Appendix for: Convolutional State Space Models for Long-range Spatiotemporal Modeling

Contents:

- **Appendix A**: Propositions
- **Appendix B**: ConvS5 Details: Parameterization, Initialization, Discretization
- **Appendix C**: Supplementary Results
- **Appendix D**: Experiment Configurations
- **Appendix E**: Datasets

A Propositions

A.1 Parallel Scan for Convolutional Recurrences

Proposition 1. *Consider a convolutional recurrence as in (7) and define initial parallel scan elements* $c_k = (c_{k,a}, c_{k,b}) := (\overline{\boldsymbol{\mathcal{A}}}, \overline{\boldsymbol{\mathcal{B}}} * \boldsymbol{\mathcal{U}}_k)$*. The binary operator* $\circledast$*, defined below, is associative.*

$$c_i \circledast c_j := (c_{j,a} \circ c_{i,a},\ c_{j,a} * c_{i,b} \ +\ c_{j,b}), \tag{14}$$

where $\circ$ *denotes convolution of two kernels,* $*$ *denotes convolution between a kernel and input, and* $+$ *is elementwise addition.*

Proof. Using that $\circ$ is associative and the companion operator of $*$, i.e. $(d \circ e) * f = d * (e * f)$ (see Blelloch [61], Section 1.4), we have:

$$\begin{aligned}
(c_i \circledast c_j) \circledast c_k &= (c_{j,a} \circ c_{i,a},\ c_{j,a} * c_{i,b} \ +\ c_{j,b}) \circledast (c_{k,a}, c_{k,b}) && (15)\\
&= \Big(c_{k,a} \circ (c_{j,a} \circ c_{i,a}),\ \ c_{k,a} * (c_{j,a} * c_{i,b} + c_{j,b}) + c_{k,b}\Big) && (16)\\
&= \Big((c_{k,a} \circ c_{j,a}) \circ c_{i,a},\ \ c_{k,a} * (c_{j,a} * c_{i,b}) + c_{k,a} * c_{j,b} + c_{k,b}\Big) && (17)\\
&= \Big((c_{k,a} \circ c_{j,a}) \circ c_{i,a},\ \ (c_{k,a} \circ c_{j,a}) * c_{i,b} + c_{k,a} * c_{j,b} + c_{k,b}\Big) && (18)\\
&= c_i \circledast (c_{k,a} \circ c_{j,a},\ \ c_{k,a} * c_{j,b} + c_{k,b}) && (19)\\
&= c_i \circledast (c_j \circledast c_k) && (20)
\end{aligned}$$

□

A.2 Computational Cost of Parallel Scan for Convolutional Recurrences

Proposition 2. *Given the effective inputs* $\overline{\boldsymbol{\mathcal{B}}} * \boldsymbol{\mathcal{U}}_{1:L} \in \mathbb{R}^{L \times H \times W \times P}$ *and a pointwise state kernel* $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{P \times P \times 1 \times 1}$*, the computational cost of computing the convolutional recurrence in Equation 7 with a parallel scan is* $\mathcal{O}\big(L(P^3 + P^2 HW)\big)$.

Proof. Following Blelloch [61], given a single processor, the cost of computing the recurrence sequentially using the binary operator $\circledast$ defined in Proposition 1 is $\mathcal{O}\big(L(T_\circ + T_* + T_+)\big)$ where $T_\circ$ refers to the cost of convolving two kernels, T_* is the cost of convolution between a kernel and input and T_+ is the cost of elementwise addition. The cost of elementwise addition is $T_+ = \mathcal{O}(PHW)$. For state kernels with resolution k_A, $T_\circ = \mathcal{O}(P^3 k_A^4)$ and $T_* = \mathcal{O}(P^2 k_A^2 HW)$. For pointwise convolutions this becomes $T_\circ = \mathcal{O}(P^3)$ and $T_* = \mathcal{O}(P^2 HW)$. Thus, the cost of computing the recurrence sequentially using $\circledast$ is $\mathcal{O}\big(L(P^3 + P^2 HW)\big)$. Since there are work-efficient algorithms for parallel scans [104], the overall cost of the parallel scan is also $\mathcal{O}\big(L(P^3 + P^2 HW)\big)$. □

Note that ConvS5's diagonalized parameterization discussed in Section 3.4 and Appendix B leads to $T_\circ = \mathcal{O}(P)$ and $T_* = \mathcal{O}(PHW)$. Therefore the cost of applying the parallel scan with ConvS5 is $\mathcal{O}(LPHW)$.

A.3 Connection Between ConvSSMs and SSMs

Proposition 3. *Consider a ConvSSM state update as in (5) with pointwise state kernel* $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{P \times P \times 1 \times 1}$*, input kernel* $\boldsymbol{\mathcal{B}} \in \mathbb{R}^{P \times U \times k_B \times k_B}$*, and input* $\boldsymbol{\mathcal{U}}(t) \in \mathbb{R}^{H' \times W' \times U}$*. Let* $\boldsymbol{\mathcal{U}}_{\text{im2col}}(t) \in \mathbb{R}^{H \times W \times U k_B^2}$ *be the reshaped result of applying the Image to Column (im2col) [64, 65] operation on the input* $\boldsymbol{\mathcal{U}}(t)$*. Then the dynamics of each state pixel of* (5)*,* $\boldsymbol{\mathcal{X}}(t)_{i,j} \in \mathbb{R}^P$*, evolve according to the following differential equation*

$$\boldsymbol{\mathcal{X}}'(t)_{i,j} = \mathbf{A}_{\text{SSM}} \boldsymbol{\mathcal{X}}(t)_{i,j} + \mathbf{B}_{\text{SSM}} \boldsymbol{\mathcal{U}}_{im2col}(t)_{i,j} \tag{21}$$

where the state matrix, $\mathbf{A}_{\text{SSM}} \in \mathbb{R}^{P \times P}$*, and input matrix,* $\mathbf{B}_{\text{SSM}} \in \mathbb{R}^{P \times (U k_B^2)}$*, can be formed by reshaping the state kernel,* $\boldsymbol{\mathcal{A}}$*, and input kernel,* $\boldsymbol{\mathcal{B}}$*, respectively.*

