Relative Positional Encoding for Transformers with Linear Complexity

Antoine Liutkus, Ondřej Cífka, Shih-Lun Wu, Umut Şimşekli, Yi-Hsuan Yang, Gael Richard

To cite this version:
Antoine Liutkus, Ondřej Cífka, Shih-Lun Wu, Umut Şimşekli, Yi-Hsuan Yang, et al.. Relative Positional Encoding for Transformers with Linear Complexity. ICML 2021 - 38th International Conference on Machine Learning, Jul 2021, Virtual Only, United States. hal-03256451

HAL Id: hal-03256451
https://hal.telecom-paris.fr/hal-03256451
Submitted on 10 Jun 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Relative Positional Encoding for Transformers with Linear Complexity

Antoine Liutkus* 1 Ondřej Cifka* 2 Shih-Lun Wu3 4 5 Umut Şimşekli 6 Yi-Hsuan Yang 3 5 Gaël Richard 2

Abstract

Recent advances in Transformer models allow for unprecedented sequence lengths, due to linear space and time complexity. In the meantime, relative positional encoding (RPE) was proposed as beneficial for classical Transformers and consists in exploiting lags instead of absolute positions for inference. Still, RPE is not available for the recent linear-variants of the Transformer, because it requires the explicit computation of the attention matrix, which is precisely what is avoided by such methods. In this paper, we bridge this gap and present Stochastic Positional Encoding as a way to generate PE that can be used as a replacement to the classical additive (sinusoidal) PE and provably behaves like RPE. The main theoretical contribution is to make a connection between positional encoding and cross-covariance structures of correlated Gaussian processes. We illustrate the performance of our approach on the Long-Range Arena benchmark and on music generation.

1. Introduction

1.1. Linear Complexity Transformers

The Transformer model (Vaswani et al., 2017) is a new kind of neural network that quickly became state-of-the-art in many application domains, including the processing of natural language (He et al., 2020), images (Dosovitskiy et al., 2020), audio (Huang et al., 2018; Pham et al., 2020) or bioinformatics (AlQuraishi, 2019) to mention just a few.

The core, novel component of the Transformer is the attention layer. It computes $M$ output values $y_m$ from $N$ input values $v_n$, all being vectors of an arbitrary dimension.

$\text{Figure 1.}$ Examples of attention patterns observed in the Performers trained for pop piano music generation (section 3.2) at inference time, for sequence length $M = N = 3072$ while training sequences have length 2048. (left) Absolute PE. (middle) Sinusoidal SPE. (right) Convolutional SPE. Note that SPE never requires computing these full attention patterns.

### Figure 1

The attention layer is a functional component of the Transformer, allowing for the computation of weighted sums of input values. Following classical non-parametric regression principles (Nadaraya, 1964; Watson, 1964), it consists in a simple weighted sum:

$$y_m = \frac{\sum_n a_{mn} v_n}{\sum_n a_{mn}},$$

where each attention coefficient $a_{mn} \in \mathbb{R}_+$ indicates how important the value $v_n$ is in the computation of the output $y_m$.

One of the main contributions of the Transformer is an original method to compute these coefficients. $D$-dimensional feature vectors $k_n$ and $d_m$ are attached to all items of the input and output sequences and are called keys and queries, respectively. Gathering them in the $N \times D$ and $M \times D$ matrices $K$ and $Q$, we get softmax dot-product attention as:

$$A = \exp \left( Q K^\top / \sqrt{D} \right) \equiv [a_{mn} = K(q_m, k_n)]_{mn},$$

where the function $\exp$ is applied element-wise. The right-hand side in (2) is a generalization introduced by Tsai et al. (2019) and Choromanski et al. (2020), where $K$ is a kernel function. Parameters pertain to how keys $k_n$, values $v_n$ and queries $q_m$ are obtained from the raw sequences, usually by time-distributed fully connected layers.

The original Transformer architecture (Vaswani et al., 2017) explicitly computes the attention matrix $A$, leading to a $O(MN)$ complexity that prevents it from scaling to very long sequence lengths. Although this is not necessarily a problem when sequence lengths are barely on the order of a few hundreds, as in some language processing tasks, it is
Whenever an efficient scheme like (3) or (4) is used, the would lead to the same output. Two core approaches were works (Bello, 2020), which encapsulate the key-value information also avoiding the computation of a full attention matrix.

1.2. Positional Encoding

In Transformer networks, the outputs are computed as linear combinations of all input values weighted by attention coefficients. In sequence modeling, it is reasonable to assume that the actual positions and should play a role in the computation, in addition to the content at these locations; otherwise, any permutation of the sequence would lead to the same output. Two core approaches were undertaken to incorporate position information:

- The original Transformer (Vaswani et al., 2017) adds this

information to the inputs of the network, i.e. before the first attention layer. This can be equivalently understood as augmenting the keys, values and queries:

\[ k_n \leftarrow k_n + \overline{k}_n, \quad v_n \leftarrow v_n + \overline{v}_n, \quad q_m \leftarrow q_m + \overline{q}_m, \]

where we write \( \overline{k}_n \in \mathbb{R}^D \) for the keys positional encoding (PE; Sukhbaatar et al., 2015) at position \( n \in \mathbb{N} \) and analogously for values and queries. Vaswani et al. propose a deterministic scheme based on trigonometric functions, which is shown to work as well as trainable embeddings.

- As an example of positional encoding in the attention domain, a relative positional encoding (RPE) was proposed by Shaw et al. (2018), building on the idea that time lags \( m - n \) are more important than absolute positional encoding (APE) for prediction. It is written as:

\[ A = \exp \left( \frac{(QK^\top + \Omega)}{\sqrt{D}} \right), \quad \Omega = \frac{1}{D} \sum_{d=1}^{D} q_{md} P_d(m - n) \]

The terms \( P_d \) now act as \( D \) different encodings for time lags selected based on the queries. This change is advocated as bringing important performance gains in many application areas and has enjoyed a widespread use since.

Although writing down the positional encoding in the attention domain is beneficial for performance (Shaw et al., 2018; Dai et al., 2019; Tsai et al., 2019), we are only aware of implementations that either require the computation of \( A \) or clustered attention schemes, which in fine decompose \( A \) into smaller attention matrices, and compute them. This is in sharp contrast to (3) and (4), which never compute the attention matrix.

Our contributions can be summarized as follows:

- We propose Stochastic Positional Encoding (SPE) as a general PE scheme in the keys domain, that enforces a particular attention pattern devised in the attention domain. This enables RPE without explicit computation of attention. To our knowledge, it is the first RPE strategy that is compatible with \( O(N) \) Transformers like Choromanski et al. (2020) and Katharopoulos et al. (2020).
- We study the impact of SPE on performance on the Long-Range Arena benchmark (Tay et al., 2021) and two music generation tasks. Since RPE was so far limited to short sequences, we believe this is the first study of its advantages on long-range predictions. Our results demonstrate better validation losses and extrapolation ability.
- We provide additional resources on our companion website, including Python implementations of SPE for PyTorch and JAX/Flax.

\[ 1 \text{somewhat related strategy is used by the recent LambdaNetworks (Bello, 2020), which encapsulate the key-value information as a so-called lambda function to be applied query-wise, hence also avoiding the computation of a full attention matrix.} \]

\[ 2 \text{https://cifkao.github.io/spe/} \]
We use bold uppercase for matrices, bold lowercase for vectors. Positional encoding:

- Draw the \( D \) independent couples \( \{ \mathbf{Q}_d, \mathbf{K}_d \}_d \) of \( M \times R \) and \( N \times R \) matrices as in section 2.1.
- Set \( \mathbf{Q} \) and \( \mathbf{K} \) as in (16) and (17).

Inference compute outputs \( \mathbf{Y} \) with the \( O(N) \) Transformer:

\[
\mathbf{Y} \leftarrow \text{diag}(\mathbf{d})^{-1} \left[ \phi \left( \mathbf{Q} \right) \left[ \phi \left( \mathbf{K} \right)^\top \mathbf{V} \right] \right] \tag{10}
\]

with \( \mathbf{d} = \phi(\mathbf{Q}) \left[ \phi \left( \mathbf{K} \right)^\top \mathbf{1}_N \right] \) and \( \phi \) discussed in (3)/(4).

2. Stochastic Positional Encoding

Index set and notation. We assume that the input/output sequences are indexed by \( n, m \in \mathbb{T} \), where \( \mathbb{T} \) is the index set. For regularly sampled sequences, we have \( \mathbb{T} = \mathbb{N} \), but more settings are possible, like irregularly sampled time series (\( \mathbb{T} = \mathbb{R} \)) or images (\( \mathbb{T} = \mathbb{N}^2 \)). In any case, the particular lists of input/output locations under consideration are written: \( \mathcal{N} \) and \( \mathcal{M} \), with respective sizes \( N \) and \( M \) (the case \( \mathcal{N} = \mathcal{M} \) is called self-attention). The corresponding keys and values are hence indexed as \( \{ \mathbf{k}_n \}_{n \in \mathcal{N}} \) and \( \{ \mathbf{v}_n \}_{n \in \mathcal{N}} \), while queries are \( \{ \mathbf{q}_m \}_{m \in \mathcal{M}} \). For convenience, we write \( a_{mn} \) for the entries of the \( M \times N \) attention matrix \( \mathbf{A} \).

We use bold uppercase for matrices, bold lowercase for vectors and a NumPy-like notation: if \( \mathbf{X}_k \) is an \( I \times J \) matrix, \( x_{k,i} \) and \( x_{k,j} \) stand for its \( i \)-th row and \( j \)-th column, respectively.

