EdgeFormer: A Parameter-Efficient Transformer for On-Device Seq2seq Generation

Tao Ge  Si-Qing Chen  Furu Wei
Microsoft
{tage,sqchen,fuwei}@microsoft.com

Abstract

We introduce EdgeFormer – a parameter-efficient Transformer for on-device seq2seq generation under the strict computation and memory constraints. Compared with the previous parameter-efficient Transformers, EdgeFormer applies two novel principles for cost-effective parameterization, allowing it to perform better given the same parameter budget; moreover, EdgeFormer is further enhanced by layer adaptation innovation that is proposed for improving the network with shared layers. Extensive experiments show EdgeFormer can effectively outperform previous parameter-efficient Transformer baselines and achieve competitive results under both the computation and memory constraints. Given the promising results, we release EdgeLM\textsuperscript{1} – the pretrained version of EdgeFormer, which is the first publicly available pretrained on-device seq2seq model that can be easily fine-tuned for seq2seq tasks with strong results, facilitating on-device seq2seq generation in practice.

1 Introduction

On-device modeling draws increasing attention for its unique advantages (Dhar et al., 2019). On the other hand, strict resource constraints prevent many neural networks performing well in the on-device setting. In Natural Language Processing (NLP), on-device sequence-to-sequence (seq2seq) generation remains challenging, especially for the Transformer (Vaswani et al., 2017) under strict resource constraints in both computation and memory.

To customize the Transformer for seq2seq tasks in the on-device setting, we propose EdgeFormer – a novel parameter-efficient Transformer of the encoder-decoder architecture. EdgeFormer is structurally similar to the standard Transformer with a deep encoder and shallow decoder, but with a modification that it uses an interleaved decoder with shared lightweight feed-forward networks, as shown in Figure 1. The modified decoder architecture allows EdgeFormer to apply two novel principles that we propose for cost-effective parameterization: 1) encoder-favored parameterization that suggests we parameterize the encoder using as many parameters as possible; 2) load-balanced parameterization that suggests we balance the load of model parameters to avoid them being either underused or overused in a neural network with shared parameterization.

Figure 1: (a) Vanilla Transformer decoder layer in which $d_{ffn} > d$; (b) Interleaved Transformer decoder layer with shared lightweight FFNs in which $d_{ffn} < d$.

In addition to cost-effective parameterization, EdgeFormer proposes and applies layer adaptation to further improve the model with tied layers, as Figure 2 shows. Inspired by parameter-efficient task transfer, we investigate 3 efficient layer adaptation approaches for improving the performance with negligible cost. We show EdgeFormer (with fewer than 10 million model parameters) largely outperforms the strong Universal Transformer baselines in the on-device setting with competitive results, and the int8-quantized EdgeFormer can perform high-quality on-device seq2seq generation within around 100ms latency (20-30 sequence length on average) using two mid-to-high-end CPU cores and less than 50MB RAM.

The contributions of this work are three-fold:

- This paper is one of the earliest work that formally studies on-device seq2seq generation by discussing its challenges and defining

\textsuperscript{1}https://github.com/microsoft/unilm/tree/master/edgelm
a practical setting with appropriate resource constraints for the evaluation.

- We propose \textsc{EDGFORMER}, a parameter-efficient Transformer with novel cost-effective parameterization and layer adaptation, achieving the state-of-the-art result in the on-device seq2seq generation setting under strict computing and memory resource constraints.

- We introduce and release \textsc{EDGELM} (the pretrained \textsc{EDGFORMER}) — the first publicly available pretrained on-device seq2seq model that can be easily fine-tuned for seq2seq tasks with strong results, which can largely reduce the effort for delivering a powerful on-device seq2seq model in practice.

2 Background: Transformer

2.1 Architecture

The Transformer follows the encoder-decoder architecture. The Transformer encoder consists of a stack of encoder layers, each of which has a self-attention module parameterized by projection matrices for the query, key, value and output: \([W^{Q}, W^{K}, W^{V}, W^{O}]\) whose shapes are all \(d \times d\), followed\(^2\) by a feed-forward network (FFN) parameterized by \(W^{f1} \in \mathbb{R}^{d \times d_{ffn}}\) and \(W^{f2} \in \mathbb{R}^{d_{ffn} \times d}\).

The Transformer decoder consists of a stack of decoder layers whose architecture is similar to an encoder layer except for an additional cross attention module between self-attention and FFN.

\(^2\)For simplicity, we omit discussing the layer normalization and residual connection that are not related with this work.

In summary, we understand that the main parameters in an encoder layer \(i\) are:

\[ \Phi_{e_i} = [W_{e_i}^{Q,K,V,O}, W_{e_i}^{f1}, W_{e_i}^{f2}] \]

\(|\Phi_{e_i}| = 4d^2 + 2d \times d_{encffn}\). For a decoder layer \(j\), its main parameters are:

\[ \Phi_{d_j} = [W_{d_j}^{Q,K,V,O}, W_{d_j}^{f1}, W_{d_j}^{f2}] \]

where \(W_{d_j}^{Q,K,V,O}\) is the cross-attention module. \(|\Phi_{d_j}| = 8d^2 + 2d \times d_{decffn}\).

