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

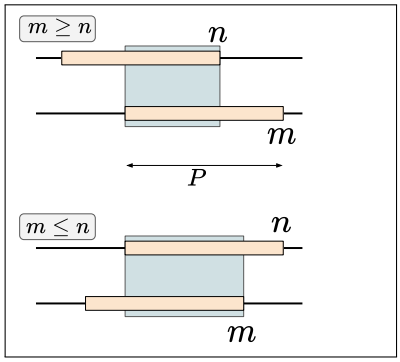


Figure 1. If $\mathbf{\Phi}_d^Q$ and $\mathbf{\Phi}_d^K$ have length P, $\overline{Q}_d$ and $\overline{K}_d$ for convolutional SPE depend on the noise Z_d over the intervals $[m-P:m]$ and $[n-P:n]$, respectively. Their correlation depends only on the shaded area, due to whiteness of Z_d. Whenever $|m-n|>P$, the two signals are uncorrelated.

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 1. 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. $\mathbb{E}[Z_d(m,r)Z_d(m',r)] = \delta_{mm'}$. Omitting the dependency on r for notational convenience (all realizations are independent), we can compute the positional attention as:

$$\begin{aligned}
\mathcal{P}_d(m,n) &= \mathbb{E}\left[\overline{Q}_d(m)\overline{K}_d(n)\right] \\
&= \mathbb{E}\left[\sum_{p,\tau=0}^{P} z_d(m-\tau)z_d(n-p)\phi_d^Q(\tau)\phi_d^K(p)\right] \\
&= \mathbb{E}\left[\sum_{p=0}^{P} z_d(n-p)^2\phi_d^Q(p+m-n)\phi_d^K(p)\right] \\
&= \sum_{p=0}^{P} \phi_d^Q(p+m-n)\phi_d^K(p),
\end{aligned}$$

where only the (p,τ) 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 $\mathbf{\Omega}$ for each feature dimension $d = 1\dots D$, which has a $\mathcal{O}(2NK)$ complexity. Then, this matrix must be multiplied by the noise matrix $\mathbf{Z}_d$ with shape $2K \times R$, leading to an overall complexity of $\boldsymbol{\mathcal{O}(DRNK^2)}$. 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 $\mathbf{z}_{d,:,r}$ of length N for each d and r, and convolving it with the filters $\boldsymbol{\phi}_d^Q$ and $\boldsymbol{\phi}_d^K$, whose length is written P.
  In the 1D case, this leads to an overall time complexity of $\boldsymbol{\mathcal{O}(DRNP)}$, which can be replaced by $\mathcal{O}(DRN\log N)$

when operating the convolutions in the frequency domain, which is advantageous for long filters.
In higher dimensions, say 2D, this becomes $\mathcal{O}(DRN_1N_2P_1P_2)$ in the original domain and $\mathcal{O}(DRN_1N_2 \log N_1 \log N_2)$ 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 $\boldsymbol{\epsilon}_d$, which has complexity $\mathcal{O}(DR)$.

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 $\overline{\mathbf{Q}}$ and $\overline{\mathbf{K}}$ 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 1. We do not include *Pathfinder* (a synthetic image classification task) or its harder variant *Pathfinder-X* 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.[1]

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 $\overline{\mathbf{Q}}$ and $\overline{\mathbf{K}}$ across all layers (but not across attention heads); for sinusoidal SPE, $\overline{\mathbf{Q}}$ and $\overline{\mathbf{K}}$ 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 2. 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 3. 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 1 405 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,[2] 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 2 048 feed-forward units, which amount to ∼80 million trainable parameters. In models that use SPE, $\overline{\mathbf{Q}}$ and $\overline{\mathbf{K}}$ 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.

[1] https://github.com/google-research/long-range-arena

[2] https://github.com/idiap/fast-transformers

Table 1. Long-Range Arena classification tasks used in this paper.

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

Table 2. Numbers of parameters of LRA models, identical for both Performer-softmax and Linear Transformer-ReLU.

	ListOps	Text	Retrieval	Image
Baseline (APE)	19 982 858	3 486 722	1 087 618	248 458
+ `sineSPE`	20 078 090	3 518 466	1 103 490	119 242
+ `convSPE`	20 117 002	3 553 282	1 120 898	133 706

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.[3] 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[4] 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 5 300 tokens to represent a song.

Training and inference. In each training epoch, we randomly crop a segment of length 2 048 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 4.

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 5. On this metric, our `convSPE` variant performs the

[3] https://github.com/YatingMusic/compound-word-transformer

[4] https://github.com/joshuachang2311/chorder

Table 3. Training times for LRA models (hours). Numbers in parentheses are from Tesla P100 GPUs, the rest from Tesla V100 GPUs.

	ListOps	Text	Retrieval	Image
Performer-softmax	1.1	4.8	1.2	4.8
Performer-softmax + `sineSPE`	(4.2)	11.7	2.9	5.0
Performer-softmax + `convSPE`	8.9	23.2	21.9	5.3
Linear Transformer-ReLU	0.6	(3.2)	0.7	4.8
Linear Transformer-ReLU + `sineSPE`	2.0	6.8	2.1	5.0
Linear Transformer-ReLU + `convSPE`	15.0	18.6	19.0	5.3

Table 4. 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.)

