Transformer-CTP: Current Token Prediction Using Cross-Attention of Queries with Current Position Information

Anonymous ACL submission

Abstract

Transformer and pre-trained language models have advanced various tasks in artificial intelligence. Typically, transformer decoders for language generation are trained using LM loss. The LM loss function predicts the next token from the outputs of previous tokens. LM loss is trained such that transformer decoders can generate autoregressively. In addition, transformers use self-attention to derive outputs, which generally assigns a high attention weight to self-tokens. Therefore, the transformer decoder may over-focus on the token $t-1$ when predicting token t because it predicts token t through self-attention up to token $t-1$. The proposed method prevents the transformer decoder from overfocusing on token $t-1$ when predicting token t. Instead of predicting token t using the output of token $t-1$, we use a new input to predict token t. We also add a CPT module to the transformer decoder, which prevents token $t-1$ from being used in the attention query by cross-attention using the new input. Moreover, we measured the performance of machine translation and document summarization to verify that the proposed methodology can mitigate overfocusing problem and improve the performance. In our experiments, the proposed methodology improved performance. Also, it can distribute the focused attention to a few specific tokens, including the self-token. The code for the experiment can be found on our GitHub[1].

1 Introduction

Transformer(Vaswani et al., 2017) has made progress in various tasks in artificial intelligence as well as natural language processing. Moreover, pre-trained transformer-based language models such as BERT (Devlin et al., 2019), GPT (Radford et al., 2019), and BART (Lewis et al., 2020) have achieved state-of-the-art (SOTA) performance on most natural language processing tasks. Among them, transformer decoder-only language models such as GPT (Radford et al., 2019), LAMMA (Touvron et al., 2023), PaLM (202), and transformer encoder–decoder pre-trained language models such as BART (Lewis et al., 2020) and T5 (Raffel et al., 2020) can perform the functions of generative AI. This is because the transformer decoder is designed to generate sequences. Therefore, for natural language sentence generation tasks, such as machine translation and text summarization, a model with a transformer decoder is generally selected. Pretrained language models with a transformer decoder-only model or a transformer encoder–decoder model exhibit SOTA performance in various generation tasks.

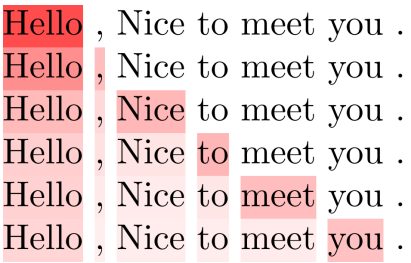


Figure 1: Visualizing the attention weights of the first layer of pre-trained GPT2.

Seq2Seq (Sutskever et al., 2014) is a mechanism comprising an encoder and a decoder. The encoder compresses the source sentence into a vector. The decoder autoregresssivly generates the sentence token-by-token through the compressed vector. The transformer has a Seq2Seq structure. Therefore, the transformer decoder also generates sentences autoregressively. The transformer is a model based on a self-attention mechanism, and the attention weight is computed by measuring the similarity between two tokens. The attention weight is used as the importance of each token in generating a correct answer. The self-attention mechanism focuses

[1] https://anonymous.4open.science/r/ARR2406_current_token_prediction

attention on the input sentence and same-sentence. The transformer decoder measures the attention score between the $t-1^{th}$ token and the sentence up to the $t-1^{th}$ token to predict the t^{th} token in the sentence. It then predicts the t^{th} token by using the weighted sum based on the attention score. This method can be used to generate sentences word-by-word while predicting the next word.

Self-attention generally has a higher attention score for the self-token(self-referenced token). This is expected because the query and key used to compute the attention score are extracted from the same token. Figure 1 shows an example of visualizing the attention score in the transformer–decoder model. For the example, we used gpt2-base (Radford et al., 2019) and extracted the attention score of the first self-attention layer for "Hello, Nice to meet you." Consequently, when generating the t^{th} token, the attention score for the $t-1^{th}$ token is often the highest. The fifth sentence in the figure predicts "you." In self-attention, the query is "meet" and the key is "Hello, Nice to meet". The highest attention score is observed for "meet" which is the same token as that for the query and key.

We hypothesize that over-focusing of self-tokens can lead to two sources of performance degradation. First, the input embedding of the $t-1^{th}$ token and output for predicting the t^{th} token may be limited by the amount of change. When deriving the results of the $t-1^{th}$ token, the attention score is often high for the $t-1^{th}$ token. Specifically, a weighted sum with a high weight on the $t-1^{th}$ token was used to derive the output of the $t-1^{th}$ token. This results in an output similar to the $t-1^{th}$ token. Because there is no guarantee that the $t-1^{th}$ token is similar to the t^{th} token, performance degradation may occur during the process of predicting the t^{th} token. Second, it may have an attention score biased toward self-token and later tokens in the sentence. This can lead to a high attention score for the self-token, even if earlier tokens are more important for predicting the next token. This is because there is a high probability that queries and keys extracted from the same vector are similar, which can cause performance degradation owing to uneven attention to the entire sentence. Therefore, we propose a way to mitigate the over-focus of self-tokens caused by the structure of the transformer decoder.

We propose a novel transformer-decoder structure that can predict the t^{th} token at the t^{th} position. Instead of predicting the t^{th} token using the output of the existing $t-1^{th}$ token, we use a new input to predict the t^{th} token. In the proposed methodology, the $t-1^{th}$ token is not directly used in the query when computing the attention score to predict the t^{th} token. This mitigates the overfocusing issues described above by distributing the biased attention score among self-tokens and later tokens. Moreover, we present text summarization and machine translation using the proposed transformer structure and verify that the proposed transformer structure leads to performance improvements.

