Supplementary Material for SKTformer: An Efficient Skeleton Transformer for Long Sequence Data

A ALGORITHMS

Algorithm 1 Skeleton Attention

```
class Skeleton_Attention(nn.Module):
 def __init__(self, num_head = 2, head_dim = 32,seq_len, left_rank = 8,right_rank = 8,
      dropout = 0.1):
    super(Skeleton_Attention, self).__init__()
    self.num_head = num_head
    self.head_dim = head_dim
    self.seq_len = seq_len
    self.left_rank = left_rank
    self.right_rank = right_rank

    self.ln_1 = nn.LayerNorm(self.num_head * self.head_dim)
    self.ln_2 = nn.LayerNorm(self.num_head * self.head_dim)

    self.drop_attn = torch.nn.Dropout(p=dropout)

    self.index_set_right = torch.randperm(self.head_dim)
    self.index_set_right = self.index_set_right[:self.right_rank]

    self.index_set_left = torch.randperm(self.seq_len)
    self.index_set_left = self.index_set_left[:self.left_rank]

 def combine_heads(self, X):
    X = X.transpose(1, 2)
    X = X.reshape(X.size(0), X.size(1), self.num_head * self.head_dim)
    return X

 def split_heads(self, X):
    X = X.reshape(X.size(0), X.size(1), self.num_head, self.head_dim)
    X = X.transpose(1, 2)
    return X

 def forward(self,Q, K, V):
    #### Row Attention ####
    if self.left_rank <= self.seq_len:
       K1 = K[:,:,self.index_set_left,:]
       V1 = V[:,:,self.index_set_left,:]
    else:
       K1 = K
       V1 = V

    dots = Q @ K1.transpose(-1,-2)
    dots = dots / math.sqrt(self.head_dim)
    attn = nn.functional.softmax(dots,dim=-1)
    attn = self.drop_attn(attn)

    #### Column Attention ####
    Q2 = Q.transpose(-1,-2)
    if self.right_rank <= self.head_dim:

       K2 = K[:,:,:,self.index_set_right]
       V2 = V[:,:,:,self.index_set_right]
    else:
       K2 = K
       V2 = V

    dots_r = Q2 @ K2
    dots_r = dots_r / math.sqrt(self.seq_len)
    attn_r = nn.functional.softmax(dots_r,dim=-1).transpose(-1,-2)
    attn_r = self.drop_attn(attn_r)

    X = self.split_heads(self.ln_1(self.combine_heads(torch.matmul(attn,V1))))/2 + self.
         split_heads(self.ln_2(self.combine_heads(torch.matmul(V2,attn_r))))/2

    return X
```

Algorithm 2 Smoother component

```
class Smoother(nn.Module):

   def __init__(self, hidden_size, seq_len, dropout = 0.5, num_head = 2,transformer_dim =
        64, fold = 1):

      super(Smoother, self).__init__()

      self.hidden_size = hidden_size
      self.seq_len = seq_len
      self.dropout = dropout
      self.num_head = num_head
      self.dim = transformer_dim
      self.fold = fold

      self.weights_fft = nn.Parameter(torch.empty(self.seq_len//2+1, self.hidden_size,2))
      nn.init.kaiming_normal_(self.weights_fft, mode='fan_in', nonlinearity='relu')

      self.tiny_conv_linear = torch.nn.Conv1d(in_channels = self.hidden_size*2 ,
          out_channels = self.hidden_size, kernel_size = 3, padding= 1, groups = 1)
      self.dropout = torch.nn.Dropout(p=self.dropout)
      self.bn_1 = nn.BatchNorm1d(self.seq_len)

   def forward(self, x):

      #### Compute Segment Average ####
      B,S,H = x.shape
      u = x.reshape(B,S,self.fold,H//self.fold)
      u = torch.mean(u,dim = -1)

      #### Fourier Convolution ####
      fft_u = fft.rfft(u, n = self.seq_len, axis = -2)
      fft_u = torch.view_as_real(fft_u)
      fft_u = fft_u.repeat(1,1,H//self.fold,1)
      self.weight_used = self.weights_fft.unsqueeze(0)
      temp_real = fft_u[...,0]*self.weight_used[...,0] - fft_u[...,1]*self.weight_used
          [...,1]
      temp_imag = fft_u[...,0]*self.weight_used[...,1] + fft_u[...,1]*self.weight_used
          [...,0]
      out_ft = torch.cat([temp_real.unsqueeze(-1),temp_imag.unsqueeze(-1)],dim = -1)
      out_ft = torch.view_as_complex(out_ft)
      m = fft.irfft(out_ft, n = self.seq_len, axis = -2)

      #### Convolution Stem ####
      input_h = torch.cat((m, x), dim = -1)
      h = self.tiny_conv_linear(input_h.permute(0,2,1)).permute(0,2,1)
      h = self.dropout(F.relu(self.bn_1(h)))

      return h
```