	# epochs	Time	Memory
`APE`	72	9.74 h	14.34 GB
`sineSPE`	78	17.92 h	29.80 GB
`sineSPE (ug)`	78	16.31 h	18.29 GB
`convSPE`	80	28.02 h	30.01 GB
`convSPE (ug)`	68	24.76 h	18.99 GB

Table 5. 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 $\leq$ 2 048, *Extrapolation*: 2 048 $<$ pos $\leq$ 3 072.

Positions	*Trained*	*Extrapolation*
`APE`	1.721 $\pm$ 0.148	3.215 $\pm$ 0.200
`sineSPE`	1.694 $\pm$ 0.148	2.396 $\pm$ 0.359
`sineSPE (ug)`	1.754 $\pm$ 0.146	1.965 $\pm$ 0.170
`convSPE`	**1.685** $\pm$ 0.151	1.932 $\pm$ 0.225
`convSPE (ug)`	1.733 $\pm$ 0.145	**1.805** $\pm$ 0.163

best both within the trained positions and on extrapolation.

C.2. Groove Continuation

Training data. The Groove2Groove MIDI dataset[5] consists of accompaniments generated by the Band-in-a-Box software (BIAB).[6] 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 Cífka 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 Cífka 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.

[5] http://doi.org/10.5281/zenodo.3958000
[6] https://www.pgmusic.com/

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 Cífka 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 the two pieces in histograms called *style profiles*, and then computes the cosine similarity between them.

The two metrics used here, *onset-duration* and *time-pitch*, differ in what kind of events they use to construct the style profile:

- The **onset-duration** profile is defined as a 2D histogram relating note onset positions to note durations. More precisely, for all notes a in a piece of music, it records a tuple of the form

  $$(\text{start}(a) \bmod 4,\ \text{end}(a) - \text{start}(a)) \in [0, 4) \times [0, 2),$$

  where $\text{start}(a)$ and $\text{end}(a)$ refer to the onset and offset time of a in beats. The expression $\text{start}(a) \bmod 4$ then represents the position of the note onset relative to the current bar, since all examples in the dataset are in a 4-beat meter. These tuples are gathered in 24×12 histogram bins (24 for onset time and 12 for duration).
- The **time-pitch** profile is also obtained as a 2D histogram, this time capturing time differences and pitch differences (intervals) between notes. The tuples it considers have the form

  $$(\text{start}(b) - \text{start}(a),\ \text{pitch}(b) - \text{pitch}(a)) \in [0, 4) \times \{-20, -19, \ldots, 20\},\ a \neq b,$$

  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_{-1}). The histogram has 24×41 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 $\mathcal{P}_d$) (*number of heads* $\times$ *key dimensions per head*. In Figure 2, we display 16 randomly picked resulting templates $\mathbf{P}_d$ for both `sineSPE` and `convSPE`, trained with gating. Details of the two variants are:

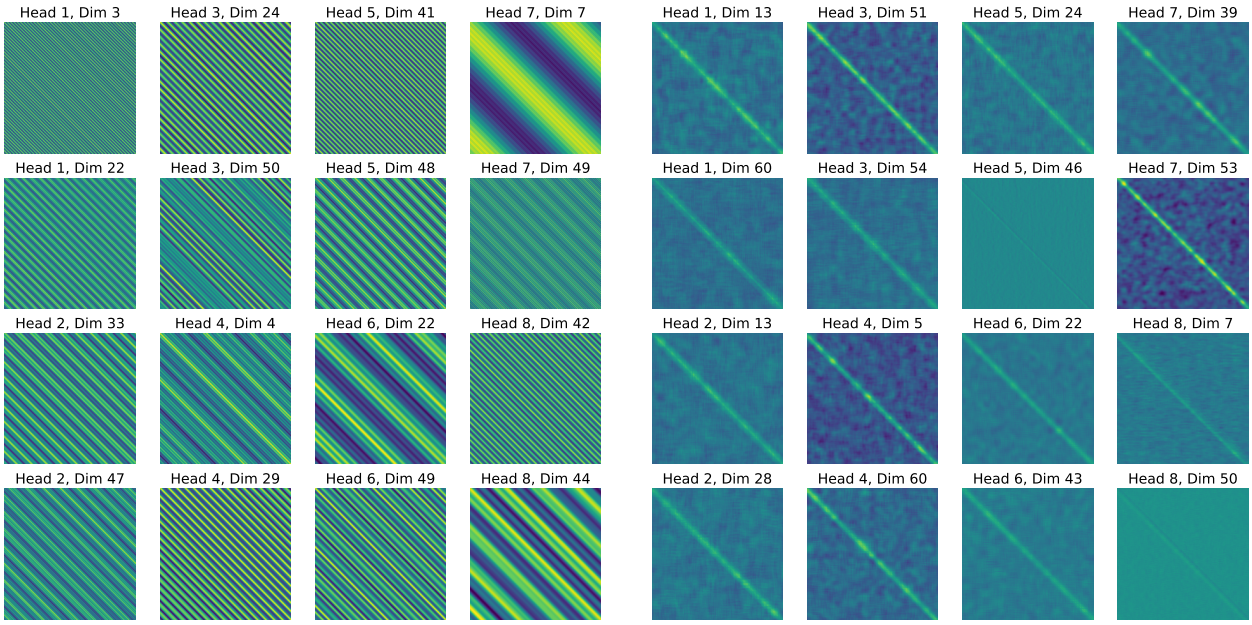
- `sineSPE`: We set the number of sines $K = 5$.
- `convSPE`: We use filters of size 128.

