
MRConv: Reparameterized Multi-Resolution Convolutions for Long Sequence Modelling

Harry Jake Cunningham*
University College London

Giorgio Giannone†
University College London
Amazon

Mingtian Zhang
University College London

Marc Peter Deisenroth
University College London

Abstract

Global convolutions have shown increasing promise as powerful general-purpose sequence models. However, training long convolutions is challenging, and kernel parameterizations must be able to learn long-range dependencies without overfitting. This work introduces reparameterized multi-resolution convolutions (MRConv), a novel approach to parameterizing global convolutional kernels for long-sequence modelling. By leveraging multi-resolution convolutions, incorporating structural reparameterization and introducing learnable kernel decay, MRConv learns expressive long-range kernels that perform well across various data modalities. Our experiments demonstrate state-of-the-art performance on the Long Range Arena, Sequential CIFAR, and Speech Commands tasks among convolution models and linear-time transformers. Moreover, we report improved performance on ImageNet classification by replacing 2D convolutions with 1D MRConv layers.

1 Introduction

Modelling sequences with long-range dependencies is critical for solving tasks such as time-series forecasting, speech recognition and language modelling. Numerous deep learning models have been designed to address this challenge by aggregating information across long contexts, including recurrent neural networks (RNNs) [2, 28], convolutional neural networks (CNNs) [37, 4, 5, 40], and Transformers [46]. Despite significant progress, these models still struggle to effectively model long sequences due to training instabilities, inadequate inductive biases to prioritize important contextual information and prohibitive computational complexities.

Recently, State Space Models (SSMs) [22, 23, 43, 25, 19] have demonstrated their ability to model extremely long sequences. SSMs corresponding to a linear time-invariant (LTI) system, can be efficiently implemented using a global depthwise convolution. The convolution kernel is parameterized according to the HiPPO framework by initializing specially constructed state matrices [20, 24]. However, despite their success, SSMs are complex models that rely on sophisticated mathematics and linear algebra to compute the convolution kernel, which for S4 [22] and S4D [23] becomes a bottleneck for the faster downstream convolution.

Inspired by their success and equivalence to global convolutions, several authors have aimed to reduce the complexity of SSMs by *parameterizing long convolution kernels directly* [40, 27, 17, 38, 29]. In general, explicitly parameterized long convolution kernels are extremely difficult to train and are

*Correspondence to jake.cunningham.21@ucl.ac.uk

†Work done at University College London.

prone to overfitting, resulting in a significant performance drop compared to SSMs. To ease training, solutions to the parameterization problem have largely focused on: 1) low-rank approximations to the convolutional kernel through implicit neural representations [40, 27, 38], the composition of multi-resolution sub-kernels [29, 42], or regularization [17] and 2) a decaying kernel structure such that weights closer to the input are larger than ones further away [29].

In this work, we introduce reparameterized multi-resolution convolutions (MRConv), a method for parameterizing global convolutional kernels for long-sequence modelling, that simplifies, outperforms, and is more efficient than SSMs. Building on previous work, we focus our attention on **structured multi-resolution sub-kernels, low-rank kernel parameterizations** and **learnable kernel decay**. We adopt the multi-resolution approach introduced by SGConv [29], constructing large kernels as the sum of smaller sub-kernels of increasing length but fixed numbers of parameters. To improve performance we use ideas from computer vision, namely *structural reparameterization*. Specifically, to diversify optimization we train each sub-kernel in parallel, summing their activations after batch normalization, before merging all parameters into a single convolution at inference. We also explore several different low-rank kernel parameterizations and how these can be linearly combined during training to learn expressive long-range kernels that perform well across a wide range of data modalities.

Our contributions are summarized as follows:

- Inspired by SGConv [29], our MRConv structure constructs global kernels as the learnable combination of low-rank sub-kernels of increasing length but equal numbers of parameters, each designed to model the input at a different resolution and protect from overfitting.
- MRConv uses a novel reparameterization scheme, allowing each sub-kernel to be trained in parallel, utilizing batch normalization and linear scaling to learn the kernel decay rate, before merging them into a single global convolution at inference.
- We demonstrate that MRConv achieves **state of the art performance** among attention-free models and linear-time Transformers across various data modalities on Long Range Arena, sCIFAR, Speech Commands and ImageNet, whilst also improving efficiency.

2 Background

State Space Models We consider discretized, linear time-invariant (LTI) state-space models (SSMs) of the form

$$x_t = Ax_{t-1} + Bu_t \quad (1)$$

$$y_t = Cx_t + Du_t, \quad (2)$$

where $x_t \in \mathbb{R}^D$ is the hidden state at time $t = 1, \dots, L$, $u_t \in \mathbb{R}$ is a scalar input signal, and $y \in \mathbb{R}$ is an output signal. Essential to the success of SSMs is initialization of the system matrices $A \in \mathbb{R}^{D \times D}$, $B \in \mathbb{R}^{D \times 1}$, $C \in \mathbb{R}^{1 \times D}$, and $D \in \mathbb{R}^{1 \times 1}$ according to the HiPPO theory that projects the input sequence onto a set of orthogonal polynomials equipped with exponential decay [20, 24].

Unrolling the recursion over the length of the time horizon L , the output $y \in \mathbb{R}^L$ can equivalently be computed as a 1D causal convolution, avoiding computation of the hidden states, which can become very memory intensive, as

$$K = [CB, CAB, \dots, CA^{L-1}B] \quad (3)$$

$$y = K * u + Du, \quad (4)$$

where $u = [u_1, \dots, u_L]^T \in \mathbb{R}^L$. Computing the convolutional kernel for S4 and S4D scales as $\mathcal{O}((L+D)\log^2(L+D))$ using fast Cauchy and Vandermonde matrix-vector products, although this bottlenecks the faster $\mathcal{O}(L \log L)$ convolution implemented using Fast Fourier Transforms (FFTs).

Convolutional Models for Sequence Modelling Interpreting SSMs as a global convolution implicitly parameterized by the system matrices, we can also consider alternative implicit parameterizations. In general, implicit parameterizations define kernel values $K[t]$ as a function of the filter locations $t = 1, \dots, L$,

$$K[t] = \alpha^{-t} f_\theta(t), \quad (5)$$

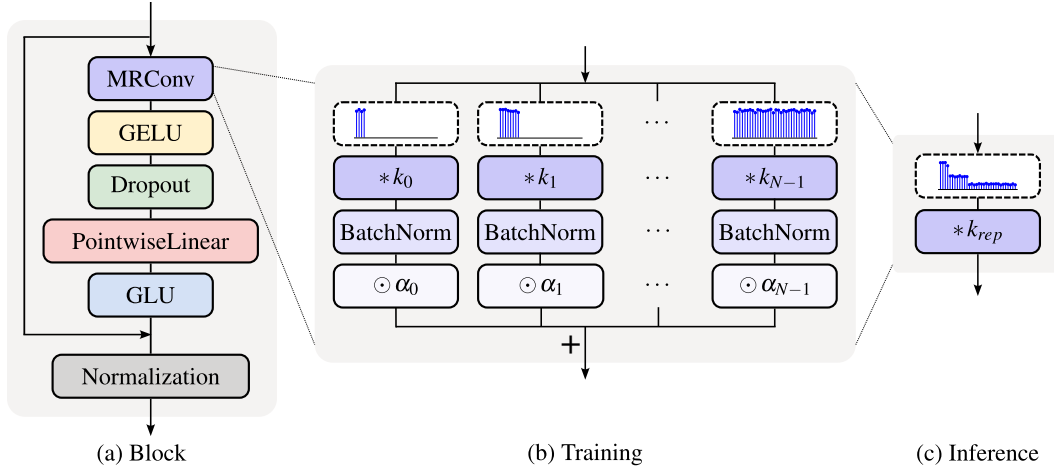


Figure 1: **Left:** The MRConv block is composed of a MRConv layer, GELU activation, pointwise linear layer, to mix the channels, and a gated linear unit. **Middle:** During training, the MRConv layer processes the input using N branches each with its own convolution kernel of increasing length and BatchNorm parameters. The output of the layer is given by pointwise multiplying each branch by α_i and summing. **Right:** At inference the branches can be reparameterised into a single convolution.

where f_θ is a parametric function with parameters $\theta \ll L$ and α is some decay constant. Several parameterizations of f_θ have been proposed including MLPs [40, 27, 38] and linear interpolation [29]. The low-rank structure of implicit parameterizations, coupled with exponential decay has proven a successful inductive bias for long-sequence modelling, ensuring the magnitude of kernel weights is greater for near information and improving generalization by preventing overfitting on irrelevant long-range dependencies.

3 Reparameterized Multi-Resolution Convolutions

We propose MRConv, a set of depthwise separable multi-resolution convolutions designed for long-sequence modelling. Addressing the need for implicit kernel parameterizations with a decay structure, we construct global kernels as the *learnable summation* of normalized multi-resolution sub-kernels of increasing length but with a constant number of parameters, using more parameters to aggregate local information. Further, we use *causal structural reparameterization* to train individual sub-kernels in parallel for diverse optimization and improved model performance, before combining them into a single kernel for efficient inference. See Figure 1 for an overview of the MRConv block.

In Section 3.1, we define 1D causal structural reparameterization and in Section 3.2 we introduce our reparameterization scheme for merging multi-resolution convolutions. In Section 3.3, we introduce 3 kernel parameterizations for generating low-rank kernels with fixed numbers of parameters but variable lengths.

3.1 Causal Structural Reparameterization

Our multi-resolution structure is based on the linearity of the convolution operator, which allows us to merge multiple branches of causal convolutions into a single convolution as

$$y[t] = \sum_{n=0}^{N-1} (u * k_n)[t] = \left(u * \left(\sum_{n=0}^{N-1} k_n \right) \right)[t] = (u * k_{rep})[t], \quad (6)$$

where k_n is the convolution kernel of the n th branch and $k_{rep} = \sum_{n=0}^{N-1} k_n$ is the *reparameterized kernel* computed by summing all n kernels together. In order to reparameterize causal convolution kernels of different sizes, we must ensure that the kernels are correctly aligned spatially before summing them, such that $k_n[\tau]$ for all kernels acts on the input at $u[t - \tau]$. To do this we pad shorter kernels of length l to the right with zeros such that the length of the kernel is the length of the longest

kernel L and then sum, such that

$$k_{rep} = \sum_{n=0}^{N-1} \text{ZeroPad}(k_n, (0, L-l)) = \sum_{n=0}^{N-1} \bar{k}_n, \quad (7)$$

where $\bar{k}_n$ corresponds to the zero-padded version of k_n . In this work, we consider two types of structural reparameterization: 1) *causal branch addition with batchnorms* for combining causal convolutions of varying length, and 2) *causal branch addition with linear rescaling* for combining causal convolutions of equal length.