Assumptions. In the remainder of this paper, we will seek an attention matrix \( \mathbf{A} \) given by:

\[
\mathbf{A} = \exp \left( \sum_{d=1}^{D} a_{mn} \mathbf{P}_d(m,n) \mathbf{k}_{nd} \right)_{mn} / \sqrt{D} , \tag{8}
\]

where \( \{ \mathbf{P}_d \}_{d=1}^{D} \) are position kernels. Defining \( \mathbf{P}_d = [\mathbf{P}_d(m,n)]_{mn} \), this can be written in matrix form as:

\[
\mathbf{A} = \exp \left( \sum_{d=1}^{D} \text{diag}(\mathbf{q}_{d}) \mathbf{P}_d \text{diag}(\mathbf{k}_{d}) / \sqrt{D} \right) , \tag{9}
\]

which is understood as having \( D \) positional attention templates \( \mathbf{P}_d \) jointly activated by the queries \( \mathbf{q}_{d} \) and keys \( \mathbf{k}_{d} \). Original RPE (7) can be seen as a special case, where some entries are kept constant.

Positional attention as covariance. The key idea for SPE is to see the attention kernel \( \mathbf{P}_d(m,n) \) as a covariance:

\[
\langle \forall \mathcal{M}, \mathcal{N} \rangle \langle \forall m, n \rangle \mathbf{P}_d(m,n) = \mathbb{E} [ \mathbf{Q}_d(m) \mathbf{K}_d(n) ] , \tag{11}
\]

where \( \mathbf{Q}_d(m) \) and \( \mathbf{K}_d(n) \) are two real and zero-mean ran-

In what follows, we will impose specific structures on the cross-covariance \( \mathbf{P}_d(m,n) \), which will in turn allow us to design random processes \( \mathbf{Q}_d \) and \( \mathbf{K}_d \) such that (11) holds. The core advantage of this construction is to allow for \( \mathbf{P}_d \) to be factorized. Let us for now assume that we construct the distributions of \( \{ \mathbf{Q}_d(m) \}, \{ \mathbf{K}_d(n) \} \) in such a way that we can sample from them (we will see how in section 2.1) and consider \( \mathcal{R} \) independent realizations of them for given \( \mathcal{M} \) and \( \mathcal{N} \), gathered in the \( M \times N \) and \( N \times R \) matrices \( \mathbf{Q}_d \) and \( \mathbf{K}_d \):

\[
\mathbf{Q}_d \equiv [\mathbf{q}_{d,m,r} \sim \mathbf{Q}_d(m) ]_{mr} , \quad \mathbf{K}_d \equiv [\mathbf{k}_{d,n,r} \sim \mathbf{K}_d(n) ]_{nr} . \tag{12}
\]

For large \( R \), by the law of large numbers, we obtain:

\[
\mathbf{P}_d \approx \left[ \mathbf{Q}_d \mathbf{K}_d^\top \right] / R . \tag{13}
\]

This leads \( \mathbf{A} \) in (9) to be given by:

\[
\mathbf{A} \approx \exp \left( \sum_{d=1}^{D} \text{diag}(\mathbf{q}_{d}) \frac{\mathbf{Q}_d \mathbf{K}_d^\top}{R} \text{diag}(\mathbf{k}_{d}) / \sqrt{D} \right) \tag{14}
\]

\[
\approx \exp \left( \sum_{d=1}^{D} \text{diag}(\mathbf{q}_{d}) \mathbf{Q}_d \mathbf{K}_d^\top \right) / \sqrt{RD} \tag{15}
\]

Here, a crucial observation is that for large \( R \), the cross terms \( \mathbf{Q}_d^\top \mathbf{K}_{d' \neq d} \) are negligible due to independence, provided that the means of the processes are selected to be zero. Finally, picking queries and keys as:

\[
\widehat{\mathbf{Q}} \leftarrow \sum_{d=1}^{D} \text{diag}(\mathbf{q}_{d}) \mathbf{Q}_d / \sqrt{RD} , \tag{16}
\]

\[
\widehat{\mathbf{K}} \leftarrow \sum_{d=1}^{D} \text{diag}(\mathbf{k}_{d}) \mathbf{K}_d / \sqrt{RD} , \tag{17}
\]

we see from (15-17) that we get back to the usual multiplicative scheme (2) with \( \mathbf{A} = \exp(\widehat{\mathbf{Q}} \mathbf{K}_d^\top / \sqrt{R}) \), where the queries/keys now have dimension \( R \) and can be used in (10) to directly get outputs without computing \( \mathbf{A} \).

The procedure is summarized in Algorithm 1: we provide a way (16-17) to achieve PE in the keys domain, such that the desired model (8) is enforced in the attention domain, pa-
rameterized by the attention kernels \( \mathcal{P}_d \). Interestingly, this is done without ever computing attention matrices, complying with \( \mathcal{O}(N) \) Transformers. The remaining challenge, which we discuss next, is to generate \( \mathcal{Q}_d \) and \( \mathcal{K}_d \) enforcing (13).

2.1. Drawing Stochastic Positional Encodings

Inspecting (11), we notice that our objective is to draw samples from \( D \) pairs of centered random processes \( \{ \mathcal{Q}_d, \mathcal{K}_d \}_d \), with a prescribed cross-covariance structure \( \mathcal{P}_d \). It is reasonable to use Gaussian processes for this purpose (Williams & Rasmussen, 2006), which have the maximum entropy for known mean and covariance. Such distributions are frequently encountered in geophysics in the co-kriging literature (Matheron, 1963; Genton & Kleiber, 2015), where scientists routinely handle correlated random fields. The particular twist of our setup are: we have a generative problem, e.g. as in Voechevsky (2008); however, as opposed to their setting, we are not directly interested in the marginal covariance function of each output, provided that the desired cross-covariance structure holds.

The most straightforward application of SPE arises when we pick \( \mathcal{P}_d(m, n) = \mathcal{P}_d(m - n) \), i.e. a stationary position kernel, which was coined in as choosing relative attention in Shaw et al. (2018) and boils down to enforcing a Toeplitz structure for the cross-covariance matrix \( \mathcal{P}_d \equiv [\mathcal{P}_d(m - n)]_{m,n} \) between \( \mathcal{Q}_d \) and \( \mathcal{K}_d \).

We propose two variants of SPE to handle this important special case, illustrated in Figure 2. The first variant yields periodic covariance functions. It can be beneficial whenever attention should not vanish with large lags, as in traffic prediction (Xue & Salim, 2020) or, as we show, in music generation. The second variant generates vanishing covariance functions; a concept which has recently been shown useful (Wang et al., 2021), and notably yields smaller validation losses in some of our experiments.

**Variant I. Relative and periodic attention (sineSPE).** In our first approach, we consider the case where \( \mathcal{P}_d \) is periodic, which gets a convenient treatment. We assume:

\[
\mathcal{P}_d(m, n) = \sum_{k=1}^{K} \lambda^2_{kd} \cos(2\pi f_{kd} (m - n) + \theta_{kd}), \tag{18}
\]

where \( K \in \mathbb{N} \) is the number of sinusoidal components and \( f_{kd} \in [0, 1]^K, \theta_{kd} \in [-\pi, \pi]^K \) and \( \lambda_{kd} \in \mathbb{R}^K \) gather their \( K \) frequencies, phases, and weights, respectively. By using the matrix notation, we can rewrite (18) as:

\[
\mathcal{P}_d = \mathbf{\Omega}(\mathcal{M}, f_d, \theta_d) \mathbf{\Omega}(\mathcal{N}, f_d, 0), \tag{19}
\]

where \( \mathbf{\Omega} \equiv [v_{\lfloor \cdot/2 \rfloor}]_{p} \in \mathbb{R}^{2K} \) denotes a twice upscaled version of a vector \( v \in \mathbb{R}^K \), \( \lfloor \cdot \rfloor \) denotes the floor operation, and for an index set \( \mathcal{I} \), \( \mathbf{\Omega}(\mathcal{I}, \mathbf{a}, \mathbf{b}) \) is a matrix of size \( |\mathcal{I}| \times 2K \), with entries (0-based indexing):

\[
[\mathbf{\Omega}(\mathcal{I}, \mathbf{a}, \mathbf{b})]_{m} = \begin{cases} 
\cos(2\pi a_k n + b_k) & \text{if } l = 2k \\
\sin(2\pi a_k n + b_k) & \text{if } l = 2k + 1
\end{cases}
\]

It can be shown that if \( \theta_d = 0 \) and \( \mathcal{M} = \mathcal{N} \), we get back to the (unique) Vandermonde decomposition for positive definite Toeplitz matrices\(^3\) (Yang et al., 2016), which boils down in our context to assuming that \( \forall \tau, \mathcal{P}_d(\tau) \geq \mathcal{P}_d(\tau) \). Since this is not always desirable, we keep the more general (19).

At this point, we can easily build \( \mathcal{Q}_d \) and \( \mathcal{K}_d \). We draw a \( 2K \times R \) matrix \( \mathcal{Z}_d \) with independent and identically distributed (i.i.d.) Gaussian entries of unit variance, and define:

\[
\mathcal{Q}_d \leftarrow \mathbf{\Omega}(\mathcal{M}, f_d, \theta_d) \mathbf{\Omega}(\mathcal{N}, f_d, 0), \mathcal{Z}_d / \sqrt{2K}, \tag{20}
\]

\[
\mathcal{K}_d \leftarrow \mathbf{\Omega}(\mathcal{M}, f_d, 0) \mathbf{\Omega}(\mathcal{N}, f_d, 0), \mathcal{Z}_d / \sqrt{2K}. \tag{21}
\]

It is easy to check that such a construction leads to (13). Its parameters are \( \{f_d, \theta_d, \lambda_{kd}\}_d \), which can be trained through stochastic gradient descent (SGD) as usual.