In accordance with the definition, all of the visualizations are plotted with the equation $\mathbf{P}_d = \overline{\mathbf{Q}}_d \overline{\mathbf{K}}_d^\top$, which we never need to explicitly compute for linear transformers. From Figure 2, 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 $\mathbf{A}$ is not computed in linear transformers, we can still obtain it *offline* by multiplying queries and keys through either $\mathbf{A} = \exp(\mathbf{Q}\mathbf{K}^\top / \sqrt{D})$ (in the case of APE, where D is the key dimensions per head), or $\mathbf{A} = \exp(\widehat{\mathbf{Q}}\widehat{\mathbf{K}}^\top / \sqrt{R})$ (in the case of SPEs); then apply row-wise softmax operation on $\mathbf{A}$ 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 3–7. 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 $3\,072$, while the training sequence length is only $2\,048$. 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 3 reveals a major drawback of `APE`: the attention of



(a) `sineSPE` (b) `convSPE`

Figure 2. Examples of $\mathbf{P}_d$ learned by SPE. X- and Y-axes are *key* and *query* positions respectively. Max position $= 2\,048$.

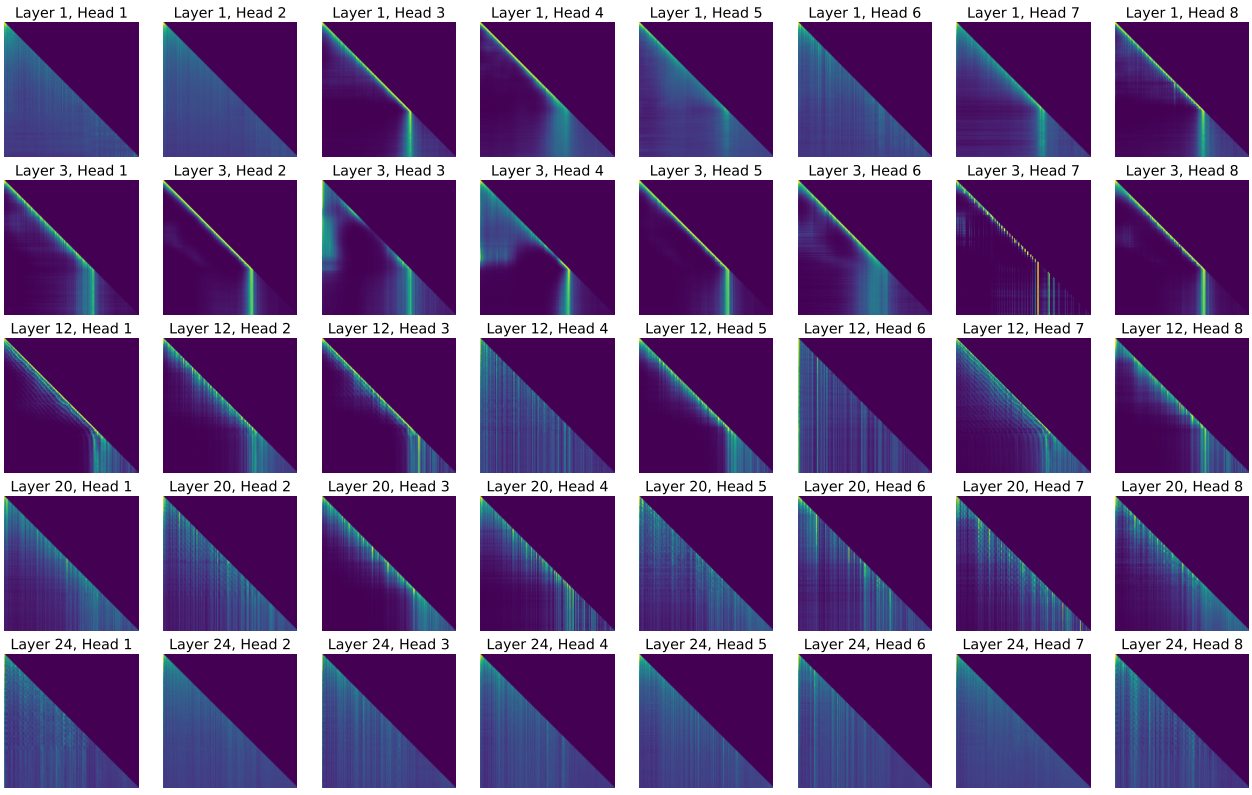


Figure 3. Full attention matrices of `APE` (baseline). X- and Y-axes are *key* and *query* positions respectively. Max position $= 3\,072$.

