GB
1024	6 GB	0.06 GB
Batch size 64		
256	3 GB	0.03 GB
1024	12 GB	0.13 GB
Batch size 320		
256	15 GB	0.15 GB
1024	60 GB	0.63 GB

We compare cache sizes required for storing input related model states between multi-head attention and EL-attention in Table 4. Our method only stores encoder output, so the memory size is 96x smaller compared to multi-head attention. Refer to Figure 1 for the relationship between encoder output and key-value pairs. These numbers are calculated based on BART$_{\text{large}}$ and half precision.

4.5. Accuracy Verification

At first, we direct load the BART released checkpoint for inference, then calculate the ROUGE score. Our reproduced results are similar to the numbers reported in their paper. Then we replace multi-head attention with EL-attention, there is no result change when using single precision (fp32). Compared result between half precision (fp16) and fp32, there are slight differences in both multi-head attention and EL-attention due to low precision. But the rouge score differences are minimal. We report all the ROUGE scores in Table 5.

5. Related Work

5.1. Transformer Speed up

Many works focus on improving inference speed for Transformer (Vaswani et al., 2017) and variant models. 1) Reducing complexity on sequence length. PoWER-BERT (Goyal et al., 2020) studies progressive word-vector elimination, Linformer (Wang et al., 2020a) proposals attention with linear complexity, Reformer (Kitaev et al., 2020) reduces complexity by locality-sensitive hash, BigBird (Zaheer et al., 2020) and LongFormer (Beltagy et al., 2020) proposes sparse attention with global tokens. Linear Transformers (Katharopoulos et al., 2020) only stores accumulated states instead of maintaining every representation. Sparse Sinkhorn Attention (Tay et al., 2020) reduces memory complexity based on differentiable sorting. 2) Reducing model size. Fixed Multi-Head Attention (Bhojanapalli et al., 2020) studies choosing d_k based on the input sequence length. One Write-Head (Shazeer, 2019) shares one head of key and value for multi-head query and achieves speed up with minor quality degradation. LayerDrop (Fan et al., 2019) enables efficient layer pruning at inference stage. 3) Non-autoregressive generation. Gu et al. (2018); Lee et al. (2018); Qi et al. (2020) speed up inference by predicting all tokens in single step instead of step-by-step generation.

Table 5. Evaluate EL-attention's impact on generation quality using CNN/DailyMail test set.

Model	ROUGE-1	ROUGE-2	ROUGE-L
BART$_{\text{large}}$	44.16	21.28	40.90
Our reproduce (fp32)	44.21	21.20	41.03
+EL-attention (fp32)	44.21	21.20	41.03
Our reproduce (fp16)	44.22	21.20	41.04
+EL-attention (fp16)	44.22	21.21	41.05

5.2. Roofline Performance Analysis

The Roofline model (Williams et al., 2009) provides an intuitive and insightful approach to identifying performance bottleneck. The attainable floating-point performance is affected by peak floating-point performance, peak memory bandwidth, and arithmetic intensity. Yang et al. (2020) constructs a hierarchical Roofline on NVIDIA GPU and extends it to support half precision and Tensor Cores, this hierarchical Roofline incorporates L1, L2, device memory, and system memory bandwidths. Wang et al. (2019) studies this problem across platforms on TPU, GPU, and CPU. Wang et al. (2020b) presents a practical methodology for collecting performance data to conduct hierarchical Roofline analysis on NVIDIA GPU.

6. Conclusion

In this work, we present EL-attention, a technology that significantly reduces memory cost and increases inference speed. EL-attention can be directly applied to existing model checkpoints without accuracy loss. Because of the massive memory savings, it might be especially helpful for efficient inference on memory limited devices, like mobile and IoT devices. We leave them as future work.

References

Beltagy, I., Peters, M. E., and Cohan, A. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Bhojanapalli, S., Yun, C., Rawat, A. S., Reddi, S., and Kumar, S. Low-rank bottleneck in multi-head attention models. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 864–873. PMLR, 13–18 Jul 2020. URL http://proceedings.mlr.press/v119/bhojanapalli20a.html.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Cho, J., Lu, J., Schwenk, D., Hajishirzi, H., and Kembhavi, A. X-LXMERT: Paint, Caption and Answer Questions with Multi-Modal Transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8785–8805, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.707. URL https://www.aclweb.org/anthology/2020.emnlp-main.707.