Causal Branch Addition with BatchNorm When merging kernels of different lengths, normalization of each branch becomes crucial due to the impact of kernel size on the output statistics of convolutions with different length kernels. Hence, when constructing multi-resolution branches we use BatchNorm after each convolution to normalize the features, which can then be merged into the preceding convolution layer at inference via

$$k_{rep} = \overline{\text{BN}_0(k_0)} + \overline{\text{BN}_1(k_1)}, \quad (8)$$

where the lengths of the normalized kernels are adjusted according to equation 7.

Causal Branch Addition with Linear Rescaling When merging kernels of the same length normalization is unnecessary. In [26], the authors argue that the scaling factors of norm layers matter most as they diversify the optimization of different branches and instead propose replacing non-linear norm layers with linear scaling layers that can be reparameterized during training. We follow this advice for merging kernels of equal length as

$$k_{rep} = \beta_0 \cdot k_0 + \beta_1 \cdot k_1. \quad (9)$$

This reduces both memory and computational costs during training as all layers are now linear and hence kernels can be reparameterized during training.

3.2 Multi-Resolution Convolutions

We now outline our reparameterization scheme as a means of training multi-resolution convolutions using branches that can be combined into a single convolution kernel for efficient inference. Let $u \in \mathbb{R}^{D \times L}$ be a D -dimensional input sequence of length L . We define the number of independent resolutions as $N = \log_2(L/l_0) + 1$ where l_0 is the size of the kernel at the first resolution. At each resolution i , we define a kernel k_i of length $l_i = l_0 2^i$ for all $i < N$. We denote the output of each convolution as $c_i = (k_i * u) \in \mathbb{R}^{D \times L}$. Following each convolution, we pass the output through a BatchNorm, $\tilde{c}_i = \text{BN}_i(c_i)$, where each resolution has its own set of BN parameters. We define the set of normalized multi-resolution convolution outputs $\tilde{c} \in \mathbb{R}^{N \times D \times L}$ as,

$$\tilde{c} = [\text{BN}_0(k_0 * u), \text{BN}_1(k_1 * u), \dots, \text{BN}_{N-1}(k_{N-1} * u)]. \quad (10)$$

Given $\tilde{c}$ we wish to combine the outputs to generate an output y , ensuring the most relevant parts of the input are highlighted. The output $y[t] \in \mathbb{R}^D$ at time step t is generated by computing a linear combination of the coefficients $\tilde{c}[t]$ at time step t according to

$$y[t] = \alpha^T \tilde{c}[t], \quad (11)$$

where $\alpha \in \mathbb{R}^{N \times D}$ is a learnable parameter. Applying α across the sequence length we define the output $y \in \mathbb{R}^{D \times L}$ as the summation

$$y = \alpha_0 \text{BN}_0(k_0 * u) + \alpha_1 \text{BN}_1(k_1 * u) + \dots + \alpha_{N-1} \text{BN}_{N-1}(k_{N-1} * u). \quad (12)$$

Further, applying our reparameterization scheme at inference, we can rewrite the above process as a single convolution by zero-padding shorter kernels and merging the BN parameters into each convolution as

$$y = u * (\alpha_0 \overline{\text{BN}_0(k_0)} + \alpha_1 \overline{\text{BN}_1(k_1)} + \dots + \alpha_{N-1} \overline{\text{BN}_{N-1}(k_{N-1})}) = u * k_{rep}, \quad (13)$$

eliminating the extra memory and computational cost of training with extra convolutions.

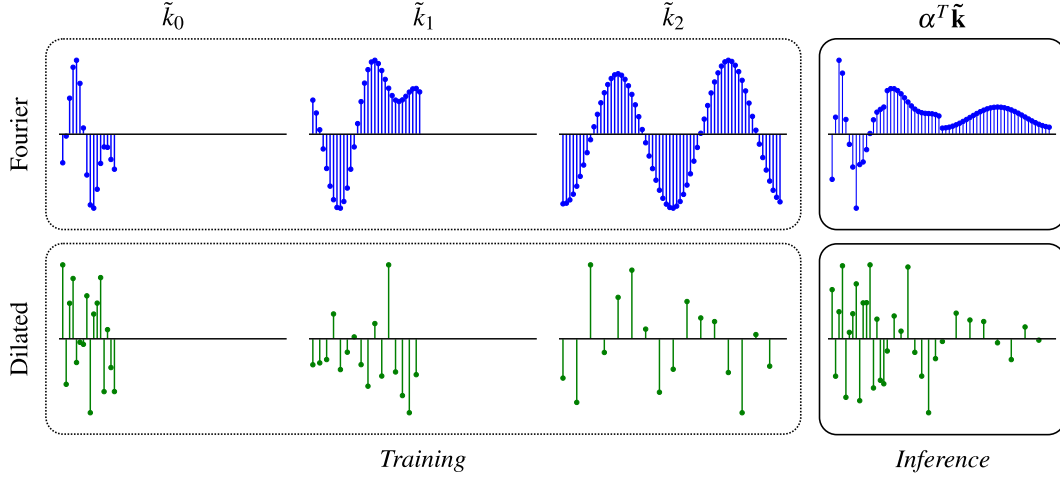


Figure 2: **Multi-resolution structural reparameterization.** During *training*, we parameterize each branch with a kernel of increasing length but fixed number of parameters. For the Fourier kernels, we use only a handful of low-frequency modes and for the dilated kernels we increase the dilation factor. At *inference*, we combine the kernels into a single kernel by merging the BN parameters with the kernel parameters and performing a learnt weighted summation.

3.3 Low-Rank Kernel Parameterization

Our multi-resolution framework is general and agnostic to the parameterization of the kernels at each resolution. In this work, we consider 3 different parameterizations:

Dilated Kernels Inspired by wavelets, dilated convolutions are a variation on standard convolutional filters where p many zeros are padded between the elements of the kernel, where $p + 1$ is known as the dilation factor. Formally, dilated convolutions are defined as

$$y[t] = (u * k_{\text{dilated}})[t] = \sum_{\tau=0}^{l-1} k[\tau]u[t - p\tau]. \quad (14)$$

They are a parameter-efficient way of increasing the receptive field of a convolution by detaching the length of the kernel from the number of parameters.

Fourier Kernels Instead of parameterizing kernels in the time domain $k \in \mathbb{R}^{D \times L}$, we instead parameterize them as complex valued kernels $\hat{k} \in \mathbb{C}^{D \times L}$ in the Fourier domain. To get k we simply take an inverse Fourier transform of $\hat{k}$, $k = \mathcal{F}^{-1}[\hat{k}]$. We can also generate long low-rank kernels by only parameterizing a small number m of low-frequency Fourier modes. In practice we use FFTs and zero-padding to achieve this, at a cost of $\mathcal{O}(L \log L)$, which is cheaper than computing the kernel in the SSM formulation,

$$k_{\text{fourier}}[t] = \text{IFFT}[\text{ZeroPad}(\hat{k}, L - m)][t]. \quad (15)$$

Sparse Kernels Similar to dilated kernels we also propose sparse kernels, where, instead of regularly spacing kernel elements at a set distance apart, we randomly sample their positions across the sequence length, which we then fix during training and inference. Given a set of kernel value locations $\mathcal{T}$ we define sparse kernels as

$$k_{\text{sparse}}[t] = \delta_{t \in \mathcal{T}} \cdot k_t, \quad (16)$$

where $\delta_t \in \mathcal{T}$ is the Kronecker delta, which equals 1 if t is in the set $\mathcal{T}$ and 0 otherwise, and k_t represents the non-zero kernel value at position t .

3.4 FFT Convolutions

Other than dilated kernel convolutions, which can be computed in $\mathcal{O}(kL)$ using the implicit GEMM algorithm [6], for all kernels we compute the depthwise convolution using FFTs reducing the

computation to $\mathcal{O}(L \log L)$ time complexity. We also make use of a number of highly optimized FFT convolution implementations, which further speeds up our work and reduces memory [17, 18].

Remark. *The time complexity of a multi-resolution convolution on a sequence of length L is at most $\mathcal{O}((\log(L/k_0) + 1)L \log L)$ during training and $\mathcal{O}(L \log L)$ during inference where each convolution is performed in the Fourier domain.*

4 Related Work

Several prior works have used multi-resolution convolutions for general sequence modelling. SGConv [29] is most similar to our approach, using a weaker form of reparameterization, by concatenating smaller low-rank sub-kernels to construct long convolution kernels. Each sub-kernel is implicitly defined by linearly interpolating dilated kernel values and a fixed kernel decay is used. We expand considerably upon their work, exploring several new reparameterization schemes, introduce improved kernel parameterizations and add learnable kernel decay.

MultiresNet [42] uses dilated convolutions with shared weights to parameterize a learnable wavelet transform and also learns a linear combination of outputs similar to our method. However, they don't consider reparameterizing their model into a single global convolution for efficient inference. Ding et al. [14] design a set of structurally parameterized kernels also using dilated kernels and parallel branches during training, however they only consider small kernel sizes and doesn't consider any kernel decay.

Recently CHELA [33] proposed to use short and long convolutions placed sequentially in conjunction with self-attention. However, in their work, in instances where they place a non-linear activation function between the convolutions, the non-linearity restricts the ability to reparameterize both convolutions into a single kernel.

5 Experiments

We now evaluate the empirical performance of MRConv against similar baseline methods on long sequence modelling tasks. We test 3 different kernel parameterizations: 1) dilated kernels, 2) Fourier kernels and 3) Fourier + sparse kernels, which we reparameterize during training using linear rescaling (see Equation 9). We select model hyperparameters to ensure similar computational complexities to comparable models such as S4 and SGConv. Our results show that MRConv is a highly effective and efficient sequence modeller, achieving SoTA performance on LRA, sCIFAR and Speech Commands, whilst being more efficient than SSMs, such as S4, and linear-time transformers.

