CHAPTER «ENGINEERING SCIENCES»

POSITIONAL ENCODING FOR TRANSFORMERS

Kateryna Antipova[1]
Hlib Horban[2]

DOI: https://doi.org/10.30525/978-9934-26-436-8-1

Abstract. The attention mechanism is a powerful and effective method utilized in natural language processing. This mechanism allows the model to focus on important parts of the input sequence. Transformer model utilizes attention mechanisms to replace recurrent and convolutional neural networks, which eliminates the need for increasingly complex operations as the distance between words in a sequence increases. However, this method is notably insensitive to positional information. Positional encoding is crucial for Transformer-like models that heavily rely on the attention mechanism. To make the models position-aware, the position information of the input words is typically incorporated to the input token embeddings as an additional embedding. *The purpose of the paper* is to conduct a systematic study to understand different position encoding methods. We briefly describe the components of the attention mechanism, its role in the Transformer model, and the encoder-decoder architecture of the Transformer. We also study how sharing position encodings across various heads and layers of a Transformer affects the model performance. *Methodology of the study* is based on general research methods of analysis and synthesis, experimental testing, and quantitative analysis to comprehensively examine and compare the efficacy and performance of different positional encoding techniques utilized in Transformer models. *The obtained results* show that using absolute and relative encodings results in similar performance

[1] Doctor of Philosophy,
Senior Lecturer at the Department of Software Engineering,
Petro Mohyla Black Sea National University, Ukraine
[2] Candidate of Technical Sciences, Associate Professor,
Associate Professor at the Department of Software Engineering
Petro Mohyla Black Sea National University, Ukraine

© Kateryna Antipova, Hlib Horban

for the model, while relative encodings worked much better with longer sentences. We found the original encoder-decoder form worked best for the tasks of machine translation and question answering. Despite using twice as many parameters as "encoder-only" or "decoder-only" architectures, an encoder-decoder model has a similar computational cost. Besides that, the number of learnable parameters can often be reduced without performance loss. *Practical implications.* Positional encoding is essential for enabling Transformer models to effectively process data by preserving sequence order, handling variable-length sequences, and improving generalization. Its inclusion significantly contributes to the success of Transformer-based architectures in various natural language processing tasks. *Value/originality.* Positional encoding is such a critical issue for Transformer-like models. However, it has not been explored how positional encoding establishes positional dependencies within a sequence. We chose to analyze several approaches to position encoding in the context of question answering and machine translation tasks because the influence of positional encoding on NLP models in terms of word order remains ambiguous and requires further exploration.

1. Introduction

The Transformer architecture has been essential for some of the biggest breakthroughs in deep learning in recent years. Especially in the field of Natural Language Processing (NLP), pre-trained autoencoding models (like BERT) and autoregressive models (like GPT-3) have continuously managed to outperform the state-of-the-art and reach human-like levels of text generation. One significant advancement introduced by the Transformer model is the use of attention layers as its main way for directing information flow.

Transformers have two major components: self-attention and a position-wise feed forward layer. The attention mechanism is a powerful and effective method utilized in NLP. However, it has been observed that this method suffers from a notable disadvantage, i.e. its permutation invariance, which makes the attention mechanism insensitive to positional information. While numerous studies have sought to enhance positional encoding and explore the effects of changing the word order, it remains unclear how positional encoding impacts NLP models in terms of word order. Notably, the word order is particularly relevant for natural language generating tasks, such as

machine translation. This significance is imposed by metrics used to evaluate generated results, such as BLEU, which are sensitive to word order.

Typically, the models are made position-aware by integrating the position information of the input words as an additional embedding to the input token embeddings. These position embeddings only depend on the location the word appears. There have been multiple works exploring different ways to include position information in Transformers, such as learnable fixed-length positional encoding, sinusoidal positional encoding and relative positional encoding. Typically, these methods assign positional encodings to words at different positions through vector addition in multiple indexing mechanisms.

In this work we undertake a systematic study to understand different position encoding methods. We briefly describe the components of the attention mechanism, its role in the Transformer model, and the encoder-decoder architecture of the Transformer. We also compare different position encodings and examine the effect of sharing position encodings across different heads and layers within a Transformer. Positional encoding is essential for enabling Transformer models to effectively process data by preserving sequence order, handling variable-length sequences, and improving generalization. Its inclusion significantly contributes to the success of Transformer-based architectures in various natural language processing tasks. However, it has not been explored how positional encoding establishes positional dependencies within a sequence. We chose to analyze several approaches to position encoding in the context of question answering and machine translation tasks because it remains unclear how positional encoding impacts NLP models from the perspective of word order.

Methodology of the study is based on general research methods of analysis and synthesis, experimental testing, and quantitative analysis to comprehensively examine and compare the efficacy and performance of different positional encoding techniques utilized in Transformer models.