**Variant II. Relative (vanishing) attention with regular sampling (convSPE).** Due to their periodic structure, the covariance functions generated by Variant I are non-vanishing. Yet, our framework is flexible enough to allow for vanishing covariance structures, which may be more desirable depending on the application (Wang et al., 2021).

As opposed to Variant I, where we imposed a specific structure on \( \mathcal{P}_d \), we will now follow an indirect approach, where \( \mathcal{P}_d \) will be implicitly defined based on our algorithmic construction. In this case, we assume that the signals are regularly sampled (typical in e.g. text, images, audio), and we will exploit the structure of Gaussian random matrices and basic properties of the convolution operation.

For ease of notation, we assume self attention, i.e. \( \mathcal{M} = \mathcal{N} \). Let \( \{\Phi^0_d, \Phi^K_d\}_d \) denote a collection of filters, which will ultimately be learned from training data. The size and the dimension of these filters can be chosen according to the input data (i.e. can be vectors, matrices, tensors). We then propose the following procedure, which leads to a Toeplitz \( \mathcal{P}_d \) by means of convolutions:

- We first draw an \( M \times R \) random matrix \( \mathcal{Z}_d \) with i.i.d. standard Gaussian entries. For multidimensional signals, \( \mathcal{Z}_d \) gathers \( R \) random vectors, matrices, cubes, etc.
- The desired \( \mathcal{Q}_d \) and \( \mathcal{K}_d \) are obtained by convolving \( \mathcal{Z}_d \) with respective filters \( \Phi^0_d \) and \( \Phi^K_d \):

\[
\mathcal{Q}_d = \mathcal{Z}_d * \Phi^0_d, \mathcal{K}_d = \mathcal{Z}_d * \Phi^K_d, \tag{22}
\]

where \( * \) denotes convolution with appropriate dimension (e.g. 1D, 2D or 3D). Using convolutions with finite filters

\(^3\)If \( \mathcal{P}_d \geq 0 \) and \( \mathcal{M} \geq N \), (19) still holds but is not unique.
ensures vanishing covariance, as proven in the appendix.

Due to the independence of the entries of \( Z_d \), for large \( R \), the product \( Z_d Z_d^\top / R \) will tend to the identity matrix. Given the fact the convolution operations in (22) can be equivalently expressed as a multiplication by triangular Toeplitz matrices constructed from the respective filters, it can be shown that, as \( R \to \infty \), \( \frac{1}{R} \tilde{Q}_d \Phi_d K_d^\top \) tends to the product of two triangular Toeplitz matrices. Hence, by using the properties of triangular Toeplitz matrices (cf. Kucerovsky et al. 2016, Theorem 2.4), we conclude that, as \( R \to \infty \), our construction yields a Toeplitz matrix \( P_d \) as desired. This approach is parameterized by the filters \( \{ \Phi^Q_d, \Phi^K_d \}_d \), which will be learned from training data through SGD.

The variety of attention patterns \( P(m-n) \) that can be obtained directly depends on the kernel sizes, which is a classical result from signal processing (Vetterli et al., 2014). Cascading several convolutions as in the VGGNet (Simonyan & Zisserman, 2014) may be a convenient way to augment the expressive power of this convolutional SPE variant.

From a more general perspective, the two operations in (22) can be understood as producing PE through filtering white noise, which is the core idea we introduce for PE. Other classical signal processing techniques may be used like using infinite impulse response filters. Such considerations are close to the ideas proposed in (Engel et al., 2020).

To summarize, the core difference between the two proposed constructions (20–21) and (22) lies in the behaviour of RPE beyond a maximum lag, implicitly defined through the frequencies \( f_d \) for (20–21) and through the sizes of the filters for (22). While the sinusoidal construction leads to a periodic RPE, the filtering construction leads to a vanishing RPE, which is called monotonic in (Wang et al., 2021). Both may be the desired option depending on the application.

2.2. Gated SPE

Although RPE and the generalization (9) we propose are novel and efficient strategies to handle position information, it may be beneficial to also allow for attention coefficients that are computed without positional considerations, simply through \( \langle q_m, k_n \rangle \). As a general gating mechanism, we propose to weight between positional and non-positional attention through a gate parameter \( \delta_d \in [0, 1] \):

\[
P_d \equiv [\delta + (1 - \delta) P_d(m,n)]_{m,n}.
\]

This gating scheme can be implemented simply by augmenting \( \tilde{Q}_d \) and \( \tilde{K}_d \) generated as above through:

\[
\tilde{q}_{d,m} \leftarrow \sqrt{1 - \delta_d} \tilde{q}_{d,m} + \sqrt{\delta_d} \epsilon_d, \tag{24}
\]

\[
\tilde{k}_{d,m} \leftarrow \sqrt{1 - \delta_d} \tilde{k}_{d,m} + \sqrt{\delta_d} \epsilon_d, \tag{25}
\]

where \( \epsilon_d \in \mathbb{R}^R \) in (24) and (25) is the same and has i.i.d. standard Gaussian entries.

In practice, we can share some SPE parameters across the network, notably across layers, to strongly reduce computing time and memory usage. In our implementation, sharing means generating a single instance of \( \tilde{Q} \) and \( \tilde{K} \) for each head, on which a layer-wise gating is applied, before achieving PE through (16–17). This is illustrated in Figure 2.
Table 1. Long-Range Arena results (higher scores are better). Mean and standard deviation of accuracy over three runs is reported, except for Performer with convolutional SPE, where only a single run was completed. For comparison, the best result reported by Tay et al. (2021), along with the name of the best-performing model (in parentheses), is included.

<table>
<thead>
<tr>
<th></th>
<th>ListOps</th>
<th>Text</th>
<th>Retrieval</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best result from Tay et al. (2021)</td>
<td>37.27</td>
<td>65.90</td>
<td>59.59</td>
<td>44.24</td>
</tr>
<tr>
<td>(Reformer)</td>
<td>(Linear Trans.)</td>
<td>(Sparse Trans.)</td>
<td>(Sparse Trans.)</td>
<td></td>
</tr>
<tr>
<td>Linear Transformer-ReLU from Tay et al.</td>
<td>18.01</td>
<td>65.40</td>
<td>53.82</td>
<td>42.77</td>
</tr>
<tr>
<td>Performer-softmax (APE)</td>
<td>17.80 ± 0.00</td>
<td>62.58 ± 0.22</td>
<td>59.84 ± 1.46</td>
<td>41.81 ± 1.16</td>
</tr>
<tr>
<td>Performer-softmax + sineSPE</td>
<td>17.43 ± 0.32</td>
<td>62.60 ± 0.50</td>
<td>60.00 ± 1.20</td>
<td>41.12 ± 1.70</td>
</tr>
<tr>
<td>Performer-softmax + convSPE</td>
<td>17.80</td>
<td>60.94</td>
<td>57.22</td>
<td>40.06</td>
</tr>
<tr>
<td>Linear Transformer-ReLU (APE)</td>
<td>17.58 ± 1.01</td>
<td>63.98 ± 0.05</td>
<td>58.78 ± 0.93</td>
<td>42.25 ± 0.01</td>
</tr>
<tr>
<td>Linear Transformer-ReLU + sineSPE</td>
<td>17.80 ± 0.00</td>
<td>64.09 ± 0.62</td>
<td>62.39 ± 0.59</td>
<td>41.21 ± 1.18</td>
</tr>
<tr>
<td>Linear Transformer-ReLU + convSPE</td>
<td>9.50 ± 1.17</td>
<td>63.23 ± 1.31</td>
<td>61.00 ± 1.34</td>
<td>39.96 ± 1.31</td>
</tr>
</tbody>
</table>

3. Experiments

3.1. Long-Range Arena

Experimental setup. We evaluate the proposed method in the Long-Range Arena (LRA; Tay et al., 2021), a benchmark for efficient Transformers, consisting of sequence classification tasks with a focus on long-range dependencies. We use the following tasks from this benchmark:

- **ListOps**: parsing and evaluation of hierarchical expressions, a longer variant of (Nangia & Bowman, 2018);
- **Text**: movie review sentiment analysis on the IMDB corpus (Maas et al., 2011);
- **Retrieval**: article similarity classification on the All About NLP (AAN) corpus (Radev et al., 2013);
- **Image**: object recognition on the CIFAR10 dataset (Krizhevsky, 2009) represented as pixel sequences.

The tasks are challenging due to the large sequence lengths, deliberately increased by choosing a character-/pixel-level representation. An overview of the tasks can be found in the appendix. We do not include *Pathfinder* (a synthetic image classification task) as we were unable to reproduce the results of Tay et al. on this task, even through correspondence with the authors.

We evaluate SPE (the gated variant) on two efficient Transformer models: the (softmax) Performer (Choromanski et al., 2020), and a Linear Transformer (Katharopoulos et al., 2020) with a ReLU feature map, i.e. choosing \( \phi(\cdot) = \max(0, \cdot) \) element-wise in (3). It should be noted that the ReLU feature map does not approximate the softmax kernel, which SPE is designed for (see assumption 8). Nevertheless, it is possible to use SPE with any feature map in practice, allowing us to include Linear Transformer-ReLU as an interesting test of generalization to alternative kernels.