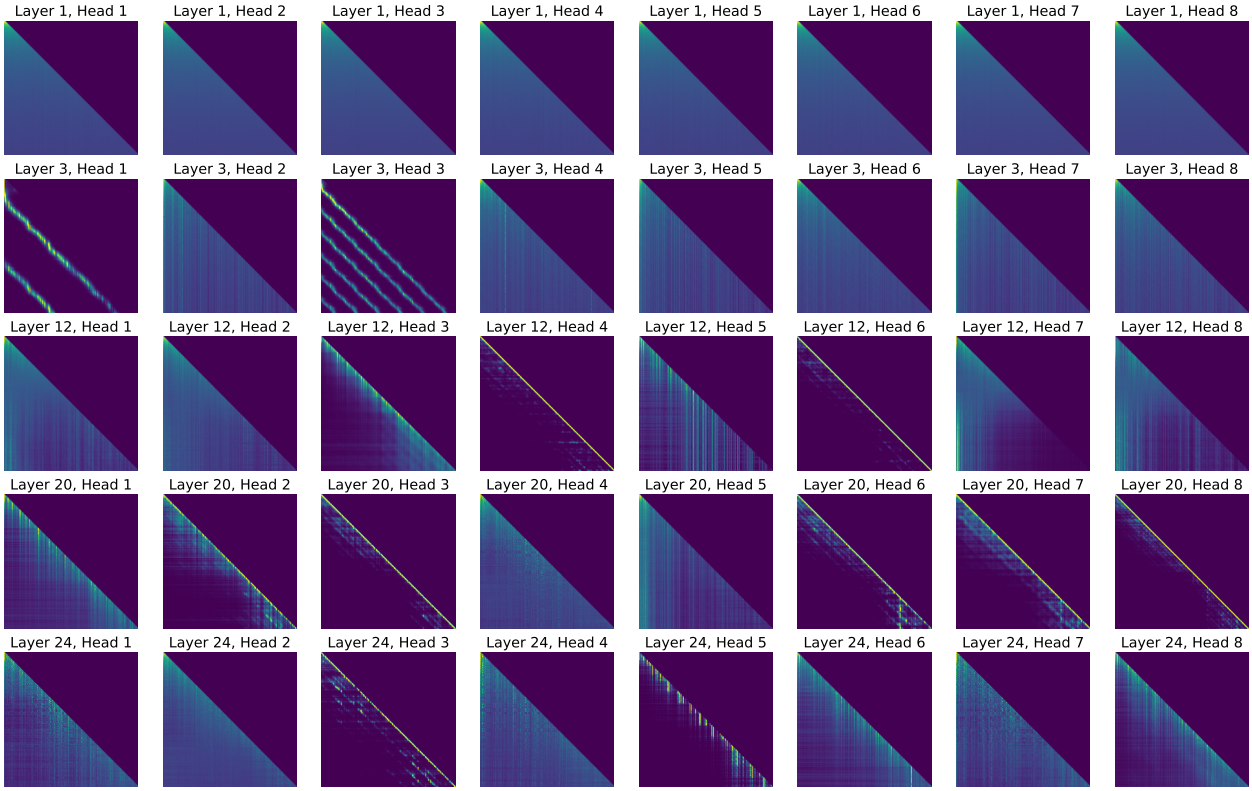


Figure 4. Full attention matrices of `sineSPE` (*with* gated SPE). Max token position $= 3\,072$.