2. Attention mechanism

The attention mechanism was first introduced within the encoder-decoder framework [1]. Attention mechanism allows the model to focus on important parts of the input sequence. The mechanism itself was introduced to solve the bottleneck problem that occurs when a fixed-length encoding vector is used, so the decoder would have limited access to the input

information. This limitation is particularly concerning for long or complex sequences, as it forces the dimensionality of their representation to be the same as for shorter or simpler sequences.

Essentially, when the generalized attention mechanism is given a sequence of words, it takes the query vector associated with a particular word in the sequence and compares it with each key in the database. In doing so, it discerns the relationship between the word being examined and others in the sequence. Then it adjusts the values based on the attention weights (computed from the scores) to concentrate on those words relevant to the query. This results in the generation of an attention output for the word in question.

The general attention mechanism makes use of three main components, namely the queries, Q, the keys, K, and the values, V. The query is analogous to the previous decoder output, s_{t-1}, while the values are analogous to the encoded inputs, h_i [1].

The general attention mechanism computes the following:

1. Each query vector, $q = s_{t-1}$, is compared to a database of keys to calculate a score value. The comparison is based on calculation of the dot product of the specific query with each key vector, k_i:

$$e_{q,k_i} = q \cdot k_i \tag{1}$$

2. A softmax operation is applied to the scores to generate the weights:

$$\alpha_{q,k_i} = softmax\left(e_{q,k_i}\right) \tag{2}$$

3. The generalized attention is subsequently calculated by taking a weighted sum of the value vectors, v_{k_i}, where each value vector is associated with a corresponding key:

$$attention\left(q,K,V\right) = \sum_i \alpha_{q,k_i} v_{k_i} \tag{3}$$

In the context of machine translation, each word of an input sequence is assigned its own query, key, and value vectors. These vectors are derived by multiplying the encoder's representation of the word under consideration with three different weight matrices generated during training.

The Transformer attention uses the following main components:

– q and k represent vectors of dimension, d_k, containing the queries and keys, respectively;

– v represents a vector of dimension, d_v, containing the values;

– Q, K, and V represent matrices packing together sets of queries, keys, and values, respectively;

– W^Q, W^K and W^V represent projection matrices that are used in generating different subspace representations of the query, key, and value matrices;

– W^O represents a projection matrix for the multi-head output.

Essentially, the attention function serves as a mapping of a query and a set of key-value pairs to produce an output.

The Transformer implements a scaled dot-product attention, following the procedure of the general attention mechanism. As the name implies, the scaled dot-product attention first computes a dot product for each query, q, with all of the keys, k. Then it divides each result by $\sqrt{d_k}$ and applies a softmax function. This process yields the weights that are used to scale the values, v.

In practice, the scaled dot-product attention can be calculated simultaneously for the entire set of queries. The matrices Q, K, and V are used as inputs for the attention function, according to the following:

$$attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4}$$

The introduction of a scaling factor $1/\sqrt{d_k}$ aims to counteract the effect of having the dot products grow large in magnitude for large values of d_k, where the application of the softmax function would then yield extremely small gradients that would lead to the vanishing gradients problem. By reducing the dot product results, the scaling factor can prevent this problem.

Although, the inclusion of the scaling factor doesn't affect the computational complexity, as that is dominated by the softmax calculation. This formula has an issue: multiplying Q and K leads to an $n*n$ matrix. Taking the row-wise softmax of an $n*n$ matrix has a complexity of $O(n^2)$, posing challenges in terms of runtime and memory usage as n can be very large. For large sequences, it quickly becomes impractical to compute self-attention over the full input, which means it is necessary to truncate or chunk the inputs.

3. Encoder and decoder

The Transformer model does not employ recurrent neural networks (RNNs), long short-term memory (LSTM) units, or convolutional neural networks (CNNs). Instead, it utilizes attention mechanisms to replace recurrence, which eliminates the need for increasingly complex operations as the distance between words in a sequence increases. The attention mechanism operates on a "word-to-word" basis, determining how each word relates to every other word in the sequence, including the word being analyzed itself.

Transformer relies solely on the use of self-attention, where the representation of a sequence (or sentence) is generated by relating different words in the same sequence. Self-attention is a specific type of attention. Compared to regular attention, self-attention focuses on a single sequence instead of relating an input to an output sequence. This way the model lets a sequence learn information about itself.

The original Transformer architecture consists of a stack of 6 layers. Each layer's output serves as the input to the subsequent layer until the final prediction is produced. The architecture includes a 6-layer encoder stack on the left and a corresponding 6-layer decoder stack on the right [15, p. 29].

On the left (see Figure 1), inputs are fed into the encoder of the Transformer through an attention sub-layer and a FeedForward Network (FFN) sub-layer. On the right, target outputs are processed through two attention sub-layers and an FFN sub-layer within the decoder.

All layers of the Transformer model retain the same structure as the original encoder layer. Each layer contains a sub-layer of multi-head attention mechanism and a sub-layer of a fully connected position-wise FFN. Each main sub-layer in the model is surrounded by a residual connection that transports the unprocessed input x of each sublayer to a layer normalization function. This way, key information such as positional encoding is not lost on the way.