2 Related Work

2.1 Transformer

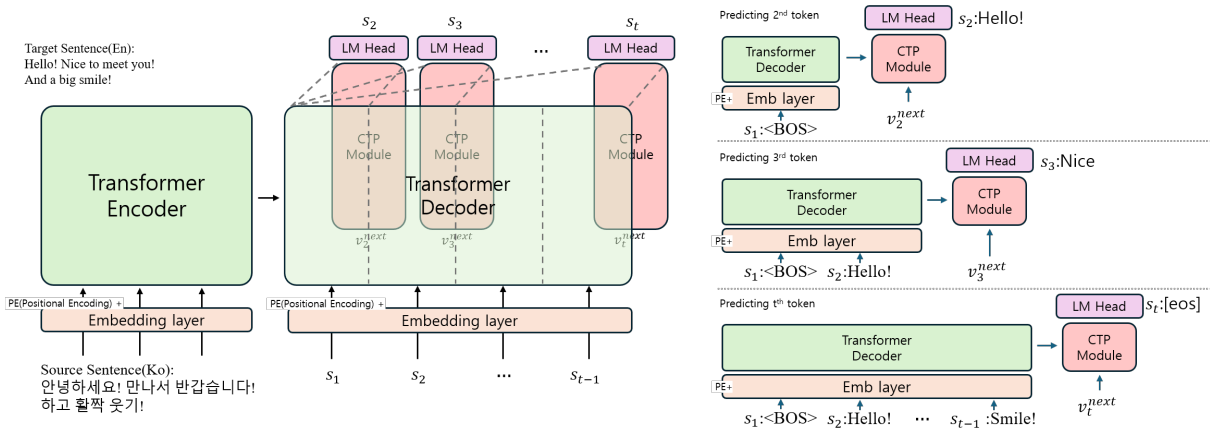
Transformer is a self-attention-based model. Dot-product attention (Luong et al., 2015) was proposed for machine translation. Dot-product attention is a mechanism for the Seq2Seq model to focus more on important tokens in the source sentence while predicting the target sentence. To select important tokens, Seq2Seq model obtained the similarity based on the dot product and used it as the attention score. The equation for the dot product attention is as follows:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

where, Q, K, and V denote the query, key, and value, respectively, and d_k denotes the dimensions of the key vector. Several transformer-based models use self-attention and cross attention with dot-product attention. Self-attention means when a query and key are extracted from the same sentence in the dot-product attention. The transformer encoder comprises a self-attention layer and feed-forward network layer. The transformer decoder comprises a masked self-attention layer, cross-attention layer and feed-forward network layer. Transformer performs residual connection and normalization in each layer. Masked self-attention causes each query to compute the attention only for the previous token. Therefore, the t^{th} token in a sentence and those after the tokens ($t+1 \sim$) are not subject to similarity computations in transformer decoder. This causes autoregressive generation during training.

2.2 LM loss in transformer decoder

Several studies of generative tasks, such as (Liu et al., 2023) and (Liu et al., 2022), use LM loss for training. The LM loss is a conditional probability-based loss function that predicts the t^{th} token given



(a) Overall process

(b) Example step-by-step generation process of decoder

Figure 2: Generation process of proposed method

the input sequence up to the $t-1^{th}$ token. The LM loss function is trained to predict the next token given the token sequence up to $t-1^{th}$ token. The trained model generates sentences token-by-token by predicting the next token and including the predicted token in the input for next time. The transformer decoder uses the output of s_{t-1} for the LM loss.

2.3 Sequence generation from mask token in transformer

In this study, we used a new input to predict the t^{th} token. The prediction of the next token from the output of a new input is similar to the prediction of a word from a mask token. Typically, masked language modeling is used to train transformer encoder language models, such as BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), and ALBERT (Lan et al., 2020). However, attempts have also been made to generate entire sentences from mask tokens.

In (Liang et al., 2023), sentence generation from mask tokens increased the generation speed of the language model. (Liang et al., 2023) generates sentences using the cross-attention of the transformer encoder and the mask token sequence. First, it predicts the length of the sentence to be generated and generates a input mask-token sequence. Replaces the mask token when the probability of the predicted word is higher than the threshold. In the next iteration, the mask token sequence is used with some assigned predictive words as the input. This process can generate sentences faster than autoregressive methods that require t iterations to predict t tokens.

In (Chen et al., 2021), proposed a transformer for speech recognition. The transformer decoder makes predictions using the mask tokens, using which proposed a non-autoregressive generation methodology. Similarly, to construct a non-autoregressive model, this method predicts the length of the correct speech. Subsequently, the mask token sequence of the predicted length is input to the decoder, which predicts the mask token in parallel.

3 Proposed Method

Our methodology does not predict s_t via the output of s_{t-1} from the target sequence $S = \{s_1, s_2, ..., s_t\}$ as in a transformer decoder. Our method uses a new input, the next input vector v_t^{next}, to predict s_t. v_t^{next} is the vector containing the t^{th} positional embedding PE_t. Therefore, it contains the t^{th} positional information. We also propose a current-token prediction (CTP) module for predicting the t^{th} token using v_t^{next}. The structure of the CTP module predicts the t^{th} token from the input of vector v_t^{next} containing the t^{th} position information. An example of this process is presented in Figure 3-(a). For clarity, the existing transformer decoder block is referred to as the original decoder block.