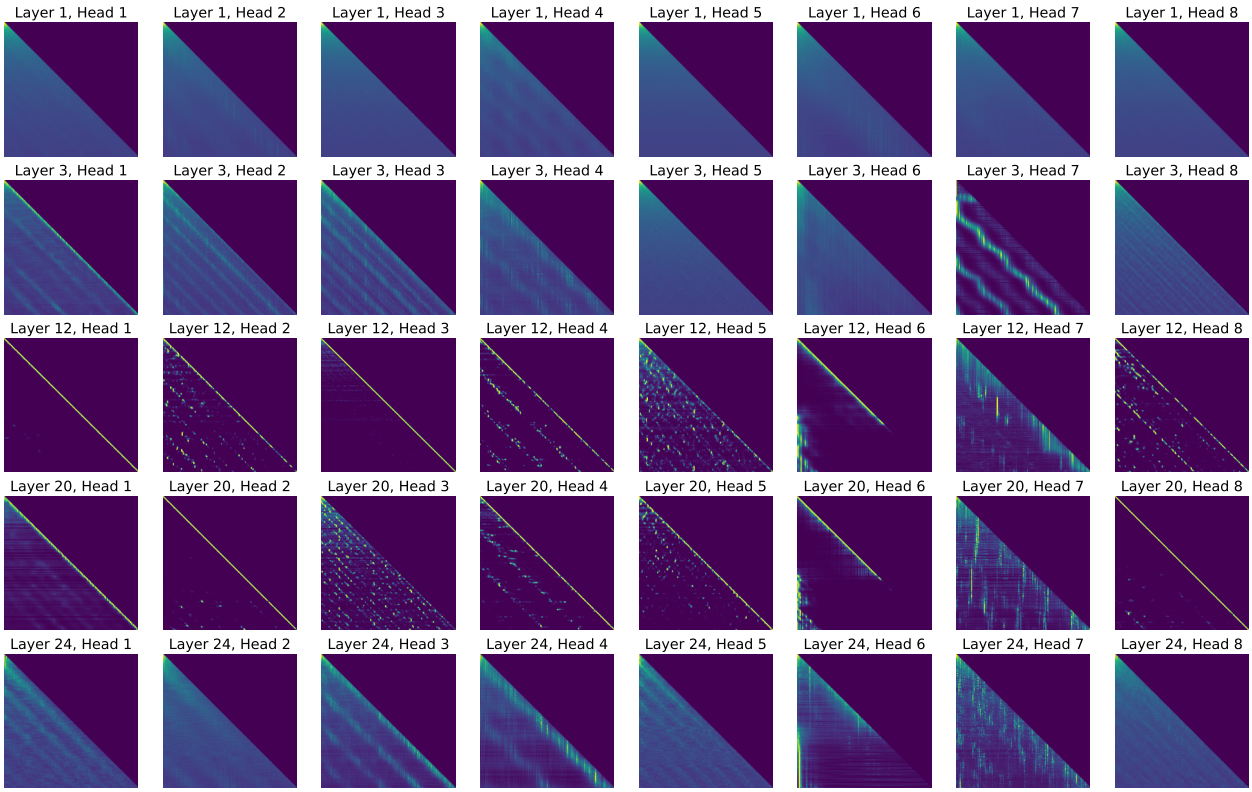


Figure 5. Full attention matrices of `sineSPE` (*without* SPE gating). Max token position $= 3\,072$.

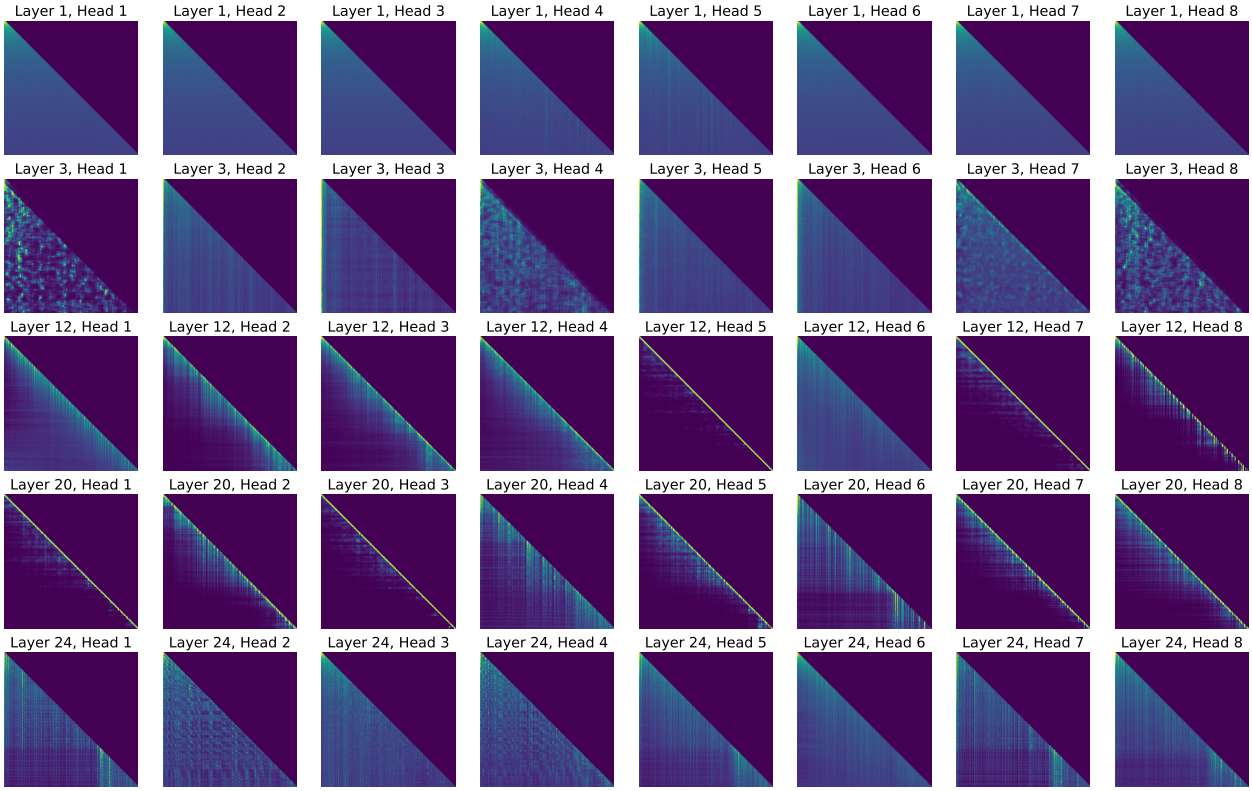


Figure 6. Full attention matrices of `convSPE` (*with* SPE gating, conv filter size $= 128$). Max token position $= 3\,072$.