Despite the identical structure of each layer in the encoder, the content of each layer differs from the previous layer. For instance, the embedding sub-layer is only present at the bottom level of the stack, and the subsequent layers do not contain an embedding layer. This ensures that the encoded input is stable through all the layers. Similarly, the multi-head attention mechanisms perform the same functions in all layers, yet they do not

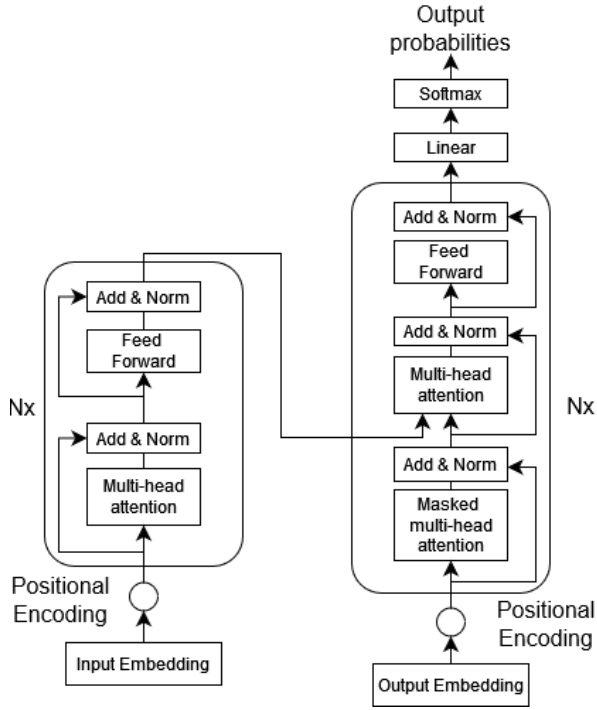

Figure 1. The Transformer architecture

perform the same tasks. Each layer learns from the previous one, exploring different ways of associating the tokens within the sequence.

Comprising eight heads, the multi-head attention sub-layer is followed by postlayer normalization, which adds residual connections to the sublayer's output and normalizes it. The input of the multi-attention sub-layer of the first layer of the encoder stack is a vector that contains the embedding and the positional encoding of each word.

Inside each head h_n of the attention mechanism, each word vector has three representations:

– A query vector that has a dimension of $d_q = 64$, which is activated and trained when a word vector x_n seeks all of the key-value pairs of the other word vectors, including itself in self-attention.

– A key vector that has a dimension of $d_k = 64$, which will be trained to provide an attention value.

– A value vector that has a dimension of $d_v = 64$, which will be trained to provide another attention value.

Each attention sub-layer and each feedforward sub-layer of the Transformer is followed by post-layer normalization (Add & Norm), which includes an add function and a normalization process itself. The add function handles the residual connections that originate from the sublayer's input. The goal of these connections is to ensure that the critical information is not lost during processing.

The input of the FFN is the $d_{model} = 512$ output of the Add & Norm of the previous sublayer. The FFN comprises two layers and applies a ReLU activation function. The output of the FFN is fed to the Add & Norm. Then the output is sent to the next layer within the encoder stack and the multi-head attention layer within the decoder stack.

The decoder layer maintains the same structure as the encoder for all the layers of the Transformer model. Each layer consists of the three sub-layers: a multi-head masked attention mechanism, a multi-head attention mechanism, and a fully connected position-wise feedforward network. The structure of each sub-layer and function of the decoder is similar to the encoder.

The decoder has a third main sub-layer, which is the masked multi-head attention mechanism. In this sub-layer output certain words are masked at a given position. Masking ensures that the Transformer bases its predictions solely on its inferences without accessing the rest of the sequence. That way, in this model, the Transformer cannot see future segments of the sequence.

The output of every sub-layer of the model has a constant dimension, d_{model}, including the embedding layer and the residual connections. This dimension can be set to another value. For the original Transformer architecture, d_{model} is set to 512.

In the original Transformer model, the input embedding sub-layer transforms the input tokens into vectors using learned embeddings. The Transformer's subsequent layers have learned word embeddings that already provide information on how the words can be associated. However, a lot of information is missing because no additional vector or information indicates a word's position in a sequence.

Generating independent positional vectors would significantly impede the training speed of the Transformer and make attention sub-layers very complex to work with. The concept entails incorporating a positional encoding value into the input embedding to describe the position of a token in a sequence, instead of having additional vectors.

4. Positional encoding

Positional representation is utilized as an inductive bias of positional relevance information by positional encoding function in the Transformer model. There have been many attempts to incorporate position information into the Transformer model.

Typically, embedding-level positional encodings build positional dependencies via adding positional encodings to embedding results [5; 7; 18]. In addition, attention-level positional encodings usually capture positional information based on an analysis of attention mechanisms and leverage delicate indexing manners, such as the relative distance to the querying word [8; 14; 17; 21]. However, it is unknown why the attention mechanism of the Transformer is position agnostic and what is the working principle behind these positional encodings.