The process for predicting s_3, "Nice" in the Korean to English translation shown in Figure 3-(b) is as follows. First, the source sentence is entered into the transformer encoder. The output of the transformer encoder and s_1 and s_2 are input to the original decoder. Then, input v_3^{next}, the input for predicting s_3, is enter to the CTP module along with the output of the original decoder block. The

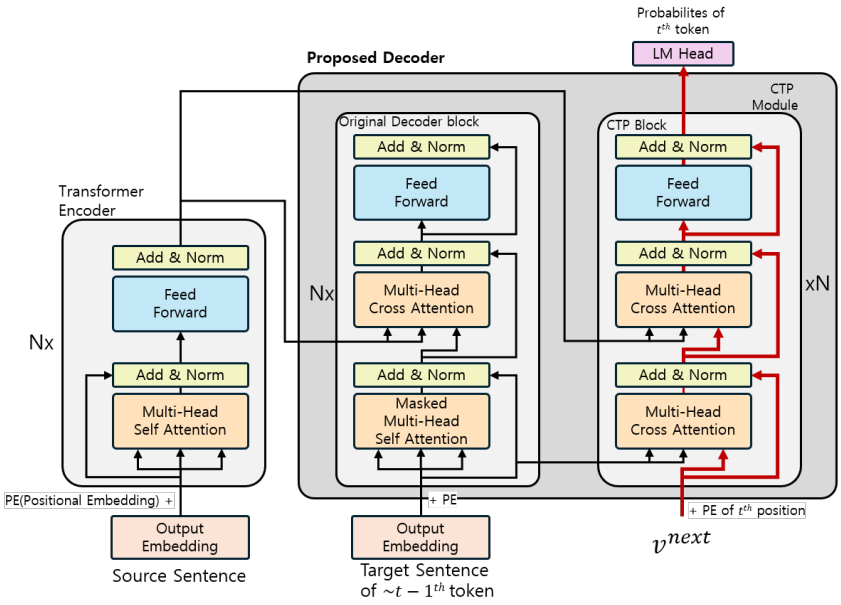


Figure 3: Overall structure of transformer-CTP

CTP module utilizes this information. The query of CPT module is extracted only from v_3^{next}. Finally, the output of the CTP module is enter into the LM head to predict s_3. The proposed methodology did not include s_2 in the query when generating the LM head input, and instead, uses v_3^{next} as a query to predict s_3. Therefore, Our method attention between v_t^{next} and $S_{<t} = \{s_1, s_2, ..., s_{t-1}\}$ instead of self-attention to s_{t-1} and $S_{<t}$ when predicting s_t. It can mitigates the excessively high attention score in s_{t-1} when predict s_{t-1}. We also expect that this will mitigate performance degradation assumed in this study.

The proposed methodology can be categorized into two main parts: the generation of the next input vector v_t^{next} and design and training of the transformer-CTP and CTP module. The details of each are described in Sections 3.1 and 3.2.

3.1 Next Input Vector v_t^{next}

The proposed method generates the next input vector v_t^{next}, which is used as the input for the CTP module when predicting s_t. The next input vector v_t^{next} contains the t^{th} positional embedding PE_t for predicting s_t at the t position. We generated v_t^{next} using two methods, both of which were used in our experiments. The first is to use the embedding of the special token <next>. Set $e_{<next>}$, the embedding of <next>, to v^{next}. Then add a t^{th}-position embedding to generate v_t^{next}. Next, we set the average of the token embeddings of the input sequence $S_{<t}$ to v^{next}. We set v^{next} as the average of the embedding vector $E_{<t} = \{e_1, e_2, ..., e_{t-1}\}$ of the input of original decoder. Similarly, we add the t^{th} position embedding to generate v_t^{next}. The equation for this process is as follows:

$$v^{next} = \{ e_{<next>} \frac{1}{t-1} \sum_{i=1}^{t-1} e_i \quad (2)$$

$$v_t^{next} = v^{next} + PE_t \quad (3)$$

e_i is the token embedding of s_i and $e_<next>$ is the embedding of the special token <next>. PE_t is the positional embedding of the t position.

3.2 CTP Module

In this study, we designed a CTP module to predict the t^{th} token using v_t^{next}. The structure of the transformer-CTP and the CTP module are shown in Figure 3. The structure of the CTP module, as in the transformer, comprises N CTP blocks. The CTP block comprises two multi-head cross-attention layers and one feed-forward layer. It also uses a residual connection and normalization after each layer. Before describing the equations for each layer, we organized the equations for the original transformer decoder.

$$H_{<t}^0 = E_{<t} \quad (4)$$

$$H_{<t}^{i+1} = DecoderBlock(H_{<t}^i, O) \quad (5)$$

The original transformer decoder returns H^N through N decoder blocks.

The first multi-head cross attention layer of the CTP block performs multi-head cross attention using the input vector of the original decoder block. In this process, the first CTP block extracts a query from v_t^{next}. And, the key and value are extracted from H^i, which is the input vector for each original decoder block. Therefore, this process can mitigate the excessively high attention score assigned to the s_{t-1}'s key. This implies that the output of the multi-head cross-attention of the CTP block is likely to be a vector that is not overly focused on s_{t-1}. In addition, the second CTP block does not receive vectors for query that are overly focused in s_{t-1} as inputs, which is expected to mitigate the overfocusing issue. Subsequently, residual connections and normalization were performed. The equation for first multi-head cross attention layer are follows:

$$C^0 = v_t^{next} \tag{6}$$

$$g^i = MHA(C^i W_i^{Q1}, H^i_{<t} W_i^{K1}, H^i_{<t} W_i^{V1}) \tag{7}$$

$$G^i = LN(g^i + C^i) \tag{8}$$

Where $MHA(\cdot)$ denote the multi-head attention. Multi-head attention was proposed in (Vaswani et al., 2017), where the attention in Equation 1 is performed by multiple heads. $W_i^{Q1}, W_i^{K1}, W_i^{V1}$ are the learnable parameters of the first multi-head cross attention layer of the i^{th} CTP block. $LN(\cdot)$ denotes layer normalization (Ba et al., 2016). In other words, the first multi-head cross attention layer of the i^{th} CTP block uses the input of the i^{th} original decoder block as the key and value.