Algorithm 3 pseudo code for Time-Series Forecasting

```
   def forward(self, x_in):
      B1,H1,C1 = x_in.shape
      for i in range(len(self.encoder)):
         attn_layer = self.encoder[i]
         #standardize the input data
         if i == 0:
            tmp_mean = torch.mean(x_in[:,:,:],dim = 1,keepdim = True)
            tmp_std = torch.sqrt(torch.var(x_in[:,:,:],dim = 1,keepdim = True)+1e0)
            x_in = (x_in - tmp_mean)/(tmp_std)

            enc_out1 = self.enc_embedding(x_in)

         enc_out1= attn_layer(enc_out1) + enc_out1

      #decoder via Fourier Extrapolation
      dec_out = self.fourierExtrapolation(post(enc_out1))
      output = (dec_out.reshape(B1,-1,C1))*(tmp_std)+tmp_mean
      return output
```

B PROOF OF LEMMA 1

A similar result, under a slightly different setting, can be found in (Cai et al., 2021). For the completeness of the paper, we provide a proof here. Firstly, we go over the definite of matrix incoherence, which is commonly used in many low-rank matrix applications.

Algorithm 4 Fourier Extrapolation

```
class fourierExtrapolation(nn.Module):
   def __init__(self,inputSize,n_harm = 8,n_predict = 96):
      super().__init__()
      self.n = inputSize
      self.n_harm = n_harm
      self.f = torch.fft.fftfreq(self.n)
      self.indexes = list(range(self.n))


      # sort indexes by frequency, lower -> higher
      self.indexes.sort(key = lambda i: torch.absolute(self.f[i]))
      self.indexes = self.indexes[:1 + self.n_harm * 2]

      self.n_predict = n_predict


      # compute init phase
      self.t = torch.arange(0, self.n + self.n_predict)
      self.t1 = self.t.unsqueeze(0).unsqueeze(-1).float().to('cuda')
      self.f = self.f.unsqueeze(0).unsqueeze(-1).to('cuda')
      self.t = self.t.unsqueeze(0).unsqueeze(-1).unsqueeze(-1)to('cuda')
      self.g = self.f[:,self.indexes,:].permute(0,2,1).unsqueeze(1)
      self.phase_init = 2 * 3.1415 * self.g * self.t


   def fourierExtrapolation(self,x):

      # x in frequency domain
      x_freqdom = torch.fft.fft(x,dim = -2)
      x_freqdom = torch.view_as_real(x_freqdom)
      # select importance frequencies
      x_freqdom = x_freqdom[:,self.indexes ,:,:]
      x_freqdom = torch.view_as_complex(x_freqdom)
      ampli = torch.absolute(x_freqdom) / self.n # amplitude
      phase = torch.angle(x_freqdom) # phase

      ampli = ampli.permute(0,2,1).unsqueeze(1)
      phase = phase.permute(0,2,1).unsqueeze(1)

      self.restored_sig = ampli * torch.cos(self.phase_init + phase)

      return torch.sum(self.restored_sig,dim = -1)
```

Definition 1 (μ-incoherence)**.** *Given a rank-r matrix $\boldsymbol{X} \in \mathbb{R}^{n\times d}$. Let $\boldsymbol{X} = \boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ be its compact singular value decomposition. $\boldsymbol{X}$ is μ-incoherent if there exists a constant μ such that*

$$\max_i \|\boldsymbol{e}_i^\top \boldsymbol{W}\| \leq \sqrt{\frac{\mu r}{n}} \qquad \text{and} \qquad \max_i \|\boldsymbol{e}_i^\top \boldsymbol{V}\| \leq \sqrt{\frac{\mu r}{d}},$$

where $\boldsymbol{e}_i$ denotes the i-th canonical basis vector.