2.2 Parameterization: Full vs Shared

Full Parameterization Full parameterization is a common parameterization approach for Transformer, meaning that each model parameter (excluding embedding) is independent without being shared by multiple modules in the network. In a forward pass, each parameter is used only once. Full parameterization allows parameters to be flexible to fit their roles well during model training.

Shared Parameterization Despite the advantages of full parameterization, it is criticized to use large numbers of parameters inefficiently, motivating shared parameterization where multiple modules in a network share parameters. For a model with shared parameterization (e.g., ALBERT), each model parameter is exploited more than once in a forward pass. The efficient parameterization can lead to a better result given the same parameterization budget, despite slowing down inference. On the other hand, given a fixed architecture (i.e., the same network depth and width), shared parameterization usually underperforms full parameterization because it has much fewer free model parameters.

Figure 2: (a) Encoder layers with shared weights (the same color) without layer adaptation: the tied weights undermine the specialities of encoder layers to process their specific inputs; (b) Bias-based Layer Adaptation (Bias-LA) employs free bias terms to adapt layers with tied weights to fit their specific roles well; (c) Adapter-LA uses a layer-specific LoRA adaptation block with rank \(r < d\) for layer adaptation; (d) Prefix-LA uses \(L\) layer-specific tokens (i.e., learnable parameters) as the prefix (dotted square) to adapt the \(m\)th layer.
3 Constraints for On-device Seq2seq

Computation On-device computer vision (CV) models tend to use 1G FLOPS (0.5G MACS) as a constraint, which is directly followed by previous work on on-device translation (Wu et al., 2020). In our work, however, we propose to relax the FLOPS constraint for typical seq2seq tasks to 2G FLOPS (1G MACS) because the latency requirement for on-device seq2seq generation is not so rigid as CV tasks and it is uncommon for an on-device seq2seq model to handle too many concurrent requests in practice. The relaxed constraint allows better prediction quality that strongly correlates with user experience for seq2seq tasks, but still ensures the CPU on edge devices to process tens of sentences per second, which is more than sufficient for on-device seq2seq model. In addition to FLOPS that is a theoretical hardware-independent measurement for computational cost, we also require the runtime latency for an input sentence (typically 20 ~ 30 tokens on average) to be within around 100ms using two mid-to-high-end CPU cores.

Memory In contrast to deploying a model on a cloud server without caring about memory cost much, there is a very strict memory constraint for an on-device model in practice, because a user’s edge device (e.g., PC) is not only for model hosting; instead, it usually runs many other (background) apps and programs at the same time besides the model. To ensure moderate memory cost, we limit the number of model parameters (excluding word embedding lookup table) up to 10 million, following previous work (Wu et al., 2020), and require the runtime memory footprint to be less than 50MB.

4 EdgeFormer

4.1 Architecture
The biggest challenge for an on-device seq2seq model is regarding the model size and memory cost. As shown in Table 1, the number of parameters of a standard Transformer-base model (d=512) is about 45 million (excluding the embedding parameters), which is far beyond the parameterization budget (10 million) and unavoidably leads to massive memory cost despite acceptable FLOPS.

EdgeFormer is proposed to address the challenge. Instead of disruptive architectural changes as previous research (Wu et al., 2020; Mehta et al., 2020; Panahi et al., 2021), EdgeFormer’s architecture basically follows the standard Transformer consisting of a 12-layer encoder and 2-layer decoder, which is efficient in decoding. We mainly discuss the model with d=512 in this paper since it can achieve good performance in the on-device setting as long as it can be appropriately parameterized. The minor architectural modification we propose for EdgeFormer is using an interleaved

---

Table 1: Top: #params and FLOPS for Transformer layers. For the encoder and vanilla decoder layer, $d_{\text{in}} = 4d$; while for the interleaved decoder layer, $d_{\text{in}} = d/4$. Bottom: #params and FLOPS of whole models, where #params excludes embedding lookup, and FLOPS is measured on a sample with src/tgt length of 30 and 32K vocabulary.
Figure 3: (a) Performance of 6+6 Transformer \((d = 512)\) on the newstest2013 English-German (En-De) translation dataset (dev set): densely parameterizing the decoder is uneconomical and much less beneficial than parameterizing the encoder; (b) Comparison of \(x+2\) Transformer with full-/shared-parameterized \(x\) encoder layers on newstest2013 En-De dataset: when \(x > 6\), the performance of the Transformer with shared parameterization only improves marginally even if \(x\) continues to increase.

decoder where attention modules are interleaved with shared lightweight\(^5\) FFNs \((d_{\text{decffn}} < d\); in this work, \(d_{\text{decffn}} = d/4\)\) in each decoder layer (shown in Figure 1). The modification is helpful for cost-effective parameterization (Section 4.2):

- The interleaved structure makes the architecture of encoder and decoder layers consistent (Ma et al., 2021), facilitating shared parameterization of attention modules throughout the encoder and decoder.

- As shown in Table 1, the lightweight FFNs that interleave attention modules in the decoder reduce FLOPS and save a large number of parameters for decoder FFNs’ parameterization that is very uneconomical.

**4.2 Cost-effective Parameterization**

Due to the tight parameterization budget (i.e., 10 million), EDGEFORMER cannot be fully parameterized as in the standard way; instead, it has to adopt shared parameterization.