5.1 Long Range Arena

The Long Range Arena (LRA) benchmark [44] evaluates the performance of sequence models on long-range modelling tasks on a wide range of data modalities and sequence lengths from 1,024 to 16,000. For all LRA tasks, we use the standard S4 block (see Figure 1) and use MRConv as a drop-in replacement for the SSM layer. We train two model variants: 1) *Base* has similar complexity and parameters to existing convolutional and SSM baselines and 2) *Large* where we scale the model with increased width or depth to match the computational budget set by more expensive quadratic attention baselines (see Table 4b).

Table 1 compares MRConv to other baseline methods. Treating the kernel parameterization as a model hyperparameter and selecting the model with the highest validation accuracy, MRConv-*Base* achieves the highest average score among sub-quadratic complexity models including S5, SGConv and modern linear-time transformer architectures such as MEGA-Chunk. Further, MRConv-*Large* matches the performance of more computationally expensive quadratic transformers whilst being much faster at inference due to reparameterization (see Table 4b). We conduct a series of ablation studies to assess the effectiveness of our kernel parameterizations, reparameterization scheme, and the importance of learnable decay. Additional implementation details can be found in Appendix D.3.

Kernel Parameterization Table 1 displays the LRA results for each proposed kernel parameterization. Dilated kernels perform exceptionally well on the Image task, outperforming all other non-input-dependent models by 1.2%. However, on information-dense tasks such as ListOps, Fourier

Table 1: **Test accuracy on the Long Range Arena Benchmarks.** We follow the standard training procedures introduced in [23]. Bold scores indicate the highest performing model on a given task and underlined the second best performing. $\times$ indicates a model did not do better than random guessing and - that a result was not available. In this table we only include results from other non-input-dependent models.

Model (Input length)	ListOps (2,048)	Text (4,096)	Retrieval (4,000)	Image (1,024)	Pathfinder (1,024)	Path-X (16,384)	Avg.
Transformer	36.37	64.27	57.46	42.44	71.40	$\times$	53.66
Transformer + SPT	59.15	88.81	90.38	76.00	88.49	88.05	81.81
<i>Linear-Time Transformers:</i>							
BST	61.49	87.63	90.51	91.07	95.75	95.28	86.96
SPADE-Chunk	60.50	90.69	91.17	88.22	<u>96.23</u>	97.60	87.40
MEGA-Chunk	58.76	<u>90.19</u>	90.97	85.80	94.41	93.81	85.66
<i>State Space Models:</i>							
S4D-LegS	60.47	86.18	89.46	88.19	93.06	91.95	84.89
S4-LegS	59.60	86.82	90.90	88.65	94.20	96.35	86.09
Liquid-S4*	62.75	89.02	91.20	89.50	94.8	96.66	87.32
S5	62.15	89.31	91.40	88.00	95.33	<u>98.58</u>	87.46
<i>Convolutional Models:</i>							
CCNN	43.60	84.08	-	88.90	91.51	$\times$	-
Long Conv	62.2	89.6	91.3	87.0	93.2	96.0	86.6
SGConv	61.45	89.20	91.11	87.97	95.46	97.83	87.17
<i>Ablations:</i>							
MRConv-B, Dilated	60.90	86.38	88.30	90.37	94.42	$\times$	78.40
MRConv-B, Fourier	62.40	89.26	<u>91.44</u>	88.55	95.03	97.82	87.42
MRConv-B, Fourier+Sparse	62.10	89.26	91.35	89.07	95.55	$\times$	79.56
MRConv-L, Dilated	61.25	88.36	89.78	<u>90.55</u>	95.22	$\times$	79.19
MRConv-L, Fourier	<u>62.45</u>	89.40	91.48	89.30	95.75	98.65	87.84
MRConv-L, Fourier+Sparse	61.65	89.42	91.35	89.15	96.64	$\times$	79.70
<i>Ours:</i>							
MRConv-B	62.40	89.26	<u>91.44</u>	90.37	95.55	97.82	87.81
MRConv-L	<u>62.45</u>	89.42	91.48	<u>90.55</u>	96.64	98.65	88.20

kernels perform better, as the sparse dilated pattern is prone to skipping important tokens. Fourier kernels perform best on average and are the only model that achieves better than random guessing on Path-X, where sparse and dilated kernels struggle due to their sparsity. The combination of sparse and Fourier kernels is also effective on natural data and performs well on Pathfinder but can lead to overfitting on discrete data modalities, such as ListOps.

MRConv Design Table 2 shows that all of our structural additions improve performance over dense kernels by 13.2% and 6.4% on the ListOps and Image tasks. We found that BatchNorms are crucial for learning the linear combination of each multi-resolution convolution, resulting in an improvement in accuracy of 5.35% and 2.36% on both tasks than without them. Interestingly, parameterizing our multi-resolution architecture with dense kernels reduces overfitting, improving the test accuracy by 3.90% and 3.75%, despite having more parameters. Path-X was the only task where we found learning the combination of multi-resolution convolutions did not improve performance and we provide extra implementation details and ablations in Appendix D.3.3.

Resolution Analysis Figure 3b shows the normalized magnitude of the weight α_i for each kernel k_i at different depths of the model after training on ListOps and CIFAR datasets. The resulting weight distribution shows that early layers learn short high-frequency kernels for local features, while deeper layers learn an even distribution across a range of longer kernels. On CIFAR, deeper layers learn low-frequency global kernels, aligning with existing observations on larger models [48]. These non-stationary filter characteristics with depth emphasise the need for effective learnable kernel decay, which is difficult to achieve with hand-tuned initializations such as in SGConv.

5.2 Pixel-Level 1D Image Classification

Next, we evaluate MRConv on the sequential CIFAR (sCIFAR) image classification task, where images are flattened to a 1D sequence of pixels. This is a challenging sequence modelling task as the model

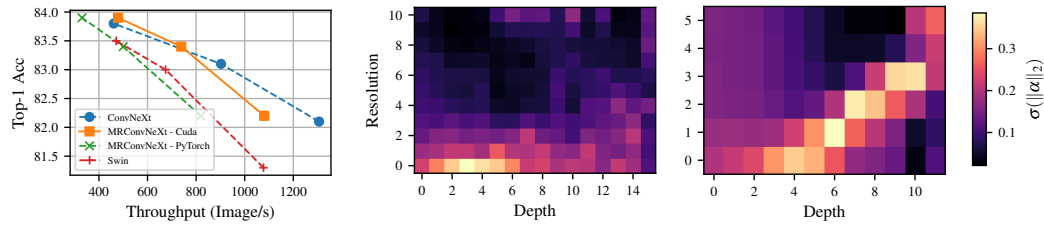


Figure 3: **Left:** ImageNet Top-1 Acc. vs. Throughput. **Right:** Distribution of α norms for each depth for MRConv trained on ListOps and CIFAR respectively. Changing composition of kernels highlights how the convolution kernels are non-stationary with respect to depth.

Model modification	ListOps			Image		
	Params	Accuracy	Change	Params	Accuracy	Change
Dense kernel	2.4M	49.25	-	6.3M	82.90	-
+ Multiresolution	4.5M	53.15	+3.90	9.4M	86.65	+3.75
+ Fourier kernel	307K	57.05	+7.80	3.8M	86.19	+3.29
+ BatchNorm	332K	62.40	+13.15	3.8M	88.55	+5.65
+ 2x Depth (MRConv, Fourier)	661K	62.45	+13.20	7.7M	89.30	+6.40

Table 2: **MRConv design albatons.** Effect of MRConv modifications on ListOps and Image tasks from LRA. For reference we note that *S4-LegS* uses 815K and 3.6M parameters and *Liquid-S4* uses 333K and 11M parameters for each task respectively.

must be able to capture pixel-level relationships at different scales, as pixels close together in 2D space can be far apart in its flattened sequence representation.

Table 3a reports the performance of MRConv compared to other baseline methods. On sCIFAR MRConv with dilated kernels significantly improves test accuracy upon the previous best model, MultiresNet, by 1.1%. Further, MRConv with dilated kernels is very parameter efficient using 10 layers and 5.7M parameters, in contrast to S4 which uses 6 layers and 7.9M parameters [22]. We also note a significant performance gap between Fourier and dilated parameterizations. We hypothesise that dilated convolutions enhance the ability of long kernels to focus on long-range sparse patterns (i.e. relationships between pixels far apart might be more important than pixels closer together), which is well suited for flattened image data which has a high correlation between neighboring pixels.

5.3 Raw Speech Classification

The Speech Commands (SC) dataset [47] contains 1s sound recordings, sampled at 16,000 Hz, of 35 spoken words in English. The task is to classify the spoken word from its sampled waveform. We also test zero-shot classification at a lower sampling rate of 8,000 Hz to test the continuous-time parameterization of each model.

Table 3b reports the results. Both the dilated and Fourier kernel parameterizations perform well, especially MRConv equipped with Fourier kernels, which outperforms all baseline models. On the zero-shot task, MRConv with Fourier kernels also performs the best. By parameterizing the kernels in the Fourier domain with a set of low-frequency modes, we ensure that the kernels are band-limited. As a result, we can downsample each kernel whilst avoiding the effects of aliasing, improving zero-shot testing performance over alternative continuous formulations, such as SGConv, which don't have any anti-aliasing guarantees. It is important to note that dilated kernels and sparse kernels are not continuous parameterizations and are therefore not suitable models for performing zero-shot changes in input resolution.

5.4 ImageNet Classification

To evaluate MRConv on a large-scale task, we employ the ImageNet classification benchmark [41], which consists of 1.28 million high-resolution training images of size 224×224 and 1000 classes. As a base architecture, we choose ConvNeXt [32], a fully convolutional model that enhances the ResNet architecture by incorporating elements from Vision Transformers. To assess MRConv, we

Model (Input length)	sCIFAR (1024)	Model (Input length)	16 kHz (16,000)	8 kHz (8,000)
Transformer	62.2			
<i>State Space Models:</i>		<i>CNNs:</i>		
S4D	89.92	InceptionNet	61.24	05.18
S5	90.10	ResNet-1	77.86	08.74
S4	91.80	XResNet-50	83.01	07.72
Liquid-S4	92.02	ConvNet	95.51	07.26
<i>Convolutional Models:</i>		<i>State Space Models:</i>		
CKConv	63.74	S4D-LegS	95.83	91.08
TrellisNet	73.42	S4-LegS	96.08	91.32
FlexConv	80.82	Liquid-S4	<u>96.78</u>	90.00
Long Conv, Geom Init	92.1	S5	96.52	<u>94.53</u>
Long Conv, Learnable Butterfly	92.5	<i>Convolutional Models:</i>		
CCNN	93.08	SGConv	96.42	94.29
MultiresNet	<u>93.15</u>	MRConv, Dilated	96.47	-
MRConv, Dilated	94.26	MRConv, Fourier	96.82	95.05
MRConv, Fourier	92.67	MRConv, Fourier+Sparse	95.21	-
MRConv, Fourier+Sparse	92.40			

(a) sCIFAR

(b) Speech Commands

Table 3: **Left:** Test accuracy on sCIFAR pixel-level 1D image classification. **Right:** Test accuracy on 35-way Speech Commands classification task [47]. Each model is trained on one-second 16kHz audio waveforms and then tested at 16kHz and 0-shot at 8kHz.

