Hypoformer: Hybrid Decomposition Transformer for Edge-friendly Neural Machine Translation

Anonymous ACL submission

Abstract

Transformer has been demonstrated to be effective in Neural Machine Translation (NMT). However, it is memory-consuming and time-consuming in edge devices, resulting in some difficulties for real-time feedback. To compress and accelerate Transformer, we propose a new *Hybrid Tensor-Train (HTT)* decomposition, which retains full rank and meanwhile reduces operations and parameters. A Transformer using HTT, named as *Hypoformer*, consistently and notably outperforms the recent light-weight SOTA methods on three standard translation tasks under different parameter and speed scales. In extreme low resource scenarios, Hypoformer has $\mathbf{7.1}$ point absolute improvement in BLEU and $\mathbf{1.27}\times$ speedup than Transformer on IWSLT'14 De-En task and $\mathbf{4.0}$ higher BLEU score on WMT'14 En-De task.

1 Introduction

Transformer (Vaswani et al., 2017) is one of the most effective models in NMT. Transformer could achieve better performance by stacking more layers and increasing the embedding dimension (Liu et al., 2020; Li et al., 2021a). However, this leads to a huge scale of parameters and a slow inference speed. Various works have been recently proposed to compress transformer, such as light-weight variants (Wu et al., 2020; Mehta et al., 2021), neural architecture search (Wang et al., 2020), parameter sharing (Reid et al., 2021), low-rank approximation (Ma et al., 2019; Liu et al., 2021; Hrinchuk et al., 2020), knowledge distillation (Kasai et al., 2020), layer reassignment (Hsu et al., 2020; Bérard et al., 2021; Li et al., 2021c).

Following the line of low-rank approximation, existing works to compress NMT models 1) are specific to a part of sub-components, *e.g., self-attention networks* (Ma et al., 2019) or *embedding layer* (Hrinchuk et al., 2020); and 2) show limited or no potential for speedup during inference (Ma et al., 2019; Hrinchuk et al., 2020). However, unlike the cloud-based system, the edge devices, such as smartphones and IoTs, need real-time feedback with constrained resources, that needs faster inference. We believe that low-rank approximation has the potential for compressing and accelerating NMT models in edge devices.

In this paper, we rethink low-rank approximation methods from a general point of view. Two representative low-rank approximation methods include Matrix Factorization (MF) and Tensor-Train decomposition (TT); the former is arguably time-efficient while the latter is parameter-efficient. However, **(1)** MF will encounter a **low-rank bottleneck** with a high compression ratio, potentially leading to moderate performance drop (Thakker et al., 2020). **(2)** Transformer compression based on *high-rank* TT decelerates inference speed due to its **quadratic computational complexity**. Interestingly, TT was claimed to retain a full matrix rank by Hrinchuk et al. (2020) while the expressive power of MF is bounded to rank R (see Table 1). We believe that *low-rank* TT is potential for both effectiveness and efficiency, while preliminary experimental results show that making use of *low-rank* TT purely results in significant performance degradation (Table 2).

To explore the potential of low-rank TT for decomposition, we propose to compensate *low-rank* TT with an auxiliary dense projection matrix, resulting in a hybrid decomposition (Figure 2E), called *Hybrid Tensor-Train (HTT) Decomposition*. Inspired by Thakker et al. (2020) which decompose a matrix into two concatenated parts — a dense projection and a low-rank MF (Figure 2D). Note that HTT still retains full matrix rank and has fewer operations and parameters than the original matrix.

We apply HTT decomposition to compress Transformer-based NMT models, called *Hypoformer*. We evaluate the Hypoformer with three standard machine translation tasks (IWSLT'14

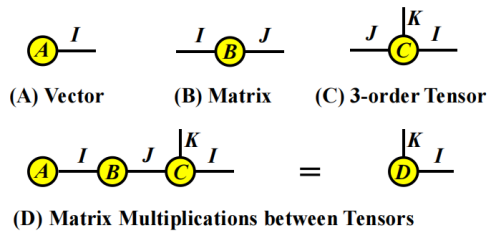

Figure 1: Given tensor graphical representations of vector (A), matrix (B) , order 3 tensor (C) and matrix multiplications between tensors (D).

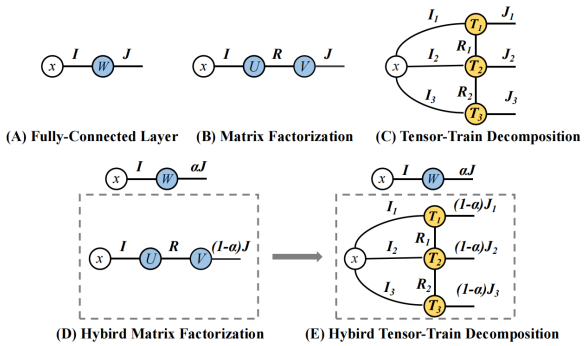

Figure 2: The operation of different decomposition with tensor representation. The *hybird* decomposition concat the output features to obtain J. α denotes the ratio of dense layer, where $\alpha \in [0, 1]$. $I = I_1 \times I_2 \times I_3$. $J = J_1 \times J_2 \times J_3$.

De-En, WMT'16 En-Ro, and WMT'14 En-De) on Raspberry Pi ARM CPU and Intel CPU. Hypoformer consistently outperforms recent light-weight state-of-the-art methods (Wu et al., 2020; Liu et al., 2020; Mehta et al., 2021; Wang et al., 2020) under various parameter scales and speed scales. Achieving similar performance on three tasks, Hypoformer can compress Transformer base model (Vaswani et al., 2017) $\mathbf{2.9} \sim \mathbf{4.5}\times$, and speedup it $\mathbf{1.9} \sim \mathbf{3.3}\times$ on Intel CPU and $\mathbf{1.8} \sim \mathbf{3.9}\times$ on Raspberry Pi ARM CPU. Notably, when the parameter scale is extreme low, Hypoformer achieves $\mathbf{7.1}$ higher BLEU score and $\mathbf{1.27}\times$ speedup than Transformer on IWSLT'14 De-En task and $\mathbf{4.0}$ higher BLEU score on WMT'14 En-De.