We adopt the configuration of Tay et al., only changing the PE and the batch sizes/learning rates to allow training on limited hardware with similar results. All other hyperparameters are kept identical to the original LRA. It is worth noting that the Image models are different from the rest in that they employ a single-layer network and only use the first position for prediction, dramatically limiting their ability to benefit from relative positional information.

Since we observe some variation between different runs, we train and evaluate each model 3 times (except for Performer with convolutional SPE, which is computationally more costly) and report the mean and standard deviation of the results.

The results of the benchmark are given in Table 1. The accuracies achieved by the baseline Linear Transformer-ReLU (APE) are similar to or surpass those reported by Tay et al., which is a clear validation of our experimental setup.

Discussion. Results on ListOps are poor overall, with accuracies around 17%. This complies with Tay et al. (2021), who reasoned that “kernel-based models [e.g. Performer, Linear Transformers] are possibly not as effective on hierarchically structured data,” leaving room for improvement. We also hypothesize this is largely due to some known issues with the training data for this task, which unfortunately have not been fixed at the time of this writing.

Regarding performance of SPE, we first notice that the sineSPE variant yields the best results on three tasks, which is a strong achievement and validates our approach, especially considering the difficulty of this evaluation benchmark. While it is only marginally better than APE for ListOps and Text, it is worth mentioning that sineSPE combined with the Linear Transformer-ReLU yields an accuracy improvement of ~3% on Retrieval compared to the best result obtained by Tay et al. (2021).

---

A model named ‘Performer’ is reported by Tay et al., but communication with the authors revealed it to be in fact equivalent to our Linear Transformer-ReLU, as it does not use random features. To avoid confusion, we refer to this model as such herein.

3Currently, the official data loader for ListOps inadvertently strips some characters from the input sequences.
Regarding convSPE, its performance in the LRA is not as remarkable as it is for the music generation experiment reported later in section 3.2. This mitigated result appears somewhat in contradiction with the discussion found in Wang et al. (2021), which presents vanishing attention as a desirable property of PE. On the contrary, we empirically observe that our non-vanishing sinusoidal version sineSPE does behave better in these particular tasks.

Finally, the superior results of APE on Image are not unexpected, given the limited ability of these models to exploit relative positions. On the contrary, the relatively good performance of SPE on this task is in fact remarkable, especially considering that the baseline systems for this task use learnable APE.

As we will see later in our music generation experiments, there are tasks where our proposed SPE clearly yields remarkable improvements. Here in the LRA, we notice that it does not result in an obvious and systematic boost in performance. This raises interesting considerations:

(i) The variance of the Monte Carlo estimator might be problematic. We are enthusiastic about the elegant formulation of stochastic feature maps as in the Performer, which was a strong inspiration. Still, we must acknowledge that their computation relies on a Monte Carlo estimator (15). We suspect that the variance of the estimator might play a role in the final performance in large dimensions, which opens up the direction of exploring variance-reduced estimation methods, rather than plain Monte Carlo.

(ii) LRA tasks might not benefit from strong (R)PE schemes. The LRA was designed to compare Transformer architectures, filling a gap in this domain and standing as the de facto standard, justifying our choice. Still, although PE is known to be important in many cases, it is not known whether it is so in the LRA tasks. We feel that there is room for such a specialized comparison, which is scheduled in our future work, possibly leading to new long-range tasks where PE is critical.

3.2. Pop Piano Music Generation

In our music generation experiments (this subsection and section 3.3), music is represented as sequences of symbols (tokens) and a Performer (Choromanski et al., 2020) is used as an autoregressive language model, which predicts a probability distribution over the next token given the past context. At test time, a new sequence is generated by iteratively sampling the next token, as commonly done in text generation.

Experimental setup. We train Performers for music generation, with 24 layers and 8 heads per layer on a dataset composed of 1,747 pop piano tracks, encoded using the recently proposed Revamped MIDI-derived format (REMI; Huang & Yang, 2020). The sequences are composed of

![Figure 3. Validation cross-entropy vs. token position on pop piano music generation task. (lower is better; the black vertical line indicates the maximum position to which the models are trained.)](image)

metrical tokens: bar, subbeat, and tempo, which represent musical timing; and note tokens: chord, pitch, duration, and volume, which describe the musical content (see the appendix for more details). We hold out 5% of the songs as the validation set.

We train the models with sequence length \( N = 2048 \), corresponding to ~1 minute of music. The only difference between our models is the PE strategy. We consider baseline APE, as well as SPE: sinusoidal or convolutional, with or without gating, resulting in 5 different models.

Results and discussion. For qualitative assessment, we first display in Figure 1 one attention pattern for each PE model: APE and (gated) sineSPE/convSPE, obtained as an average over 20 from-scratch generations for a chosen (layer, head). More similar plots can be found in appendix. Interestingly, we notice that for early layers, APE attention does not go much beyond training sequence length. This behaviour is not found in SPE variants, which consistently attend to all positions. Another remarkable feature of the proposed model (only displayed in the appendix) is that gating as described in section 2.2 visually disables PE altogether for some layers/heads, in which case attention is global.

Since the literature suggests that RPE improves generalization performance (Shaw et al., 2018; Zhou et al., 2019; Rosendahl et al., 2019), we display validation cross-entropy computed with teacher forcing (Williams & Zipser, 1989) in Figure 3, as a function of the target token position. The values would indicate how well the models predict the token at a certain position given the preceding tokens, for tracks in the validation set. We notice that all SPE variants, especially convSPE, behave much better than APE for token positions beyond 2048. This suggests that SPE inherits this celebrated advantage of RPE (Huang et al., 2018) while being applicable to much longer sequences.

Recently, Wang et al. (2021) defined metrics for the evaluation of PE, suggesting that translation invariance and monotonicity are desirable properties. The former states that the distances of two arbitrary \( \tau \)-offset position embeddings should be identical, while the latter states that neighboring
positions should be assigned with position embeddings that are closer than faraway ones. Following their identical word probing methodology, we report these metrics in Figure 4. As expected, SPE variants greatly outperform APE in terms of translation invariance. However, monotonicity does not seem a very relevant criterion in our music application, as can be seen when comparing scores in Figures 3 and 4. It seems that music modeling can benefit from non-vanishing attention patterns. In any case, SPE scores are remarkably stable across positions, contrarily to APE, which rapidly degrades beyond the training length.

3.3. Groove Continuation

In this experiment, we evaluate Performers on a groove continuation task. After training on a dataset where each example has a uniform style (‘groove’), we prime the model with a short prompt (2-bar musical fragment) and let it generate a continuation. We then observe whether the generated continuation matches the style of the prompt.

Experimental setup. The models (24-layer Performers with 8 attention heads) are trained on an accompaniment dataset comprising 5 522 samples in 2 761 different musical styles, encoded in a token-based format adopted from Cifka et al. (2020) and detailed in the appendix. All SPE-based models use gating in this experiment. Unlike the previous experiment, which leverages long training sequences, we consider training sequences of length \( N = 512 \), corresponding to 2–10 bars. At test time, the model is prompted with 2 bars in a style not seen during training and new tokens are sampled to complete the sequence to a length of 1 024, i.e. twice the training length.

We use two musically motivated style similarity metrics – time-pitch and onset-duration proposed by Cifka et al. (2019; 2020) – to quantify the similarity of the generated continuation to the prompt. When listening to the generated music, we perceptually notice a drift in quality along time. For this reason, we divide each generated sample into four successive chunks of identical duration and evaluate them independently. The results are displayed in Figure 5.

Discussion. We clearly see that SPE substantially outperforms APE in both metrics. Although APE visibly does manage to generate close to the desired style at the beginning of the sequence, this similarity strongly degrades over time. Both sineSPE and convSPE are much more stable in this regard, confirming the result from section 3.2 that SPE extrapolates better beyond the training sequence length. This matches our informal perceptual evaluation.6

This experiment suggests that exploiting a local neighborhood is a robust way to process long sequences. This could appear as contradicting the use of long-range Transformers, but we highlight that gating is used here, enabling some heads to exploit long-term attention independently from position. Further comparisons with local attention schemes (e.g. Dai et al., 2019; Hofstätter et al., 2020) could be interesting, although they were not included here due to Tay et al. (2021) suggesting that they are clearly inferior, at least in the LRA setting.

4. Related Work

This paper is concerned with PE (Sukhbaatar et al., 2015), as a way to embed the position of each token as part of its features. This idea is a core ingredient for many subsequent groundbreaking studies (Gehring et al., 2017; Vaswani et al., 2017), and has been the actual topic of many investigations.

---

6Examples: https://cifkao.github.io/spe/
Absolute Positional Encoding (APE) based on sinusoids from Vaswani et al. (2017) is the most widely used for Transformer-like architectures. However, PE \( q(n) \) and \( K(n) \) in (5) can also be trained as in BERT (Devlin et al., 2019; Liu et al., 2019). Although the original Transformer only includes PE at the input layer, it may be included at all layers (Dehghani et al., 2019; Lan et al., 2020).

Relative positional encoding (RPE; Shaw et al., 2018) is a way to leverage relative positions. It came with a \( O(N^2D) \) space complexity, which was reduced to \( O(N^2) \) in Huang et al. (2018); He et al. (2020). Considering log-distances was proposed in Raffel et al. (2020). Several variants for RPE were introduced (Huang et al., 2020; Wang et al., 2021). They all apply learned RPE in the attention domain. Using fixed embedding functions was also considered for RPE (Pham et al., 2020), and masking RPE is used in Kim et al. (2020) to promote local attention.