The second multi-head cross-attention layer and feed-forward network layer of the CTP block is the same as the original transformer decoder. The second multi-head cross attention layer performs cross attention with the output of the transformer encoder. The output of the first multi-head cross-attention layer of the CTP block was used to extract the query. It also uses the output of the transformer encoder as key and value. The equations for this process are as follows:

$$m^i = MHA(G^i W_i^{Q2}, O W_i^{K2}, O W_i^{V2}) \tag{9}$$

$$M^i = LN(m^i + G^i) \tag{10}$$

$$C^{i+1} = LN(FFNN(M^i) + M^i) \tag{11}$$

where W_i^{Q2}, W_i^{K2}, and W_2^{V1} are the learnable parameters of the second multi-head cross-attention layer of the i^{th} CTP block; O is the output of the transformer encoder; and $FFNN(\cdot)$ is a feed forward network.

Equations 7–11 represent the CTP block cycles. The CTP module returns C^N through N CTP blocks and N original decoder block. Finally, Transformer-CTP predict s_t from the output of the CTP module and LM head. The transformer-CTP trains the LM loss using the cross-entropy loss of the prediction and gold. The training objectives of the proposed transformer-CTP are as follows:

$$L = -\sum_{t=1}^{T} log(s_t | v_t^{next}, s_1, ..., s_{t-1}, I) \tag{12}$$

The bold red line in Figure. 3 represents the flow of v_t^{next}. In other words, the red line flow of CTP module, where v_t^{next} is directly used as the input or output of each CTP block layer, is used as the input of next CTP block. The weights of the layers in this red line flow are not shared with the layers of the original decoder block. The weights of the other layers(black line in the CTP module) share weights with the layers of the original decoder block. In other words, $W_i^{K1}, W_i^{K2}, W_i^{V1}, W_i^{V2}$ of CTP module equation share weights with the original decoder block.

3.3 Parallel Processing on Training phase

Parallel processing of the transformer decoder during the training process is an important feature of the transformer. The transformer decoder can be trained non-autoregressively through masked multi-head self-attention during the training process. The proposed methodology also enables parallelism in the training process by masking multi-head cross-attention. For parallel processing, the first multi-head cross attention layer of the CTP block is constructed as a masked multi-head cross attention. The sequence of the next input vectors $V^{next} = \{v_2^{next}, v_3^{next}, ...v_T^{next}\}$. We use V^{next} as a query in the masked multi-head cross-attention. We used original decoder block of the target sentence $S_{<T}$ as the key and value. This allows non-autoregressive parallel training of the CTP module.

Model	WMT'14 En->De BLEU	WMT'14 De->EN BLEU	WMT'16 Ro->En BLEU	WMT'14 En->Fr BLEU	WMT'14 Fr->En BLEU
$BART_s$ (Our impl)	23.44	28.05	30.18	36.09	31.33
$BART_s$-CTP_{avg}	**23.73**	28.33	**30.48**	36.13	31.40
$BART_s$-$CTP_{<next>}$	23.59	**28.45**	30.43	**36.58**	**31.57**
$BART_b$ (Our impl)	24.41	**29.18**	30.71	36.93	32.32
$BART_b$-CTP_{avg}	24.40	29.10	**31.00**	**37.08**	**32.35**
$BART_b$-$CTP_{<next>}$	**24.60**	29.17	30.93	37.02	32.31

Table 1: BLEU of $BART$-CTP and $BART$ in machine translation. This experiment train from scratch without pre-trained model

Model	XSUM R-1	XSUM R-2	XSUM R-L
$BART_b$ (Our impl)	31.23	**11.33**	24.99
$BART_b$-CTP_{avg}	31.24	11.24	25.06
$BART_b$-$CTP_{<next>}$	**31.27**	11.24	**25.12**

Table 2: ROUGE of $BART$-CTP and $BART$ in document summarization. This experiment train from scratch without pre-trained model

4 Experiment

4.1 Dataset

To evaluate the performance of the proposed methodology, we evaluated the performance of machine translation and document summarization. For machine translation, we conducted experiments on three datasets: WMT'14 (Bojar et al., 2014) En-De, WMT'14 (Bojar et al., 2014) En-Fr, and WMT'16 (Bojar et al., 2016) Ro-En.

We also experimented with the XSUM (Narayan et al., 2018) dataset for document summarization. Detailed descriptions and statistics for each dataset are provided in the Appendix A.

4.2 Experiment Setting

In this paper, we propose a novel transformer structure called the transformer-CTP. We trained the downstream task from scratch with the transformer-CTP, which was not pre-trained. In addition, we did not perform pre-training due to the limitations of the research equipment. Instead, we evaluated the performance using a pre-trained weight. The training steps for the experiments was 1M update steps or 10 epochs, whichever was shorter. The performance of deep learning is affected by many factors such as model size, amount of training, hyperparameters, and generation settings. Therefore, to accurately compare the proposed methodology with the baseline, we conducted direct experiments on both under a fixed setting. For fair performance evaluation, we used detokenized sacreBLEU (Post, 2018) using mteval-v14.pl and ROUGE score.

Train From Scratch. We experimented similarly to the setup in Transformer paper (Vaswani et al., 2017). The transformer-based language model used in our experiments was BART (Lewis et al., 2020). BART has an approximately similar structure to the transformer, except that the ReLU activation function is replaced with GeLU, the parameter initialization is set to N(0,0.2) and using trainable positional embedding. We used a shared vocab size of 37000 for bilingual machine translation. In the document summary, we trained with a vocab size of 30000. We also experimented with BART-small, which has the same model size as that of the Transformer paper (Vaswani et al., 2017) and BART-base. The detailed experimental setup is described in Appendix B. Since the XSUM dataset has a long input document length due to the nature of document summarization, we only experiment with the BART-base that can take up to 1024 sequence lengths.