Model	2D Bias	FLOPs	Top 1 Acc.	Model	Avg.	Speed Text Image	
<i>ConvNeXt-T</i>	✓	4.5G	82.1	S4	86.09	1×	1×
SGConvNeXt-T	✗	4.3G	82.0	MEGA-Chunk ($\mathcal{O}(L)$)	85.66	0.3×	0.7×
MRConvNeXt-T	✗	4.3G	82.2	MRConv-B	87.77	0.4×	0.6×
<i>ConvNeXt-S</i>	✓	8.7G	83.1	MRConv-B (Rep)	87.77	1.5×	1.3×
SGConvNeXt-S	✗	8.3G	83.4	MEGA ($\mathcal{O}(L^2)$)	88.21	0.1×	0.5×
MRConvNeXt-S	✗	8.3G	83.4	MRConv-L	88.20	0.2×	0.3×
<i>ConvNeXt-B</i>	✓	15.4G	83.8	MRConv-L (Rep)	88.20	0.9×	0.7×
SGConvNeXt-B	✗	14.6G	83.9				
MRConvNeXt-B	✗	14.6G	83.9				

(a) ImageNet Classification

(b) LRA Compute Comparison

Table 4: **Left:** Top-1 test accuracy on ImageNet classification [41]. **Right:** Inference time speed comparison between *Base* and *Large* versions of MRConv and linear and quadratic attention versions of *MEGA* [34]. We denote *Rep* as reparameterized models. By scaling MRConv up with more parameters we match the performance of MEGA with quadratic attention, whilst also being more efficient.

replace the standard 7x7 2D convolutional layers in each block with 1D MRConv blocks, flattening the 2D features in each layer to 1D sequences. We denote our model MRConvNeXt and use the same hyperparameter settings for each Tiny/Small/Base model from ConvNeXt without any changes.

Comparing MRConvNeXt with ConvNeXt and SGConvNeXt [29], another 1D convolution model, MRConvNeXt equipped with Fourier + Sparse kernels achieves SoTA performance, outperforming ConvNeXt at every model size. As highlighted by [29], 1D FFT convolutions use fewer FLOPs than standard convolutions, although empirically, the throughput decreases. Using optimized CUDA kernels for 1D FFT convolutions, we close the gap between theoretical and empirical throughput as shown in Figure 3a, comfortably outperforming Swin [31] a powerful vision transformer, and improving the throughput-accuracy frontier when compared to standard ConvNeXt.

6 Discussion & Conclusion

In this work, we introduced MRConv, a simple yet effective method for parameterizing long-convolution kernels. We develop three kernel parameterizations and demonstrate through experimentation how each approach is suited to different data modalities. Further, we show the importance of having a learnable decay due to differing model characteristics with depth. Finally, we highlight MRConv's leading performance on LRA, sCIFAR, Speech Commands and ImageNet classification.

However, our model is not without its limitations. Training with parallel branches requires computing many more convolutions, increasing memory usage and slowing down training. Currently, parallel training is necessary due to the presence of batch normalization, which is non-linear during training. For future work, we aim to remove batch normalization, potentially through initialization [8] or linear rescaling (Equation 9), allowing for reparameterization during training, significantly reducing training costs. Our model also lacks input dependency. Whilst this does not affect performance on natural data, such as images and audio, on discrete information-dense sequences, such as text, linear-time transformers still outperform MRConv (see Table 1). For future work, we propose introducing input dependency into our model, using either Hyena recurrences [38] or by combining with self-attention similar to MEGA [34]. Finally, unlike SSMs, our model doesn't support fast autoregressive inference by construction. However, we note an equivalence between kernels constructed as the sum of Fourier basis functions and SSMs has already been established [24]. We propose converting MRConv equipped with Fourier kernels into a multi-resolution SSM as future work.

References

- [1] I. Amos, J. Berant, and A. Gupta. Never train from scratch: Fair comparison of long-sequence models requires data-driven priors. *arXiv preprint arXiv:2310.02980*, 2023.
- [2] M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. In *International conference on machine learning*, pages 1120–1128. PMLR, 2016.
- [3] S. Arunachalam, S. Khairnar, and B. Desale. The fast fourier transform algorithm and its application in digital image processing. *New J Chem*, 35(5):1004–1010, 2013.
- [4] S. Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.
- [5] S. Bai, J. Z. Kolter, and V. Koltun. Trellis networks for sequence modeling. *arXiv preprint arXiv:1810.06682*, 2018.
- [6] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.
- [7] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35: 16344–16359, 2022.
- [8] S. De and S. Smith. Batch normalization biases residual blocks towards the identity function in deep networks. *Advances in Neural Information Processing Systems*, 33:19964–19975, 2020.
- [9] X. Ding, Y. Guo, G. Ding, and J. Han. Acnet: Strengthening the kernel skeletons for powerful cnn via asymmetric convolution blocks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1911–1920, 2019.
- [10] X. Ding, T. Hao, J. Tan, J. Liu, J. Han, Y. Guo, and G. Ding. Resrep: Lossless cnn pruning via decoupling remembering and forgetting. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4510–4520, 2021.
- [11] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13733–13742, 2021.
- [12] X. Ding, H. Chen, X. Zhang, J. Han, and G. Ding. Repmlpnet: Hierarchical vision mlp with re-parameterized locality. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 578–587, 2022.
- [13] X. Ding, X. Zhang, J. Han, and G. Ding. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11963–11975, 2022.

- [14] X. Ding, Y. Zhang, Y. Ge, S. Zhao, L. Song, X. Yue, and Y. Shan. Unireplknet: A universal perception large-kernel convnet for audio, video, point cloud, time-series and image recognition. *arXiv preprint arXiv:2311.15599*, 2023.
- [15] N. B. Erichson, O. Azencot, A. Queiruga, L. Hodgkinson, and M. W. Mahoney. Lipschitz recurrent neural networks. *arXiv preprint arXiv:2006.12070*, 2020.
- [16] M. Fathi, J. Pilault, P.-L. Bacon, C. Pal, O. Firat, and R. Goroshin. Block-state transformer. *arXiv preprint arXiv:2306.09539*, 2023.
- [17] D. Y. Fu, E. L. Epstein, E. Nguyen, A. W. Thomas, M. Zhang, T. Dao, A. Rudra, and C. Ré. Simple hardware-efficient long convolutions for sequence modeling. *arXiv preprint arXiv:2302.06646*, 2023.
- [18] D. Y. Fu, H. Kumbong, E. Nguyen, and C. Ré. Flashfftconv: Efficient convolutions for long sequences with tensor cores. *arXiv preprint arXiv:2311.05908*, 2023.
- [19] A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.
- [20] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33:1474–1487, 2020.
- [21] A. Gu, C. Gulcehre, T. Paine, M. Hoffman, and R. Pascanu. Improving the gating mechanism of recurrent neural networks. In *International Conference on Machine Learning*, pages 3800–3809. PMLR, 2020.
- [22] A. Gu, K. Goel, and C. Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.
- [23] A. Gu, K. Goel, A. Gupta, and C. Ré. On the parameterization and initialization of diagonal state space models. *Advances in Neural Information Processing Systems*, 35:35971–35983, 2022.
- [24] A. Gu, I. Johnson, A. Timalsina, A. Rudra, and C. Ré. How to train your hippo: State space models with generalized orthogonal basis projections. *arXiv preprint arXiv:2206.12037*, 2022.
- [25] R. Hasani, M. Lechner, T.-H. Wang, M. Chahine, A. Amini, and D. Rus. Liquid structural state-space models. *arXiv preprint arXiv:2209.12951*, 2022.
- [26] M. Hu, J. Feng, J. Hua, B. Lai, J. Huang, X. Gong, and X.-S. Hua. Online convolutional re-parameterization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 568–577, 2022.
- [27] D. M. Knigge, D. W. Romero, A. Gu, E. Gavves, E. J. Bekkers, J. M. Tomczak, M. Hoogendoorn, and J.-J. Sonke. Modelling long range dependencies in nd: From task-specific to a general purpose cnn. *arXiv preprint arXiv:2301.10540*, 2023.
- [28] J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber. A clockwork rnn. In *International conference on machine learning*, pages 1863–1871. PMLR, 2014.
- [29] Y. Li, T. Cai, Y. Zhang, D. Chen, and D. Dey. What makes convolutional models great on long sequence modeling? *arXiv preprint arXiv:2210.09298*, 2022.
- [30] S. Liu, T. Chen, X. Chen, X. Chen, Q. Xiao, B. Wu, T. Kärkkäinen, M. Pechenizkiy, D. Mocanu, and Z. Wang. More convnets in the 2020s: Scaling up kernels beyond 51x51 using sparsity. *arXiv preprint arXiv:2207.03620*, 2022.
- [31] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.
- [32] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022.
- [33] Z. Liu, S. Li, L. Wang, Z. Wang, Y. Liu, and S. Z. Li. Short-long convolutions help hardware-efficient linear attention to focus on long sequences. *arXiv preprint arXiv:2406.08128*, 2024.
- [34] X. Ma, C. Zhou, X. Kong, J. He, L. Gui, G. Neubig, J. May, and L. Zettlemoyer. Mega: moving average equipped gated attention. *arXiv preprint arXiv:2209.10655*, 2022.