As a strong baseline for shared parameterization, UNIVERSAL TRANSFORMER lets all its \(M\) encoder layers share 1 group of encoder layer parameters and all its \(N\) decoder layers share 1 group of decoder layer parameters:

\[
\Phi_{e1} = \Phi_{e2} = \cdots = \Phi_{eM} \\
\Phi_{d1} = \Phi_{d2} = \cdots = \Phi_{dN}
\]

Although UNIVERSAL TRANSFORMER is a popular solution to shared parameterization, it is still not cost-effective for two reasons:

First, UNIVERSAL TRANSFORMER uses (over) half of total parameters to parameterize the decoder, which is uneconomical. As shown in Figure 3a, given a fixed architecture (6+6 Transformer, \(d = 512\)), densely parameterizing the decoder results in much less performance gain than parameterizing the encoder. This suggests we use as many parameters as possible to parameterize the encoder for the performance.

Second, UNIVERSAL TRANSFORMER does not consider load balance of model parameters, which was a rarely discussed problem until the recent emergence of Mixture-of-Expert models (Fedus et al., 2021). For the Transformers with a deep encoder and shallow decoder, UNIVERSAL TRANSFORMER’s parameterization method will overburden parameters in the encoder but underutilize parameters in the decoder. For example, for a 12+2 UNIVERSAL TRANSFORMER, a parameter in the encoder is used 12 times, while a parameter in the decoder is used only twice in a forward pass. As shown in Figure 3b, moderately reusing parameters (e.g., when \(x \leq 4\)) helps better utilize the parameters, resulting in significant performance gain without increasing parameters. However, as the shared parameters are overused (when \(x > 6\)), the performance gain will become marginal, which is intuitive because a parameter’s capability is limited. This suggests we balance the load of parameters to

---

\(^5\)As observed by Kasai et al. (2020), reducing \(d_{\text{decffn}}\) does not hurt the result much, as shown in Table 8 in Appendix A.
avoid them being either overused or underused. 
Based on the above insights, we parameterize 
\textsc{EdgeFormer} in the following two novel principles 
for cost-effective parameterization:

**Encoder-favored Parameterization** For \textsc{EdgeFormer}, we parameterize its encoder using as 
many parameters as possible: except a small number of parameters \((d^2/2)\) for all lightweight FFNs 
in the decoder, we use almost all parameters in our 
budget to parameterize the encoder. For attention 
modules in the decoder, we let them reuse (i.e., share) parameters with the attention modules in 
the encoder since attention modules in both the en-
coder and decoder work in the same mechanism 
and can be effectively shared (Dong et al., 2019). 
Thanks to the interleaved decoder architecture that 
makes the structure of encoder and decoder layers 
consistent, we let the self-attention module in a de-
coder layer share parameters with its corresponding odd layer in the encoder, and let its cross-attention 
module share with the corresponding even layer in 
the encoder, inspired by Ma et al. (2021):

\[
W_{d_j}^{[Q,K,V,O]} \overset{\text{tied}}{=} W_{e_{2j-1}}^{[Q,K,V,O]} \quad (1 \leq j \leq 2)
\]

\[
W_{d_j}^{[Q,K,V,O]} \overset{\text{tied}}{=} W_{e_{2j}}^{[Q,K,V,O]} \quad (1 \leq j \leq 2)
\]

**Load-balanced Parameterization** We try pa-
rameterizing \textsc{EdgeFormer} with a balanced load 
for each model parameter so that each parameter 
could be as equally exploited as possible in a for-
ward pass. Given the parameterization budget and 
the load balance principle, we create 2 groups of 
encoder FFN parameters equally shared by all en-
coder layers, 1 group of decoder FFN parameters 
shared by light FFNs in the decoder, and 4 groups 
of attention parameters are shared throughout the 
encoder and decoder. Except for parameters in the 
coder FFNs that are used 6 times, other parameters 
are all used 4 times in a forward pass, resulting 
in a load balanced parameterization:

\[
W_{e_i}^{[Q,K,V,O]} \overset{\text{tied}}{=} W_{e_{i+4}}^{[Q,K,V,O]} \quad (1 \leq i < 9)
\]

\[
W_{e_i}^{[f_1,f_2]} \overset{\text{tied}}{=} W_{e_{i+2}}^{[f_1,f_2]} \quad (1 \leq i < 11)
\]

\[
W_{d_j}^{[f_1,f_2]} \overset{\text{tied}}{=} W_{d_1}^{[f_1,f_2]} \quad (1 \leq j \leq 2)
\]

4.3 Layer Adaptation

Shared parameterization causes layers with tied 
weights to become less specialized, as discussed in 
Section 1. To allow tied layers to be better adapted 
to their corresponding roles, we propose layer ad-
aptation to further enhance \textsc{EdgeFormer}. Inspired 
by parameter-efficient task transfer methods, we in-
vestigate three efficient layer adaption approaches:

**Bias-based Layer Adaptation (Bias-LA)** Inspired by BitFit (Ben Zaken et al., 2021) fine-
tuning with only bias terms, we unite all bias terms 
of each layer and use them to specialize the layers 
with tied weights, as shown in Figure 2(b). As Bit-
Fit, bias-based layer adaptation introduces very few 
additional parameters without inference overhead.