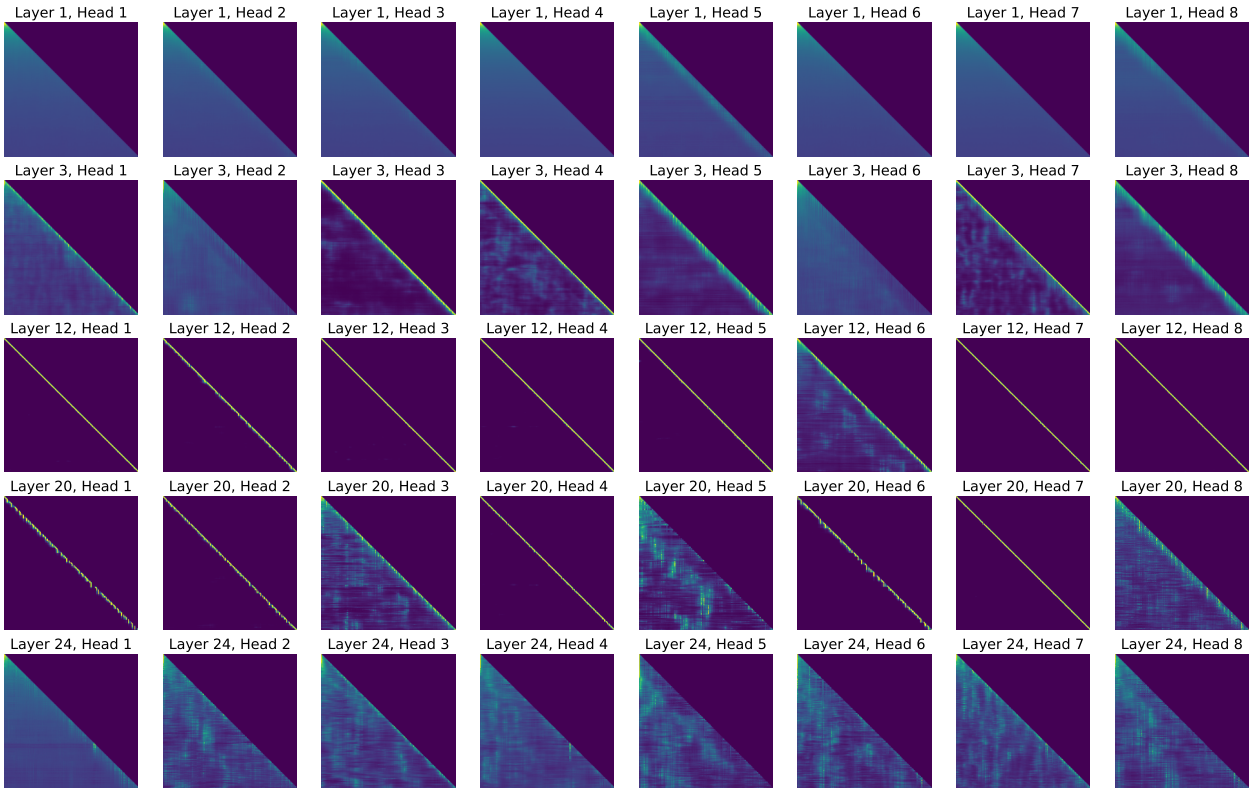


Figure 7. Full attention matrices of `convSPE` (*without* SPE gating, conv filter size $= 512$). Max token position $= 3\,072$.

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 5 here).

Next, comparing Figures 4 and 5, it is obvious that gated SPE gives the model the freedom to *switch off* PE in some heads to achieve global attention (see Figure 4), whereas the attention of ungated `sineSPE` (Figure 5) largely stays periodic, which might not be always desirable. The same can be said for `convSPE` (Figures 6 and 7). The gated `convSPE` is able to look much further back in the middle layers than its ungated counterpart.

D.2. Attention Visualization: CIFAR10

Figure 8 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 ∼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., $\mathbf{A} = \exp(\mathbf{Q}\mathbf{K}^\top/\sqrt{D})$ for APE, and $\mathbf{A} = \exp(\widehat{\mathbf{Q}}\widehat{\mathbf{K}}^\top/\sqrt{R})$ 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 6. 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 $\mathbf{Z}_d$, 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_{train} at training time, resulting in a trained model, and then evaluate its performance using a possibly different value R_{test}. The results are displayed in Figure 9.

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_{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.

References

Böck, S., Korzeniowski, F., Schlüter, J., Krebs, F., and Widmer, G. Madmom: A new Python audio and music signal processing library. In *Proc. ACM International Multimedia Conf.*, pp. 1174–1178, 2016.