The contributions of this work are as follows: **(1)** Based on the analysis of previous low-rank approximation methods (Oseledets, 2011; Thakker et al., 2020), we propose a novel *Hybrid Tensor-Train (HTT) Decomposition*, which can retain full matrix rank but with fewer operations and parameters. **(2)** Applying HTT decomposition to compress and accelerate Transformer, *Hypoformer* consistently outperforms the recent light-weight SOTA methods (Wu et al., 2020; Mehta et al., 2021) with a higher compression ratio, speedup, and better BLEU score. **(3)** In edge devices with extremely limited resource, Hypoformer can retain $\mathbf{95}\%$ performance with $\mathbf{1/12}$ parameters, setting a new SOTA in extremely-small NMT models.

2 Preliminaries

Tensor Representation First, we introduce the graphical representations of tensors. An n-order tensor can be denoted as nodes with n legs. Each leg is labeled by the length (an associated positive integer) of the corresponding dimension, and the orientation of the leg does not matter. As shown in Figure 1, the vecotr $A \in \mathbb{R}^I$, the matrix $B \in \mathbb{R}^{I\times J}$ and the 3-order tensor $C \in \mathbb{R}^{J\times K\times I}$ are denoted as (A), (B), and (C) respectively.

Tensor Operation Matrix multiplication can be performed between two tensors. Two nodes share one leg means that these two tensors perform matrix multiplication in the corresponding dimension. A tensor can perform matrix multiplication with one or more tensors by sharing legs, and the order of execution does not matter. As show in Figure 1(D), three tensors perform two matrix multiplications, and the result is the matrix $D \in \mathbb{R}^{K\times I}$. Figure 2(A) illustrates a fully connected layer. That is, the input vector $X \in \mathbb{R}^I$ and the weight matrix $W \in \mathbb{R}^{I\times J}$ perform matrix multiplication, and the result is a vector $\in \mathbb{R}^J$.

Matrix Factorization is a common and effective technique to compress deep neural networks. To compress the fully connected layer, the weight matrix $W \in \mathbb{R}^{I\times J}$ is decomposed into the product of two smaller matrices $U \in \mathbb{R}^{I\times R}$ and $V \in \mathbb{R}^{R\times J}$, as shown in Figure 2(B). Matrix Factorization (MF) controls the compression ratio by adjusting the hyperparameter rank R.

Tensor-Train Decomposition (Oseledets, 2011) gives a method to represent the high-order tensor with a low-rank approximation. In Tensor-Train (TT) decomposition, each element in the original tensor can be represented as the product of two vectors and a series of matrices. For the weight matrix $W \in \mathbb{R}^{(\prod_{k=1}^n I_k)\times(\prod_{k=1}^n J_k)}$, where $I = \prod_{k=1}^n I_k$ and $J = \prod_{k=1}^n J_k$, it can be reshaped as a 2n-order tensor $\mathcal{X} \in \mathbb{R}^{I_1\times\cdots\times I_n\times J_1\times\cdots\times J_n}$. Then, the ele-

Method	Time Complexity	Space Complexity	Max Rank	Potential Risk
FC	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$	N	-
MF	$\mathcal{O}(NR)$	$\mathcal{O}(NR)$	R	low rank bottleneck
TT	$\mathcal{O}(DN^{1+\frac{1}{D}}R^2)$	$\mathcal{O}(DN^{\frac{2}{D}}R^2)$	N	quadratic computational complexity
HMF	$\mathcal{O}(\alpha N^2)$	$\mathcal{O}(\alpha N^2)$	$< N$	can't achieve full rank
HTT	$\mathcal{O}(\alpha N^2)$	$\mathcal{O}(\alpha N^2)$	N	

Table 1: The comparison of Time and Space Complexity between Different decomposition methods for a $N \times N$ Fully Connected layer. **Max Rank** denotes that the approximated matrix with these decomposition can theoretically achieve the maximum rank. α indicates the ratio of dense part, where $\alpha \in [0, 1]$. R denotes the rank of MF and The TT-ranks of TT. D indicates the TT-cores number of TT. HMF and HTT: we only retain Time and Space Complexity of the dense part, which their decomposition part is different.

ment in $\mathcal{T}$ is defined as follow:

$$\mathcal{T}(i_1, ..., i_n, j_1, ..., j_n) = T_1(:i_1, j_1, :)\, T_2(:i_2, j_2, :) \cdots T_n(:, i_n, j_n, :) \quad (1)$$

where $T_k \in \mathbb{R}^{R_{k-1} \times I_k \times J_k \times R_k}$ are referred to as TT-cores, which are smaller tensors as shown in Figure 2(C). The set $\{R_k\}_{k=0}^n$ is called TT-ranks, and $R_0 = R_n = 1$. Usually, $R_1, \ldots, R_{n-1}$ are equal, denoted as R. TT decomposition controls the compression ratio by adjusting TT-ranks.

3 Hybrid Tensor-Train Decomposition

In this section, we first discuss the limitation of MF and TT, as well as their expressivity comparison in the case with limited parameters. Then, we introduce *Hybrid Tensor-Train decomposition*, which can retain full matrix rank but with fewer operations and parameters.

3.1 Rethinking MF and TT

Low-rank bottleneck in MF The rank of MF is essential to its expressivity (Thakker et al., 2020). To obtain high compression ratio, MF needs extremely small rank, which will lead to Low-rank bottleneck. Technically, for a matrix $W \in \mathbb{R}^{I \times J}$, the compression ratio is $\frac{IJ}{(I+J)R}$. Furthermore, lower rank of MF has protential to lead to performance drop. For example, (Novikov et al., 2015) found that compressing a dense layer with a rank 50 to 1 MF layer, which show significant drop of performance. Noach and Goldberg (2020) also found a similar conclusion in compressing BERT (Devlin et al., 2019) with MF.