- [35] S. Massaroli, M. Poli, D. Y. Fu, H. Kumbong, R. N. Parnichkun, A. Timalisina, D. W. Romero, Q. McIntyre, B. Chen, A. Rudra, et al. Laughing hyena distillery: Extracting compact recurrences from convolutions. *arXiv preprint arXiv:2310.18780*, 2023.
- [36] N. Nonaka and J. Seita. In-depth benchmarking of deep neural network architectures for ecg diagnosis. In *Machine Learning for Healthcare Conference*, pages 414–439. PMLR, 2021.
- [37] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.
- [38] M. Poli, S. Massaroli, E. Nguyen, D. Y. Fu, T. Dao, S. Baccus, Y. Bengio, S. Ermon, and C. Ré. Hyena hierarchy: Towards larger convolutional language models. *arXiv preprint arXiv:2302.10866*, 2023.
- [39] D. W. Romero, R.-J. Brintjes, J. M. Tomczak, E. J. Bekkers, M. Hoogendoorn, and J. C. van Gemert. Flexconv: Continuous kernel convolutions with differentiable kernel sizes. *arXiv preprint arXiv:2110.08059*, 2021.
- [40] D. W. Romero, A. Kuzina, E. J. Bekkers, J. M. Tomczak, and M. Hoogendoorn. Ckconv: Continuous kernel convolution for sequential data. *arXiv preprint arXiv:2102.02611*, 2021.
- [41] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.
- [42] J. Shi, K. A. Wang, and E. Fox. Sequence modeling with multiresolution convolutional memory. In *International Conference on Machine Learning*, pages 31312–31327. PMLR, 2023.
- [43] J. T. Smith, A. Warrington, and S. W. Linderman. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*, 2022.
- [44] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.
- [45] T. Trinh, A. Dai, T. Luong, and Q. Le. Learning longer-term dependencies in rnns with auxiliary losses. In *International Conference on Machine Learning*, pages 4965–4974. PMLR, 2018.
- [46] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [47] P. Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.
- [48] Q. Zhang, D. Ram, C. Hawkins, S. Zha, and T. Zhao. Efficient long-range transformers: You need to attend more, but not necessarily at every layer. *arXiv preprint arXiv:2310.12442*, 2023.
- [49] S. Zuo, X. Liu, J. Jiao, D. Charles, E. Manavoglu, T. Zhao, and J. Gao. Efficient long sequence modeling via state space augmented transformer. *arXiv preprint arXiv:2212.08136*, 2022.

Appendix

A Additional Background	13
A.1 Structural Reparameterization	13
B Additional Methods	14
B.1 Multi-Head Convolutions	14
B.2 Reduced-Dimension Convolutions	14
B.3 Connections to SSMs	14
C Implementation Details	15
C.1 Kernel Parameterization	15
C.2 Multi-Resolution Convolutions	15
D Experiments and Configurations	16
D.1 Runtime Comparison	16
D.2 Default Hyperparameters	17
D.3 Long Range Arena	18
D.4 Pixel-Level 1D Image Classification	22
D.5 Raw Speech Classification	23
D.6 ImageNet Classification	24

A Additional Background

We provide some additional background information that was not included in the main body of the paper.

A.1 Structural Reparameterization

It has been shown that training multi-branch convolutional layers with different combinations of paths, scales and complexities can enrich the feature space, improving performance over a single convolutional layer [9–14, 26]. However, such architectures lead to increased training and inference costs. Developed in the vision community, *structural reparameterization* exploits the ability to merge multi-branch convolutional architectures during training into a single convolution at inference by applying a set of equivalent transformations to the parameters of the convolutional layers. In general, structural reparameterization trades off additional training costs for performance.

A key component of structural reparameterization is the use of BatchNorms (BNs),

$$\text{BN}(x) = \frac{x - \mu}{\sqrt{\sigma + \epsilon}} * \gamma + \beta \quad (17)$$

where μ is the accumulated mean, σ the standard deviation, γ the learned scaling factor and β the learned bias. BNs provide training time non-linearity, causing the kernels to undergo different optimization dynamics than training an equivalent reparameterized convolution. After training BN parameters can be merged with the the preceding convolution layer parameters as,

$$\bar{W} = \frac{\gamma}{\sigma} W, \quad \bar{b} = -\frac{\mu\gamma}{\sigma} + \beta \quad (18)$$

Recently, structural reparameterization has been used to train large kernels alongside a small kernel, forming an implicit ensemble of models with differing receptive fields, successfully increasing the size of the receptive field without losing the ability to capture small-scale dependencies between inputs [30, 14, 13].

B Additional Methods

Here we discuss some additional methods that we experimented with but did not include in the main body of the paper.

B.1 Multi-Head Convolutions

Similar to multi-head attention, SSMs and prior long-convolution methods generate a *multi-head structure* by applying independent long convolutions to separate copies of the input. Each copy of the input is termed a *head*.

Remark. For an input $u \in \mathbb{R}^{D \times L}$, a multi-head convolution with H heads processes the input H times and has HD number of independent depthwise convolutions.

B.2 Reduced-Dimension Convolutions

It has been shown that during training long-convolutions converge to equivalent low-dimensional SSMs and has therefore proven beneficial to apply independent long convolutions over *groups of channels* instead of each channel individually [35]. This reduces the dimensionality of the convolution requiring fewer convolutional filters. Given an input $u \in \mathbb{R}^{D \times L}$, we apply *reduced-dimension* convolutions by:

1. Splitting the dimension D into G groups of size $M = D/G$ such that $u \in \mathbb{R}^{G \times M \times L}$.
2. Applying a set of M depthwise separable convolutions with kernel $k \in \mathbb{R}^{M \times L}$ independently over $G \times M$.
3. Concatenating the outputs $y \in \mathbb{R}^{G \times M \times L}$ over the groups G such that $y \in \mathbb{R}^{D \times L}$.

Remark. For an input $u \in \mathbb{R}^{D \times L}$, a reduced-dimension convolution with G groups reduces the number of independent depthwise convolutions to D/G .

B.3 Connections to SSMs

We establish a relationship between our Fourier parameterised kernels and SSMs. Let $k \in \mathbb{R}^L$ be a real-valued kernel of length L , sampled at $1/L$ Hz and parameterized in the Fourier domain as,

$$k[t] = \frac{1}{L} \sum_{k=0}^{L-1} \hat{c}_k \exp\left(2\pi i \frac{tk}{L}\right) \quad (19)$$

where $\hat{c}_k \in \mathbb{C}$ are complex valued parameters and $\hat{c}_k = \hat{c}_{-k}^*$ such that kernel values k are real.

From [24] we can represent the above convolution kernel as the following SSM,

$$\mathbf{A}_{nk} = \begin{cases} -2 & n = k = 0 \\ -2\sqrt{2} & n = 0, k \text{ odd} \\ -2\sqrt{2} & k = 0, n \text{ odd} \\ -4 & n, k \text{ odd} \\ 2\pi k & n - k = 1, k \text{ odd} \\ -2\pi n & k - n = 1, n \text{ odd} \\ 0 & \text{otherwise} \end{cases} \quad \mathbf{B}_n = \begin{cases} 2 & n = 0 \\ 2\sqrt{2} & n \text{ odd} \\ 0 & \text{otherwise} \end{cases} \quad (20)$$

Theorem ([24] 6.). As state size $N \rightarrow \infty$, the SSM in Eq 20 is a time-invariant orthogonal state space model defined by the truncated Fourier basis functions, orthonormal on $[0, 1]$, $\{p_n\}_{n \geq 0} = [1, c_0(t), s_0(t), \dots]$, where $c_m(t) = \sqrt{2} \cos(2\pi mt)$ and $s_m(t) = \sqrt{2} \sin(2\pi mt)$ for $m = 0, \dots, N/2$.

Hence, as the state size $N \rightarrow \infty$, the corresponding convolution kernel to the above SSM is a linear combination of truncated Fourier basis functions with support on the unit interval $[0, 1]$, controlled by the vector of coefficients $\mathbf{C}$ whose parameters correspond to $\hat{c}$ in our explicit formulation in Eq 19. It is interesting to note that whilst we exploit the FFT to generate our convolution in $\mathcal{O}(L \log L)$, to generate the equivalent kernel given the recurrent form of the SSM takes $\mathcal{O}((L + D) \log^2(L + D))$ time using Cauchy and Vandermonde matrix multiplications, which, even with efficient CUDA implementations, results in considerably slower performance than highly optimized FFTs (see Table 2).

C Implementation Details

We provide algorithmic implementations of both the dilated and Fourier kernel parameterizations and the aggregation step in MRConv.

C.1 Kernel Parameterization

Algorithm 1 MRConv, Dilated

```
1: Input: Input sequence  $\mathbf{u} : [B, D, L]$ , depth  $N$ 
2: Output:  $\tilde{\mathbf{c}} : [B, D, L, N]$ 
3: for  $i = 1$  to  $N$  do
4:    $\mathbf{h}_i \leftarrow \text{ImplicitFilter}(D, i)$  //  $[D, 2^i]$ 
5:    $\mathbf{c}_i \leftarrow \text{DepthwiseCausalConv}(\mathbf{u}, \mathbf{h}_i)$  //  $[B, D, L, 1]$ 
6:    $\tilde{\mathbf{c}}_i \leftarrow \text{BN}_i(\mathbf{c}_i)$  //  $[B, D, L, 1]$ 
7: end for
8:  $\tilde{\mathbf{c}} = \text{Concat}(\{\tilde{\mathbf{c}}_i\}_{i=1}^N)$ 
```

Algorithm 2 MRConv, Fourier

```
1: Input: Input sequence  $\mathbf{u} : [B, D, L]$ , depth  $N$ , Complex Kernels  $\hat{\mathbf{k}} : [D, L]$ 
2: Output:  $\mathbf{c} : [B, D, L, N]$ 
3: for  $i = 1$  to  $N$  do
4:    $\mathbf{h}_i \leftarrow \text{IFFT}[\text{ZeroPad}(\hat{\mathbf{k}}_i)]$ 
5:    $\mathbf{c}_i \leftarrow \text{DepthwiseCausalConv}(\mathbf{u}, \mathbf{h}_i)$  //  $[B, D, L, 1]$ 
6:    $\tilde{\mathbf{c}}_i \leftarrow \text{BN}_i(\mathbf{c}_i)$  //  $[B, D, L, 1]$ 
7: end for
8:  $\tilde{\mathbf{c}} = \text{Concat}(\{\tilde{\mathbf{c}}_i\}_{i=1}^N)$ 
```

C.2 Multi-Resolution Convolutions

Algorithm 3 MRConv, Aggregation

```
1: Input: Input sequence  $\mathbf{u} : [B, D, L]$ , weights  $\alpha : [N, D]$ , depth  $N$ .
2: Output:  $\mathbf{y} : [B, D, L]$ 
3:  $\tilde{\mathbf{c}} \leftarrow \text{MultiResConv}(\mathbf{u}, N)$  //  $[B, D, L, N]$ 
4:  $\mathbf{y} \leftarrow \alpha^T \tilde{\mathbf{c}}$  //  $[B, D, L]$ 
```

D Experiments and Configurations

In this section we provide additional experimental details, including model architecture, hyperparameters and task details. We also include some further ablation studies not included in the main body of the paper.