**Adapter-based Layer Adaptation (Adapter-LA)** Adapter-based approaches (Houlsby et al., 2019) in-
troduce adapter modules for NLP task transfer without 
full fine-tuning. We borrow this idea for layer 
adaptation by introducing an independent adapter 
module for each layer. Specifically, we adopt the 
recently proposed LoRA (Hu et al., 2021) as our 
layer adapter, as Figure 2(c) shows. In our experiments, 
we apply the layer adapter to \(W^Q\) and \(W^V\), 
as the original paper of LoRA suggests.

**Prefix-based Layer Adaptation (Prefix-LA)** Inspired by recent work (Li & Liang, 2021; Lester 
et al., 2021) using a prefix/prompt for task transfer, 
we introduce \(L\) tokens with learnable parameters 
as a specific prefix for each layer to adapt layers 
with tied weights, as shown in Figure 2(d). The pre-
fixes are only used for keys and values in attention 
modules, which will not introduce much inference 
overhead as long as \(L\) is moderately set.

Following the encoder-favored principle in Sec-
tion 4.2, we only apply LA to encoder layers.

5 Experiments

5.1 Experimental Setting

We mainly evaluate our approach in Machine Trans-
lation (MT). We select the most popular MT bench-
mark – WMT14 English-German (En-De) transla-
tion task, which is also a touchstone for seq2seq 
evaluation, as our main test bed. To compare with 
previous work, we also evaluate WMT14 English-
French (En-Fr) translation. We follow the standard 
way to train and evaluate WMT14 En-De and En-Fr. As Ott et al. (2018), we use a joint 
source-target dictionary of 32K Byte Pair Encoding 
(BPE) for En-De, and 40K BPE for En-Fr. We 
mainly use sacreBLEU (Post, 2018) for evaluation.

We select Universal Transformer which is the 
most popular and a strong baseline of parameter-
<table>
<thead>
<tr>
<th>Model</th>
<th>#Params</th>
<th>FLOPS</th>
<th>sacreBLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teacher</td>
<td>176 M</td>
<td>6.7 G</td>
<td>29.3</td>
</tr>
<tr>
<td>6+6 Transformer (full enc, full dec)</td>
<td>44 M</td>
<td>1.8 G</td>
<td>28.5</td>
</tr>
<tr>
<td>6+6 Transformer (full enc, shared dec)</td>
<td>23 M</td>
<td>1.8 G</td>
<td>28.2</td>
</tr>
<tr>
<td>6+6 Transformer (full dec, shared enc)</td>
<td>28 M</td>
<td>1.8 G</td>
<td>27.3</td>
</tr>
<tr>
<td>12+2 Transformer (full enc, full dec)</td>
<td>46 M</td>
<td>1.9 G</td>
<td>28.5</td>
</tr>
<tr>
<td>12+2 Transformer (full enc, shared dec)</td>
<td>42 M</td>
<td>1.9 G</td>
<td>28.4</td>
</tr>
<tr>
<td>12+2 Transformer (full dec, shared enc)</td>
<td>12 M</td>
<td>1.9 G</td>
<td>27.2</td>
</tr>
<tr>
<td>12+2 UT</td>
<td>7.4 M</td>
<td>1.9 G</td>
<td>27.0</td>
</tr>
<tr>
<td>12+2 UT ($d_{lin} = 2560$)</td>
<td>8.5 M</td>
<td>2.1 G</td>
<td>27.2</td>
</tr>
<tr>
<td>12+2 UT ($d_{actin} = 3072$)</td>
<td>8.5 M</td>
<td>2.3 G</td>
<td>27.4</td>
</tr>
<tr>
<td>12+2 UT ($d_{actin} = 3072$)</td>
<td>8.5 M</td>
<td>2.0 G</td>
<td>27.0</td>
</tr>
<tr>
<td>EDGEFORMER w/o LA (^2)</td>
<td>8.6 M</td>
<td>1.8 G</td>
<td>27.7 (^{(1)})</td>
</tr>
<tr>
<td>EDGEFORMER (Bias-LA)</td>
<td>8.6 M</td>
<td>1.8 G</td>
<td>27.8</td>
</tr>
<tr>
<td>EDGEFORMER (Adapter-LA) ($r = 32$)</td>
<td>9.4 M</td>
<td>1.8 G</td>
<td>28.0 (^{(2)})</td>
</tr>
<tr>
<td>EDGEFORMER (Prefix-LA) ($L = 8$)</td>
<td>8.6 M</td>
<td>1.9 G</td>
<td>28.0 (^{(2)})</td>
</tr>
</tbody>
</table>

Table 2: WMT14 En-De results. To fairly compare with Universal Transformer (UT) that is originally smaller than EDGEFORMER, we also test UT with larger FFNs to make its model size comparable to EDGEFORMER. \(^{(i)}\) denotes $p < 0.05$ in significance test compared with the model marked with \(^{(i)}\).