Fan, A., Grave, E., and Joulin, A. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*, 2019.

Goyal, S., Choudhury, A. R., Raje, S., Chakaravarthy, V., Sabharwal, Y., and Verma, A. Power-bert: Accelerating bert inference via progressive word-vector elimination. In *International Conference on Machine Learning*, pp. 3690–3699. PMLR, 2020.

Gu, J., Bradbury, J., Xiong, C., Li, V. O., and Socher, R. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1l8BtlCb.

Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. Teaching machines to read and comprehend. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/afdec7005cc9f14302cd0474fd0f3c96-Paper.pdf.

Huang, C.-Z. A., Vaswani, A., Uszkoreit, J., Shazeer, N., Simon, I., Hawthorne, C., Dai, A. M., Hoffman, M. D., Dinculescu, M., and Eck, D. Music transformer. *arXiv preprint arXiv:1809.04281*, 2018.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are RNNs: Fast autoregressive transformers with linear attention. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 13–18 Jul 2020. URL http://proceedings.mlr.press/v119/katharopoulos20a.html.

Kitaev, N., Kaiser, L., and Levskaya, A. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgNKkHtvB.

Lee, J., Mansimov, E., and Cho, K. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1173–1182, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1149. URL https://www.aclweb.org/anthology/D18-1149.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7871–7880, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.703. URL https://www.aclweb.org/anthology/2020.acl-main.703.

Liu, D., Yan, Y., Gong, Y., Qi, W., Zhang, H., Jiao, J., Chen, W., Fu, J., Shou, L., Gong, M., et al. Glge: A new general language generation evaluation benchmark. *arXiv preprint arXiv:2011.11928*, 2020.

Narayan, S., Cohen, S. B., and Lapata, M. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1797–1807, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1206. URL https://www.aclweb.org/anthology/D18-1206.

Ott, M., Edunov, S., Baevski, A., Fan, A., Gross, S., Ng, N., Grangier, D., and Auli, M. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American*

Chapter of the Association for Computational Linguistics (Demonstrations), pp. 48–53, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-4009. URL https://www.aclweb.org/anthology/N19-4009.

Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., and Tran, D. Image transformer. In *International Conference on Machine Learning*, pp. 4055–4064. PMLR, 2018.

Qi, W., Gong, Y., Jiao, J., Yan, Y., Liu, D., Chen, W., Tang, K., Li, H., Chen, J., Zhang, R., et al. Bang: Bridging autoregressive and non-autoregressive generation with large scale pretraining. *arXiv preprint arXiv:2012.15525*, 2020.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://www.aclweb.org/anthology/D16-1264.

Shazeer, N. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Shleifer, S. and Rush, A. M. Pre-trained summarization distillation. *arXiv preprint arXiv:2010.13002*, 2020.

Tay, Y., Bahri, D., Yang, L., Metzler, D., and Juan, D.-C. Sparse sinkhorn attention. In *International Conference on Machine Learning*, pp. 9438–9447. PMLR, 2020.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., and Metzler, D. Long range arena : A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=qVyeW-grC2k.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Vaswani, A., Bengio, S., Brevdo, E., Chollet, F., Gomez, A., Gouws, S., Jones, L., Kaiser, Ł., Kalchbrenner, N., Parmar, N., Sepassi, R., Shazeer, N., and Uszkoreit, J. Tensor2Tensor for neural machine translation. In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas (Volume 1: Research Track)*, pp. 193–199, Boston, MA, March 2018. Association for Machine Translation in the Americas. URL https://www.aclweb.org/anthology/W18-1819.

Wang, S., Li, B., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020a.

Wang, Y., Yang, C., Farrell, S., Kurth, T., and Williams, S. Hierarchical roofline performance analysis for deep learning applications. *arXiv preprint arXiv:2009.05257*, 2020b.

Wang, Y. E., Wei, G.-Y., and Brooks, D. Benchmarking tpu, gpu, and cpu platforms for deep learning. *arXiv preprint arXiv:1907.10701*, 2019.