Secondly, we resolve the sampling strategy. We consider a clear rank-r matrix $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, i.e., no additive noise and the rank is exact. Without loss of generality, we assume $n \geq d$. Provided $\boldsymbol{X}$ is μ-incoherent, by (Chiu & Demanet, 2013, Theorem 1.1), Skeleton approximation recovers $\boldsymbol{X}$ exactly, i.e.,

$$\boldsymbol{X} = \boldsymbol{C}\boldsymbol{U}\boldsymbol{R},$$

with probability at least $1 - \mathcal{O}(n^{-2})$ if we uniformly sample $\mathcal{O}(\mu r \log n)$ rows and columns to form the submatrices $\boldsymbol{C}$ and $\boldsymbol{R}$.

Thirdly, we resolve the error bound estimation. For the noisy matrix $\boldsymbol{X} + \boldsymbol{E}$, we directly apply (Hamm & Huang, 2021, Corollary 4.3). Thus, we have

$$\|\boldsymbol{X} - \hat{\boldsymbol{C}}\hat{\boldsymbol{U}}\hat{\boldsymbol{R}}\| \leq \mathcal{O}\left(\sqrt{\frac{nd}{l_C l_R}}\right)\|\boldsymbol{E}\|,$$

where $\hat{\boldsymbol{C}}$ and $\hat{\boldsymbol{R}}$ are sampled from the noisy matrix, $\hat{\boldsymbol{U}}$ is the pseudo-inverse of their intersection, and l_C (resp. l_R) is the number of columns (resp. rows) being sampled in $\hat{\boldsymbol{C}}$ (resp. $\hat{\boldsymbol{R}}$).

Note that this error bound assumes good column and row sampling, i.e., the clear submatrices corresponding to $\hat{\boldsymbol{C}}$ and $\hat{\boldsymbol{R}}$ can recover $\boldsymbol{X}$ exactly. Therefore, by combining the above two results, we show the claim in Lemma 1.

C PROOF OF LEMMA 2

As $f(t)$ is the convolution function of $\{x_t\}$ and $\{l_t\}$, from the definition of convolution for $t = 1, 2, ..., n$ we have

$$f(t) = \sum_{i=1}^{t} l_{t+1-i} x_i$$

and

$$f(t) - f(t-1) = \underbrace{\sum_{i=1}^{t-1} (l_{i+1} - l_i) x_i}_{:=(a_t)} + l_1 x_t. \tag{9}$$

By Hoffelding inequality, term (a) satisfies the following inequality with $\varepsilon > 0$.

$$\mathbb{P}(|(a_t)| \geq \varepsilon) = \mathbb{P}\left(\left| \sum_{i=1}^{t-1} (l_{i+1} - l_i) x_i \right| \geq \varepsilon \right) \leq \exp\left(- \frac{2\varepsilon^2}{(t-1) b_{\max}^2 \cdot \frac{1}{n^2} \sigma^2} \right) \tag{10}$$

Combine (10) with the union bound over $t = 1, 2, ..., n$ and the following (10) holds with probability at least $1 - \delta/2$:

$$\max_t |(a_t)| \leq b_{\max} \sigma \sqrt{\frac{1}{2n} \log\left(\frac{2n}{\delta}\right)} \tag{11}$$

Similarly, with probability $1 - \delta/2$, we have

$$\max_t |l_1 x_t| \leq a_{\max} \sigma \sqrt{\frac{1}{2n^2} \log\left(\frac{2}{\delta}\right)}. \tag{12}$$

Therefore, via (11) and (12), with probability at least $1 - \delta$, we have

$$\max_t |f(t) - f(t-1)| \leq b_{\max} \sigma \sqrt{\frac{1}{2n} \log\left(\frac{2n}{\delta}\right)} + a_{\max} \sigma \sqrt{\frac{1}{2n^2} \log\left(\frac{2}{\delta}\right)} \tag{13}$$

D PROOF OF LEMMA 3

The proof contains two parts. In the first part, we view the data sequence as a function of index t and construct the coefficients and orthogonal polynomials for function approximation. In the second part, we show such coefficients can be computed with Fourier convolution i.e. (5)).

Function Approximation. We reformulate the matrix $\boldsymbol{X}\boldsymbol{S}$ as follow:

$$\boldsymbol{X}\boldsymbol{S} = \left[\bar{\boldsymbol{x}}_1 \boldsymbol{e} \quad \bar{\boldsymbol{x}}_2 \boldsymbol{e} \quad \cdots \quad \bar{\boldsymbol{x}}_r \boldsymbol{e} \right],$$

where $\boldsymbol{e} \in \mathbb{R}^{1 \times s}$ is the one vector and $\bar{\boldsymbol{x}}_i \in \mathbb{R}^{n \times 1}$ is the average from $(s(i-1)+1)$-th column to (si)-th column of $\boldsymbol{X}$.