Cífka, O., Şimşekli, U., and Richard, G. Supervised symbolic music style translation using synthetic data. In

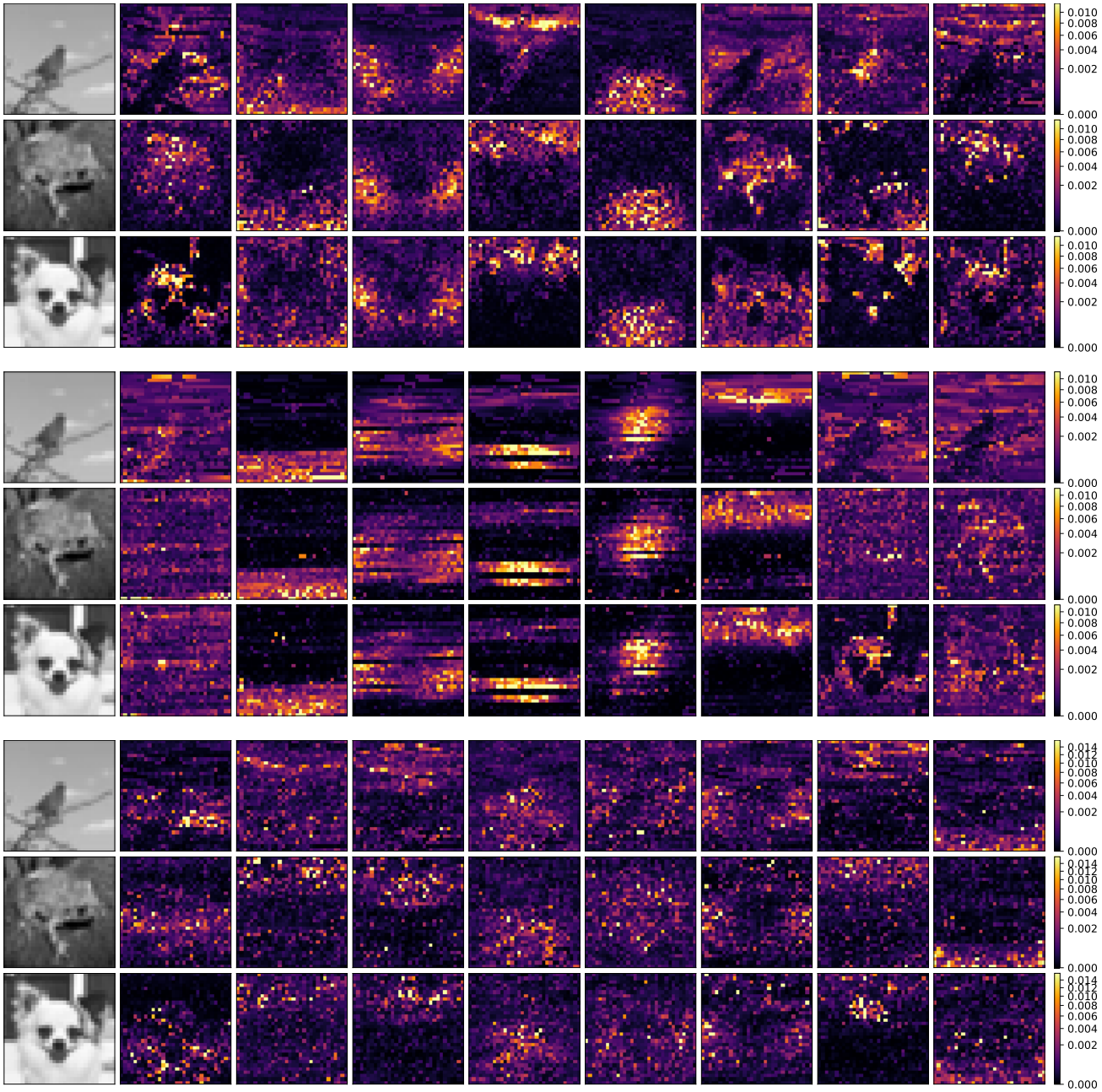
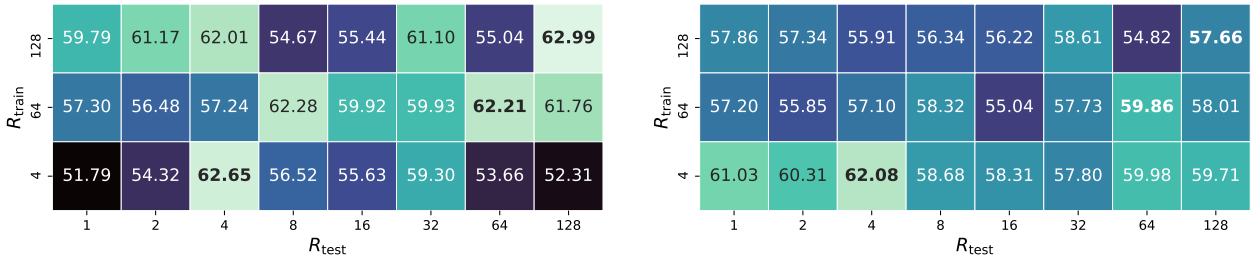


Figure 8. CIFAR10 attention maps for 3 variants of Linear Transformer-ReLU: learnable APE (top), `sineSPE` (middle), and `convSPE` (bottom). Each row displays the input image, followed by attention weights of the 8 respective heads for each pixel, with the special `CLS` token as the query.