Positional encodings are typically added to the embeddings of a sequence right after the embedding layers. They are usually a set of predefined or trainable vectors, indexed with absolute position numbers. After that, the contextual representation o_i can be presented as following:

$$o_i = \alpha\left(x_i + b_i, X + B\right)\left(X + B\right)W_{V,} \tag{5}$$

where $b_i \in R^d$ denotes the positional embedding for the *i*-th word, $B \in R^{L\times d}$ represents the packed positional embeddings for the whole sequence [22].

Positional encodings at the attention level are based on a key insight into the attention mechanism. Since they are employed in attention layers, they can access both the query and the key. This way positional information can be encoded in a more complex manner, such as using the relative distance between the query and the key for indexing. Considering a relative positional encoding as an example, the contextual representation oi can be calculated as [17]:

$$o_i = \sum_{j=1}^{L} \alpha_{ij}\left(v_j + r_{ij}^{v}\right) \tag{6}$$

$$\alpha_{ij} = \alpha\left(q_i, k_j + r_{ij}^k\right), \tag{7}$$

where $r_{ij}^k, r_{ij}^v \in R^{d_k}$ are trainable positional encodings for the *j*-th key and *j*-th value respectively. These encodings are indexed based on the distance between x_i and x_j [22].

Sine and cosine functions are used to generate different frequencies for the positional encoding (PE) for each position and each dimension *i* of the $d_{model} = 512$ of the word embedding vector [18; 19]:

$$PE_{(pos\ 2i)} = sin\frac{pos}{10000^{\frac{2i}{d_{model}}}} \tag{8}$$

$$PE_{(pos\ 2i+1)} = cos\frac{pos}{10000^{\frac{2i}{d_{model}}}} \tag{9}$$

Figure 2 shows the heatmap of sinusoidal positional encoding method.

The fixed position embeddings are subsequently combined with the word embeddings of the input sequence accordingly. Although the sinusoidal version with predefined wavelength has unique extrapolability which allows it to encode longer sequential positions than pre-training samples, it does not always perform well on downstream tasks, due to the lack of learnability and flexibility [17].

Absolute position encoding includes computing a positional encoding for each token and adding it to the input content embedding to incorporate positional information into the original sequence. This method was introduced in [18] for Transformers and it has been commonly utilized in the follow-up works. There are two variations of the absolute position encodings: fixed and learned.

Given an input sentence: $X = \{x_1, x_2, \ldots, x_n\} \in R^{n \times d}$, where *n* is the number of words and *d* is the dimension of word embeddings, the attention computes the output of the *i*-th token as:

$$\alpha_{ij} = \frac{(x_i + p_i)W^Q\left(\left(x_j + p_j\right)W^K\right)^T}{\sqrt{d}} \tag{10}$$

Here, $p_i \in R^d$ is a position embedding for the *i*-th token, obtained by fixed or learned encodings.

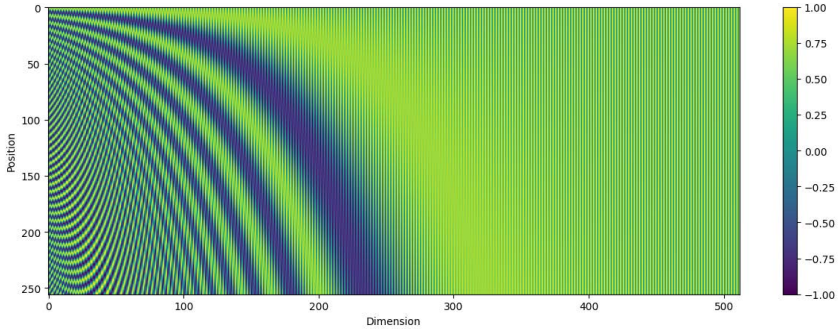

Figure 2. Sinusoidal positional encoding with the maximum sequence length of 256. Each position row represents the embedding vector as positional representation

Many successful variants of pre-trained Transformer models, such as BERT [5] and RoBERTa [12], incorporate the entire embedding matrix as trainable parameters [7]. To ensure a fixed number of training parameters, the maximum length of a sequence, denoted as L_{max}, must be predefined before training. Despite lacking the inductive property, this data-driven approach has proven effective for various NLP tasks. Unlike the fixed sinusoidal position encoding, a learnable position embedding matrix is not injected at each block for Transformer due to a large number of additional parameters.

Figure 3 shows the heatmap of learnable absolute positional encoding method.

Unit order across different languages is quite different. English uses a subject-verb-object ordering, that is rather fixed, but all possible orderings have been argued to occur in other languages. So, the question is whether it is useful to share position information across languages. Multiple studies observe mixed results with language specific position embeddings in the context of transferring monolingual models to multiple languages: for most languages it helps, but for some it seems harmful. In multilingual models position embeddings are shared by default [5].