Keys-domain vs attention domain. Doing PE in the keys domain introduces position-content cross terms that are advocated as noisy and not beneficial in Ke et al. (2020) and replaced by Unified PE, i.e. PE in the attention domain. This is also called disentangled attention in He et al. (2020) and already proposed in Tsai et al. (2019) through separable content-position attention kernels. All of these studies require the explicit computation and storage of \( A \).

Non-integer positions were considered for structured inputs. Tree-based PE was proposed both for APE (Shiv & Quirk, 2019; Xiao et al., 2019; Ma et al., 2019) and RPE (O mote et al., 2019). Positional encoding of robots within arbitrary polygons is found in Bose et al. (2019).

Dynamical models for PE. Attention for machine translation was introduced in Bahdanau et al. (2016), which was retrospectively understood in Ke et al. (2020) as using recurrent neural nets (RNN) for PE. In Chen et al. (2018), the hidden states of encoder RNNs are said to contain enough position information to skip explicit PE. Neishi & Yoshinaga (2019) builds on this view, but explicitly describes the idea for the first time. Their contribution is to replace the additive PE in (5) by an RNN. In the same vein, Liu et al. (2020) generates PE using (neural) ordinary differential equations.

Convolutional contexts. Our convSPE variant involves convolving random noise. First, this can be related to Mohamed et al. (2019), who use convolutional neural networks for queries and keys computation. Second, the connections between convolutions and stationary processes have recently been highlighted by Xu et al. (2020) as enforcing PE.

Multiplicative PE. Various levels of content-position interactions are formalized in (Tsai et al., 2019). Multiplicative strategies were proposed for both RPE (Huang et al., 2020) and APE (Dai et al., 2019). The latter was generalized in Tsai et al. (2019). All these require the explicit computation of the attention matrix. Wang et al. (2020a) presents a scheme that is close to our sinusoidal variant, but without the stochastic part that is the key to go from (14) to (15).

The limits of APE and RPE were highlighted by some authors. In Wang & Chen (2020), the best performing models exploit both absolute and relative positions. In Irie et al. (2019) and Tsai et al. (2019), it is found that removing APE altogether in the causal decoder part of Transformer-based architectures leads to comparable/better performance. It is also not clear which one is best between incorporating PE in the raw input signal (and hence propagating it through the value entries) or using it anew on the queries and keys only, as we do. Our choice is backed by Tsai et al. (2019).

5. Conclusion

We propose a new Stochastic Positional Encoding (SPE), based on filtering random noise. As we show, the procedure generalizes relative PE and is a principled means to enforce any prescribed (but trained) cross-covariance structure, which we demonstrated should be the central concern in dot-product attention. In our experiments, we show that SPE brings an interesting gain in performance for large-scale transformer models (Choromanski et al., 2020; Katharopoulos et al., 2020), as compared to classical (sinusoidal) PE. This was expected, because RPE (Shaw et al., 2018) is often advocated as beneficial. However, no way to incorporate it for long sequences was available so far and this is the core contribution of this paper. The natural future directions for our study are (i) Signal-dependent PE that incorporates the input sequence as an additional input for SPE, (ii) Non-stationary PE that utilizes both relative and absolute positions, (iii) Extending our approach to arbitrary attention kernels, e.g. defined implicitly through their (random) mappings as in (4). Indeed, SPE as it is presented here holds theoretically for dot-product attention kernels only, but our results given in Table 1 suggest that this generalizes, asking an interesting research question.

Acknowledgements

This work was supported by the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 765068 (MIPFrontiers) and in part by the French government under management of Agence Nationale de la Recherche as part of the “Investissements d’avenir” program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute).

We would like to thank Yi Tay, Mostafa Dehghani and Philip Pham for their help with troubleshooting the Long-Range Arena, and Krzysztof Choromanski for clarifications about the Performer.
Relative Positional Encoding for Transformers with Linear Complexity

References


Relative Positional Encoding for Transformers with Linear Complexity


Relative Positional Encoding for Transformers with Linear Complexity


Relative Positional Encoding for Transformers with Linear Complexity
Supplementary Material

Introduction
This document comprises additional information that could not be included in the paper due to space constraints. It is structured as follows. In appendix A, we provide some further theoretical developments. In appendix B, we detail the experimental setup on the Long Range Arena. In appendix C, we detail our music generation experiments. Finally, we provide additional results in appendix D.

Our source code is available at:

https://github.com/aliutkus/spe/

See also the companion website:

https://cifkao.github.io/spe/

A. Theory
A.1. Convolutional SPE Leads to Vanishing Attention

In the main document, we claim that the convolutional variant leads to vanishing attention. We shortly prove this claim here. For ease of notation, the proof is given in the 1D case, but extends trivially to higher dimensions. The core idea is illustrated in Figure 6. Convolutional SPE yields:

\[
\overline{Q}_d(m, r) = \sum_{p=0}^{P} Z_d(m - p, r) \phi_d^Q(p),
\]

\[
\overline{K}_d(n, r) = \sum_{p=0}^{P} Z_d(n - p, r) \phi_d^K(p),
\]

where \(Z_d\) is a white Gaussian noise process, i.e. \(\mathbb{E}[Z_d(m, r)Z_d(m', r)] = \delta_{nn'}\). Omitting the dependency on \(r\) for notational convenience (all realizations are independent), we can compute the positional attention as:

\[
\mathcal{P}_d(m, n) = \mathbb{E}[\overline{Q}_d(m)\overline{K}_d(n)]
\]

\[
= \mathbb{E}\left[\sum_{p,\tau=0}^{P} z_d(m - \tau)z_d(n - p)\phi_d^Q(\tau)\phi_d^K(p)\right]
\]

\[
= \mathbb{E}\left[\sum_{p=0}^{P} z_d(n - p)^2 \phi_d^Q(p + m - n)\phi_d^K(p)\right]
\]

\[
= \sum_{p=0}^{P} \phi_d^Q(p + m - n)\phi_d^K(p),
\]

where only the \((p, \tau)\) values such that \(n - p = m - \tau\) remain, all other cross terms \(\mathbb{E}[Z_d(m)Z_d(m' \neq m)]\) disappearing due to whiteness of \(Z_d\). Filters are taken as 0-valued outside of \([0 : P]\). As can be seen, whenever \(|m - n| > P\), we get \(\mathcal{P}_d(m, n) = 0\), because \(\phi_d^K(p + (m - n)) = 0\).

A.2. Complexity

In this section, we detail the additional complexity caused by the proposed SPE method.

- **Sinusoidal SPE** first requires the computation of the modulation matrices \(\Omega\) for each feature dimension \(d = 1 \ldots D\), which has a \(O(2NK)\) complexity. Then, this matrix must be multiplied by the noise matrix \(Z_d\) with shape \(2K \times R\), leading to an overall complexity of \(O(DRNK^2)\). Since \(K\) is typically very small in our experiments, SineSPE can be seen as quite light in terms of both time and space complexity.

- **Convolutional SPE** involves drawing a new noise signal \(z_{d:n}r\) of length \(N\) for each \(d\) and \(r\), and convolving it with the filters \(\phi_d^Q\) and \(\phi_d^K\), whose length is written \(P\). In the 1D case, this leads to an overall time complexity of \(O(DRN)\), which can be replaced by \(O(DRN \log N)\).
when operating the convolutions in the frequency domain, which is advantageous for long filters.

In higher dimensions, say 2D, this becomes $O(DRN_1N_2P_1P_2)$ in the original domain and $O(DRN_1N_2 \log N_1 \log N_2)$ in the frequency domain, where $(N_1, N_2)$ and $(P_1, P_2)$ are the shapes of noise and filters, respectively.

- The bottleneck of gating is the generation of random noise $e_d$, which has complexity $O(DR)$.

Note that this complexities of course must be multiplied by the number of heads considered, up to 8 in our experiments.

As can be seen, the complexities of the sinusoidal and convolutional variants are similar, depending on the length $P$ of the filters and the number $K$ of sinusoids.

Still, other aspects come into the play. First, the convolutional version requires generating noise whose size scales as $N$, while the sinusoidal version requires much smaller $2K$-large noise matrices. Second, only a very small number of sinusoids was required in our experiments, whereas the convolutional version required longer contexts, so that we often had $2K \ll P$ in practice. Finally, although this may change in the near future, deep learning frameworks like PyTorch do not easily integrate convolutions in the frequency domain.

Sample-wise noise sharing. In practice, SPEs do not need to be drawn anew for each example. The most straightforward trick to reduce memory and computational footprint of the method is to share $\bar{Q}$ and $\bar{K}$ among all examples in each mini-batch, as we do in all our experiments. This can bring significant memory savings when SPE is used as a drop-in addition to networks trained with large batch sizes.

B. Experimental Setup: Long-Range Arena

An overview of the Long-Range Arena (Tay et al., 2021) tasks is given in Table 2. We do not include Pathfinder-X (a synthetic image classification task) or its harder variant Pathfinder-G. In this paper as we were unable to reproduce the results of Tay et al. on this task. All the datasets are described in detail in Tay et al. and available from the official LRA repository.⁷

In all LRA experiments, we employ gated SPE with $R \in \{32, 64\}$. We consistently use $K = 10$ for sinusoidal (periodic) SPE and filters of length 128 for convolutional SPE. For convolutional SPE, we share $\bar{Q}$ and $\bar{K}$ across all layers (but not across attention heads); for sinusoidal SPE, $\bar{Q}$ and $\bar{K}$ are unique to each layer and head; in both cases, layer-specific gating is employed. Baseline experiments employ the same absolute positional encodings as Tay et al. (learnable APE for Image and sinusoidal APE for the remaining tasks). In models employing SPE, APE is removed.