D.1 Runtime Comparison

To evaluate the computational efficiency of MRConv we benchmark its runtime against multi-head attention, FlashAttention [7], and S4D [23]. The evaluation was conducted on an NVIDIA A100-40GB GPU where each model has a hidden dimension of 768 and we use a batch size of 64. Both FlashAttention and MRConv, which uses FlashFFTConv [18], utilize `bf16` precision. For MRConv we use an initial kernel size of size 128. The runtime was computed as an average over 1000 forward passes. We get out of memory errors for MHA at sequence length ≥ 4096 .

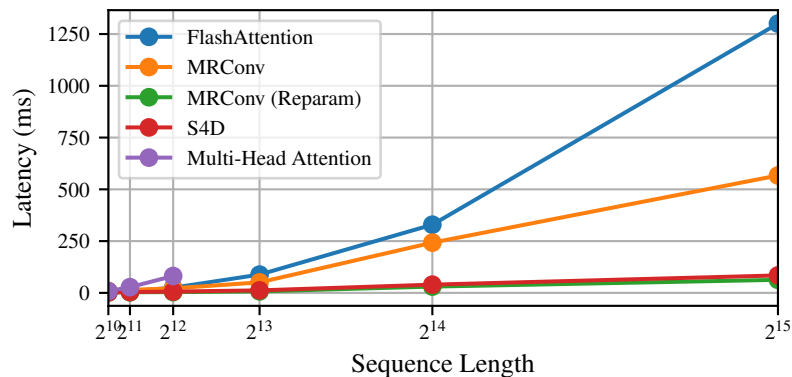


Figure 4: **Runtime Comparison.** Runtime comparison of MRConv versus PyTorch’s Multi-Head attention (MHA) implementation and FlashAttention at inference with increasing sequence length.

D.2 Default Hyperparameters

Base Models Table 5 presents the highest performing hyperparameters for each base model used for each experiment. For all experiments we ensure that the *total number of trainable parameters* stays comparable with baseline methods.

Dataset	Kernel Type	Depth	Features	Kernel Size	Bidirectional	Norm	Prenorm	Dropout	Kernel LR	LR	WD	Batch Size	Epochs
ListOps	Fourier	8	128	2	✗	BN	✗	0.05	0.001	0.003	0.05	50	40
Text	Fourier	6	256	1	✗	BN	✓	0.05	0.001	0.005	0.05	16	32
Retrieval	Fourier	6	256	1	✗	BN	✓	0.05	0.001	0.003	0.05	64	20
Image	Dilated	6	512	8	✗	LN	✗	0.1	0.001	0.0045	0.05	50	200
Pathfinder	Fourier+Sparse	6	256	16	✓	BN	✓	0.1	0.001	0.005	0.03	64	200
Path-X	Fourier	6	256	64	✓	BN	✓	0	0.001	0.004	0.03	16	50
sCIFAR	Dilated	10	512	8	✗	LN	✗	0.2	0.001	0.0045	0.05	50	300
SC	Fourier	6	128	32	✓	BN	✓	0.1	0.001	0.005	0.05	16	40

Table 5: Experiment hyperparameters for MRConv-Base variants

Large Models Table 6 presents the highest performing hyperparameters for each large model used in LRA. For this set of experiments we ensure that the *computational resources* (throughput and memory requirements) at inference are comparable with baseline methods that use quadratic attention.

Dataset	Kernel Type	Depth	Features	Kernel Size	Bidirectional	Norm	Prenorm	Dropout	Kernel LR	LR	WD	Batch Size	Epochs
ListOps	Fourier	16	128	1	✗	BN	✗	0.05	0.001	0.003	0.05	50	40
Text	Fourier+Sparse	6	384	1	✗	BN	✓	0.1	0.001	0.005	0.05	16	32
Retrieval	Fourier	6	384	1	✗	BN	✓	0	0.001	0.003	0.05	64	20
Image	Dilated	10	512	8	✗	LN	✗	0.2	0.001	0.0045	0.05	50	200
Pathfinder	Fourier+Sparse	12	256	32	✓	BN	✓	0.05	0.001	0.005	0.03	64	200
Path-X	Fourier	12	256	64	✓	BN	✓	0	0.001	0.004	0.03	16	50

Table 6: Experiment hyperparameters for MRConv-Large variants

D.2.1 Optimization

We follow the optimization approach presented in [23], which uses the AdamW optimizer with a global learning rate and weight decay and a separate smaller learning rate with no weight decay specifically for the kernel parameters. All experiments use a cosine annealing learning rate schedule with linear warmup.

D.2.2 Compute Infrastructure

All LRA, sCIFAR and Speech Commands experiments were run using a single 40GB A100 GPU apart from Retrieval and Path-X and Speech Commands where we use 2 40GB A100s.

D.3 Long Range Arena

Here we provide context and implementation details for each of the LRA tasks. In our work we use the now standard pre-processing steps from [22].

1. **ListOps**: The dataset consists of sequences of nested mathematical operations, including brackets, lists of numbers and operators `min`, `max`, `median` and `summod`. In total, there are 17 unique token values, including possible integers, which are encoded as one-hot vectors. The task is to compute the integer result of the operations, e.g. `[min 7 4 [max 2 5 ] 3 4 [median 0 1 6 9 , 2 ]]` $\rightarrow$ 1. This is described as a 10-class classification problem where each class represents an integer result of the expression. The sequences are of unequal length and shorter sequences within a batch are zero-padded up to a maximum length of 2,000. There are 96,000 training examples, 2,000 validation examples and 2,000 test sequences.
2. **Text**: The task is binary text classification on whether a movie review is positive or negative. The dataset is constructed from the IMDB reviews dataset. Each review is encoded on the character level as one-hot vectors, with 129 possible values. Encoding on the character level naturally simulates a longer input sequence. Sequences are of unequal length and are either zero-padded or truncated to a maximum length of 4,000. There are 25,000 training examples, no validation examples and 25,000 test examples.
3. **Retrieval**: The task is to identify if two papers share a citation link. The dataset is based on the ACL Anthology Network dataset. The input is constructed by concatenating each document and compressing each sequence individually, before passing the compressed representations to a linear classifier. Similar to the text classification setup, text inputs are encoded on the character level as one-hot vectors with 97 unique values. Input sequences of unequal length are either zero-padded or truncated to a maximum length of 4,000, which after concatenation makes the total sequence length 8,000. There are 147,086 training pairs, 18,090 validation pairs and 17,437 test pairs.
4. **Image**: The task is image classification using the CIFAR-10 dataset, flattened into a 1D sequence of length 1024 and grayscaled. Each pixel value is normalized to have zero mean and unit variance. The dataset consists of ten classes. There are 45,000 training examples, 5,000 validation examples and 10,000 test examples.
5. **Pathfinder**: The dataset consists of 32×32 grayscale images which contain 2 small circles and several dashed lines. The images are then flattened to a 1D sequence of length 1,024 and normalized to be in the range $[-1, 1]$. The task is to identify whether the circles are connected by one of the dashed lines or not, formulated as a binary classification problem. There are 160,000 training examples, 20,000 validation examples and 20,000 test examples.
6. **Path-X**: The task is a harder version of the pathfinder challenge, where the input images are 128×128 , which when flattened form a sequence of length 16,384, and there are more distraction lines. This makes the Path-X challenge significantly harder than Pathfinder.

D.3.1 Hyperparameter Sweeps

For all experiments we performed large hyperparameter sweeps on the following parameters:

- Kernel size: [1, 2, 4, 8, 16, 32]
- Learning Rate: [0.03, 0.05, 0.07]
- Dropout: [0, 0.1, 0.2, 0.3]

D.3.2 Extended LRA Results

We provide further LRA results that include quadratic attention Transformers and citations in Table 7.

D.3.3 Path-X Implementation Details

We found that due to the extreme length of the input sequences on Path-X, (16,384), MRConv required a large initial kernel size to ensure the longer kernels had a large enough rank such that they wouldn't just collapse to a bias term. However, using large initial kernel sizes also led to overfitting on local information, and we didn't perform better than random guessing. Additionally, We observed that

Model (Input length)	ListOps (2,048)	Text (4,096)	Retrieval (4,000)	Image (1,024)	Pathfinder (1,024)	Path-X (16,384)	Avg.
<i>Quadratic-Time Transformers:</i>							
Transformer [46]	36.37	64.27	57.46	42.44	71.40	X	53.66
Transformer + SPT [1]	59.15	88.81	90.38	76.00	88.49	88.05	81.81
MEGA [34]	63.14	<u>90.43</u>	91.25	90.44	96.01	97.98	88.21
<i>Linear-Time Transformers:</i>							
BST [16]	61.49	87.63	90.51	91.07	95.75	95.28	86.96
SPADE-Chunk [49]	60.50	90.69	91.17	88.22	<u>96.23</u>	97.60	87.40
MEGA-Chunk [34]	58.76	90.19	90.97	85.80	<u>94.41</u>	93.81	85.66
<i>State Space Models:</i>							
S4D-LegS [23]	60.47	86.18	89.46	88.19	93.06	91.95	84.89
S4-LegS [22]	59.60	86.82	90.90	88.65	94.20	96.35	86.09
Liquid-S4 [25]	<u>62.75</u>	89.02	91.20	89.50	94.8	96.66	87.32
S5 [43]	62.15	89.31	91.40	88.00	95.33	<u>98.58</u>	87.46
<i>Convolutional Models:</i>							
CCNN [40]	43.60	84.08	-	88.90	91.51	X	-
Long Conv [17]	62.2	89.6	91.3	87.0	93.2	96.0	86.6
SGConv [29]	61.45	89.20	91.11	87.97	95.46	97.83	87.17
<i>Ablations:</i>							
MRConv-B, Dilated	60.90	86.38	88.30	90.37	94.42	X	78.40
MRConv-B, Fourier	62.40	89.26	<u>91.44</u>	88.55	95.03	97.82	87.42
MRConv-B, Fourier+Sparse	62.10	89.26	<u>91.35</u>	89.07	95.55	X	79.56
MRConv-L, Dilated	61.25	88.36	89.78	<u>90.55</u>	95.22	X	79.19
MRConv-L, Fourier	62.45	89.40	91.48	<u>89.30</u>	95.75	98.65	87.84
MRConv-L, Fourier+Sparse	61.65	89.42	91.35	89.15	96.64	X	79.70
<i>Ours:</i>							
MRConv-B	62.40	89.26	<u>91.44</u>	90.37	95.55	97.82	87.81
MRConv-L	62.45	89.42	91.48	<u>90.55</u>	96.64	98.65	<u>88.20</u>

Table 7: **Test accuracy on the Long Range Arena Benchmarks [44].** We follow the standard training procedures introduced in [23]. Bold scores indicate the highest performing model on a given task and underlined the second best performing. **X** indicates a model did not do better than random guessing and - that a result was not available. In this table we only include results from other non-input-dependent models.

learning the linear combination of each sub-kernel resulted in some overfitting when compared to using a fixed kernel decay.