However, one limitation of absolute position encoding is that it requires a fixed length of input sequence and does not directly account for the relative

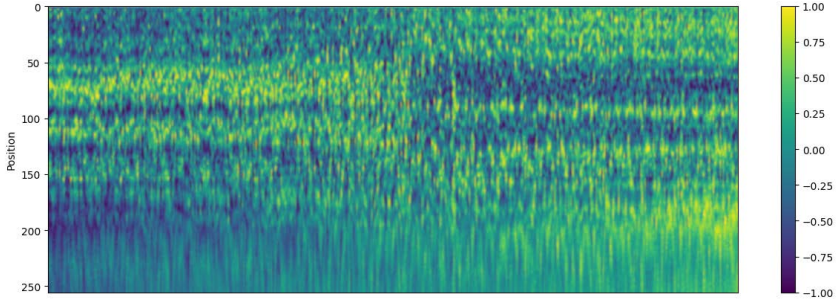

Figure 3. Learnable absolute positional encoding with the maximum sequence length of 256

positions between words [2]. Several approaches were proposed to enhance positional representation by adding relative position information into the attention score computation stage to improve performance of Transformer based models [4; 6; 8; 17].

Due to the fixed context length, the model is unable to capture any longer-term dependency longer than the predefined context length. Furthermore, the fixed-length segments are formed by selecting a consecutive sequence of symbols without considering the sentence or any other semantic boundary. Hence, the model lacks necessary contextual information needed to well predict the first few symbols, leading to inefficient optimization and inferior performance.

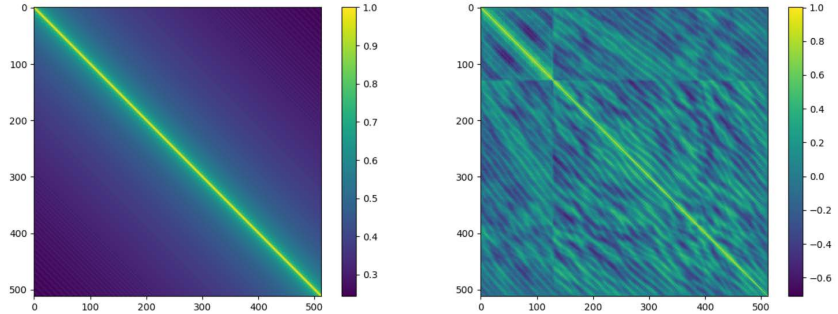

Figure 4. Similarity of sinusoidal and learnable positional encodings

The authors of [17] suggested using relative position encoding in lieu of absolute position encoding, and incorporating position embeddings to the key and optionally value projections, rather than the input. Their findings indicate that this alternative method of encoding position information improves performance on machine translation tasks. The authors of [23] further simplified this approach by removing the position embeddings in value projections, resulting in enhanced performance in language modeling tasks. Both approaches use a vector representation to encode positional information.

Relative positional encoding produces a vector $r_{i,j}$ or a scalar value $\beta_{i,j}$ that depends on the relative distance of tokens. These methods specifically apply such a vector or bias to the attention head, enabling the adjustment of the corresponding attentional weight according to the relative distance between two tokens:

$$\alpha_{ij} = \frac{x_i W^Q \left(x_j W^K + r_{i,j}^K \right)^T}{\sqrt{d}} \tag{11}$$

$$\alpha_{ij} = \frac{(x_i W^Q)\left(x_j W^K \right)^T + \beta_{i,j}}{\sqrt{d}} \tag{12}$$

Here, the first mode uses a vector $r_{i,j}$ while the second uses a scalar value $\beta_{i,j}$, for infusing relative distance into attentional weight. Relative position representation reduces the number of parameters to $(2K+1)d$ by dropping the interactions between tokens with a distance greater than K.

Let's consider the runtime storage complexity of embedding methods for a transformer model with *m* layers, *h* attention heads per layer, and maximum sequence length of *n*. For each position in the sequence, a positional encoding vector is required. Therefore, the storage complexity of positional encoding is $O(nd)$. Each token in the sequence is embedded into a vector. If the vocabulary size is *V* and the embedding dimension is *d*, then the storage complexity of token embeddings is $O(Vd)$.

Absolute positional encoding introduces embedding parameters with size of *nd*, relative positional encoding − with size of $mh(2n-1)d$. All position embedding methods introduce a small number of additional parameters to a model. Therefore, relative positional encoding introduces $mh(2n-1)d, 12 \cdot 12 \cdot (2 \cdot 512 - 1) = 147K$ parameters at maximum. Comparing

this to the number of parameters in large models like BERT (e.g., 108M parameters), it's indeed negligible. This illustrates that the overhead introduced by relative positional encoding is relatively small compared to the overall size of the model.

The cross correlation between position and token embeddings can result in weaker performance of additive absolute position embeddings, so instead it was proposed to add both absolute and relative positional information-based attention directly in each head [9]. So, the Transformer only uses the word embedding as input. In the self-attention module, different types of correlations are separately computed to reflect different aspects of information, including word contextual correlation and absolute (and relative) positional correlation. Each kind of correlation has its own parameters and is added together to generate the attention distribution. This design successfully eliminates the randomness in word-to-position or position-to-word correlations and provides greater expressiveness in characterizing the relationship between a pair of words or positions.