The numbers of parameters of the models presented in the main document are shown in Table 3. We can see that SPE-based models have at most 3.1% more parameters than the baselines. In the Image column, the numbers for SPE-based models are about 50% lower due to the fact that the baselines on this task employ learnable APE.

We use code from the official LRA repository, including the authors’ Transformer implementation, modified as necessary to incorporate SPE. We keep the same training configuration as provided by the LRA authors, but decrease the batch sizes (from 256 to 96 for Image and from 32 to 8 for the rest) and learning rates so as to fit within 16 GB of GPU memory. Our modified code and configuration files are available in our source code repository.

B.1. Resource usage

The typical training times of the LRA models are displayed in Table 4. Note that the times may not be comparable across models or tasks due to evaluation (which may be time-consuming) being done more frequently in some runs than others.

The total training time was 1405 h (189 runs in total), out of which 273 h (61 runs) were spent on attempts to reproduce the results of Tay et al. (2021) using Performer-softmax, Linear Transformer-ReLU and vanilla Transformer. Some of these preliminary experiments were distributed over 1–3 Tesla V100 GPUs with 32 GB of memory each. The final models were all trained on a single Tesla V100 or P100 GPU with 16 GB of memory.

C. Experimental Setup: Music Generation

Our music Performers are implemented using the pytorch-fast-transformers package,⁸ modified as necessary to incorporate SPE. The modified code and configuration files are available in our code repository.

All models have 24 layers with model dimension 512, 8 attention heads and 2048 feed-forward units, which amount to ~80 million trainable parameters. In models that use SPE, $\bar{Q}$ and $\bar{K}$ are shared across all layers (but not across attention heads); layer-specific gating is employed for models trained with gated SPE.

The models are trained with the Adam optimizer. We schedule the learning rate with linear warmup, followed by cosine decay. Full details of hyperparameters can be found in the provided configuration files.

---

⁷https://github.com/google-research/long-range-arena

⁸https://github.com/idiap/fast-transformers
According to Hsiao et al. (2021), the piano performances
(measure). We leave 5% (87 songs) as the validation set.

C.1. Pop Piano Music Generation

Training data. The pop piano MIDI dataset we use is derived from the one provided in Hsiao et al. (2021), open-sourced on GitHub. It consists of 1,747 pure piano performances of various Japanese, Korean, and Western pop songs, amounting to a total duration of ~100 hours. All the songs are in 4/4 time signature, namely four beats per bar (measure). We leave 5% (87 songs) as the validation set.

According to Hsiao et al. (2021), the piano performances are originally collected from the Internet in the MP3 (audio) format. Hsiao et al. further employed Onsets and Frames piano transcription (Hawthorne et al., 2018), madmom beat tracking tool (Böck et al., 2016), and chorder rule-based chord detection to transcribe the audio into MIDI format with tempo, beat, and chord information.

Data representation. The representation adopted here is largely identical to the Revamped MIDI-derived (REMI) encoding by Huang & Yang (2020), except that an extended set of chord tokens (described below) is used. REMI encodes a piano piece into a sequence composed of two types, metrical and note, of tokens. The metrical tokens are:

- **bar**: Marks the start of a musical bar.
- **subbeat**: Marks the musical timing within a bar. A bar is divided into 16 subbeats, which is equivalent to 4 beats. This symbolic timing provides an explicit time grid for sequence models to model music.
- **tempo**: Determines the pace (in beats per minute, or bpm) at which the piece is played, varied per bar. The range of tempo tokens is [32, 224] bpm, in steps of 3 bpm for quantization.

The note tokens are:

- **pitch**: Marks a note played. The 88 pitch-es correspond to each key on the piano.
- **duration**: Denotes the length of a played note, ranging from 1/2 to 16 subbeats, in steps of 1/2 subbeat.
- **volume (or, velocity)**: Denotes how loud a note is played. A total of 24 volume levels are considered.
- **chord**: Marks a change on the accompanying chord. Each chord is described by its root note and quality, e.g., C-Maj7, E-min. A total of 133 distinct chord tokens are found in the dataset.

Please note that a single note played is represented by a co-occurring triple of (pitch, duration, volume). The aforementioned tokens constitute a vocabulary of size ~340 for our REMI encoding. On average, we need a sequence with 5300 tokens to represent a song.

Training and inference. In each training epoch, we randomly crop a segment of length 2048 from each sample, and shift the pitches of the entire segment by ~6 to 6 semitones randomly (this is called transposition in music) as data augmentation. We use batch size = 4, and set the learning rate to 0.0001 for APE and 0.0002 for all SPE models. For sineSPE, we choose the number of sines $K = 5$; for convSPE, the convolutional filter size is set to be 128, 512 for the gated and ungated variants respectively.

Detailed resource usage of each model is shown in Table 5. During inference, we employ nucleus sampling (Holtzman et al., 2019) with $p = 0.9$ and softmax temperature $t = 1.2$. No post-processing on enforcing the grammatical correctness of the generated sequence is done.

Validation loss of the models trained on this task is listed in Table 6. On this metric, our convSPE variant performs the

<table>
<thead>
<tr>
<th>Name</th>
<th>Dataset</th>
<th>Input</th>
<th>Length</th>
<th>Goal</th>
<th># classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListOps</td>
<td>ListOps</td>
<td>expression with operations on lists of digits</td>
<td>2k</td>
<td>evaluate expression</td>
<td>10</td>
</tr>
<tr>
<td>Text</td>
<td>IMDB</td>
<td>movie review as byte string</td>
<td>8k</td>
<td>classify sentiment</td>
<td>2</td>
</tr>
<tr>
<td>Retrieval</td>
<td>AAN</td>
<td>pair of articles as byte strings</td>
<td>2 x 4k</td>
<td>detect citation link</td>
<td>2</td>
</tr>
<tr>
<td>Image</td>
<td>CIFAR10</td>
<td>8-bit gray-scale 32 x 32 image as byte string</td>
<td>1k</td>
<td>recognize object</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 2. Long-Range Arena classification tasks used in this paper.

<table>
<thead>
<tr>
<th>Name</th>
<th>Dataset</th>
<th>Input</th>
<th>Length</th>
<th>Goal</th>
<th># classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListOps</td>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Retrieval</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Image</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Numbers of parameters of LRA models, identical for both Performer-softmax and Linear Transformer-ReLU.

[10] https://github.com/joshuachang2311/chorder
Table 4. Training times for LRA models (hours). Numbers in parentheses are from Tesla P100 GPUs, the rest from Tesla V100 GPUs.

<table>
<thead>
<tr>
<th>Model</th>
<th>ListOps</th>
<th>Text</th>
<th>Retrieval</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performer-softmax</td>
<td>1.1</td>
<td>4.8</td>
<td>1.2</td>
<td>4.8</td>
</tr>
<tr>
<td>Performer-softmax + sineSPE</td>
<td>4.2</td>
<td>11.7</td>
<td>2.9</td>
<td>5.0</td>
</tr>
<tr>
<td>Performer-softmax + convSPE</td>
<td>8.9</td>
<td>23.2</td>
<td>21.9</td>
<td>5.3</td>
</tr>
<tr>
<td>Linear Transformer-ReLU</td>
<td>0.6</td>
<td>(3.2)</td>
<td>0.7</td>
<td>4.8</td>
</tr>
<tr>
<td>Linear Transformer-ReLU + sineSPE</td>
<td>2.0</td>
<td>6.8</td>
<td>2.1</td>
<td>5.0</td>
</tr>
<tr>
<td>Linear Transformer-ReLU + convSPE</td>
<td>15.0</td>
<td>18.6</td>
<td>19.0</td>
<td>5.3</td>
</tr>
</tbody>
</table>

Table 5. Resource usage of models trained on pop piano music generation, on a Tesla V100 GPU with 32GB of memory. # of epochs and time to the checkpoint with the lowest validation loss are displayed. (ug: trained without SPE gating.)

<table>
<thead>
<tr>
<th>Model</th>
<th># epochs</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>APE</td>
<td>72</td>
<td>9.74 h</td>
<td>14.34 GB</td>
</tr>
<tr>
<td>sineSPE</td>
<td>78</td>
<td>17.92 h</td>
<td>29.80 GB</td>
</tr>
<tr>
<td>sineSPE (ug)</td>
<td>78</td>
<td>16.31 h</td>
<td>18.29 GB</td>
</tr>
<tr>
<td>convSPE</td>
<td>80</td>
<td>28.02 h</td>
<td>30.01 GB</td>
</tr>
<tr>
<td>convSPE (ug)</td>
<td>68</td>
<td>24.76 h</td>
<td>18.99 GB</td>
</tr>
</tbody>
</table>

Table 6. Validation cross-entropy for models trained for pop piano music generation (mean and standard deviation) over all sequences. (ug: trained without SPE gating). Trained: pos ≤ 2048, Extrapolation: 2048 < pos ≤ 3072.