To address the training instabilities, we simplified our reparameterization scheme by concatenating kernels together instead of summing them as is in SGConv, and using a fixed kernel decay rate throughout. These changes helped to stabilize the training process and improved generalization, as shown in Table 8.

Model modification	Path-X	
	Accuracy	Change
MRConv-B, Fourier	50.00	-
+ Concatenate sub-kernels	96.19	+ 46.19
+ Fixed kernel decay	97.83	+ 47.83

Table 8: **Path-X design ablations.** We find that concatenating kernels together instead of summing dramatically improved performance on Path-X from random guessing (50%) to 96.19% accuracy and using a fixed kernel decay reduced overfitting, further improving performance to 97.83%.

D.3.4 Convergence Plots

We provide additional training plots corresponding to the ablations reported in Table 2. In particular, the convergence plots highlight the difficulty in training long dense convolution kernels which overfit to the training data, as seen in Figure 5.

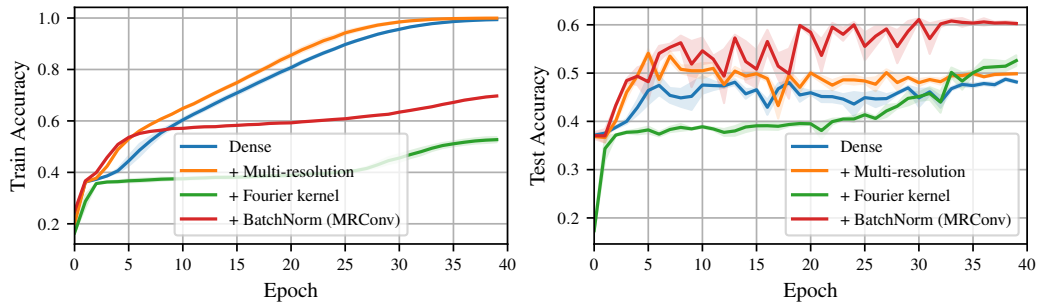


Figure 5: **Convergence Plots on ListOps.** Corresponds to ablations in Table 2

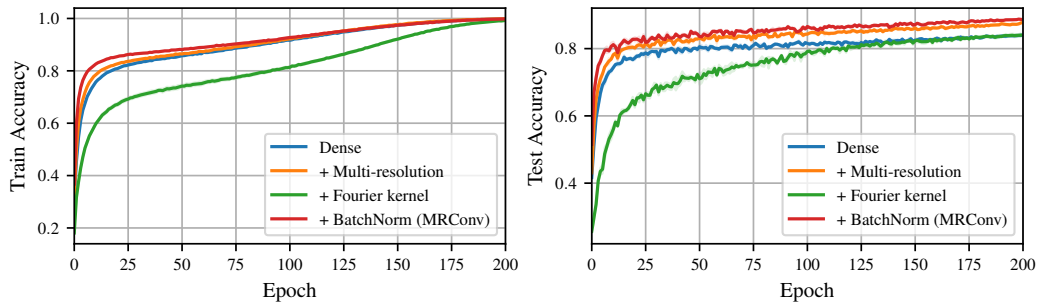


Figure 6: **Convergence Plots on sCIFAR.** Corresponds to ablations in Table 2

D.3.5 Initial Resolution Ablation

We perform an additional ablation study on the effects of different initial kernel sizes on the ListOps and Image datasets from LRA. Table 9 reports our results, highlighting how kernel size can have an effect of final accuracy, with shorter kernel sizes favoring more discrete data types such as found in the ListOps task and longer kernel sizes smoother data such as flattened images used in the sCIFAR task.

Kernel Type	ListOps			Image		
	l_0	N	Accuracy	l_0	N	Accuracy
Fourier Kernel	1	11	61.80	8	8	86.69
	2	10	62.40	16	7	88.39
	4	9	61.10	32	6	88.55

Table 9: **Initial kernel size ablations.** Effect of the initial kernel size l_0 on ListOps and Image tasks from LRA. We note that the initial kernel size also effects the number of independent resolutions $N = \log_2(L/l_0) + 1$ which we include in the table for clarity.

D.3.6 LayerNorm Ablation

Whilst LayerNorm cannot be used in conjunction with structural parametrization since it remains a non-linear operation during inference, we provide a further ablation where we substitute BatchNorm for LayerNorm. Our results in Table 10 show that the performance of LayerNorm is comparable to BatchNorm. This further emphasises the suitability of BatchNorm for structural reparameterization, on top of being able to reparameterize BatchNorm at inference.

D.3.7 MLP Kernels

Further to the Dilated, Sparse and Fourier kernel parameterizations suggested introduced in Section 3.3, we also include experimental results performed using multi-resolution kernels implicitly parameterized by an MLP of fixed size. Kernels parameterized by MLPs have previously shown to be

Norm Type	ListOps	Image
BatchNorm	62.40	89.30
LayerNorm	60.10	87.80

Table 10: **Normalization ablations.** Effect of different normalization layers. We find that there is very little difference between using LayerNorm or BatchNorm but emphasize the benefit of using BatchNorm which enables structural reparameterization.

successful in sequence modelling tasks [40, 39, 38]. Our results in Table 11 show however that MLP based kernels when placed within our multi-resolution framework underperform both dilated and Fourier kernels.

Kernel Type	ListOps		Image	
	Params	Accuracy	Params	Accuracy
Dilated	759K	59.25	3.5M	90.37
Fourier	332K	62.40	3.8M	88.55
Fourier+Sparse	420K	62.25	4.0M	89.07
MLP	580K	60.08	3.6M	85.71

Table 11: **Kernel parameterization ablations.** Performance of different kernel parameterizations on ListOps and Image tasks from LRA.

D.4 Pixel-Level 1D Image Classification

The task is 10-way image classification using the CIFAR-10 dataset. The task is identical to the LRA Image task except that flattened full colour images are used as input instead. Table 12 reports additional results and citations.

Model (Input length)	sCIFAR (1024)
Transformer [46]	62.2
<i>RNNs:</i>	
LSTM [21]	63.01
LipschitzRNN [15]	64.2
r-LSTM [45]	72.2
<i>State Space Models:</i>	
S4D [23]	89.92
S5 [43]	90.10
S4 [22]	91.80
Liquid-S4 [25]	92.02
<i>Convolutional Models:</i>	
CKConv [40]	63.74
TrellisNet [5]	73.42
FlexConv [39]	80.82
Long Conv, Geom Init [17]	92.1
Long Conv, Learnable Butterfly [17]	92.5
CCNN [27]	93.08
MultiresNet [42]	93.15
MRConv, Dilated	94.26
MRConv, Fourier	92.67
MRConv, Fourier+Sparse	92.40

Table 12: **Test accuracy on sCIFAR 1D Image classification.** Citations refer to the original model where the baseline values is taken from otherwise additional citations indicates work in which the baseline value is reported.

D.4.1 Model Depth Ablation

We perform an ablation study comparing models with increasing depth on sCIFAR with dilated convolutions. Table 13, shows that increasing the model depth has a positive impact on model performance.

Model	Layers	Parameters	sCIFAR
MRConv, Dilated	6	3.4M	93.57
	8	4.6M	94.03
	10	5.7M	94.26

Table 13: **Depth ablation study on sCIFAR dataset.** Increasing the number of layers has a positive effect on model performance.

D.5 Raw Speech Classification

The task is to classify one of 35 words from the dataset of 1s audio recordings, released by [47], sampled at 16kHz. Sequences are all of the same length (16,000). The zero-shot classification task is constructed by temporally downsampling the validation and test datasets using naive decimation, reducing the signal length to 8,000. There are 24,482 training examples, 5,246 validation examples and 5,247 test examples. Following [43], the data is normalized to be zero-mean and have a standard deviation of 0.2. Table 14 reports citations for each of the baseline methods.

Model (Input length)	16 kHz (16,000)	8 kHz (8,000)
<i>CNNs:</i>		
InceptionNet [36]	61.24	05.18
ResNet-18 [36]	77.86	08.74
XResNet-50 [36]	83.01	07.72
ConvNet [36]	95.51	07.26
<i>State Space Models:</i>		
S4D-LegS [23]	95.83	91.08
S4-LegS [22]	96.08	91.32
Liquid-S4 [25]	<u>96.78</u>	90.00
S5 [43]	96.52	<u>94.53</u>
<i>Convolutional Models:</i>		
SGConv [29]	96.42	94.29
MRConv, Dilated	96.47	-
MRConv, Fourier	96.82	95.05
MRConv, Fourier+Sparse	95.21	-

Table 14: **Test accuracy on 35-way Speech Commands classification task [47].** Each model is trained on one-second 16kHz audio waveforms and then tested at 16kHz and 0-shot at 8kHz. Results for the baselines are reported in [23].

D.6 ImageNet Classification

We use the same training settings as in the original ConvNeXt work by Liu et al. [32]. ConvNeXt uses several downsampling layers and hence the feature maps once flattened correspond to sequences of length [3136, 784, 196, 49] at each stage. When setting the initial kernel size, we take inspiration from RepLKNet [14] which reparameterizes a small 5×5 kernel into a larger one, and hence we ensure our smallest kernel size is ≥ 25 and use no more than 5 resolutions. Hence, for each stage we used kernel lengths [196, 49, 28, 28]. We used $8 \times \text{V100}$ for training. We provide an extended results table below.