Quadratic computational complexity in TT Since the computational complexity of TT is $\mathcal{O}(DN^{1+\frac{1}{D}}R^2)$, TT decomposition has a *quadratic computational complexity* with respect to the TT-ranks R (Table 1). Fortunately, the quadratic computational complexity issue could be tolerated in the case TT has relatively small TT-ranks. In this paper, we avoid using TT with relatively high TT-ranks. For example, to compress a 512×512 dense layer, we used a TT-cores=3 and TT-ranks=2 TT layer, which can compress $512\times$ parameters and $8\times$ operations. As shown in Table 2, we don't directly apply original TT layers to the Transformer in this work, because the Low-rank TT Transformer suffers a performance degradation and the high-rank TT Transformer decelerates the inference speed due to quadratic computational complexity.

Expressivity between MF and TT In the case TT has relatively small TT-ranks, *TT is more expressive than MF with comparable amount of parameters.* For example, Novikov et al. (2015) found that compressing a dense layer with a TT-ranks = 1 TT layer outperforms a rank = 50 LMF layer and achieves more significant compression. This means that the rank of LMF can't be set too small for preserving performance. Hrinchuk et al. (2020) found that TT has *full matrix rank* while compressing a matrix, and doesn't reduce the expressivity.

3.2 Hybrid Tensor-Train Decomposition

TT decomposition has potential since any TT-parameterized matrix has full matrix rank (Hrinchuk et al., 2020). Considering that the *high-rank* TT decomposition is time-consuming, we aim to explore *low-rank* TT in this paper. However, the preliminary experimental results in Table 2 show purely making use of *low-rank* TT decomposition results in a large performance drop.

To compensate *low-rank* TT in terms of expressive power, we propose Hybrid Tensor Train decomposition (HTT) to mix TT decomposition with an auxiliary dense projection matrix [1]. The reasons

[1] A related work is Thakker et al. (2020) which proposes

Model	Params	Speed	BLEU
TT Transformer†	1.0 M	55.3 tokens/s	8.6
TT Transformer‡	5.8 M	37.6 tokens/s	23.2
MF Transformer	5.8 M	62.7 tokens/s	22.5
HTT Tranformer	5.7 M	50.5 tokens/s	22.5
Hypoformer	5.7 M	60.1 tokens/s	**23.6**

Table 2: The performance comparison among decomposed Transformers in TT, MF, HTT, Hypoformer format on WMT'14 En-De. Hypoformer obtains the best BLEU score compared with other decomposition methods. Speed denotes translation speed [tokens/s] on the Intel CPU. † indicates the low-rank (TT-ranks=2) TT Transformer. ‡ indicates the high-rank (TT-ranks=32) TT Transformer.

are twofold. First, HTT can relieve the quadratic computational issue since TT-rank is small. Secondly, HTT is expressive thanks to the compensation from a dense matrix. HTT consists of two parts: a dense part and a Tensor-Train part (Figure 2(E)). The formula is as follows:

$$\begin{aligned} &W = [W_{dense}, W_{tt}] \\ &\mathcal{W}_{tt}\left(i_1, ..., i_n, j_1, ..., j_n\right) \\ &= \mathcal{G}_1\left(1, i_1, j_1, :\right) \cdots \mathcal{G}_n\left(:, i_n, j_n, 1\right) \end{aligned} \tag{2}$$

where $W \in \mathbb{R}^{I \times J}$ can be regarded as the embedding layer or the weight matrix of Fully Connected layer. $W_{dense} \in \mathbb{R}^{I \times \alpha J}$, $W_{tt} \in \mathbb{R}^{I \times (1-\alpha) J}$, and $\alpha \in [0, 1]$ is used to control the proportion of the dense and TT part. W_{tt} can be derived from tensor $\mathcal{W}_{tt}$ by reshaping. TT cores $\mathcal{G}_k \in \mathbb{R}^{R_{k-1} \times I_k \times J_k \times R_k}$, n is the numbers of TT-cores. we use 2 or 3 TT-cores in this paper. $\prod_{k=1}^{n} I_k = I$, $\prod_{k=1}^{n} J_k = (1-\alpha)J$, and $k = 1, ..., n$.

The space/time complexity and expressive power (in terms of rank) of the above decompositions are shown in Table 1. Note that *HTT could conduct flexible decomposition* : There exisits a hyperparameter α in HTT to adjust the compression ratio of original matrices.

4 Hypoformer

In this section, we first show the overall structure between the Transformer and our Hypoformer in Figure 3. In this work, we don't apply HTT decomposition to FFN, because FFN is less important in Transformer and impact the model performance limited (Hsu et al., 2020; Yang et al., 2020). Based on HTT Transformer, we apply the LMF layer to replace the HTT layer in FFN, which brought a further improved speed (Table 2). Therefore, we decide to use MF layer in FFN. Then, we introduce the three *Hypoformer* sub-layers including HTT Embedding, HTT Self-Attention, and LMF FFN.

4.1 HTT Embedding

The embedding layer is a very large matrix, which can't be overlooked. For example, it accounts for about $1/4$ of the model parameters in WMT'14 En-De. previous works (Wu et al., 2019) used a joint source-target vocabulary with a large embedding $W \in \mathbb{R}^{v \times d}$. where v is the length of the joint source-target vocabulary, d is the embedding dimension.

Hrinchuk et al. (2020) showed that directly using Tensor-Train embedding to replace the original embedding layer would obviously damage the performance of the machine translation model. In order to address it, we propose a new word embedding method named Hybrid Tensor-Train Embedding (HTT EMB) that mixes the dense embedding and sparse embedding. Suppose the number of words in the vocabulary is v, and the dimension of word embedding is d. HTT EMB is obtained by concatenating a low-dimensional dense embedding matrix and a TT-matrix with 3 TT cores, which can be formulated as follows:

$$W_E = [W_e, W_{tt}] \tag{3}$$

where $W_e \in \mathbb{R}^{v \times \alpha d}$, $W_{tt} \in \mathbb{R}^{v \times (1-\alpha) d}$, and $\alpha \in [0, 1]$ is used to control the proportion of the two kind of embeddings.