Next, we focus on vector $\bar{\boldsymbol{x}}_j$ and view its t-th element as the output of a function $h^j(t) = \bar{\boldsymbol{x}}_{jt}$. Via analysis in (Gu et al., 2020, Appendixes C and D), we can form an approximation on $h^j(t)$ as follow:

$$h^j_{[x \leq t]}(x) \approx \sum_{i=1} c_i^j(t) g_i(x), \tag{14}$$

where $\{g_i\}$ is a sequence of orthogonal polynomial and $[c_1^j(t), c_2^j(t), ..., c_s^j(t)] := \boldsymbol{c}_t^j \in \mathbb{R}^{1 \times s}$ satisfy

$$\frac{d}{dt} \boldsymbol{c}(t)^j = \frac{1}{t} \boldsymbol{c}(t)^j \boldsymbol{A}_0 + \frac{1}{ts \log n} h(t) \boldsymbol{b}_0 \tag{15}$$

where $\boldsymbol{A}_0 \in \mathbb{R}^{s\times s}$ and $\boldsymbol{b}_0 \in \mathbb{R}^{1\times s}$ are predefined matrix and vector respectively. Equation (15) is corresponding to the case with $\lambda_n = s\log n$ in (Gu et al., 2020).

We then use Forward Euler approach to discretize it:

$$\hat{\boldsymbol{c}}(t)^j = \hat{\boldsymbol{c}}(t-1)^j(\frac{1}{t}\boldsymbol{I} + \frac{1}{t}\boldsymbol{A}_0) + \frac{1}{ts\log n}h(t)\boldsymbol{b}_0, . \tag{16}$$

Via standard error analysis of Forward Euler approach, we have

$$\begin{aligned}
\boldsymbol{c}(t+1)^j &= \boldsymbol{c}(t)^j + \frac{1}{t}\boldsymbol{c}(t)^j\boldsymbol{A}_0 + \frac{1}{ts\log n}h(t)\boldsymbol{b}_0 + \frac{d^2}{dt^2}\boldsymbol{c}(t)^j|_{t=\xi} \\
&= \boldsymbol{c}(t)^j + \frac{1}{t}\boldsymbol{c}(t)^j\boldsymbol{A}_0 + \frac{1}{ts\log n}h(t)\boldsymbol{b}_0 + \frac{1}{\xi s\log n}h(\xi)'\boldsymbol{b}_0 \\
&= \boldsymbol{c}(t)^j + \frac{1}{t}\boldsymbol{c}(t)^j\boldsymbol{A}_0 + \frac{1}{ts\log n}h(t)\boldsymbol{b}_0 + \mathcal{O}\left(\frac{1}{ts\log n}\right),
\end{aligned}$$

where $\xi \in [t, t+1]$.

It implies that for $t = 1, 2, ..., n$,

$$\|\hat{\boldsymbol{c}}(t)^j - \boldsymbol{c}(t)^j\| \le \mathcal{O}\left(\frac{\log t}{s\log n}\right). \tag{17}$$

Combine (17) with the similar proof procedure in (Gu et al., 2020, Proposition 6), if $h^j(x)$ is quadratic spline interpolation on $\{\bar{\boldsymbol{x}}_{jt}\}$, we obtain

$$\|\bar{\boldsymbol{x}}_{jt} - \sum_{i=1}^{s}\hat{\boldsymbol{c}}_i(t)g_i(x)\| \le \mathcal{O}\left(t\log n/\sqrt{s}\right) = \mathcal{O}\left(t\log n\sqrt{\frac{r}{d}}\right). \tag{18}$$

The desirable result in Lemma 3 is obtained by repeatedly using (18) with $j = 1, 2, ..., r$.