Using Pre-trained Weight. We experimented with the weight of an already pre-trained language model; specifically, we used the weights of the pre-trained BART-base. When using a pre-trained language model, a large number of new layers can cause performance degradation such as adding CPT module to pre-trained model. Therefore, we share more weights in this experimental setup. In section 3.2, only $W_i^{K1}, W_i^{K2}, W_i^{V1}, W_i^{V2}$ shared weights with the original decoder layer. However, in this experimental setup, we share the entire weight of the CPT module. In other words, the transformer-CTP loaded all learnable parameters

Model	WMT'14 En->De BLEU	WMT'14 De->EN BLEU	WMT'16 Ro->En BLEU	WMT'14 En->Fr BLEU	WMT'14 Fr->En BLEU
Transformers (Vaswani et al., 2017)	27.3	–	–	39.2	–
GLAT (Qian et al., 2021)	25.21	29.84	32.00	–	–
CMLMC (Huang et al., 2022)	26.40	30.92	34.13	–	–
JANUS-NAR (Liang et al., 2022)	26.40	30.90	34.36	–	–
$BART_b$ (Our impl)	27.05	30.23	**34.89**	38.87	33.36
$BART_b$-CTP_{avg}	**27.47**	**31.01**	34.20	38.93	**33.68**
$BART_b$-$CTP_{<next>}$	26.84	30.72	34.11	**39.43**	33.63

Table 3: BLEU of $BART_b$-CTP and $BART_b$ in machine translation. This experiment using pre-trained weight

Model	XSUM R-1	XSUM R-2	XSUM R-L
$BART_b$ (Our impl)	**41.01**	**19.03**	**33.63**
$BART_b$-CTP_{avg}	40.91	18.59	33.16
$BART_b$-$CTP_{<next>}$	40.85	18.57	33.18

Table 4: ROUGE of $BART_b$-CTP and $BART_b$ in document summarization. This experiment using pre-trained weight

from the pre-trained language model except for the next token <next> embedding.

4.3 Experiment Result

The results of the proposed method trained from scratch are presented in Table 1 and Table 2. CTP_{avg} denotes an experiment with the average of the previous token embeddings as v^{next}. $CTP_{<next>}$ is the experiment with the embedding of special token <next> as v^{next}. Most of the experimental results show that using the CTP module improves performance. However, in XSUM, the performance improvement was small, and ROGUE-2 exhibited a performance decrease. We expect that converting the decoder to the proposed methodology will have a relatively small impact on the performance change because of the nature of document summaries, where the number of words input to the transformer encoder is relatively large.

The experimental results obtained using the pre-trained weight are shown in Table 3 and Table 4. The proposed methodology improved the performance of WMT'14 En-De and En-Fr. Both methodologies showed performance enhancements. In many case, the experiment with CTP_{avg} performs better. This is different from the "train from scratch" experiment. We predict that the addition of the untrained <next> token embedding is the cause of the lower performance. However, the performance degradation for WMT'16 Ro->En and XSUM. A common characteristic of these two datasets was their small size. We observed overfitting on WMT'16 and XSUM while performing experiments using the proposed methodology. In most experiments, the training loss decreased faster than that at the baseline, but the validation score decreased after a certain time in WMT'16 and XSUM. However, in WMT'14, there was no overfitting and good performance. Therefore, in WMT'16 and XSUM, we used the weight decay to 0.1 of the Adam optimizer (Kingma and Ba, 2015) to mitigate overfitting; however, this was not sufficient. This issue is further discussed in Section 4.4.

4.4 Train Loss and Overfitting

While experimenting with the proposed methodology, we observed that it had a lower training loss than the baseline in most cases. We also found that overfitting tended to occur more easily. The overfitted model had a lower performance despite the lower training loss. In general, overfitting is easier with higher model complexity and smaller data size. Using $BART_b - CTP$ on small datasets such as XSUM and WMT'16 En-Ro fulfills this requirement. Therefore, overfitting was frequently observed in XSUM and WMT'16 En-Ro during the experiments. The initial settings of the XSUM and WMT'16 En-Ro experiments were run with a weight decay of 0.01. In this case, did not overfit $BART_b$, but frequently overfitted $BART_b - CTP$. Therefore, we converted the weight decay to 0.1 in XSUM and WMT'16 En-Ro, but observed weak overfitting in some cases.

However, this does not mean that the model exhibits a low performance. Typically, the datasets used for pre-training are hyperlarge-scale; there-

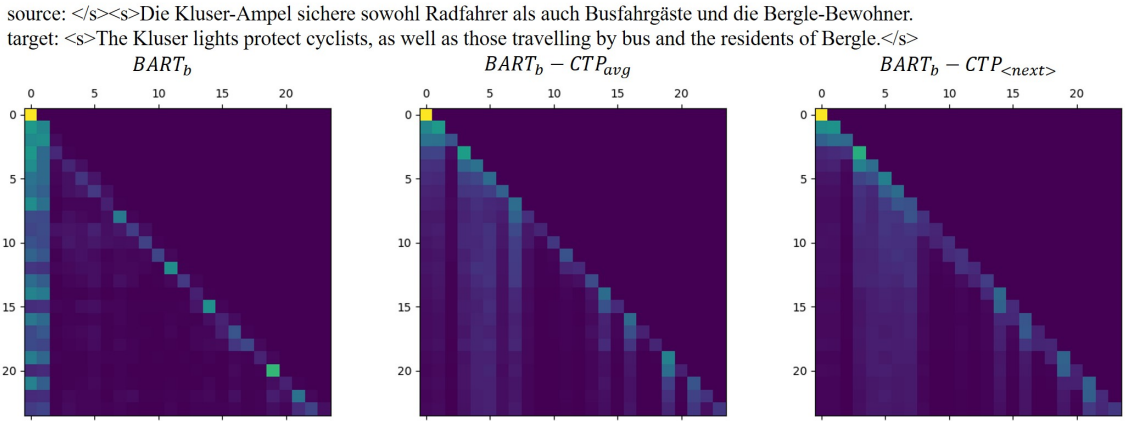


Figure 4: Visualizing the attention weights of the first layer of $BART_b - CTP$ train from scratch.

fore, a low training loss can be seen as a positive factor for training. According to Table 7, WMT'14 is very large compared with the other datasets. So, We did not observe any tendency of overfitting on WMT'14. The proposed methodology showed better performance on WMT'14. Therefore, we expect that the proposed methodology will be beneficial for very large datasets. An example and explanation of the low training loss are provided in Appendix D.