4.2 HTT Self-attention

In the original Transformer, the self-attention allows the input X to apply three projections to obtain Query, Key and Value representations. The self-attention can be computed as:

$$\begin{aligned} A =& \frac{X W_q W_k^T X^T}{\sqrt{d}} \\ \text{Attention}(Q, K, V) =& \text{softmax}(A) X W_v \end{aligned} \tag{4}$$

where $X \in \mathbb{R}^{N \times d}$ is the input from previous layer, N is the sequence length, d is the model dimension. A is the attention matrix. The three weight matrix $W_q, W_k, W_v \in \mathbb{R}^{d \times d}$ can be column-wise concate-

a Hybrid Matrix Factorization (HMF). HMF decomposes a matrix into two parts — a 'narrow' matrix part and a MF part (Figure 2(D)). It is claimed that HMF can *obtain a higher rank than MF under the same parameters*

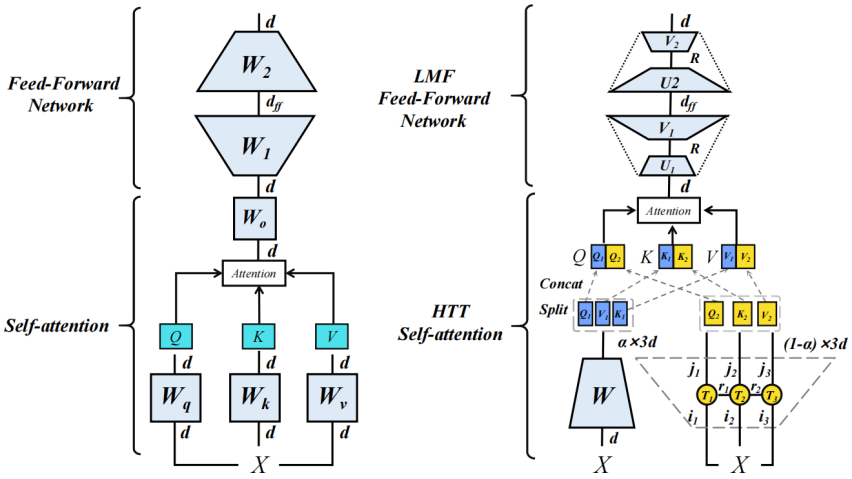

Figure 3: The structure comparison of the Transformer and The Hypoformer. We focus on the weight matrices of structure and overlook some operations like softmax etc.

nated as a single matrices $W \in \mathbb{R}^{d\times 3d}$, which can be formulated as:

$$W = [W_q, W_k, W_v] \tag{5}$$

Therefore, what we need to compress is the W matrix and it consists of only a HTT layer. HTT splits the W matrix into a dense part and a Tensor-Train part with 3 TT cores (Figure 2(E)). This procedure can be represented as follows:

$$W = [W_{dense}, W_{tt}] \tag{6}$$

where $W_{dense} \in \mathbb{R}^{d\times 3\beta d}$, $W_{tt} \in \mathbb{R}^{d\times 3(1-\beta)d}$, similar to HTT EMB, $\beta \in [0, 1]$ is used to control the proportion of the dense part and TT part.

In HTT SAN, the input X first goes through a dense layer, and the output is split into three equal parts Q_1, K_1, V_1. Second, the input X goes through a low-rank TT layer, and the output is also split into three equal parts Q_2, K_2, V_2. Finally, we obtain the Q, K and V by connecting related slices. The procedure can be formulated as:

$$\begin{aligned} Q_1, K_1, V_1 &= \text{Split}(XW_{\text{dense}}, 3) \\ Q_2, K_2, V_2 &= \text{Split}(XW_{\text{tt}}, 3) \\ Q &= \text{Concat}(Q_1, Q_2), \\ K &= \text{Concat}(K_1, K_2), \\ V &= \text{Concat}(V_1, V_2), \end{aligned} \tag{7}$$

4.3 LMF Feed-Forward Network

The function of the Feed-Forward Network (FFN) is to make the input X perform a non-linear transformation. It can be defined as:

$$\text{FFN}(X) = \text{ReLU}\left(XW_1 + b_1\right)W_2 + b_2 \tag{8}$$

where $W_1 \in \mathbb{R}^{d\times d_{ff}}$, $W_2 \in \mathbb{R}^{d_{ff}\times d}$, $b_1 \in \mathbb{R}^{d_{ff}}$ and $b_2 \in \mathbb{R}^d$. d is the model dimension, d_{ff} is the dimension of the FFN. To compress the FFN, we propose Low-rank Matrix Factorized FFN (LMF FFN). Unlike the two dense layers in the FFN, LMF FFN consists of four dense layers, which can de defined as follows:

$$\text{LMF-FFN}(X) = \text{ReLU}\left(XU_1V_1 + b_1\right)U_2V_2 + b_2 \tag{9}$$

where $U_1 \in \mathbb{R}^{d\times R}$, $V_1 \in \mathbb{R}^{R\times d_{ff}}$, $U_2 \in \mathbb{R}^{d_{ff}\times R}$, $V_2 \in \mathbb{R}^{R\times d}$, $b_1 \in \mathbb{R}^{d_{ff}}$ and $b_2 \in \mathbb{R}^d$. R is the rank of the matrix factorization.

5 Experiment

5.1 Datasets and Evaluation

We evaluate our methods on three standard translation benchmark datasets: IWSLT'14 De-En (De-En), WMT'14 En-De (En-De), and WMT'16 En-Ro (En-Ro). For De-En, we use the same setup as in Liu et al. (2020), which consists of 160K sentence pairs and 10K joint byte pair encoding (BPE) (Sennrich et al., 2016) vocabulary. For En-De , we follow the setup as in Liu et al. (2020), which includes 3.9M training sentence pairs for and 37K joint BPE vocabulary. For En-Ro ,we follow the setup of Lee et al. (2018), which includes 610K training sentence pairs. For evaluation, we use beam search decoding in three tasks, where there is beam 5 for De-En and beam 4 and length penalty 0.6 for En-De and En-Ro. The performance is measured by case-sensitive tokenized BLEU (Papineni et al., 2002) for all translation tasks.