Model	2D Bias	Params	FLOPs	Throughput (Image/s)	Top 1 Acc.
ConvNeXt-T [32]	✓	29M	4.5G	1306.3	82.1
Swin-T [31]	✓	29M	4.5G	1077.4	81.3
SGConvNeXt-T [29]	✗	29M	4.3G	819.7	82.0
MRConvNeXt-T, Fourier	✗	30M	4.3G	1080.3	82.1
MRConvNeXt-T, Fourier + Sparse	✗	32M	4.3G	1080.3	82.2
ConvNeXt-S [32]	✓	50M	8.7G	902.6	83.1
Swin-S [31]	✓	50M	8.7G	674.5	83.0
SGConvNeXt-S [29]	✗	51M	8.3G	500.0	83.4
MRConvNeXt-S, Fourier	✗	53M	8.3G	739.1	83.3
MRConvNeXt-S, Fourier + Sparse	✗	57M	8.3G	739.1	83.4
ConvNeXt-B [32]	✓	89M	15.4G	460.9	83.8
Swin-B [31]	✓	88M	15.4G	470.9	83.5
SGConvNeXt-B [29]	✗	90M	14.6G	330.4	83.9
MRConvNeXt-B, Fourier	✗	93M	14.6G	478.7	83.8
MRConvNeXt-B, Fourier + Sparse	✗	98M	14.6G	478.7	83.9

Table 15: **Extended ImageNet classification results.** Other than throughput results, which we computed ourselves, baseline results were taken from the cited paper.

D.6.1 FLOPs Calculation

We provide a further theoretical analysis on the number of FLOPs required to compute a 1D FFT convolution and a regular 2D convolution. In all comparisons, we will use a batch size of 1 and a channel dimension of 1. An FFT convolution involves: i) a forward FFT of the input, ii) an elementwise product between the input and the kernel, and iii) an inverse FFT of the resulting convolution. The total time complexity is $\mathcal{O}(2L \log L + L)$. In practice, we use the real-valued FFT implemented via the Cooley-Tukey algorithm, which uses FLOPs [3]. On the other hand, regular 2D depthwise separable convolutions with a kernel size of k have a complexity of $\mathcal{O}(k^2 HW)$, requiring $2k^2 HW$ FLOPs with 2 FLOPs needed for each multiply-accumulate operation.

Next, we calculate the number of FLOPs to compute a 2D 7×7 convolution, as utilized in ConvNeXt and a global 1D convolution as utilized in MRConv, for a sequence of increasingly larger (flattened) images. Our results show that our 1D convolutions use fewer FLOPs than the comparable 2D convolution used by ConvNeXt. We note that MRConv uses the same number of FLOPs as SGConv [29].

Conv Type	Conv Size	(16,16) (256)	(32, 32) (1024)	(64, 64) (4096)	(128, 128) (16,384)	(256, 256) (65,536)	(512, 512) (262,144)
2D	7×7	25.1K	100K	401K	1.61M	6.42M	25.7M
1D	7×7	8.7K	43.0K	205K	0.95M	4.32M	19.4M

Table 16: **Theoretical FLOPs.** Comparison of theoretical FLOPs between 2D convolutions and 1D convolutions implemented using the FFT. We display both the 2D image size and the length of the equivalent flattened 1D image.

D.6.2 Throughput

The throughput is measured as the number of images processed per second on a single 24GB 3090 GPU, by using the largest batch size that could fit in memory for a given model and averaging over 50,000 test images.

We include additional results comparing the throughput of MRConvNeXt using both PyTorch implemented FFT convolutions and optimized CUDA FFT convolutions from [17]. Use of optimized CUDA kernels results in a 41.5% throughput increase on average. We also include an enlarged version of Figure 3a which plots Top-1 accuracy against throughput.

Model	FFT Conv. Imp.	Throughput (Image/s)
MRConvNeXt-T	PyTorch	819.7
	CUDA	1080.3
MRConvNeXt-S	PyTorch	500.0
	CUDA	739.1
MRConvNeXt-B	PyTorch	330.4
	CUDA	478.7

Table 17: **ImageNet classification throughput comparison.** We compare the throughput of MRConvNeXt when using PyTorch implemented FFT convolutions and fast CUDA implemented FFT convolutions [17]

D.6.3 Visualization of MRConvNeXt-T kernels

In Figure 7 we provide visualization of the reparameterized convolution kernels from MRConvNeXt-T for both Fourier and Fourier + Sparse parameterizations.

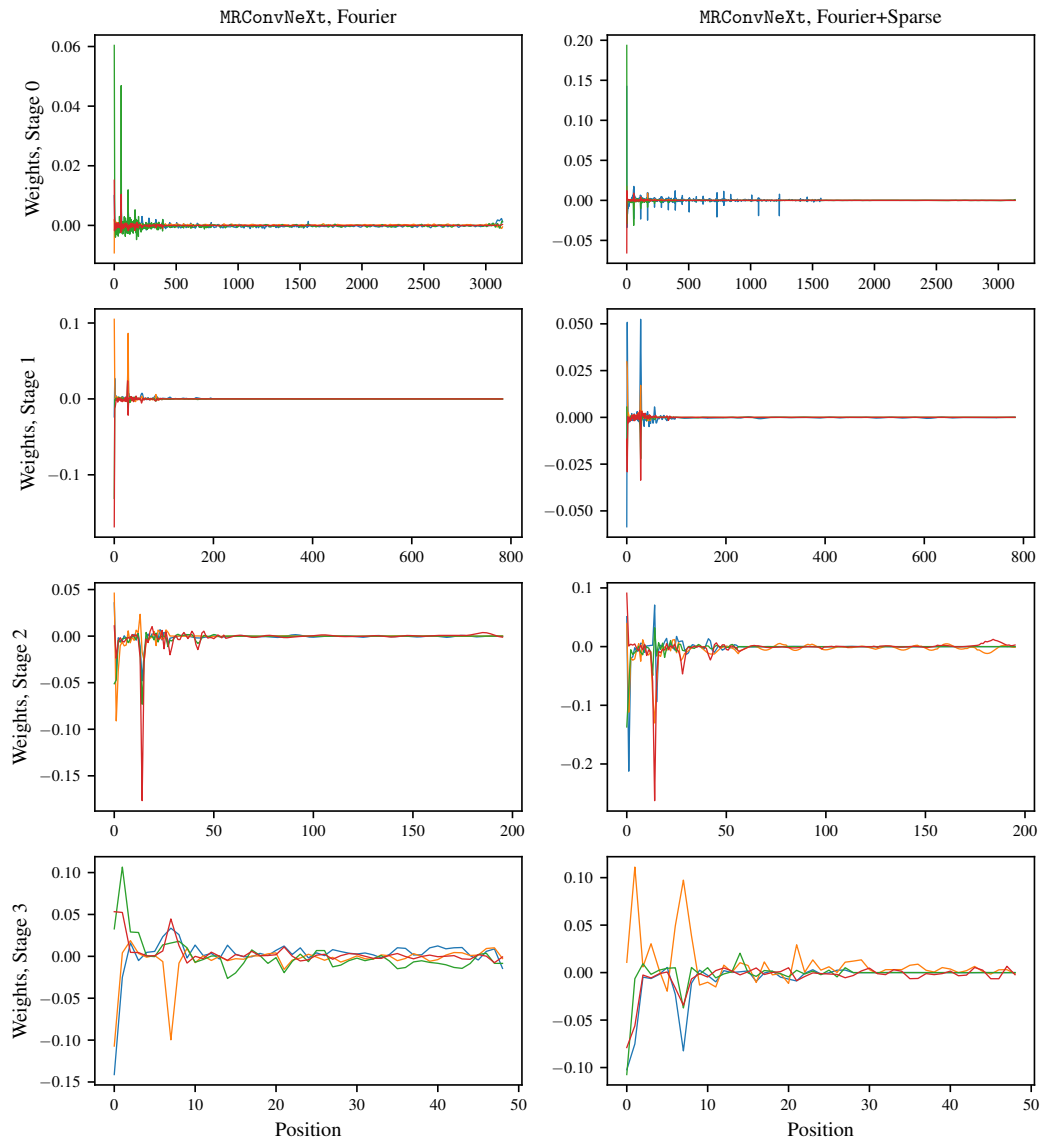


Figure 7: Visualization of learned Kernels from MRConvNeXt at different stages for both Fourier and Fourier + Sparse parameterizations.

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope?

Answer: [Yes]

Justification: In the paper, we claim that we introduce a new multi-resolution convolution model that both outperforms current attention-free methods and linear-time Transformers and is more efficient than SSMs. We support these claims through extensive experimentation on benchmark

datasets, including Long Range Arena (a standard benchmark dataset for long sequence models), sequential CIFAR, Speech Commands and ImageNet classification. To ensure fair comparison we ensure similar computational complexities and parameter counts when comparing models. We also provide several ablation studies supporting how our architectural features improve performance over standard baselines.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: We discuss the limitations of our model in the Section 6, clearly outlining the increased computational resources required for training and the lack of input dependency required for language modelling, proposing solutions to these problems as future work. We also include an extra ablations section in the appendix for the Path-X dataset, where we show that learning the decay rate can cause overfitting on this specific task.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory Assumptions and Proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper does not include any proofs. We do however outline the mathematical derivation of our model as clearly as possible in the main paper.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental Result Reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We ensure that all experimental results are reproducible by providing tables with hyperparameters in Appendix D.2 and references to prior works from which we use the same experimental setup. We also include a clear diagram of our architecture in Figure 1, algorithmic details in Appendix C and mathematical notation throughout the main paper to reproduce our model.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
 - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
 - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We use open-sourced datasets and release the code used to run our experiments.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental Setting/Details

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: We ensure that all experimental results are reproducible by providing tables with hyperparameters in Appendix D.2 and include specific optimizer settings and task specific training details in the Appendix.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment Statistical Significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: We don't provide error bars as we were working with a strict computational budget, which didn't enable us to run hyper-parameter sweeps over multiple seeds. On the larger ImageNet experiments, we lacked the computational resources to do any hyper-parameter tuning at all and were only able to perform a single training run for each model.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments Compute Resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: In the Appendix we provide details on the hardware used to run each experiment.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code Of Ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines>?

Answer: [Yes]

Justification: We read the guidelines and to the best of our knowledge our work complies with them.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader Impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA]

Justification: Our work develops a new sequence model with no direct societal impact.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: Our work does not have a high for misuse.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cite all datasets and code used.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. New Assets

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: We don't release new assets with our work.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: Our work does not involve crowdsourcing nor research with human subjects

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: Our work does not involve crowdsourcing nor research with human subjects

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.