Coefficients via Fourier Convolution. The remaining task is to show that $\{\hat{\boldsymbol{c}}(t)^j\}$ can be generated via Fourier convolution. To simplify the notation, we denote $\boldsymbol{A} = \frac{1}{t}\boldsymbol{I} + \frac{1}{t}\boldsymbol{A}_0$ and $\boldsymbol{b} = \frac{1}{t\log n}\boldsymbol{b}_0$ and (16) becomes

$$\hat{\boldsymbol{c}}(t)^j = \hat{\boldsymbol{c}}(t-1)^j\boldsymbol{A} + h(t)\boldsymbol{b}. \tag{19}$$

We then repeatedly use (19) from $t = 1, 2, ...$ and one may verify

$$\begin{aligned}
\hat{\boldsymbol{c}}_t^j &= \sum_{i=1}^{t-1}\boldsymbol{b}\boldsymbol{A}^{t-i}h(i) = \sum_{i=1}^{t-1}\boldsymbol{b}\boldsymbol{A}^{t-i}\bar{\boldsymbol{x}}_{ji} \\
\Rightarrow \quad \boldsymbol{C}^j &= \bar{\boldsymbol{A}}_j * (\bar{\boldsymbol{x}}_j\boldsymbol{e}),
\end{aligned} \tag{20}$$

where

$$\boldsymbol{C}^j = \begin{bmatrix}\hat{\boldsymbol{c}}_1^j \\ \hat{\boldsymbol{c}}_2^j \\ \vdots \\ \hat{\boldsymbol{c}}_n^j\end{bmatrix} \in \mathbb{R}^{n\times s}, \text{ and } \bar{\boldsymbol{A}}_j = \begin{bmatrix}\boldsymbol{b} \\ \boldsymbol{b}\boldsymbol{A} \\ \vdots \\ \boldsymbol{b}\boldsymbol{A}^{n-1}\end{bmatrix} \in \mathbb{R}^{n\times s}. \tag{21}$$

Next we repeatedly use (20) from $j = 1, 2, .., r$, and one has

$$\begin{aligned}
&\underbrace{\begin{bmatrix}\boldsymbol{C}^1 & \boldsymbol{C}^2 & \cdots & \boldsymbol{C}^r\end{bmatrix}}_{:=\boldsymbol{X}^{\text{smooth}}} = \underbrace{\begin{bmatrix}\bar{\boldsymbol{A}}_1 & \bar{\boldsymbol{A}}_2 & \cdots & \bar{\boldsymbol{A}}_r\end{bmatrix}}_{:=\boldsymbol{L}_0} * \underbrace{\begin{bmatrix}\bar{\boldsymbol{x}}_1\boldsymbol{e} & \bar{\boldsymbol{x}}_2\boldsymbol{e} & \cdots & \bar{\boldsymbol{x}}_r\boldsymbol{e}\end{bmatrix}}_{=\boldsymbol{X}\boldsymbol{S}} \\
\Rightarrow \quad & \boldsymbol{X}^{\text{smooth}} = \boldsymbol{L}_0 * \boldsymbol{X}\boldsymbol{S} \\
\Rightarrow \quad & \boldsymbol{X}^{\text{smooth}} = \mathcal{F}^{-1}\left(\mathcal{F}(\boldsymbol{L}_0)\cdot\mathcal{F}(\boldsymbol{X}\boldsymbol{S})\right) \\
\Rightarrow \quad & \boldsymbol{X}^{\text{smooth}} = \mathcal{F}^{-1}\left(\boldsymbol{L}\cdot\mathcal{F}(\boldsymbol{X}\boldsymbol{S})\right),
\end{aligned}$$

where we use the fact that $\boldsymbol{L}$ is constructed in frequency domain in Fourier convolution in Eq. (5).

Table 7: Experimental results on varying r, s_1 and s_2. Best result is in boldface and second best is underlined. And Ablation experiments for each components

(a) Experimental results on varying r parameter in smoothing component.

r	LisOps	Text	Retrieval	Image	Pathfinder	Average
1	37.30	65.25	78.65	51.36	71.23	60.76
8	38.30	69.27	**83.26**	53.90	75.82	64.11
16	**38.62**	**70.02**	83.21	**54.20**	**76.15**	**64.44**
32	38.19	69.27	82.05	53.73	75.58	63.76
64	37.89	69.73	81.79	51.28	75.52	63.24

(b) Experimental results on varying s_1 parameter in Row Attention.

s_1	LisOps	Text	Retrieval	Image	Pathfinder	Average
8	38.30	69.27	83.26	53.90	75.82	64.11
32	**38.44**	**70.85**	**83.41**	**54.92**	77.97	**65.12**
64	37.88	70.53	83.02	51.22	78.02	64.33
128	37.33	69.24	81.58	49.08	78.12	63.07
256	37.02	65.72	79.30	46.24	**78.14**	61.29

(c) Experimental results on varying s_2 parameter in Column Attention.

s_2	LisOps	Text	Retrieval	Image	Pathfinder	Average
1	37.32	55.28	57.37	40.97	66.25	51.44
4	37.82	52.05	72.58	46.74	73.17	57.47
8	**38.30**	69.27	83.26	53.90	75.82	64.11
16	37.77	**70.24**	**83.42**	**54.11**	77.92	**64.73**
32	37.62	68.32	80.11	51.66	**78.18**	62.98

(d) The SKT $(r, s_1, s_2 = 8)$ is used as baseline. The differences by removing each component from the baseline model are reported.