Table 6. Evaluation of PEs metrics. `T:` translation invariance, `M:` monotonicity (lower is better). `ug`: models trained without SPE gating.

Query positions	**0 < pos ≤ 1 024**			**1 024 < pos ≤ 2 048**			**2 048 < pos ≤ 2 560** (extrapolation)		
Query-key offset	<128	<512	<1 024	<128	<512	<1 024	<128	<512	<1 024
APE	T: 0.4335 M: **0.0152**	T: 0.2063 M: 0.0625	T: 0.1845 M: **0.0616**	T: 0.9142 M: **0.0193**	T: 0.6953 M: 0.0413	T: 0.6458 M: **0.0713**	T: 0.9599 M: 0.3974	T: 0.8959 M: 0.2429	T: 0.5886 M: 0.1637
sineSPE	T: 0.1660 M: 0.2893	T: 0.3078 M: 0.4406	T: 0.3527 M: 0.4283	T: 0.1337 M: 0.2826	T: 0.2504 M: 0.4063	T: 0.3228 M: 0.4167	T: 0.2167 M: 0.3253	T: 0.3599 M: 0.4060	T: 0.4147 M: 0.3913
sineSPE (ug)	T: **0.0141** M: 0.6295	T: 0.0242 M: 0.1844	T: 0.0231 M: 0.1582	T: **0.0135** M: 0.6238	T: 0.0206 M: 0.1623	T: 0.0190 M: 0.1061	T: **0.0105** M: 0.6189	T: 0.0196 M: 0.1609	T: 0.0163 M: 0.0994
convSPE	T: 0.3422 M: 0.1781	T: 0.5637 M: 0.2242	T: 0.6389 M: 0.2189	T: 0.3209 M: 0.1735	T: 0.6239 M: 0.3624	T: 0.7648 M: 0.4192	T: 0.3462 M: 0.1486	T: 0.6135 M: 0.3247	T: 0.7025 M: 0.2740
convSPE (ug)	T: 0.2828 M: 0.1234	T: **0.0192** M: **0.0249**	T: **0.0107** M: 0.0620	T: 0.3334 M: 0.1505	T: **0.0188** M: **0.0253**	T: **0.0109** M: 0.1254	T: 0.2207 M: **0.1342**	T: **0.0171** M: **0.0217**	T: **0.0106** M: **0.0989**



(a) `sineSPE` (b) `convSPE`

Figure 9. 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 $R_{\text{test}} = R_{\text{train}}$. Higher (brighter) is better.

Proc. International Society for Music Information Retrieval Conf., pp. 588–595, 2019. doi: 10.5281/zenodo.3527878. URL https://doi.org/10.5281/zenodo.3527878.

Cífka, O., Şimşekli, U., and Richard, G. Groove2Groove: One-shot music style transfer with supervision from synthetic data. *IEEE/ACM Transactions on Audio, Speech and Language Processing*, 28:2638–2650, 2020. doi: 10.1109/TASLP.2020.3019642. URL https://hal.archives-ouvertes.fr/hal-02923548.

Donahue, C., Mao, H. H., Li, Y. E., Cottrell, G. W., and McAuley, J. Lakhnes: Improving multi-instrumental music generation with cross-domain pre-training. In *ISMIR*, 2019.

Hawthorne, C., Elsen, E., Song, J., Roberts, A., Simon, I., Raffel, C., Engel, J., Oore, S., and Eck, D. Onsets and Frames: Dual-objective piano transcription. In *Proc. Int. Society for Music Information Retrieval Conf.*, 2018.

Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. The curious case of neural text degeneration. In *Proc. International Conference on Learning Representations*, 2019.

Hsiao, W.-Y., Liu, J.-Y., Yeh, Y.-C., and Yang, Y.-H. Compound Word Transformer: Learning to compose full-song music over dynamic directed hypergraphs. In *Proc. AAAI Conf. Artificial Intelligence*, 2021.

Huang, Y.-S. and Yang, Y.-H. Pop Music Transformer: Generating music with rhythm and harmony. In *Proc. ACM International Multimedia Conf.*, 2020.

Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., and Metzler, D. Long Range Arena: A benchmark for efficient Transformers. In *Proc. Int. Conf. Learning Representations*, 2021. URL https://openreview.net/forum?id=qVyeW-grC2k.

Wang, B., Shang, L., Lioma, C., Jiang, X., Yang, H., Liu, Q., and Simonsen, J. G. On position embeddings in BERT. In *Proc. Int. Conf. Learning Representations*, 2021. URL https://openreview.net/forum?id=onxoVA9FxMw.