5 Analysis

We propose a Transformer-CTP for mitigating the excessive attention score measured on s_{t-1} while predicting s_t. Figure 4 shows the attention weights in the first decoder layer for $BART_b$ and first CTP module layer for $BART_b$–CTP. As can be seen, the proposed method has a more distributed attention score. In $BART_b$, we can observe that high attention scores are clustered into a few tokens at the beginning and a few at the end. However, both the proposed method have relatively distributed attention scores. Additionally, $BART_b$-$CTP_{<next>}$ further mitigates the excessive attention scores in the early tokens. This confirms that the proposed methodology can mitigate an excessive attention score at s_{t-1}, which is an issue in this study.

6 Ablation Study

We evaluated the performance with and without positional embedding. We compared the performance without including the t^{th} position to construct v_t^{next} as shown in Equation 6. The experimental results are listed in Table 5. The results showed that the performance without positional embedding was lower than that of the baseline in most cases. We

Model	WMT'14 En->De BLEU	WMT'14 Ro->En BLEU
$BART_s$	23.44	30.18
$BART_s$-CTP_{avg}	23.73	30.48
-Positional Embedding	23.33	30.07
$BART_s$-$CTP_{<next>}$	23.59	30.43
-Positional Embedding	23.33	30.02

Table 5: BLEU score according to positional embedding

can expected that $CTP_{<next>}$ without positional embedding might be problematic, because it has the same value at all positions. CTP_{avg} has a different value at each position, because the average number of target tokens is different at each position. However, CTP_{avg} without positional embedding also exhibited a lower performance than the baseline.

7 Conclusion

In this study, we mitigated the problem of overfocusing on the last input token to predict the next token in the transformer decoder. To mitigate the overfocusing problem, we used the next input vector with the next position information as the input. We also designed a CTP module and transformer-CTP to predict the next token from the next input vector. The proposed method showed better performance when trained from scratch without pre-training. We also experimented using pre-trained weights, and the performance improved on some datasets. In addition, the transformer-CPT uses well-distributed attention weights. We expect that the proposed method will help to build language models that can better focus on important tokens.

Model	#Prameter	Train time
$BART_s$	64M	x1.00
$BART_s$-CTP_{avg}	83M	x1.38
$BART_s$-$CTP_{<next>}$	83M	x1.38
$BART_b$	129M	x1.00
$BART_b$-CTP_{avg}	171M	x1.42
$BART_b$-$CTP_{<next>}$	171M	x1.43

Table 6: Model size and train time in our experiment of WMT'14 En-De

Limitation

Transformer-CPT uses larger model sizes than the transformer and requires more computation. The Transformer-CPT has the same size as the embedding, encoder, and LM head of the transformer. However, it is approximately 1.6 times of number of parameter in the transformer decoder ,excluding the weight shared with the original decoder. The increase in computation time was more noticeable in the non-autoregressive training phase. In the inference phase, where tokens are generated token by token, the total number of queries increases from n to n+1. However, in the non-autoregressive training phase, the total number of queries increases from n to 2n. Table 6 lists the training times for the experiment on the same machine.

In our experiments, we observed that the proposed methodology tended to overfit. In our experiments, there were cases of poor performance on some datasets, despite the lower training loss. We believe that a training loss lower than the baseline is advantageous for large datasets, such as in pre-training. However, there are concerns regarding overfitting in some small datasets. Future research is required on the application and development of methods to address overfitting.

Unfortunately, we were unable to perform any pre-training because of the limitations of our experimental resources. The transformer-CPT returns the output of the original decoder block and the output of the CPT module for the next token v_t^{next}. However, in this study, we only trained the model using the output of the next token v_t^{next}, because we did not perform pre-training. In addition, we did not perform error backpropagation to the last original decoder block because the output of the last original decoder block was not used as an input to the CPT module. We expect that pre-training research that uses or improves our proposed methodology will exhibit better performance and address our limitations.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *Preprint*, arXiv:1607.06450.

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA. Association for Computational Linguistics.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurélie Névéol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, pages 131–198, Berlin, Germany. Association for Computational Linguistics.

Nanxin Chen, Shinji Watanabe, Jesús Villalba, Piotr Żelasko, and Najim Dehak. 2021. Non-autoregressive transformer for speech recognition. *IEEE Signal Processing Letters*, 28:121–125.

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, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Xiao Shi Huang, Felipe Perez, and Maksims Volkovs. 2022. Improving non-autoregressive translation models without distillation. In *International Conference on Learning Representations*.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. 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*, pages 7871–7880, Online. Association for Computational Linguistics.

Xiaobo Liang, Zecheng Tang, Juntao Li, and Min Zhang. 2023. Open-ended long text generation via masked language modeling. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 223–241, Toronto, Canada. Association for Computational Linguistics.