Model	LisOps	Text	Retrieval	Image	Pathfinder	Average
Baseline	38.30	69.27	83.26	53.90	75.82	64.11
Fourier Conv.	-0.47	-4.04	-13.98	-5.64	-6.59	-6.14
Conv Stem	-0.13	-0.55	-1.51	-1.76	-9.47	-0.88
Column Attn.	-1.16	-8.00	-9.16	-10.63	-12.45	-8.28
Row Attn.	-0.38	-1.92	-1.97	-2.64	-2.56	-1.89

E Model Parameters Impact

SKTformer introduces three extra hyperparameters, r, s_1 and s_2. We test the influence when varying them and report results in Table 7. We use SKTformer $(r, s_1, s_2 = 8)$ as the baseline model and other parameters are reported in Table 10 in Appendix F.

Influence of r in Fourier Convolution. The r parameter is used to determine the number of segment-averages to compute in (5). The smaller r leads the matrix with more duplicate columns, and more details information is lost. On the other hand, according to Lemma 3, the larger r would potently decrease the memorization ability and yield a high approximation error. In Table 7a, the best performance is observed when $r = 8$ or $r = 16$. For the case with $r = 1$, the token matrix is smoothed to rank one matrix, and the average accuracy drops 3.55 from the best setting. When the r value goes larger than 16, the accuracy in all experiments slightly decreases. We believe it is due to the over-fitting since the smoothed token matrix contains more flexibility and more irrelevant information training dataset is learned.

Influence of sample number s_1 in Row Attention. In Row Attention part, we randomly sample s_1 from key and value tokens. Table 7b reports that the optimal sampling amounts are different among tasks. In Pathfinder task, the optimal result is associated with $s_1 = 256$, while the best performance of other tasks the reached with $s_1 = 32$. Pathfinder task requires learning extreme long-range dependence (the connectivity between two circles far away from each other). The lack of enough tokens leads to inaccurate long-range dependence estimation and damages the final results. For the tasks like Image or Retrieval, the modest range dependence may already be enough to get promising performance, and we thus could use fewer token samples.

Influence of sample number s_2 in Column Attention. In Column Attention, s_2 columns are selected. The experiment results are shown in Table 7c. When setting $s_2 = 1$, average performance decreases by 13.24%. Similar behavior is also observed in the first row of Table 7a with $r = 1$. The information loss due to lack of rankness limits the final performance. In an average sense, $s_2 = 16$ gives the best result, and further increasing in s_2 slightly harms the accuracy in all tasks except Pathfinder.

F Experiment Configurations

In this section, we report the configurations for the experiments in Sections 4.1, 4.2, and 4.3.

Table 8: Experiment Configuration of SKTformer $(r, s_1, s_2 = 8)$.

Parameters	ListOps	Text	Retrieval	Image	Pathfinder
Epoch	5	30	15	60	100
Learning Rate	1e-4	1e-4	1e-4	1e-3	1e-4
Weight Decay	0	1e-2	1e-2	1e-2	1e-2
Batch Size	32	32	32	256	256
r, s_1, s_2	8,8,8	8,8,8	8,8,8	8,8,8	8,8,8
dropout in embedding	0	0.5	0.1	0.1	0
dropout in attention	0	0.1	0.1	0.1	0
dropout in smoother	0	0.5	0.1	0.5	0.5

Table 9: Experiment Configuration of SKTformer (best).

Parameters	ListOps	Text	Retrieval	Image	Pathfinder
Epoch	10	30	15	60	100
Learning Rate	1e-4	1e-4	1e-4	1e-3	1e-4
Weight Decay	1-2	1e-2	1e-2	1e-2	1e-2
Batch Size	32	32	32	256	256
r, s_1, s_2	8,8,8	8,8,8	8,32,32	8,16,16	8,128,32
dropout in embedding	0	0.5	0.1	0.5	0.1
dropout in attention	0	0.1	0.1	0.1	0.1
dropout in smoother	0	0.5	0.1	0.5	0.5

Table 11: Experiment Configuration for Ablation.