<table>
<thead>
<tr>
<th>FFNs</th>
<th>Load</th>
<th>#Params</th>
<th>FLOPS</th>
<th>sacreBLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 FFNs ($d_{ffn} = 2048$) (^1)</td>
<td>6-6</td>
<td>8.6 M</td>
<td>1.8 G</td>
<td>27.7</td>
</tr>
<tr>
<td>3 FFNs ($d_{ffn} = 1536$)</td>
<td>4-4-4</td>
<td>9.1 M</td>
<td>1.6 G</td>
<td>27.4 (^{(1)})</td>
</tr>
<tr>
<td>4 FFNs ($d_{ffn} = 1024$)</td>
<td>3-3-3-3</td>
<td>8.6 M</td>
<td>1.4 G</td>
<td>27.2 (^{(1)})</td>
</tr>
<tr>
<td>2 FFNs ($d_{ffn} = 2048$)</td>
<td>1-1-1</td>
<td>8.6 M</td>
<td>1.8 G</td>
<td>27.5 (^{(1)})</td>
</tr>
<tr>
<td>2 FFNs ($d_{ffn} = 2048$)</td>
<td>1-11</td>
<td>8.6 M</td>
<td>1.8 G</td>
<td>27.4 (^{(1)})</td>
</tr>
</tbody>
</table>

Table 3: Performance of EDGEFORMER with various encoder FFN parameterization on WMT14 En-De. Load 6-6 means the 2 groups of FFN parameters are used 6 times each, while Load 1-11 means 1 group of FFN is used once, and the other is used 11 times.

As discussed in Section 4.2, the advantage of EDGEFORMER over UT comes from its cost-effective parameterization. The encoder-favored principle is again supported by comparing 6+6 Transformers’ results in Table 2, which is consistent with the observation on the dev set in Figure 3a. To further understand the effectiveness of load-balanced parameterization principles, we conduct an ablation study by adjusting encoder FFNs in EDGEFORMER. Table 3 shows the results of EDGEFORMER with various FFN parameterization. As we reduce $d_{ffn}$ (e.g., to 1536 or 1024), we can increase the group of encoder FFN parameters and reduce their load given a fixed parameterization budget. However, such a strategy leads to a clear degradation of sacreBLEU. One reason is that the FFN parameters of a reduced load (3-4 times) are not so fully utilized as the baseline (6 times) despite other reasons such as the differences of network shape (e.g., $d_{lin}$). To minimize the effects of other factors, we compare the first group with a balanced...
Figure 4: The effects of (a) rank $r$ in Adapter-LA, and (b) prefix length $L$ in Prefix-LA on the performance in WMT14 En-De. Note that $r = 64$ will lead to exceed our parameterization budget despite better performance.

Table 4: Result comparison to previous parameter-efficient Transformers that have fewer parameters than the baseline Transformer (around 45M parameters). "-" means that the metrics are unavailable or not comparable in the original paper. The underlines denote that the metrics cannot meet the on-device requirement. Note that all the models in this table do not apply Seq-KD.

<table>
<thead>
<tr>
<th>Model</th>
<th>#params</th>
<th>FLOPS</th>
<th>En-De</th>
<th>En-Fr</th>
</tr>
</thead>
<tbody>
<tr>
<td>6+6 Transformer</td>
<td>44M</td>
<td>1.8G</td>
<td>28.3</td>
<td>41.0</td>
</tr>
<tr>
<td>12+2 Transformer</td>
<td>46M</td>
<td>1.9G</td>
<td>28.4</td>
<td>41.4</td>
</tr>
<tr>
<td>12+2 UT</td>
<td>7.4M</td>
<td>1.9G</td>
<td>26.2</td>
<td>39.2</td>
</tr>
<tr>
<td>DeLighT</td>
<td>31.4M</td>
<td>-</td>
<td>27.6</td>
<td>39.6</td>
</tr>
<tr>
<td>Shapeshifter</td>
<td>8.2M</td>
<td>-</td>
<td>26.6</td>
<td>40.8</td>
</tr>
<tr>
<td>Lite Transformer (small)</td>
<td>2.9M</td>
<td>0.2G</td>
<td>22.5</td>
<td>35.3</td>
</tr>
<tr>
<td>Lite Transformer (medium)</td>
<td>11.7M</td>
<td>0.7G</td>
<td>25.6</td>
<td>39.1</td>
</tr>
<tr>
<td>Lite Transformer (big)</td>
<td>17.3M</td>
<td>1.0G</td>
<td>26.5</td>
<td>39.6</td>
</tr>
<tr>
<td>EdgeFormer w/o LA</td>
<td>8.6M</td>
<td>1.8G</td>
<td>26.5</td>
<td>39.8</td>
</tr>
<tr>
<td>EdgeFormer (Adapter-LA)</td>
<td>9.4M</td>
<td>1.8G</td>
<td>26.9</td>
<td>40.5</td>
</tr>
<tr>
<td>EdgeFormer (Prefix-LA)</td>
<td>8.6M</td>
<td>1.9G</td>
<td>26.8</td>
<td>40.3</td>
</tr>
</tbody>
</table>

parameter load (i.e., 6-6) and the last group with an imbalanced parameter load (1-11 or 11-1), showing load-balanced parameterization is consistently better than the imbalanced counterparts.

After discussing parameterization, we then analyze the effects of layer adaptation on the results by mainly focusing on Adapter-LA and Prefix-LA that both show performance gain. Figure 4 shows the effects of the rank $r$ in Adapter-LA and prefix length $L$ in Prefix-LA. As $r$ increases, the model performance will gradually improve. However, when $r$ becomes large (e.g., $r \geq 64$), it will exceed our parameterization budget and thus the gain will become meaningless. As for prefix length $L$ in Prefix-LA, it is different from $r$ that it will not keep improving the results as it increases: the gain can hardly be observed after some length (e.g., $L = 8$), which is similar to the observation in prefix-tuning (Li & Liang, 2021). Therefore, we use $r = 32$ and $L = 8$ as the default setting to report the results of Adapter-LA and Prefix-LA.