Xiaobo Liang, Lijun Wu, Juntao Li, and Min Zhang. 2022. JANUS: Joint autoregressive and non-autoregressive training with auxiliary loss for sequence generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8050–8060, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Emmy Liu, Aditi Chaudhary, and Graham Neubig. 2023. Crossing the threshold: Idiomatic machine translation through retrieval augmentation and loss weighting. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15095–15111, Singapore. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Yixin Liu, Pengfei Liu, Dragomir Radev, and Graham Neubig. 2022. BRIO: Bringing order to abstractive summarization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2890–2903, Dublin, Ireland. Association for Computational Linguistics.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal. Association for Computational Linguistics.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. 2018. 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*, pages 1797–1807, Brussels, Belgium. Association for Computational Linguistics.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

Lihua Qian, Hao Zhou, Yu Bao, Mingxuan Wang, Lin Qiu, Weinan Zhang, Yong Yu, and Lei Li. 2021. Glancing transformer for non-autoregressive neural machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1993–2003, Online. Association for Computational Linguistics.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Huggingface's transformers: State-of-the-art natural language processing. *Preprint*, arXiv:1910.03771.

Dataset	# Data			Avg. # Word	
	Train	Validation	Test	Source	Target
WMT'14 En-De	4.5M	3k	3k	23.0	21.3
WMT'14 En-Fr	40.8M	3k	3k	25.0	28.6
WMT'16 En-Ro	610k	2k	2k	23.1	23.4
XSUM	204k	11k	11k	373.9	21.1

Table 7: Dataset statistics information

Setting	$BART_s$-CTP	$BART_B$-CTP
Vocab size	WMT: 37000 XSUM: 30000	WMT: 37000 XSUM: 30000
Layers	6	6
Attention Heads	8	12
Attention Heads size	64	64
Hidden size	512	768
FFNN inner hidden size	3072	2048
Maximum sequence length	512	1024
Dropout	0.1	0.1

Table 8: Model setting in our experiment.

Hyper-parameter	WMT'14 En-De, En-Fr	WMT'16 En-Ro, XSUM
Learning Rate	5e-5	5e-5
Batch size	16	16
Epochs/Steps	1M steps	10 epoch
Optimizer	AdamW	AdamW
Weight Decay	0.01	0.1
AdamW ϵ	1e-6	1e-6
AdamW β_1, β_2	0.9,0.999	0.9,0.999
Scheduler	linear	linear
Beam size	5	5

Table 9: Hyperparameter setting in our experiment.

A Dataset Information

We experimented with machine translation and document summarization using WMT'14, WMT'16, and XSUM. The WMT dataset is a machine-translation dataset comprising various sources, such as news commentaries and parliamentary minutes. XSUM is a document summarization dataset containing short summaries of BBC online articles. WMT'14 En-De and En-Fr are large datasets; therefore, we trained them with 1 M steps. WMT'16 and XSUM were small; therefore, we trained on 10 epochs, which is approximately 350 k steps. Detailed statistics for each dataset are listed in Table 7.

B Detailed Experiment Setting

We experimented with a similar setup to the transformer (Vaswani et al., 2017): in the model without prior learning, we used a shared vocab size of 37000 in the machine translation task. For document summarization, we used a vocab size of 30000 since it is monolingual. We used the same settings for the CTP block and original decoder block. The same settings as those of the transformer (Vaswani et al., 2017) were used in the small model. We used the same settings as in BART-base. Table 8 lists the model settings. When using pre-trained weights, we used the same settings for the pre-trained BART-base, including vocab size.

All the experiments were conducted once with the same random seed, and experiments on various datasets and settings demonstrated the effectiveness of the proposed methodology. We validate its effectiveness by experimenting with different datasets and models. All experiments were performed us-