Parameters	ListOps	Text	Retrieval	Image	Pathfinder
Learning Rate	1e-4	1e-4	1e-4	1e-3	1e-4
Weight Decay	0	1e-2	1e-2	1e-2	1e-2
Batch Size	32	32	32	256	256
r, s_1, s_2	8,8,8	8,8,8	8,8,8	8,8,8	8,8,8
dropout in embedding	0	0.5	0.1	0.1	0
dropout in attention	0	0.1	0.1	0.1	0
dropout in smoother	0	0.5	0.1	0.5	0.5

G Additional Results on LRA

We have already provided the average of 5 runs with different random seeds in Table 1. Here we also provide the standard deviations for these experiments in Table 12.

Table 12: Accuracy on Long Range Arena (LRA) with standard errors shown in parenthesis. All results are averages of 5 runs with different random seeds.

Model	LisOps	Text	Retrieval	Image	Pathfinder
SKTformer $(r, s_1, s_2 = 8)$	38.30 (0.40)	69.27 (0.83)	83.26 (0.45)	53.90 (1.54)	75.82 (0.97)
SKTformer (best)	39.15 (0.48)	71.58 (0.95)	83.73 (0.61)	57.73 (1.83)	78.20 (1.32)

H Dataset and Implementation Details

In this subsection, we summarize the details of the datasets used in this paper as follows:

LRA datasets: **ListOps**(2K length mathematical expression task which investigates the parsing ability); **Text** (up to 4K byte/character-level document classification task that tests capacity in character compositionality); **Retrieval** (byte/character-level document matching task, which exams the information compression ability with two 4K length sequence); **Image** (pixel-wise sequence image classification based on the CIFAR-10 dataset); **Pathfinder** (long-range spatial dependency

Table 10: Experiment Configuration for Model Parameters Impact.

Parameters	ListOps	Text	Retrieval	Image	Pathfinder
Epoch	5	30	15	60	100
Learning Rate	1e-4	1e-4	1e-4	1e-3	1e-4
Weight Decay	0	1e-2	1e-2	1e-2	1e-2
Batch Size	32	32	32	256	256
dropout in embedding	0	0.5	0.1	0.1	0
dropout in attention	0	0.1	0.1	0.1	0
dropout in smoother	0	0.5	0.1	0.5	0.5

Table 13: Details of time series benchmark datasets.

DATASET	LENGTH	DIMENSION	FREQUENCY
ETTM2	69680	8	15 MIN
EXCHANGE	7588	9	1 DAY
WEATHER	52696	22	10 MIN
ELECTRICITY	26304	322	1H
ILI	966	8	7 DAYS
TRAFFIC	17544	863	1H

identification task. The input images contain two small points/circles and dash-line paths. The model needs to identify whether two points/circles are connected);The LRA has several desirable advantages that made us focus on it as the evaluation benchmark: **generality** (only requires the encoder part); **simplicity** (data augmentation and pretraining are out of scope); **challenging long inputs** (difficulty enough and room to improve); **diversity aspects** (tasks covering math, language, image, and spatial modeling); and **lightweight** (run with low resource requirement).

Time series datasets:1) ETT (Zhou et al., 2021a) dataset contains two sub-dataset: ETT1 and ETT2, collected from two separated counties. Each of them has two versions of sampling resolutions (15min & 1h). ETT dataset contains multiple time series of electrical loads and one time sequence of oil temperature. 2) Electricity[3] dataset contains the electricity consumption for more than three hundred clients with each column corresponding to one client. 3) Exchange (Lai et al., 2018) dataset contains the current exchange of eight countries. 4) Traffic[4] dataset contains the occupation rate of freeway systems in California, USA. 5) Weather[5] dataset contains 21 meteorological indicators for a range of one year in Germany. 6) Illness[6] dataset contains the influenza-like illness patients in the United States. Table 13 summarizes all the features for the six benchmark datasets. They are all split into the training set, validation set and test set by the ratio of 7:1:2 during modeling.

GLUE datasets: The GLUE benchmark covers various natural language understanding tasks and is widely used in evaluating transfering ability. The tasks can be devided in to two types, single-sentence tasks (SST-2 and CoLA), and sentence-pair tasks (MNLI, QQP,QNLI,STS-B,MRPC,RTE). Following the same settings in (Devlin et al., 2018), we exclude WNLI task.