Finally, we compare EdgeFormer with recent work on parameter-efficient Transformer modeling. To keep consistency of the training and evaluation protocols with previous work, we here give up using Seq-KD to train the models, and report BLEU (Papineni et al., 2002) for comparison. Specifically, we compare with DeLighT (Mehta et al., 2020), Shapeshifter (Panahi et al., 2021) and Lite Transformer (Wu et al., 2020), and show the results in Table 4. However, it is notable that the results are not strictly comparable because the previous studies have their own focus and setting, which are different from ours. For example, DeLighT and Lite Transformer focus much more on FLOPS than the model size, thus they do not a desirable tradeoff between the model quality and size; while Shapeshifter’s goal is minimizing the model size despite an additional 10% ~ 20% inference overhead. Regardless of these factors that
### Table 5: Runtime results for int8-quantized EDGEFORMER, in which Latency 1 and Latency 2 denote the average latency per sentence measured on the Intel® Xeon® E-2288G CPU and Qualcomm SM8150 Snapdragon 855 CPU, respectively. We run through the test set with batch size=1, and use greedy decoding instead of beam search.

<table>
<thead>
<tr>
<th>Model</th>
<th>Disk size (7zip)</th>
<th>Peak Memory</th>
<th>Latency 1</th>
<th>Latency 2</th>
<th>sacreBLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>EdgeFormer (Adapter-LA, 32k vocab)</td>
<td>28MB</td>
<td>60MB</td>
<td>65ms</td>
<td>114ms</td>
<td>27.2</td>
</tr>
<tr>
<td>EdgeFormer (Adapter-LA, 8k vocab)</td>
<td>15MB</td>
<td>47MB</td>
<td>59ms</td>
<td>101ms</td>
<td>27.1</td>
</tr>
<tr>
<td>CoNLL-14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EdgeFormer (Adapter-LA, 2k vocab)</td>
<td>11MB</td>
<td>42MB</td>
<td>51ms</td>
<td>98ms</td>
<td>50.8</td>
</tr>
</tbody>
</table>

Table 6: The performance of EDGELM in comparison with the baselines. + denotes that the results are from Chen et al. (2019).

<table>
<thead>
<tr>
<th>Model</th>
<th>#Param</th>
<th>FLOPS</th>
<th>CoNLL14 $F_{0.5}$</th>
<th>XSum</th>
<th>SQuAD-NQG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-base</td>
<td>44M</td>
<td>1.8G</td>
<td>50.1</td>
<td>31.2</td>
<td>24.9</td>
</tr>
<tr>
<td>Pretrained 12+2 UT ($d_{ffn} = 2048$)</td>
<td>7.4M</td>
<td>1.4G</td>
<td>50.8</td>
<td>36.0</td>
<td>29.2</td>
</tr>
<tr>
<td>Pretrained 12+2 UT ($d_{ffn} = 3072$)</td>
<td>9.4M</td>
<td>1.9G</td>
<td>51.1</td>
<td>36.7</td>
<td>29.7</td>
</tr>
<tr>
<td>EDGELM</td>
<td>9.4M</td>
<td>1.3G</td>
<td>52.0$^{(1)}$</td>
<td>37.2$^{(1)}$</td>
<td>20.6$^{(1)}$</td>
</tr>
</tbody>
</table>

5.3 Runtime Evaluation

We conduct experiments in WMT14 En-De translation and CoNLL-14 Grammatical Error Correction (GEC) benchmark for runtime latency and memory evaluation using onnxruntime\(^7\) that supports efficient seq2seq decoding. We apply int8-quantization to EDGEFORMER and test latency on 2 devices: a 2-core Intel® Xeon® E-2288G CPU (in PC), and a 2-core Qualcomm SM8150 Snapdragon 855 CPU (in Pixel 4), which are both current mid-to-high end CPUs launched 2-3 years ago.

Table 5 shows runtime evaluation results. With int8-quantization and smaller vocabulary, EDGEFORMER can not only meet the on-device seq2seq requirements but also maintain its good performance, demonstrating its practical values.

6 EdgeLM – The Pretrained EdgeFormer

Given the promising results, we introduce EDGELM – the pretrained\(^8\) EDGEFORMER (Adapter-LA) with 8K sentenpiece vocabulary with factorized embedding ($d_{embed} = 128$) through the same self-supervised task (i.e., masked span infilling) as T5 (Raffel et al., 2019) and make it publicly available for downstream on-device seq2seq task fine-tuning.

We evaluate EDGEML in the benchmarks of three popular seq2seq tasks: CoNLL-14 for Grammatical Error Correction (GEC), XSum (Narayan et al., 2018) for Abstractive Summarization, and SQuAD-NQG (Du et al., 2017) for Question Generation. According to Table 6, EDGEML achieves significantly better performance than the pretrained UT models as well as the Transformer-base model trained from scratch. We believe that EDGEML, as the first publicly released on-device seq2seq pretrained model, can largely facilitate on-device seq2seq generation in practice.

7 Related Work

On-device seq2seq generation in NLP is a research area that has been less explored than on-device CV and NLU (Tambe et al., 2021). Besides the general techniques like pruning, compression, quantization and knowledge distillation (Fan et al., 2019; 8We include pretraining details in the Appendix B.