Model	IWSLT'14 De-En					WMT'14 En-De				
	Params	**Ratio**	$S_{(Pi)}$	$S_{(Intel)}$	**BLEU**	**Params**	**Ratio**	$S_{(Pi)}$	$S_{(Intel)}$	**BLEU**
Transformer	36.8 M	1.0×	1.0×	1.0×	34.5	61.0 M	1.0×	1.0×	1.0×	27.3
Lite Transformer	13.9 M	2.7×	1.5×	1.5×	33.6	33.6 M	1.8×	0.8×	1.1×	26.5
HAT Transformer	35.2 M	1.1×	1.8×	1.7×	34.5	46.2 M	1.3×	1.5×	1.7×	26.9
DeLighT	19.9 M	1.9×	1.0×	0.8×	34.4	23.3 M	2.6×	1.3×	1.2×	26.7
Hypoformer 12-2	**8.4 M**	**4.4**×	3.5×	2.7×	**34.8**	**21.2 M**	**2.9**×	1.5×	1.6×	**27.5**
Hypoformer 12-1	8.6 M	4.3×	**3.9**×	**3.3**×	34.4	23.6 M	2.6×	**1.8**×	**2.8**×	27.4

Table 3: Comparison of compression ratio and speedup with Light-Weight Transformers on IWSLT'14 De-En and WMT'14 En-De. Best performance is **bolded**. **Params**: the whole model parameters including embedding layer. **Ratio**: dividing the parameters of Light-Weight Transformers by parameters of Transformer. $S_{(Pi)}$ and $S_{(Intel)}$: the inference speedup of Raspberry Pi-4 CPU and Intel CPU, speedup indicates the division between the inference speed of Light-Weight Transformers and Transformer. **Hypoformer X-X**: the X numbers of encoder and decoder Hypoformer with knowledge distillation. The results of Transformer (Vaswani et al., 2017) are from original paper.

Model	Params	Ratio	$S_{(Pi)}$	$S_{(Intel)}$	BLEU
Transformer‡	62.1 M	1.0×	1.0×	1.0×	**34.3**
DeLighT‡	22.0 M	2.8×	0.6×	1.2×	**34.3**
Subformer	20.0 M	3.1×	-	-	34.1
Hypoformer 12-2	13.9 M	4.5×	3.0×	1.9×	**34.3**
Hypoformer 12-2	**5.6 M**	**11.1**×	**3.7**×	**3.0**×	32.7

Table 4: Results on WMT'16 En-Ro. ‡ indicates the results is our implementation from Mehta et al. (2021)

5.2 Model Settings

Deep Encoder, Shallow Decoder The vanilla Transformer (Vaswani et al., 2017) adopts 6 encoder layers and 6 decoder layers. Besides the **6-6** setting, we choose a deep encoder shallow decoder setting that assigns 12 encoder layers and 2/1 decoder layers. Assigning more layers on encoders than decoders is beneficial for inference speed while maintaining its performance (Li et al., 2021c; Bérard et al., 2021) – this sometimes needs knowledge distillation (Kasai et al., 2020).

Knowledge Distillation Deep to shallow Transformer can benefit more from Sequence-level Knowledge Distillation (SKD) (Kim and Rush, 2016), which has been widely verified in previous work (Li et al., 2021b; Kasai et al., 2020). For the training stage, we use the method of SKD to generate training data. Firstly, we train a teacher model with ground truth data. Then we utilize the teacher model to predict the training data and obtain SKD data. Especially, we use ground truth and SKD data together in student training. In this work, we use 12-2 Transformer base model as our *teacher model* of knowledge distillation.

Our structure Our Hypoformer are based on a **12-2** or **12-1** Transformer base model(Vaswani et al., 2017) with the initialization of Admin (Liu et al., 2020). We use three of our proposed decomposition sub-layers (HTT EMD, HTT SAN and LMF FFN) to replace the original sub-layers (Embedding, The self-attention and FFN). We control the parameters of the model by adjusting the hyperparameters of the decomposition sub-layers and model dimensions. HTT EMB and HTT SAN have three hyperparameters: the dense part ratio α, the TT-cores D and TT-ranks R in TT parts. HMF FFN has one hyperparameter: rank R. The detailed model settings are given in Table 7.

5.3 Experimental Details

Baselines and Implementation We compare our Hypoformer with recent light-weight SOTA methods, including: Transformer (Vaswani et al., 2017), Transformer with the initialization of Admin (Liu et al., 2020), Lite Transformer (Wu et al., 2020), HAT Transformer (Wang et al., 2020), DeLighT (Mehta et al., 2021), Subformer (Reid et al., 2021). The implementation of all models uses Fairseq (Ott et al., 2019), including baselines and our methods. We reproduce the results of baselines following the setting from their papers or download the trained models from their official GitHub.

Speed Measures We do not use FLOPs as a speed metric because Wang et al. (2020) found that FLOPs does not reflect the measured latency in autoregressive Transformer. The inference speed metric we used is *tokens/s*, which means the number of tokens translated per second. We sample 50 sentences of an average sequence length (23

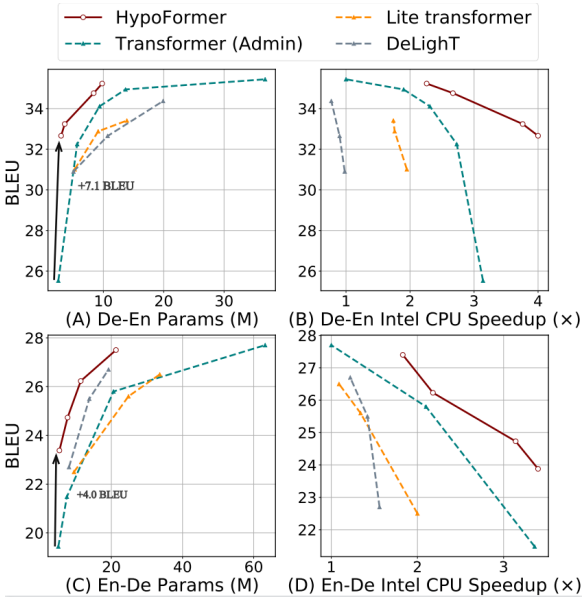

Figure 4: The comparison of performance between Hypoformer and SOTA light-weight methods on IWSLT'14 De-En **(Top)** and WMT'14 En-De **(Down)**. **Left**: under different parameters. **Right**: under different inference speedup of Intel CPU. Transformer (Admin): Transformer with the initialization of Admin (Liu et al., 2020). We scaled down Transformer (Admin) by adjusting the model dimension. We reproduced results of different settings in Lite Transformer and DeLighT's paper.