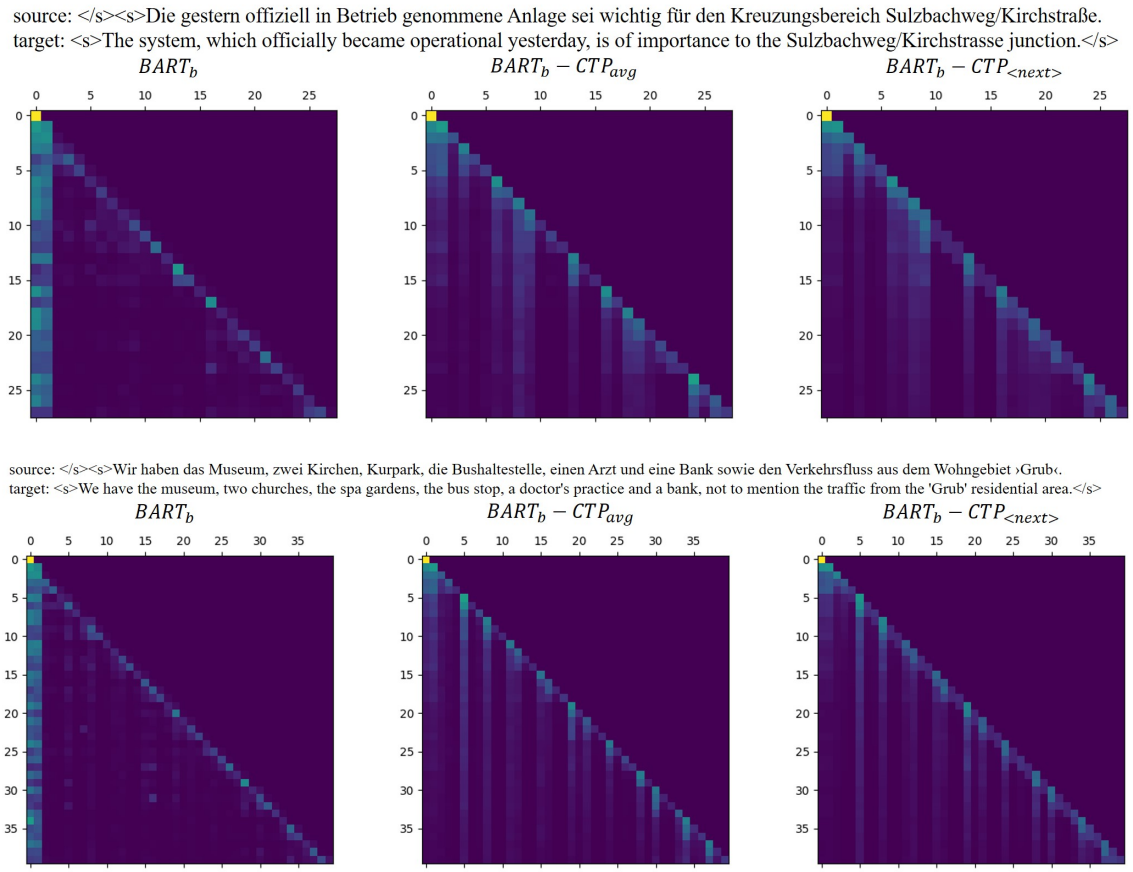


Figure 5: Attention weight visualization of first layer of $BART_b$-CTP without pre-trained model.

ing fixed hyperparameters. And we experimented with a single RTX 3090 or RTX4090. The comparison groups were tested using the same machine. The values of the hyperparameters used in the experiments are listed in Table 9. For performance evaluation, we used scareBLEU and ROUGE from huggingface packages (Wolf et al., 2020).

C Attention Weight Visualization

We visualized the attention weights on additional data to verify that the proposed methodology mitigated the overfocusing problem. The results are presented in Figure 5. Similar results were observed for other data. In BART, the first two tokens and the last token have biased attention weights. In the proposed methodology, the attention weight biased toward the first two tokens is reduced, and is further reduced as the number of tokens increases. In addition, spreads out the attention score concentrated on a few specific tokens, including the self token.

D Train Loss

While experimenting with the proposed methodology, we found that it had a low training loss in most experiments. Figure 6 shows the training loss graph. As shown in the figure, the initial training loss tended to be higher for the proposed methodology; however, the proposed methodology gradually exhibited a lower training loss. This was observed in most experiments, even in cases with similar or lower performances, such as XSUM. We predicted that the proposed methodology would train faster, but may be more easily overfitted. Further research on pretraining and studies with overfitting prevention methodologies are required to confirm this.

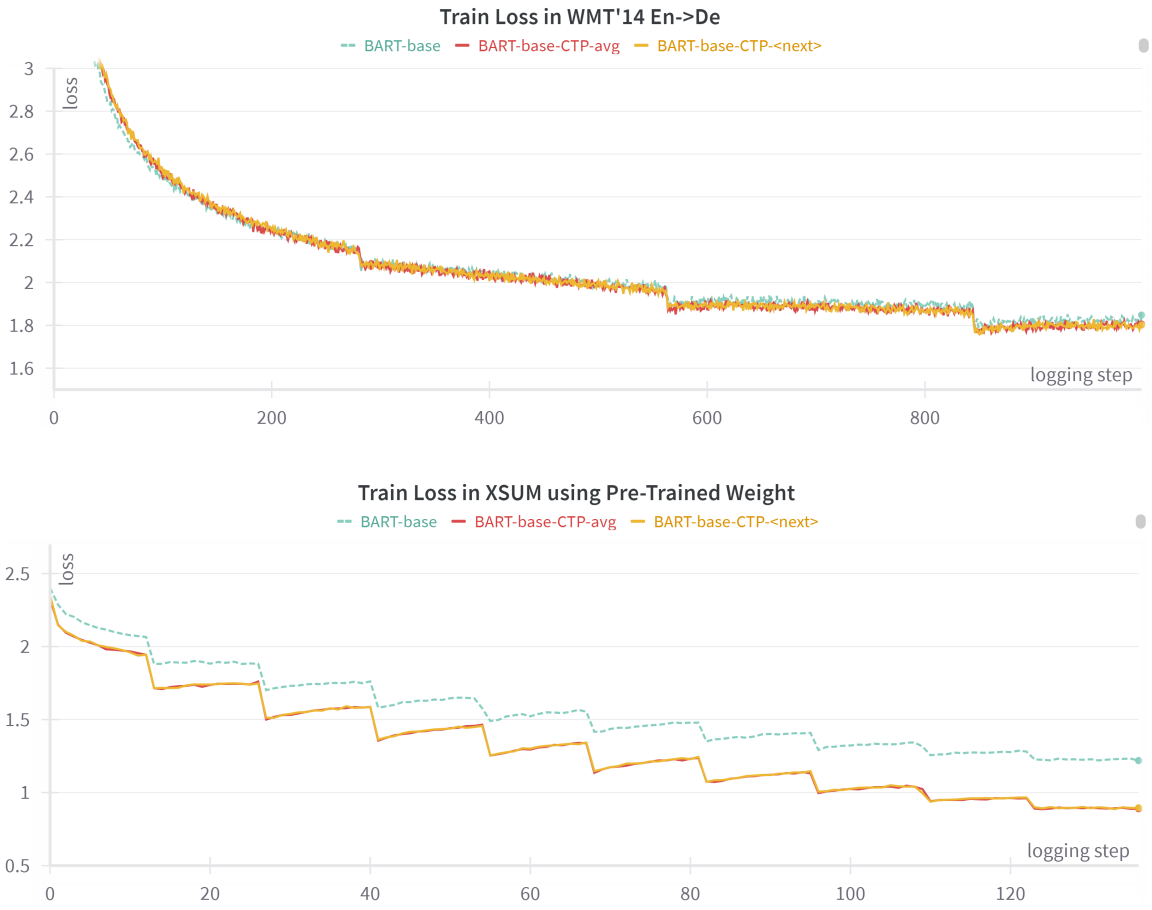


Figure 6: Train loss graph in XSUM and WMT'14 En-De dataset. we logging average loss of every 1000 steps