<table>
<thead>
<tr>
<th>Positions</th>
<th>Trained</th>
<th>Extrapolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>APE</td>
<td>1.721 ± 0.148</td>
<td>3.215 ± 0.200</td>
</tr>
<tr>
<td>sineSPE</td>
<td>1.694 ± 0.148</td>
<td>2.396 ± 0.359</td>
</tr>
<tr>
<td>sineSPE (ug)</td>
<td>1.754 ± 0.146</td>
<td>1.965 ± 0.170</td>
</tr>
<tr>
<td>convSPE</td>
<td><strong>1.685 ± 0.151</strong></td>
<td>1.932 ± 0.225</td>
</tr>
<tr>
<td>convSPE (ug)</td>
<td>1.733 ± 0.145</td>
<td><strong>1.805 ± 0.163</strong></td>
</tr>
</tbody>
</table>

Table 5. Resource usage of models trained on pop piano music generation, on a Tesla V100 GPU with 32GB of memory. # of epochs and time to the checkpoint with the lowest validation loss are displayed. (ug: trained without SPE gating.)

<table>
<thead>
<tr>
<th>Model</th>
<th># epochs</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>APE</td>
<td>72</td>
<td>9.74 h</td>
<td>14.34 GB</td>
</tr>
<tr>
<td>sineSPE</td>
<td>78</td>
<td>17.92 h</td>
<td>29.80 GB</td>
</tr>
<tr>
<td>sineSPE (ug)</td>
<td>78</td>
<td>16.31 h</td>
<td>18.29 GB</td>
</tr>
<tr>
<td>convSPE</td>
<td>80</td>
<td>28.02 h</td>
<td>30.01 GB</td>
</tr>
<tr>
<td>convSPE (ug)</td>
<td>68</td>
<td>24.76 h</td>
<td>18.99 GB</td>
</tr>
</tbody>
</table>

Table 6. Validation cross-entropy for models trained for pop piano music generation (mean and standard deviation) over all sequences. (ug: trained without SPE gating). Trained: pos ≤ 2048, Extrapolation: 2048 < pos ≤ 3072.

<table>
<thead>
<tr>
<th>Positions</th>
<th>Trained</th>
<th>Extrapolation</th>
</tr>
</thead>
<tbody>
<tr>
<td>APE</td>
<td>1.721 ± 0.148</td>
<td>3.215 ± 0.200</td>
</tr>
<tr>
<td>sineSPE</td>
<td>1.694 ± 0.148</td>
<td>2.396 ± 0.359</td>
</tr>
<tr>
<td>sineSPE (ug)</td>
<td>1.754 ± 0.146</td>
<td>1.965 ± 0.170</td>
</tr>
<tr>
<td>convSPE</td>
<td><strong>1.685 ± 0.151</strong></td>
<td>1.932 ± 0.225</td>
</tr>
<tr>
<td>convSPE (ug)</td>
<td>1.733 ± 0.145</td>
<td><strong>1.805 ± 0.163</strong></td>
</tr>
</tbody>
</table>

Data representation. We use a representation similar to the one proposed by Cifka et al. (2020), but adapted to a multi-track (multi-instrument) setting. Specifically, we encode a piece of music as a sequence of the following types of event tokens, each with two integer arguments:

- **note_on**(track, pitch): Begins a new note at the given pitch (0–127).
- **note_off**(track, pitch): Ends the note at the given pitch (0–127).
- **time_shift**(beats, offset): Advances current time by a given number of beats and then sets the offset within the beat, given as the number of ticks from its beginning (0–11). Maximum possible shift is (2, 0).

The track numbers range from 1 to 3, where 1 is always bass and 2 is always drums. The vocabulary of the model then consists of 794 tokens (3 × 128 note-ons, 3 × 128 note-offs, 24 time shifts, and 2 beginning-/end-of-sequence markers).

The main differences to the representation described in Section C.1 are a more compact encoding of timing, no representation of musical dynamics (for simplicity), and support for multiple tracks (not originally proposed by Cifka et al., 2020 but introduced here inspired by Donahue et al., 2019).

Training and inference. During training, each example is pre-processed and encoded as described above and the resulting token sequence is truncated to a length of 512. We train each model for a total of 24 epochs.

At test time, we sample with a softmax temperature of 0.6. We disallow sampling tokens that would result in invalid sequences (i.e. spurious note-offs, backward time shifts) in order to ensure that the generated sequence can be correctly decoded.

best both within the trained positions and on extrapolation.

C.2. Groove Continuation

Training data. The Groove2Groove MIDI dataset consists of accompaniments generated by the Band-in-a-Box software (BIAB). We only use the training section of the Groove2Groove MIDI dataset and perform a custom training/validation/test split such that each section contains a unique set of BIAB styles (2,761 for training and 50 each for validation and testing). The code necessary to download, pre-process and split the dataset is included in the repository.

We convert each accompaniment to a trio consisting of bass, drums and another randomly selected accompaniment track (e.g. piano, guitar). We then perform random data augmentation by skipping measures at the beginning, dropping some of the instruments, and transposition (pitch-shifting by −5 to +5 semitones). All randomization is done anew in each epoch.

Data representation. We use a representation similar to the one proposed by Cifka et al. (2020), but adapted to a multi-track (multi-instrument) setting. Specifically, we encode a piece of music as a sequence of the following types of event tokens, each with two integer arguments:

- **note_on**(track, pitch): Begins a new note at the given pitch (0–127).
- **note_off**(track, pitch): Ends the note at the given pitch (0–127).
- **time_shift**(beats, offset): Advances current time by a given number of beats and then sets the offset within the beat, given as the number of ticks from its beginning (0–11). Maximum possible shift is (2, 0).

The track numbers range from 1 to 3, where 1 is always bass and 2 is always drums. The vocabulary of the model then consists of 794 tokens (3 × 128 note-ons, 3 × 128 note-offs, 24 time shifts, and 2 beginning-/end-of-sequence markers).

The main differences to the representation described in Section C.1 are a more compact encoding of timing, no representation of musical dynamics (for simplicity), and support for multiple tracks (not originally proposed by Cifka et al., 2020 but introduced here inspired by Donahue et al., 2019).

Training and inference. During training, each example is pre-processed and encoded as described above and the resulting token sequence is truncated to a length of 512. We train each model for a total of 24 epochs.

At test time, we sample with a softmax temperature of 0.6. We disallow sampling tokens that would result in invalid sequences (i.e. spurious note-offs, backward time shifts) in order to ensure that the generated sequence can be correctly decoded.
Various training details. Hyperparameter tuning was mostly performed in preliminary experiments (~100 runs); these were mostly done on other variants of the dataset and with different sequence lengths (ranging from 256 to 20 k); this includes experiments discarded due to bugs discovered during or after training. Learning rates between 0.0001 and 0.0008 and batch sizes between 1 and 24 were considered. For SPE, we considered both the gated and ungated variants with as many realizations as fit in memory (between 16 and 64). Model selection was based on validation loss and informal perceptual evaluation. Only a minimal attempt at further learning rate tuning was made for the final set of models with length 512, which did not appear to be particularly sensitive to it, and we chose to keep the initial learning rate 0.0004, which was found to perform well in all cases.

The models included in the main document – APE, sineSPE and convSPE – all use a batch size of 10 and finished training in about 3 h, 5 h and 6 h, respectively, using 9.7 GB, 14.4 GB and 14.8 GB of GPU memory. The total training time including all preliminary experiments was 852 hours.

Evaluation metrics. We use the objective metrics proposed by Cifka et al. (2019; 2020) to measure the style similarity between the generated continuation and the file from which the prompt was extracted. Given two pieces of music, each metric gathers musical event statistics of from which the prompt was extracted. Given two pieces similarity between the generated continuation and the file

where $a, b$ is a pair of notes and $\text{pitch}(\cdot)$ represents the pitch of a note as its MIDI note number (the number of semitones from $C_0$). The histogram has $24 \times 41$ bins (24 for time lags between 0 and 4 beats and 41 bins for intervals between $-20$ and 20 semitones).

In both cases, the 2D histograms are flattened to vectors before computing cosine similarities.

D. Additional Results

D.1. Attention Visualization: Music Generation

In this section, we display attention patterns produced by our pop piano music generation models.

Learned positional templates. We share the SPE modules across all layers of the Performer, but not across the attention heads, resulting in 512 learned positional kernels $P_d$ (number of heads $\times$ key dimensions per head). In Figure 7, we display 16 randomly picked resulting templates $P_d$ for both sineSPE and convSPE, trained with gating. Details of the two variants are:

- **sineSPE**: We set the number of sines $K = 5$.
- **convSPE**: We use filters of size 128.

In accordance with the definition, all of the visualizations are plotted with the equation $P_d = \mathbf{Q}_d \mathbf{K}_d^\top$, which we never need to explicitly compute for linear transformers. From Figure 7, we can observe that sineSPE learns to exploit a wide range of frequencies, and that convSPE is effective within small query-key offsets corresponding to the filter size, as expected.

Full Attention. Although the full attention matrix $A$ is not computed in linear transformers, we can still obtain it offline by multiplying queries and keys through either $A = \exp(QK^\top/\sqrt{D})$ (in the case of APE, where $D$ is the key dimensions per head), or $A = \exp(QK^\top/\sqrt{R})$ (in the case of SPEs); then apply row-wise softmax operation on $A$ as normalization.

Here, we present the (softmax-ed) attention matrices in the 1st, 3rd, 12th, 20th, and 24th (last) layers of all the five models trained on pop piano music generation in Figures 8–12. These are computed from one of each model’s random from-scratch music generations. To examine the models’ extrapolation ability, we let them generate a sequence of length 3072, while the training sequence length is only 2048. The attention matrices are lower-triangular due to causal masking. For better visualization, the color of each pixel is adjusted through $\min\{1, a_{mn}^{0.4}/0.02^{0.4}\}$ in the plots, where $a_{mn} \in [0, 1]$ is the softmax-ed attention score.