for IWSLT and 30 for WMT) to test speed. We run these samples 10 times and remove 10% the fastest and slowest results and average the rest 80% results. We test the speed on two representative devices: Raspberry Pi-4 with an ARM Cortex-A72 CPU, 1 core Intel Xeon E5-2678 v3 @ 2.50GHz CPU. We evaluate the inference speed with a batch size of 1 to simulate the inference of edge devices.

5.4 Experimental Results

Hypoformer smaller, faster and better performing than recent light-weight SOTA methods. In Table 3 and 4, we first compare the results between Hypoformer with previous works in case of compressing Transformer base model (Vaswani et al., 2017) on De-En, En-De and En-Ro tasks. Under the similar or even better performance on De-En and En-De, Hypoformer (12-2) compresses Transformer $2.9 \sim 4.4 \times$ parameters and Hypoformer (12-1) accelerates Transformer $2.5 \sim 2.8 \times$ on Intel CPU and $1.8 \sim 3.9\times$ on Raspberry Pi CPU, which also outperform recent light-weight the SOTA methods in compression, acceleration and performance.

Hypoformer consistently better than recent light-weight methods under different parameters, inference speed. Previous works evaluate their methods on different model scales on De-En and En-De tasks. To further verify the effectiveness of our method, we also compare Hypoformer to these model scales. As shown in Figure 4(A) and (C), Hypoformer outperforms Transformer (Admin), lite transformer and DeLighT with fewer parameters and better performance matching the similar parameter scale. For the inference speedup (Figure 4(B) and (D)), Hypoformer achieves a significantly higher speedup than lite transformer and DeLighT when scaled down the model size. Overall, Hypoformer can better meet the needs of real-time scenes in edge devices.

Hypoformer gets more competitive performance under Extremely low parameters. Notably, Hypoformer achieve 7.1 and 4.0 higher BLEU score than Transformer (Admin) when two models scale down to about 2.8M and 5.1M parameters on De-En and En-De (Figure 4(A) and (C)). These results indicate that Hypoformer's effectiveness is more significant in extremely low-resource scenarios.

5.5 Ablation Study

In Table 5, we show the ablation study of how does the vanilla Transformer evolve into Hypoformer from scheme **1**∼**5**, and evaluate the impact of Hypoformer's three components respectively. From scheme **1**∼**5**, we observe that Transformer with initialization of Admin and deep shallow structure bring an improvement of performance and $1.6\times$ speedup. We further apply our Hypofomer on this setting (scheme **3**), which leads to a performance degradation but can compress $3.8\times$ parameters and 21% speedup. However, the gap of performance between Hypoformer and Transformer can be made up with knowledge distillation (scheme **3**).

From scheme **6**∼**8**, we observe that only compressing embedding with HTT EMB can reduce embedding parameters and have no impact on speed because we restore the look-up table embedding in the processing of model inference. HTT SAN (Only compressing self-attention) and HMF FFN (Only compressing FFN) can both reduce about 31% model parameters and promote about 11% speedup. In the aspect of performance, three components combined do not bring a significant drop of performance and get higher compression and acceleration compared to adding three components respectively.

	Model	#Params	#Total	$S_{(Intel)}$	BLEU
1	Transformer	31.6 M	36.8 M	1.0 ×	34.5
2	+ Admin	31.6 M	36.8 M	1.0 ×	34.9
3	+ 12-2	31.6 M	36.8 M	1.9 ×	35.4
4	+ Hypoformer	7.2 M	9.8 M	2.3 ×	33.3
5	+ Hypoformer†	7.2 M	9.8 M	2.3 ×	35.2
6	+ HTT EMB †	31.6 M	34.2 M	2.0 ×	36.0
7	+ HTT SAN †	21.7 M	24.3 M	2.1 ×	35.5
8	+ LMF FFN †	22.2 M	24.8 M	2.1 ×	35.4

Table 5: Ablation study on IWSLT'14 De-En. #Params and #Total indicate the model parameters without and with embedding respectively, † denotes that the models apply knowledge distillation. For scheme **1**∼**5**, (+) indicates that a result includes all preceding methods. For scheme **6**∼**8**, (+) indicates that a proposed decomposition applies on scheme **2** with knowledge distillation.

Dataset	Model	E-D	Params	Ratio	$S_{(Intel)}$	BLEU
De-En	Transformer	6-6	36.8 M	1.0×	1.0×	34.9
	Transformer	12-2	36.8 M	1.0×	1.9×	**35.4**
	Hypoformer†	12-2	**9.8 M**	**3.8**×	**2.3**×	35.2
En-De	Transformer	6-6	63.2 M	1.0×	1.0×	27.7
	Transformer	12-2	65.3 M	1.0×	**1.5**×	**28.3**
	Hypoformer†	12-2	**21.2 M**	**3.0**×	**1.5**×	27.5
En-Ro	Transformer	6-6	61.4 M	1.0×	1.0×	**34.5**
	Transformer	12-2	64.2 M	1.0×	1.6×	34.4
	Hypoformer†	12-2	**13.9 M**	**4.4**×	**1.9**×	34.3

Table 6: The compression ratio and the speedup comparison with deep to shallow Transformer and Hypoformer on three tasks. All models initialize with Admin. **E-D**: the numbers of encoder (**E**) and decoder (**D**). † denotes the model apply knowledge distillation.