The relative positional encoding denotes the position of a word within a sentence, either through a vector representation [4; 17] or a scalar representation [14]. In the scalar approach, the embedding vectors of relative positions and the word embedding have the same dimension. Scalars are used to encode relative position between query and key indices by directly incorporating them into the attention scores matrix. These position embeddings are streamlined, with each "embedding" essentially being a scalar added to the corresponding logit for computing attention weights. For efficiency, they also share the position embedding parameters across all layers in our model, though within a given layer each attention head uses a different learned position embedding. Typically, only a fixed number of embeddings is learned, with each corresponding to a range of possible offsets between keys and queries.

The approach used in the T5 Transformer consists of bucketing function and bucket embedding [14]. The bucketing function categorizes the relative positions into different buckets using a fixed heuristic algorithm, and the bucket embedding component maps each bucket to a learnable scalar value. This "scalar embedding" is then added to the corresponding context score to derive the final attention score, determining the self-attention weights. In contrast, Adaptive-T5 adopts a learnable bucketing

function that automatically adapts to the dependency range specific to the learning task [21].

In addition to just the input layer, some authors suggest that the injection of position information to every layer leads to even better performance for the Transformer. Since it is nontrivial to modify or replace backbone of model structure during the fine-tuning stage, some research works propose auxiliary tasks or data augmentation approaches to leverage absolute or relative position information without modifying model structure [11; 13; 20].

An ideal position encoding approach should satisfy the following three properties [11]:

1. Inductive. It should have the capability to handle sequences longer than any encountered during training.

2. Data-driven. The position encoding should be trainable from the data, allowing it to adapt to different tasks and datasets.

3. Parameter efficient. The number of trainable parameters introduced by the encoding should be limited to prevent excessive model size, which could hinder generalization.

Table 1

Comparing position representation methods

Methods	Inductive	Data-driven	Parameter efficient
Sinusoidal PE [18]	+	-	+
Absolute PE [5]	-	+	-
Relative PE [17]	-	+	+
Combined absolute and relative PE [9]	-	+	+
Relative scalar approach [14]	+	+	-
Adaptive-T5 [21]	+	+	-

Transformer language models (LMs) with sinusoidal position embeddings exhibit very weak extrapolation capabilities. The position embedding method causes this failure to extrapolate. Recent alternatives to the original sinusoidal position method have improved extrapolation. However, the better of these, relative scalar approach and Adaptive-T5, are considerably slower than the sinusoidal approach and use extra memory and parameters.

Although some approaches lack the inductive property, many of them are still effective for many NLP tasks, due to the fact that the large sequence length can be enforced at inference.

To construct position embeddings in a more data-driven way, many Transformer variants include these embeddings as learnable model parameters in the training stage. This data-driven approach has a drawback of imposing a fixed maximum length of input sequences, along with the computational and memory overhead of additional parameters. A relative position representation helps to reduce the number of parameters. Also, in addition to just the input layer, the injection of position information to every layer leads to even better performance for the Transformer [3].

5. Experiments

In this section, we present our experimental results comparing different position encoding approaches. We consider training Transformer models from scratch for question answering and machine translation. We compare the following positional encoding approaches:

– w/o PE: without any positional encoding, to provide another reference of positional encoding ability;

– SPE: sinusoidal positional encoding [18];

– LAPE: learnable absolute positional embedding, to evaluate the effect of trainable positional encoding [5];

– RPE: relative positional embedding [17];

– CARPE: combined absolute and relative positional encoding [9];

– RSA: relative scalar approach [14];

– AT5: adaptive version of T5's relative positional encoding [21].

For the question answering task, we pre-train the model on Ukrainian Web Corpus (ukTenTen22). The Ukrainian Web Corpus is a Ukrainian corpus made up of texts collected from the Internet. The corpus belongs to the TenTen corpus family which is a set of web corpora built using the same method with a target size 10+ billion words. Data for the Ukrainian Web 2022 corpus consists of texts from May 2014, July–August 2020, and October–December 2023. The Wikipedia part is from December 2020 and 2022. The final size of the corpus contains 9.5+ billion words.

We set a sequence length of 758, a batch size of 128 sequences. We use language-independent tokenizer, Sentence Piece model [10],

with 45,000 token vocabulary to encode input text. Performance of the model is measured by F1 score.

The F1 score is not as strict as the Exact Match score (EM), and more closely resembles human judgment as far as the similarity of two answer strings. It measures the word overlap between the labeled and the predicted answer. It thus symmetrically represents both precision and recall in one metric. While EM is calculated on the character level, F1 is calculated on individual word level. We find the performance of the EM and F1 scores to be highly correlated so we report the F1 score alone.