Figure 8 reveals a major drawback of APE: the attention of
Figure 7. Examples of $P_d$ learned by SPE. X- and Y-axes are key and query positions respectively. Max position = 2048.

Figure 8. Full attention matrices of APE (baseline). X- and Y-axes are key and query positions respectively. Max position = 3072.
Figure 9. Full attention matrices of \textsc{sineSPE (with gated SPE)}. Max token position = 3,072.

Figure 10. Full attention matrices of \textsc{sineSPE (without SPE gating)}. Max token position = 3,072.
Figure 11. Full attention matrices of convSPE (with SPE gating, conv filter size = 128). Max token position = 3072.

Figure 12. Full attention matrices of convSPE (without SPE gating, conv filter size = 512). Max token position = 3072.
tokens beyond position 2,048 (the training sequence length) seems to concentrate around 2,048 in earlier layers, rather than paying global or local attention. Such behavior is not seen in any of our SPE models. This potentially explains APE’s poor generalization to long sequences suggested by the stark increase in validation loss after position 2,048 (see Figure 3 in the main paper, and Table 6 here).

Next, comparing Figures 9 and 10, it is obvious that gated SPE gives the model the freedom to switch off PE in some heads to achieve global attention (see Figure 9), whereas the attention of ungated sineSPE (Figure 10) largely stays periodic, which might not be always desirable. The same can be said for convSPE (Figures 11 and 12). The gated convSPE is able to look much further back in the middle layers than its ungated counterpart.

D.2. Attention Visualization: CIFAR10

Figure 13 displays attention maps extracted from models trained on the LRA CIFAR10 task. Note that these are one-layer networks, and classification is done by prepending a special CLS token to the sequence of pixel values and using the output at this first position as input to a feed-forward classifier. Consequently, only the attention map at this single position (which is the one we display here) matters. (The model is therefore de facto not using self-attention, but rather attention with a single query and many keys. This removes the distinction between relative and absolute positions, which might explain why trainable APE performs better than SPE on this task.)

D.3. Evaluation of Desired PE Properties

We employ identical word probing and the associated metrics introduced in Wang et al. (2021) to compare the translation invariance and monotonicity properties of APE and our SPEs. The other properties mentioned in that work, namely symmetry and direction balance, are not evaluated here since the attention is uni-directional in our case. The models are also trained on pop piano music generation.

The metrics are calculated from attention matrices of each head in the 1st layer, averaged over all possible identical-token sequences (i.e., a sequence composed of repeated, same tokens; there are \( \sim 340 \) of them for our REMI vocabulary). To eliminate the impact of applying row-wise softmax with causal masking on the translation invariance property, we compute the metrics on the unnormalized attention matrices, i.e., \( A = \exp(QK^\top/\sqrt{D}) \) for APE, and \( A = \exp(QK^\top/(\sqrt{R}) \) for SPEs. Various combinations of query positions and query-key offsets are considered to examine whether the PE properties stay consistent when we extrapolate to longer sequences, as well as to look into their behavior in local and long-range attention spans.

We report the scores of the best-performing (i.e., lowest-scoring) head of each model in Table 7. From the table, we can notice that the PE properties of APE often deteriorate drastically in cases of extrapolation. On the contrary, the scores of ungated SPE models, i.e., models in which we enforce the incorporation of positional information in every layer, remain remarkably consistent throughout the positions. The evaluation here provides additional evidence for the extrapolation capability of SPEs.

D.4. Impact of the Number \( R \) of Realizations

In the main document, we discussed how SPE asymptotically leads to the desired cross-covariance structure as \( R \) grows to infinity. In this section, we empirically study how performance is affected by that parameter in practice. A first thing to highlight is that each training batch yields a new set of realizations for the noise \( Z_r \), so that the network sees the right attention pattern on average.

However, we may wonder whether how the number of realizations \( R \) impacts training and test performance. One can indeed notice that \( R \) may totally be set differently during training and inference, since it has no impact on the shape of the actual parameters/structure of the model. For this reason, we performed an ablation study where we use different values for \( R_{\text{train}} \) at training time, resulting in a trained model, and then evaluate its performance using a possibly different value \( R_{\text{test}} \). The results are displayed in Figure 14.

We can notice that the result achieved with \( R_{\text{test}} = R_{\text{train}} \) (highlighted in bold) is consistently close to the best result for the same \( R_{\text{train}} \), and conversely, choosing \( R_{\text{test}} \neq R_{\text{train}} \) often leads to a poor result. In other words, training and testing with the same \( R \) appears to be favorable for consistently good performance.

Another remarkable fact is that a higher \( R \) does not seem to imply better performance, even when \( R_{\text{test}} = R_{\text{train}} \). On the contrary, convSPE achieved by far the highest accuracy with \( R = 4 \). This unexpected result seems contradictory to the fact that it means noisier attention patterns. Further investigation is required to explain this phenomenon, but we conjecture that this additional noise in the attention patterns leads to increased robustness of the trained model, helping generalization.
Figure 13. CIFAR10 attention maps for 3 variants of Linear Transformer-ReLU: learnable APE (top), sineSPE (middle), and convSPE (bottom). Each row displays the input image, followed by attention weights of the 8 respective heads for each pixel, with the special CLS token as the query.

<table>
<thead>
<tr>
<th>Query positions</th>
<th>0 &lt; pos ≤ 1024</th>
<th>1024 &lt; pos ≤ 2048</th>
<th>2048 &lt; pos ≤ 2560 (extrapolation)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 &lt; pos ≤ 512</td>
<td>&lt; 1024</td>
<td>&lt; 2048</td>
</tr>
<tr>
<td></td>
<td>1024 &lt; pos ≤ 2048</td>
<td>&lt; 1024</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2048 &lt; pos ≤ 2560 (extrapolation)</td>
<td>&lt; 2048</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt; 1024</td>
<td>&lt; 2048</td>
<td>&lt; 2560</td>
</tr>
<tr>
<td></td>
<td>&lt; 2048</td>
<td>&lt; 2560</td>
<td></td>
</tr>
<tr>
<td></td>
<td>&lt; 2560</td>
<td></td>
<td></td>
</tr>
<tr>
<td>APE</td>
<td>T: 0.4335</td>
<td>T: 0.1845</td>
<td>T: 0.6953</td>
</tr>
<tr>
<td></td>
<td>M: 0.0152</td>
<td>T: 0.0625</td>
<td>M: 0.0616</td>
</tr>
<tr>
<td></td>
<td>T: 0.9142</td>
<td>T: 0.0413</td>
<td>M: 0.0113</td>
</tr>
<tr>
<td></td>
<td>M: 0.3974</td>
<td>M: 0.2429</td>
<td>M: 0.1637</td>
</tr>
<tr>
<td>sineSPE</td>
<td>T: 0.2063</td>
<td>T: 0.1337</td>
<td>T: 0.2504</td>
</tr>
<tr>
<td></td>
<td>M: 0.2893</td>
<td>M: 0.4060</td>
<td>M: 0.4167</td>
</tr>
<tr>
<td></td>
<td>T: 0.2167</td>
<td>T: 0.3599</td>
<td>M: 0.4147</td>
</tr>
<tr>
<td>sineSPE (ug)</td>
<td>T: 0.0141</td>
<td>T: 0.0231</td>
<td>T: 0.0206</td>
</tr>
<tr>
<td></td>
<td>M: 0.6295</td>
<td>M: 0.1582</td>
<td>M: 0.1623</td>
</tr>
<tr>
<td></td>
<td>T: 0.0105</td>
<td>T: 0.0196</td>
<td>M: 0.0163</td>
</tr>
<tr>
<td></td>
<td>M: 0.6189</td>
<td>M: 0.1609</td>
<td>M: 0.0994</td>
</tr>
<tr>
<td>convSPE</td>
<td>T: 0.3422</td>
<td>T: 0.3209</td>
<td>T: 0.6239</td>
</tr>
<tr>
<td></td>
<td>M: 0.1781</td>
<td>M: 0.3078</td>
<td>M: 0.7648</td>
</tr>
<tr>
<td></td>
<td>M: 0.2242</td>
<td>M: 0.4060</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M: 0.2189</td>
<td>M: 0.4283</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M: 0.1735</td>
<td>M: 0.4063</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M: 0.3624</td>
<td>M: 0.4167</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M: 0.4192</td>
<td>M: 0.4147</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M: 0.1486</td>
<td>M: 0.3247</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M: 0.3247</td>
<td>M: 0.7025</td>
<td></td>
</tr>
<tr>
<td>convSPE (ug)</td>
<td>T: 0.2828</td>
<td>T: 0.0107</td>
<td>T: 0.3334</td>
</tr>
<tr>
<td></td>
<td>M: 0.1234</td>
<td>M: 0.0249</td>
<td>M: 0.1505</td>
</tr>
<tr>
<td></td>
<td>M: 0.0620</td>
<td>M: 0.1254</td>
<td>M: 0.1254</td>
</tr>
<tr>
<td></td>
<td>M: 0.0253</td>
<td>M: 0.1342</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M: 0.0217</td>
<td>M: 0.0989</td>
<td></td>
</tr>
</tbody>
</table>

Figure 14. Accuracy of Performer-softmax with SPE on the LRA Text task, with different numbers of realizations $R$ during training/testing. Each value is the result of a single run. Highlighted in bold are values obtained with $R_{test} = R_{train}$. Higher (brighter) is better.