5.6 Analysis

Deep to Shallow Setting. Table 6 shows the results of Hypoformer and Transformer (Admin) on three tasks. First, we note that the deep to shallow Transformer (12-2) can get similar performance with symmetric Transformer (6-6) but speedup about 1.6 ×, which is similar to the conclusion of previous studies (Li et al., 2021c). *Hypoformer can further compress and accelerate Transformer (12-2) with matching similar performance.* Compared to Transformer (12-2) baseline, our method can compress $3.0 \sim 4.4\times$, accelerate $0 \sim 21\%$ speed in Intel CPU with a slight performance drop on three tasks. The speedup could be further improved if our HTT module has been optimized, which is also the focus of our future work.

Knowledge Distillation Analysis. *Distillation is important.* As shown in Figure 5, we compared the results of Hypoformer and the model without distillation. We found that knowledge distillation can improve our models by about 2 BLEU scores which is a significant improvement.

Regularization Analysis. *Low parameter models need less Regularization.* With the same model of 11.4 M parameters, Hypoformer achieve better BLEU scores (25.7→26.2) with using less dropout (0.1→0.05) and label-smoothing (0.1→0.05) on WMT'14 En-De.

6 Related Work

Low-rank Approximation is a common and effective technique to compress the deep neural networks. Matrix Factorization, which expresses a matrix $W \in \mathbb{R}^{I\times J}$ as a product of two smaller matrices $U \in \mathbb{R}^{I\times R}$ and $V \in \mathbb{R}^{R\times J}$. Hybrid Matrix Factorization (Thakker et al., 2020) is an extension of MF, which inspired our Hybrid Tensor Train decomposition. Tensor-Train decomposition is one part of our methods, which has proved its effectiveness in compression in previous works(Novikov et al., 2015; Yang et al., 2017; Hrinchuk et al., 2020; Liu et al., 2021).

Light-Weight Transformer is a focused topic recently. There is an increasing demand for the deployment of Transformer in resource-constrained scenarios. Recent works explore different methods to compress and accelerate Transformer, including light-weight variants (Wu et al., 2020; Mehta et al., 2021), Neural Architecture Search (Wang et al., 2020), Parameter sharing (Reid et al., 2021), Low-rank approximation (Ma et al., 2019; Liu et al., 2021; Hrinchuk et al., 2020) deep encoder shallow decoder structure (Hsu et al., 2020; Bérard et al., 2021; Li et al., 2021c).

7 Conclusion

In this paper, we presented *Hybrid Tensor-Train (HTT) Decomposition*, which can retain full matrix rank but with fewer operations and parameters. We explore the application of HTT decomposition to compress and accelerate Transformer for edge devices. In three standard translation tasks, Hypoformer consistently better than recent light-weight SOTA methods under various parameter and speed scales. In edge devices with extremely limited resource, Hypoformer can retain $\mathbf{95}\%$ performance with $\mathbf{1/12}$ parameters, setting a new SOTA in extremely-small NMT models.

References

Alexandre Bérard, Dain Lee, Stéphane Clinchant, Kweonwoo Jung, and Vassilina Nikoulina. 2021. Efficient inference for multilingual neural machine translation. *ArXiv*, abs/2109.06679.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Oleksii Hrinchuk, Valentin Khrulkov, Leyla Mirvakhabova, Elena Orlova, and Ivan V. Oseledets. 2020. Tensorized embedding layers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 4847–4860. Association for Computational Linguistics.

Yi-Te Hsu, Sarthak Garg, Yi-Hsiu Liao, and Ilya Chatsviorkin. 2020. Efficient inference for neural machine translation. *CoRR*, abs/2010.02416.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A. Smith. 2020. Deep encoder, shallow decoder: Reevaluating the speed-quality tradeoff in machine translation. *CoRR*, abs/2006.10369.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *EMNLP*.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1173–1182. Association for Computational Linguistics.

Bei Li, Ziyang Wang, Hui Liu, Quan Du, Tong Xiao, Chunliang Zhang, and Jingbo Zhu. 2021a. Learning light-weight translation models from deep transformer. In *AAAI*.

Bei Li, Ziyang Wang, Hui Liu, Quan Du, Tong Xiao, Chunliang Zhang, and Jingbo Zhu. 2021b. Learning light-weight translation models from deep transformer. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 13217–13225. AAAI Press.

Yanyang Li, Ye Lin, Tong Xiao, and Jingbo Zhu. 2021c. An efficient transformer decoder with compressed sub-layers. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 13315–13323. AAAI Press.

Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. 2020. Understanding the difficulty of training transformers. In *EMNLP*.

Peiyu Liu, Ze-Feng Gao, Wayne Xin Zhao, Zhi-Yuan Xie, Zhong-Yi Lu, and Ji-Rong Wen. 2021. Enabling lightweight fine-tuning for pre-trained language model compression based on matrix product operators. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 5388–5398. Association for Computational Linguistics.

Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Ming Zhou, and Dawei Song. 2019. A tensorized transformer for language modeling. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 2229–2239.

Sachin Mehta, Marjan Ghazvininejad, Srinivasan Iyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2021. Delight: Deep and light-weight transformer. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Matan Ben Noach and Yoav Goldberg. 2020. Compressing pre-trained language models by matrix decomposition. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing, AACL/IJCNLP 2020, Suzhou, China, December 4-7, 2020*, pages 884–889. Association for Computational Linguistics.

Alexander Novikov, D. Podoprikhin, A. Osokin, and D. Vetrov. 2015. Tensorizing neural networks. In *NIPS*.

I. Oseledets. 2011. Tensor-train decomposition. *SIAM J. Sci. Comput.*, 33:2295–2317.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. 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: Human Language Technologies, NAACL-HLT 2019,*

Minneapolis, MN, USA, June 2-7, 2019, Demonstrations, pages 48–53. Association for Computational Linguistics.

Yu Pan, Maolin Wang, and Zenglin Xu. 2021. Tednet: A pytorch toolkit for tensor decomposition networks. *CoRR*, abs/2104.05018.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*, pages 311–318. ACL.