\(^6\)We include experiments details of GEC in Appendix A.
\(^7\)https://github.com/microsoft/onnxruntime

\(^8\)We include pretraining details in the Appendix B.
Xu et al., 2020; Li et al., 2022) that are orthogonal to our effort, parameter-efficient Transformer-based seq2seq modeling is the most related research branch to ours. In this branch, UNIVERSAL TRANSFORMER (Dehghani et al., 2018) uses cross-layer sharing method, which is the most popular solution to parameter efficiency. Takase & Kiyono (2021) extends UNIVERSAL TRANSFORMER by studying different ways for layer sharing, and Reid et al. (2021) proposes to free the first and last encoder layer and widen the intermediate layers for better performance. However, both the approaches consider parameter-efficiency only without caring about latency becoming worse.

In addition to work improving parameter efficiency by weight sharing, there is research that studies lightweight model architecture for seq2seq learning where early work (Gehring et al., 2017; Wu et al., 2019) mainly focuses on CNNs, while recent efforts have tended to switch to attention-based models such as Mehta et al. (2020). Also, low-rank factorization has been studied intensively to make the model tiny (Zhang et al., 2021; Panahi et al., 2021); and hardware-aware network architecture search with elastic modeling (Wang et al., 2020) has been proposed recently for facilitating deployment of seq2seq models on various devices. Among previous studies, the work of Wu et al. (2020) is the most related to ours, which studies seq2seq generation in an on-device setting. However, it sets the computational constraint for on-device seq2seq to be the same with the CV tasks, which is too strict and unnecessary, as discussed in Section 3. As a result, their models focus on FLOPS optimization much more than memory, leading to an undesirable tradeoff between the quality and model size for the practical on-device seq2seq setting which should care about memory much more than latency. In contrast, our work carefully evaluates bottleneck constraints, and proposes appropriate models with parameterization and layer adaptation innovations, largely improving the results for practical on-device seq2seq generation.

8 Conclusion and Future Work

We formally study on-device seq2seq generation, including defining its practical resource constraint setting and proposing an appropriate modeling technology EDGEFORMER. The cost-effective parameterization and layer adaptation innovations in EDGEFORMER both prove effective to improve the results with negligible computation and memory cost, achieving state-of-the-art results in the on-device seq2seq generation setting. Our released pretrained EDGEFORMER – EDGE LM can be easily fine-tuned for downstream seq2seq tasks, largely facilitating on-device seq2seq generation in practice.

For future work, we plan to further study load-balanced parameterization for parameter-efficient models, which is an interesting and new but seemingly profound machine learning research problem: instead of naively assuming that all the parameters are equal in this preliminary study, we suspect that parameters in different modules (e.g., parameters in the self-attn and FFN; or parameters in different layers) should be under different amounts of load. We look forward to in-depth research on this problem, which might be helpful to deepen our understanding of neural networks.

9 Limitations

EDGEFORMER is a preliminary model proposed for on-device seq2seq generation setting, which still has much room for improvement. For example, as mentioned in Section 8, the current load balance mechanism naively assumes that the number of times that a parameter is used in a forward pass is equal to its load, which may not be always true because parameters in different modules are different: some parameters may be effectively used more times than others, which requires deeper understanding of neural network and the Transformer.

Acknowledgments

We thank all the anonymous reviewers for their valuable comments. We thank Xiaohu Tang, Fucheng Jia, Yifan Yang and Huiqiang Jiang for their help in runtime evaluation, and thank Shuming Ma, Ting Cao, Fan Yang, Qifeng Yin, Yuqing Yang and Lidong Zhou in Microsoft Research Asia for the discussion and helpful comments. We also appreciate the support from Wenbing Li, Yufeng Li, Bowen Bao, Ye Wang, Sunghoon Choi, Scott McKay and Emma Ning in Microsoft AI Frameworks for onnxruntime, and appreciate the feedback and valuable suggestions from Joshua Burkholder, Xun Wang, Weixin Cai and Zhang Li in Microsoft Office Intelligence regarding detailed constraints for on-device seq2seq generation in real-word applications.
References


A Details of Evaluations for MT and GEC

For MT, we follow the setting of Ott et al. (2018) to train the model on the WMT14 datasets. For En-De, the training set contains 4.5M parallel sentence pairs. We use newstest2013 as our dev set. For En-Fr, there are 36M parallel sentence pairs for training, and we use newstest2012+2013 as the dev set.

In GEC evaluation, we follow previous work (Chen et al., 2020; Zhou et al., 2021) to use the BEA-19 restricted setting, training with Lang-8 (Mizumoto et al., 2011), FCE (Yannakoudakis et al., 2011) and WI+LOCNESS (Bryant et al., 2019). We validate on CoNLL-13 shared task dataset (Ng et al., 2013), and test on CoNLL-14 shared task dataset (Ng et al., 2014). After de-duplicating, we have around 900K sentence pairs for training. Both the dev (CoNLL-13) and test (CoNLL-14) have 1.3K samples. We train a sentencepiece (Kudo & Richardson, 2018) model of 2K vocabulary for tokenization, and evaluate with the metric of Max-Match (Dahlmeier & Ng, 2012) $F_{0.5}$.

For model training configuration, we show in Table 7; The ablation study into the reduction of $d_{\text{deffn}}$ is presented in Table 8.