Williams, S., Waterman, A., and Patterson, D. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Yang, C., Kurth, T., and Williams, S. Hierarchical roofline analysis for gpus: Accelerating performance optimization for the nersc-9 perlmutter system. *Concurrency and Computation: Practice and Experience*, 32(20):e5547, 2020.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., and Ahmed, A. Big bird: Transformers for longer sequences. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 17283–17297. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/c8512d142a2d849725f31a9a7a361ab9-Paper.pdf.

A. Pseudocode

To better understand differences between EL-attention and multi-head attention in generation, we list their pseudocode for the comparison. For simplicity, bias term and some non-essential operations are omitted.

Algorithm 1 shows the generation process for a typical encoder-decoder model. First, input is encoded by model, then decoder starts repetitively executing, one token is generated per step. The high level logic is the same for both attention methods, differences are in the cache and computation.

The generation process with multi-head attention is listed in Algorithm 2. First, the encoder output is repeated(h) beam size times to match query's first dimension. Second, the key and value cache are built for every layer. So its cache size is linear to number of the decoder layer L, beam size x and batch size b.

EL-attention (See Algorithm 3) does not need to repeat the encoder output and get rid of the per layer cache for storing key and value. To achieve these benefits, EL-attention shifts some computation on key and value to the query side. Due to the length of query is always one which is much shorter than the length of key and value in most cases, it reduces the computations significantly.

EL-attention can achieve faster speed due to time saved from reorder_cache(), two torch.bmm operations, and the ability to use much larger batch size.

B. Proof for EL-Attention

In this section, we will present the proof that EL-attention can have the same result as multi-head attention via the choice of FFN functions. And again, there is no explicit conversion on key and value.

Recall that, by expanding Equation 1 and 2 from the paper, multi-head attention can be formulated as:

$$\begin{aligned}&\text{MultiHead}(Q,K,V)=\sum_{i=1}^{h}\overbrace{\text{softmax}(\frac{Q_iK_i^T}{\sqrt{d_k}})}^{\text{Prob}_i}V_iW_i^O\\&\text{where } Q_i=QW_i^Q+b_i^Q,\ K_i=KW_i^K+b_i^K,\\&\quad V_i=VW_i^V+b_i^V,\text{ and } H=K=V\end{aligned}\tag{6}$$

Here, Q $\in\mathbb{R}^{1\times d_m}$, H $\in\mathbb{R}^{n\times d_m}$, $W_i^Q, W_i^K, W_i^V\in\mathbb{R}^{d_m\times d_k}$ and $W_i^O\in\mathbb{R}^{d_k\times d_m}$. We include bias term in this proof, it is omitted in previous equations for simplification.

The multiplication of single head query and single head key can be replaced by the multiplication of expanded query and original key, derived as:

$$\begin{aligned}Q_iK_i^T&=(QW_i^Q+b_i^Q)(KW_i^K+b_i^K)^T\\&=(QW_i^Q+b_i^Q)((KW_i)^T\\&\quad+(QW_i^Q+b_i^Q)(b_i^K)^T\\&=\text{FFN}_i^Q(Q)K^T+Q_i(b_i^K)^T\\&\text{where FFN}_i^Q(Q)=(QW_i^Q+b_i^Q)(W_i^K)^T\\&\text{and } Q_i=QW_i^Q+b_i^Q\end{aligned}\tag{7}$$

Below is the EL-attention conversion from single head attention result to final output in multi-head attention:

$$\begin{aligned}\text{Prob}_i\cdot V_i\cdot W_i^O&=\text{Prob}_i(VW_i^V+b_i^V)W_i^O\\&=\text{Prob}_i(VW_i^V)W_i^O\\&\quad+\text{Prob}_i\cdot\text{Repeat}(b_i^V)\cdot W_i^O\\&=\text{FFN}_i^O(X)+b_i^VW_i^O\\&\text{where FFN}_i^O(X)=XW_i^VW_i^O\\&\text{and } X=\text{Prob}_i\cdot V\\&\text{and Repeat}(b_i^V)\text{ is broadcasting dim}\end{aligned}\tag{8}$$

To ensure the equivalence to multi-head attention, we adjust EL-attention as:

$$\begin{aligned}&\text{EL}(Q,K,V)=\sum_{i=1}^{h}\text{FFN}_i^O(\text{Prob}_i\cdot V)+\sum_{i=1}^{h}b_i^VW_i^O\\&\text{where Prob}_i=\text{softmax}(\frac{\text{FFN}_i^Q(Q)K^T+Q_i(b_i^K)^T}{\sqrt{d_k}})\\&\quad\text{and } H=K=V\end{aligned}\tag{9}$$

By leveraging the associative property of matrix multiplication, Equation 6 and Equation 9 are interchangeable.