Machel Reid, Edison Marrese-Taylor, and Yutaka Matsuo. 2021. Subformer: Exploring weight sharing for parameter efficiency in generative transformers. *CoRR*, abs/2101.00234.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

Urmish Thakker, Jesse G. Beu, Dibakar Gope, Ganesh S. Dasika, and Matthew Mattina. 2020. Rank and run-time aware compression of nlp applications. *ArXiv*, abs/2010.03193.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. 2017. Attention is all you need. *arXiv*.

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020. Hat: Hardware-aware transformers for efficient natural language processing. In *ACL*.

Felix Wu, Angela Fan, Alexei Baevski, Yann N. Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. 2020. Lite transformer with long-short range attention. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Yilin Yang, Longyue Wang, Shuming Shi, Prasad Tadepalli, Stefan Lee, and Zhaopeng Tu. 2020. On the sub-layer functionalities of transformer decoder. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 4799–4811. Association for Computational Linguistics.

Yinchong Yang, Denis Krompass, and Volker Tresp. 2017. Tensor-train recurrent neural networks for video classification. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 3891–3900. PMLR.

A Appendix for "Hypoformer: Hybrid Decomposition Transformer for Edge-friendly Neural Machine Translation"

Figure 5: Distillation analysis in different model size on IWSLT'14 De-En task.

A.1 Experiment: Training Setting

Our models follow the training setups of (Liu et al., 2020). We trained our models with 1 RTX 2080Ti GPU on IWSLT'14 De-En, 8 Telsa V100 GPUs on WMT'14 En-De and 4 For WMT'16 En-Ro. The last 10 model checkpoints are averaged for testing on En-de/En-Ro and the lowest perplexity checkpoint for testing on De-En. The detailed hyperparameters is listed in Table 8 and 9. Our TT decomposition code based on TedNet (Pan et al., 2021).

A.2 Related Work: Sequence-Level Knowledge Distillation

Sequence-Level Knowledge Distillation (Kim and Rush, 2016) is a branch of knowledge distillation for seq2seq model. The generated sequences from the teacher model can be treated as the sequence-level knowledge to guide the student network training, which can make up the performance of deep shallow transformers (Kasai et al., 2020; Li et al., 2021b).

The goal of Knowledge Distillation is to transfer the knowledge learned from a large teacher network

Tasks	E-D	d_{model}	d_{ff}	HTT EMB†			HTT SAN			LMF FFN	Speed(CPU)	Params	BLEU
				α	D_1	R_1	α	D_2	R_2	R			
De-En	12-2	512	1024	0.5	3	4	0.25	3	2	64	95 tokens/s	9.8 M	35.2
	12-1	512	1024	0.5	3	4	0.25	3	2	64	140 tokens/s	8.6 M	34.4
	12-2	512	1024	0.5	3	4	0.25	3	2	32	112 tokens/s	8.4 M	34.8
	12-2	256	1024	0.5	3	4	0.25	3	4	32	158 tokens/s	3.6 M	33.4
	12-2	256	1024	0.125	3	4	0.25	3	4	16	168 tokens/s	3.0 M	32.7
En-De	12-2	512	2048	0.25	3	16	0.5	2	4	128	42 tokens/s	21.3 M	27.5
	12-1	512	2048	0.25	3	16	0.5	2	4	256	49 tokens/s	23.6 M	27.4
	12-2	384	2048	0.25	3	16	0.5	2	3	96	59 tokens/s	11.4 M	26.2
	12-2	256	1024	0.25	3	16	0.5	2	3	96	90 tokens/s	7.7 M	24.7
	12-1	256	1024	0.25	3	16	0.5	2	3	96	96 tokens/s	7.1 M	23.9
	12-2	256	512	0.125	3	16	0.5	2	3	64	87 tokens/s	5.4 M	23.4
En-Ro	12-2	512	2048	0.125	3	32	0.25	3	4	96	48 tokens/s	13.9 M	34.3
	12-2	256	1024	0.25	3	16	0.25	3	4	64	76 tokens/s	5.6 M	32.7

Table 7: The overall results of Hypoformer in three tasks. **E-D** indicates the numbers of encoder (**E**) and decoder (**D**). d_{model} denotes the model dimensions, d_{ff} indicates the hidden state dimensions of FFN. † indicates that the TT-ranks of HTT EMB don't impact the inference speed because we restore the look-up table embedding in the processing of model inference.

Hyperparameter	**Value**
label smoothing	0.1
max tokens	4096
distributed world size	1
update frequency	1
dropout rate	[0.05, 0.1, 0.2]
embedding dim	[512, 256]
ffn dim	1024
Attn heads	4
optimizer	radam
Adam-betas	(0.9, 0.98)
lr	7e-4
lr scheduler	invert sqrt
warmup lr	1e-7
warmup updates	6000
max updates	20K
fp16	True
fp16 scale window	256
threshold loss scale	0.03125
initialization	Admin
beam	5
length penalty	1.0

Table 8: Training setting of De-En.

Hyperparameter	**Value**
label smoothing	[0.05, 0.1]
max tokens	4096
distributed world size	8
update frequency	1
dropout rate	[0, 0.05, 0.1]
embedding dim	[512, 384, 256]
ffn dim	[2048, 1024, 512]
Attn heads	8
optimizer	radam
Adam-betas	(0.9, 0.98)
lr	1e-3
lr scheduler	invert sqrt
warmup lr	1e-7
warmup updates	8000
max updates	60K
fp16	True
fp16 scale window	256
fp16 scale tolerance	0.25
threshold loss scale	0.03125
initialization	Admin
beam	4
length penalty	0.6

Table 9: Training setting of En-De and En-Ro.

to the small student network. The procedure of KD can formulated like:

$$\mathcal{L}_{\mathrm{KD}} = \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \mathcal{L}\left(f^T(x,y), f^S(x,y)\right) \quad (10)$$

where $\mathcal{L}(\cdot)$ is a loss function to evaluate the gap between the teacher predict $f^T(x,y)$ and the student predict $f^S(x,y)$. In machine translation, (x,y) is a pair of inputs of the source and target language. $(\mathcal{X}, \mathcal{Y})$ is the represents the training dataset.