<table>
<thead>
<tr>
<th>Configurations</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of epochs</td>
<td>1000</td>
</tr>
<tr>
<td>Devices</td>
<td>8 Nvidia V100 GPU</td>
</tr>
<tr>
<td>Max tokens per GPU</td>
<td>20,000</td>
</tr>
<tr>
<td>Update Frequency</td>
<td>4</td>
</tr>
<tr>
<td>Optimizer</td>
<td>Adam $(\beta_1=0.9, \beta_2=0.99, \epsilon=1 \times 10^{-4})$</td>
</tr>
<tr>
<td>Learning rate</td>
<td>$1 \times 10^{-3}$</td>
</tr>
<tr>
<td>Learning rate scheduler</td>
<td>inverse sqrt</td>
</tr>
<tr>
<td>Warmup</td>
<td>4000</td>
</tr>
<tr>
<td>Weight decay</td>
<td>0.00001</td>
</tr>
<tr>
<td>Loss Function</td>
<td>label smoothed cross entropy (label-smoothing=0.1)</td>
</tr>
<tr>
<td>Dropout</td>
<td>[0.1, 0.2] for MT, [0.3, 0.4, 0.5] for GEC</td>
</tr>
</tbody>
</table>

Table 7: Training details for EDGEFORMER for NMT and GEC.

<table>
<thead>
<tr>
<th>$d_{\text{deffn}}$</th>
<th>#Param</th>
<th>sacreBLEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>2048</td>
<td>46M</td>
<td>26.5</td>
</tr>
<tr>
<td>512</td>
<td>37M</td>
<td>26.4</td>
</tr>
<tr>
<td>256</td>
<td>35M</td>
<td>26.4</td>
</tr>
<tr>
<td>128</td>
<td>34M</td>
<td>26.4</td>
</tr>
</tbody>
</table>

Table 8: The ablation study into the reduction of $d_{\text{deffn}}$ on a standard 6+6 Transformer on the dev set.
<table>
<thead>
<tr>
<th>Model</th>
<th>#Param</th>
<th>FLOPS</th>
<th>CoNLL-14</th>
<th>Xsum</th>
<th>QG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>$F_{0.5}$</td>
<td>$F_{0.3}$</td>
<td></td>
</tr>
<tr>
<td>Transformer</td>
<td>44M</td>
<td>1.8G</td>
<td>50.1</td>
<td>31.2</td>
<td>10.7</td>
</tr>
<tr>
<td>Pretrained UT ($d = 512, d_{ffn} = 3072$)</td>
<td>9.4M</td>
<td>1.9G</td>
<td>51.1</td>
<td>36.7</td>
<td>14.9</td>
</tr>
<tr>
<td>EDGELM ($d = 512$)</td>
<td>9.4M</td>
<td>1.3G</td>
<td>52.0</td>
<td>37.2</td>
<td>15.4</td>
</tr>
<tr>
<td>Pretrained UT ($d = 384, d_{ffn} = 2432$)</td>
<td>5.5M</td>
<td>1.1G</td>
<td>50.1</td>
<td>31.9</td>
<td>11.5</td>
</tr>
<tr>
<td>EDGELM ($d = 384$)</td>
<td>5.5M</td>
<td>0.8G</td>
<td>50.1</td>
<td>32.3</td>
<td>11.7</td>
</tr>
<tr>
<td>Pretrained UT ($d = 768, d_{ffn} = 4608$)</td>
<td>21.2M</td>
<td>4.2G</td>
<td>52.8</td>
<td>37.3</td>
<td>15.6</td>
</tr>
<tr>
<td>EDGELM ($d = 768$)</td>
<td>21.4M</td>
<td>2.9G</td>
<td>53.1</td>
<td>37.9</td>
<td>15.9</td>
</tr>
</tbody>
</table>

Table 11: The comparison between pretrained Universal Transformer (UT) and EdgeLM with Adapter-based layer adaptation ($r = d/8$). We enlarge UT’s $d_{ffn}$ to let its model size comparable with its EdgeFormer counterpart with the same $d$ for fair comparison given the same parameter budget though introducing additional cost.

<table>
<thead>
<tr>
<th>Vocab</th>
<th>$d_{emb}$</th>
<th>#Param (including embedding)</th>
<th>CoNLL-14</th>
<th>Xsum</th>
<th>QG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>$F_{0.5}$</td>
<td>$F_{0.3}$</td>
<td></td>
</tr>
<tr>
<td>spm2k</td>
<td>512</td>
<td>10.4M</td>
<td>52.7</td>
<td>36.3</td>
<td>14.8</td>
</tr>
<tr>
<td>spm8k</td>
<td>128</td>
<td>10.5M</td>
<td>52.0</td>
<td>37.2</td>
<td>15.4</td>
</tr>
<tr>
<td>spm16k</td>
<td>64</td>
<td>10.5M</td>
<td>51.6</td>
<td>37.1</td>
<td>15.1</td>
</tr>
</tbody>
</table>

Table 12: EDGELM ($d = 512$) with Adapter-LA ($r = 32$) using different vocabulary and $d_{emb}$. Except the model with spm2k whose $d_{emb} = d$, the models with spm8k and spm16k use factorized embedding to prevent increasing the total parameters.

To morphological and syntactic information of a token; when $d_{emb}$ becomes small with factorized embedding, it may not accurately capture the morphological and syntactic information of the token.