I EXPERIMENTS ON THE SMOOTHNESS EFFECT OF FOURIER CONVOLUTION

In this section, we verify Fourier convolution component in the Smoother block can reduce the incoherence value in the early training stage. We use SKTformer with $(r, s_1, s_2 = 8)$ as the test model and test on an NLP dataset: Text, and a vision dataset: Pathfinder. We compute the μ-incoherence value [7] of the token matrix before and after the Fourier convolution (denoted as $\mu_{\boldsymbol{X}}$ and $\mu_{\boldsymbol{X}^{\text{smooth}}}$, respectively) for each samples in the validation dataset. Since we do not explicitly

[3] https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams 20112014

[4] http://pems.dot.ca.gov

[5] https://www.bgc-jena.mpg.de/wetter/

[6] https://gis.cdc.gov/grasp/fluview/fluportaldashboard.html

[7] Incoherence is defined by Definition 1 in Appendix B.

force the token matrix to be low-rank required by Definition 1, we report the incoherence value for different rankness settings ($\mathrm{rank} = 16$ and $\mathrm{rank} = 32$) approximately, and the mean and standard deviation of incoherence value can be found in Table 14. The average incoherence value reduced 30% after the Fourier convolution in both datasets. Moreover, We observe that the standard deviation significantly decreases, which suggests the Fourier convolution may also potentially stabilize the training procedure.

Table 14: The average incoherence parameters after 100 training steps with standard errors shown in the parenthesis.

Dataset	$\mu_{\boldsymbol{X}}$ ($\mathrm{rank} = 32$)	$\mu_{\boldsymbol{X}^{\mathrm{smooth}}}$ ($\mathrm{rank} = 32$)	$\mu_{\boldsymbol{X}}$ ($\mathrm{rank} = 16$)	$\mu_{\boldsymbol{X}^{\mathrm{smooth}}}$ ($\mathrm{rank} = 16$)
Text	2.75 (0.027)	2.05 (0.007)	3.98 (0.046)	3.23 (0.038)
Pathfinder	3.83 (0.221)	1.99 (0.001)	4.88 (0.264)	3.48 (0.001)

J Illustration on Effect of the Smoother and Skeleton Attention in Token Matrix

In this section, an illustration of the Smoother and Skeleton Attention part is shown in Figure 2. We smooth the input token matrix to ensure the sampling in rows and columns containing more local and/or global information. Thus, sampling several rows and columns from the smoothed token matrix can be more effective than the samples from the original token matrix.

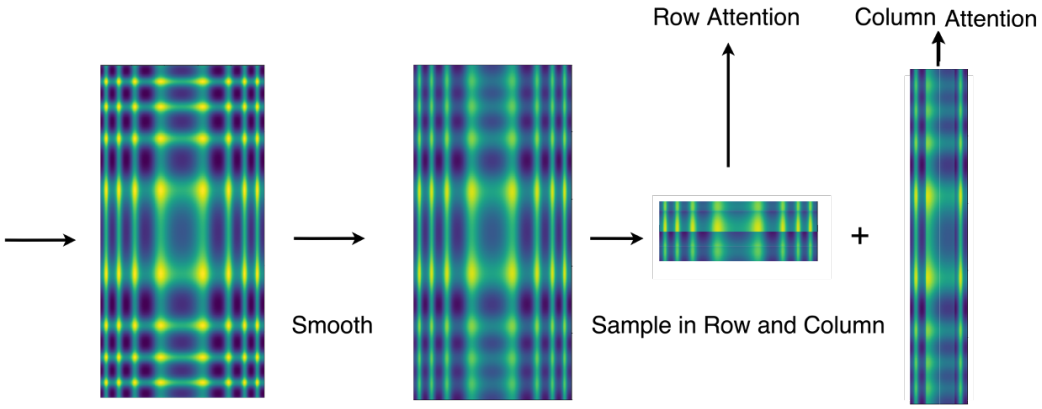

Figure 2: Illustration on effect of the Smoother and Skeleton Attention on Token Matrix.

K Transfer Learning

Table 15: The training configurations for Pretraining and GLUE tasks

	Pre-training	GLUE
Max Steps	1000K	-
Max Epochs	-	[4,20]
Learning Rate	1e-4	[5e-5,1e-4]
Batch Size	256	[16,32]
Warm-up Steps	5000	-
Sequence Length	512	128
Learning Rate Decay	-	Linear
Clip	-	1
Dropout	-	0.1