For the machine translation task we pre-train the model on Multi30K-uk dataset using English-to-Ukrainian language pairs. Multi30K is a modification of the Flickr30K dataset with German translations of English annotations. Multi30K-uk is a variation of this dataset manually translated for Ukrainian language by [16]. The dataset consists of 31K English-Ukrainian sentence pairs, 357K tokens in English and 276K tokens in Ukrainian.

We set a sequence length of 758, a batch size of 128 sequences. We use Sentence Piece tokenizer with 72,000 token vocabulary to encode input text. We test the corresponding model and report the BLEU score output by SacreBLEU with default setting. Following [18] we use a 6-layer Transformer with encoder-decoder architecture.

BLEU (Bilingual Evaluation Understudy) is an algorithm designed to assess the quality of machine-translated text from one natural language to another. Quality is considered to be the correspondence between a machine's output and that of a human: the closer a machine translation is to a professional human translation, the better it is. BLEU was among the earliest automated metrics to claim a high correlation with human evaluation, and remains widely used due to its cost-effectiveness.

BLEU computes scores for individual translated segments by comparing them against a set of high-quality reference translations, which are then averaged across the entire corpus to gauge the overall translation quality.

However, comparing BLEU scores can be difficult because every decoder has its own implementation. Tokenization can also be handled in different ways. SacreBLEU aims to solve these problems by wrapping the original reference implementation together with other useful features, such as: automatic download of common WMT test sets, and support of different

tokenizers for BLEU. SacreBLEU provides a computation of shareable, comparable, and reproducible BLEU scores.

We present our results on the question answering and machine translation tasks for different position encoding methods in Table 2.

We conduct experiments on the Transformer model without positional encodings (w/o PE) to assess the importance of positional information. The outcomes reveal a notable decrease in performance, indicating the critical role of positional information in Transformer, which relies solely on the position insensitive attention mechanism.

Table 2

Comparing different position encoding methods

Model	QA (F1)	MT (SacreBLEU)
w/o PE	51.4	18.59
SPE	57.9	20.55
LAPE	59.0	28.74
RPE	64.8	29.3
CARPE	63.9	29.17
RSA	63.8	36.3
AT5	77.2	40.6

We included two experimental setups: one where the position encoder was present in all blocks, and another where it was only present at the input block. Our experiments show that incorporating absolute position encodings into attention matrices with different parameters for each head yields significant improvement compared to adding them directly to the input. This underscores the significance of where position information is integrated in the Transformer, providing an explanation for the gap in performance between absolute and relative position encodings.

Adding position encoders at all blocks yields better performance than only at the input block, except for the fixed-length position embedding approach. Possibly, this is due to over-fitting caused by learnable parameters introduced by this approach.

We found the original encoder-decoder architecture worked best for our models. Though an encoder-decoder model uses twice as many parameters as "encoder-only" or "decoder-only" architectures, it has a

similar computational cost. Furthermore, we observed that sharing the parameters in the encoder and decoder did not lead to a significant decrease in performance, despite reducing the total parameter count by half.

6. Conclusion

Non-recurrent Transformer models are not sensitive to position. The primary reason is that there is no inherent encoding of positional information for input units, making the models permutation equivalent. This problem explains why existing models are accompanied by a sinusoidal encoding or embedding layer at the input. Nonetheless, this approach has obvious limitations: the sinusoidal encoding lacks flexibility as it is manually designed and does not contain any trainable parameters, whereas the position embedding restricts the maximum length of input sequences. Besides that, it has not been explored how positional encoding builds positional dependencies for a sequence.

There is a range of work comparing and analyzing position information approaches, because positional encoding is such a critical issue for Transformer-like models. We chose to analyze several of these approaches in the context of question answering and machine translation tasks using Ukrainian corpora. We observed similar performance for absolute and relative encodings, although relative encodings are superior for longer sentences. Absolute position embeddings yield worse results mainly because they are added at the input. The superior performance of some relative position encoding methods can be attributed to their per-head addition to attention matrix rather than the position information being relative or absolute. Therefore, encoding position to attention matrix per-head results in superior performance for both absolute and relative information. A simple per-head position attention method achieves the state-of-the-art performance on NLP tasks and is more computationally efficient than existing approaches.

The original encoder-decoder architecture for the Transformer model proved to be the most suitable for the chosen NLP tasks. Though an encoder-decoder uses twice as many parameters as "encoder-only" or "decoder-only" architectures, it has a similar computational cost. Besides that, the number of learnable parameters can often be reduced without performance loss.

We would also like to stress how important it is to collect high-quality datasets for low-resource languages such as Ukrainian. Such datasets on

their own lead to a significant improvement of results accuracy for machine translation and other NLP tasks.

In the future, we aim to explore alternative operations that use word order to implicitly encode positional information, thus shedding light on how it impacts the performance of NLP models.