Please note that some bias terms can be omitted when training a new model. Like the bias term b_i^K that adding the same value for all attention positions, and b_i^V that contributing constant information to the output, it is independent of query/key/value and the sum of all elements in Prob$_i$'s last dimension is always one.

Algorithm 1 Generation Process

```
Input: data $src\_tokens$, beam size $x$
Output: $tokens$
$encoder\_outs$ = forward_encoder($src\_tokens$)
Initialize $previous\_output[:]$ = BOS
Initialize $tokens = array$
for $t = 0$ to $T$ do
  $logits$ = forward_decoder($previous\_output, encoder\_outs$)
  $previous\_output, order\_index$ = sample($logit, x$)
  $tokens$ = reorder($tokens, order\_index$)
  $tokens[t, :] = previous\_output$
end for
```

Algorithm 2 forward_decoder with multi-head attention

```
function forward_decoder(previous_output, h)
  if $cache$ is None then
    $h$ = repeat($h$) {repeat beam size times}
  else
    reorder_cache() {Cache size: O(2BLSD), where B is beam size, L is decoder layer, S is sequence length, D is model dimension.}
  end if
  $x$ = embedding($previous\_output$)
  for $i = 0$ to layers $L$ do
    $x$ = self_attention($x$)
    $x$ = encoder_decoder_attention($x, h, h$)
    $x$ = mlp($x$)
  end for
  return predict_on_vocab($x, unembedding\_weight$)
end function

function encoder_decoder_attention(query, key, value)
  $\{query \in [bx, 1, d_m]\}$
  if $cache[i, k]$ is None then
    $cache[i, k]$ = reshape(torch.mm($key, W_k^i$))
    $cache[i, v]$ = reshape(torch.mm($value, W_v^i$))
  end if
  $k = cache[i, k]$ $\{k \in [bxh, d_k, n]\}$
  $v = cache[i, v]$ $\{v \in [bxh, n, d_k]\}$
  $\{q \in [bxh, 1, d_k]\}$
  $q$ = reshape(torch.mm($query, W_q^i$))
  $weights$ = torch.bmm($q, k$)
  $prob$ = softmax($weights$)
  $attn$ = torch.bmm($prob, v$)
  $attn$ = torch.mm($attn, W_o^i$)
end function
```

Algorithm 3 forward_decoder with EL-attention

```
function forward_decoder(previous_output, h)
  if $cache$ is not None then
    reorder_cache() {Cache size: O(SD), where S is sequence length, D is model dimension. Which is 2BL times less. }
  end if
  $k$ = reshape($h$) $\{k \in [b, d_m, n]\}$
  $v$ = reshape($h$) $\{v \in [b, n, d_m]\}$
  $x$ = embedding($previous\_output$)
  for $i = 0$ to layers $L$ do
    $x$ = self_attention($x$)
    $x$ = encoder_decoder_attention($x, k, v$)
    $x$ = mlp($x$)
  end for
  return predict_on_vocab($x, unembedding\_weight$)
end function

function encoder_decoder_attention(query, k, v)
  $\{query \in [bx, 1, d_m]\}$
  {No heavy op for building multi-head key/value.}
  {Encoder output is directly used as key and value, and shared among all layers.}
  $q$ = reshape(torch.mm($query, W_q^i$)) $\{q \in [bx, h, d_k]\}$
  $\{W_k^i \in [h, d_k, d_m]\}$
  $q$ = reshape(torch.bmm($q, W_k^i$)) $\{q \in [b, hx, d_m]\}$
  $weights$ = torch.bmm($q, k$)
  $prob$ = softmax($weights$)
  $attn$ = torch.bmm($prob, v$) $\{attn \in [b, hx, d_m]\}$
  $\{W_v^i \in [h, d_m, d_k]\}$
  $attn$ = reshape(torch.bmm($attn, W_v^i$))
  $attn$ = torch.mm($attn, W_o^i$)
end function
```