References:

1. Bahdanau D., Cho K., & Bengio Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. DOI: https://doi.org/10.48550/arXiv.1409.0473

2. Chen L., Varoquaux G., & Suchanek F. (2023). The Locality and Symmetry of Positional Encodings. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 14313–14331, Singapore. Association for Computational Linguistics. DOI: https://doi.org/10.18653/v1/2023.findings-emnlp.955

3. Chen P., Tsai H., Bhojanapalli S., Chung H. W., Chang Y., & Ferng C. (2021). A Simple and Effective Positional Encoding for Transformers. *Conference on Empirical Methods in Natural Language Processing*. DOI: https://doi.org/10.48550/arXiv.2104.08698

4. Dai Z., Yang Z., Yang Y., Carbonell J. G., Le Q. V., & Salakhutdinov R. (2019). Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. *Annual Meeting of the Association for Computational Linguistics*. DOI: https://doi.org/10.48550/arXiv.1901.02860

5. Devlin J., Chang M., Lee K., & Toutanova K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *North American Chapter of the Association for Computational Linguistics*, vol. 1, pp. 4171–4186. DOI: https://doi.org/10.18653/v1/N19-1423

6. Dufter P., Schmitt M., & Schütze H. (2021). Position Information in Transformers: An Overview. *Computational Linguistics*, vol. 48, pp. 733–763. DOI: https://doi.org/10.48550/arXiv.2102.11090

7. Gehring J., Auli M., Grangier D., Yarats D., & Dauphin Y. (2017). Convolutional Sequence to Sequence Learning. DOI: https://doi.org/10.48550/arXiv.1705.03122

8. Huang Z., Liang D., Xu P., & Xiang B. (2020). Improve Transformer Models with Better Relative Position Embeddings. DOI: https://doi.org/10.48550/arXiv.2009.13658

9. Ke G., He D., & Liu T. Y. (2020). Rethinking positional encoding in language pre-training. DOI: https://doi.org/10.48550/arXiv.2006.15595

10. Kudo T., & Richardson J. (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. DOI: https://doi.org/10.48550/arXiv.1808.06226

11. Liu X., Yu H., Dhillon I.S., & Hsieh C. (2020). Learning to Encode Position for Transformer with Continuous Dynamical Model. DOI: https://doi.org/10.48550/arXiv.2003.09229

12. Liu Y., Ott M., Goyal N., Du J., Joshi M., Chen D., Levy O., Lewis M., Zettlemoyer L., & Stoyanov V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. DOI: https://doi.org/10.48550/arXiv.1907.11692

13. Pham T. M., Bui T., Mai L., & Nguyen A. M. (2020). Out of Order: How important is the sequential order of words in a sentence in Natural Language Understanding tasks? DOI: https://doi.org/10.48550/arXiv.2012.15180

14. Raffel C., Shazeer N., Roberts A., Lee K., Narang S., Matena M., & Liu P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, vol. 21(140), pp. 1–67. DOI: https://doi.org/10.48550/arXiv.1910.10683

15. Rothman D. (2022). *Transformers for Natural Language Processing*. 2nd edition. Packt Publishing, 564 p.

16. Saichyshyna N., Maksymenko D., Turuta O., Yerokhin A., & Babii A. (2023). Extension Multi30K: Multimodal Dataset for Integrated Vision and Language Research in Ukrainian. In *Proceedings of the Second Ukrainian Natural Language Processing Workshop (UNLP)*, pp. 54–61, Dubrovnik, Croatia. Association for Computational Linguistics. DOI: https://doi.org/10.18653/v1/2023.unlp-1.7

17. Shaw P., Uszkoreit J., & Vaswani A. (2018). Self-attention with relative position representations. DOI: https://doi.org/10.48550/arXiv.1803.02155

18. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, vol. 30. DOI: https://doi.org/10.48550/arXiv.1706.03762

19. Wang G., Lu Y., Cui L., Lv T., Florencio D., & Zhang C. (2022). A Simple yet Effective Learnable Positional Encoding Method for Improving Document Transformer Model. In *Findings of the Association for Computational Linguistics: AACL-IJCNLP 2022*, pp. 453–463.

20. Wang W., Bi B., Yan M., Wu C., Bao Z., Peng L., & Si L. (2019). StructBERT: Incorporating Language Structures into Pre-training for Deep Language Understanding. DOI: https://doi.org/10.48550/arXiv.1908.04577

21. Wu J., Zhang R., Mao Y., & Chen J. (2021). On Scalar Embedding of Relative Positions in Attention Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35(16), pp. 14050–14057. DOI: https://doi.org/10.1609/aaai.v35i16.17654

22. Xilong Zh., Ruochen L., Jin L., & Xuefeng L. (2023). Interpreting Positional Information in Perspective of Word Order. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, vol. 1, pp. 9600–9613. Toronto, Canada. Association for Computational Linguistics. DOI: https://doi.org/10.18653/v1/2023.acl-long.534

23. Yang Z., Dai Z., Yang Y., Carbonell J., Salakhutdinov R. R., & Le Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, vol. 32. DOI: https://doi.org/10.48550/arXiv.1906.08237