Proof. Let $\mathbf{U}_{\text{im2col}} \in \mathbb{R}^{Uk_b^2 \times HW}$ denote the result of performing the im2col operation on the input $\boldsymbol{\mathcal{U}}(t)$ for convolution with the kernel $\boldsymbol{\mathcal{B}}$. Reshape this matrix into the tensor $\boldsymbol{\mathcal{U}}_{\text{im2col}}(t) \in \mathbb{R}^{H \times W \times Uk_b^2}$. Reshape $\boldsymbol{\mathcal{U}}_{\text{im2col}}(t)$ once more into the tensor $\boldsymbol{\mathcal{V}}(t) \in \mathbb{R}^{H \times W \times U \times k_B \times k_B}$.

Now, we can write the evolution for the individual channels of each pixel, $\boldsymbol{\mathcal{X}}'(t)_{i,j,k}$, in (5) as

$$\boldsymbol{\mathcal{X}}'(t)_{i,j,k} = \sum_{l=0}^{P-1} \boldsymbol{\mathcal{A}}_{k,l,0,0} \boldsymbol{\mathcal{X}}(t)_{i,j,l} + \sum_{q=0}^{U-1} \sum_{m=0}^{k_B-1} \sum_{n=0}^{k_B-1} \boldsymbol{\mathcal{B}}_{k,q,m,n} \boldsymbol{\mathcal{V}}(t)_{i,j,q,m,n}. \tag{22}$$

Let $\mathbf{A}_{\text{SSM}} \in \mathbb{R}^{P \times P}$ be a matrix with rows, $\mathbf{A}_{\text{SSM},i} \in \mathbb{R}^P$, corresponding to a flattened version of the output features of $\boldsymbol{\mathcal{A}}$, i.e. $\boldsymbol{\mathcal{A}}_i \in \mathbb{R}^{P \times 1 \times 1}$. Similarly, reshape $\boldsymbol{\mathcal{B}}$ into a matrix $\mathbf{B}_{\text{SSM}} \in \mathbb{R}^{P \times (Uk_B^2)}$ where the rows, $\mathbf{B}_{\text{SSM},i} \in \mathbb{R}^{Uk_B^2}$ correspond to a flattened version of the output features of $\boldsymbol{\mathcal{B}}$, i.e. $\boldsymbol{\mathcal{B}}_i \in \mathbb{R}^{U \times k_B \times k_B}$.

Then we can rewrite (22) equivalently as

$$\begin{aligned}
\boldsymbol{\mathcal{X}}'(t)_{i,j,k} &= \sum_{l=0}^{P-1} \boldsymbol{\mathcal{A}}_{k,l,0,0} \boldsymbol{\mathcal{X}}(t)_{i,j,l} + \sum_{q=0}^{U-1} \sum_{m=0}^{k_B-1} \sum_{n=0}^{k_B-1} \boldsymbol{\mathcal{B}}_{k,q,m,n} \boldsymbol{\mathcal{V}}(t)_{i,j,q,m,n} && (23)\\
&= \sum_{l=0}^{P-1} \mathbf{A}_{\text{SSM},k,l} \boldsymbol{\mathcal{X}}(t)_{i,j,l} + \sum_{v=0}^{Uk_B^2-1} \mathbf{B}_{\text{SSM},k,v} \boldsymbol{\mathcal{U}}_{\text{im2col}}(t)_{i,j,v} && (24)\\
&= \mathbf{A}_{\text{SSM},k}^T \boldsymbol{\mathcal{X}}(t)_{i,j} + \mathbf{B}_{\text{SSM},k}^T \boldsymbol{\mathcal{U}}_{\text{im2col}}(t)_{i,j} && (25)
\end{aligned}$$

□

B ConvS5 Details: Parameterization, Discretization, Initialization

B.1 Background: S5

S5 Parameterization and Discretization S5 [20] uses a diagonalized parameterization of the general SSM in (3).

Let $\mathbf{A}_{\text{S5}} = \mathbf{V}\mathbf{\Lambda}_{\text{S5}}\mathbf{V}^{-1} \in \mathbb{R}^{P\times P}$ where $\mathbf{\Lambda}_{\text{S5}} \in \mathbb{C}^{P\times P}$ is a complex-valued diagonal matrix and $\mathbf{V} \in \mathbb{C}^{P\times P}$ corresponds to the eigenvectors. Defining $\tilde{\mathbf{x}}(t) = \mathbf{V}^{-1}\mathbf{x}(t)$, $\tilde{\mathbf{B}} = \mathbf{V}^{-1}\mathbf{B}$, and $\tilde{\mathbf{C}} = \mathbf{C}\mathbf{V}$ we can reparameterize the SSM of (3) as the diagonalized system:

$$\frac{\mathrm{d}\tilde{\mathbf{x}}(t)}{\mathrm{d}t} = \mathbf{\Lambda}_{\text{S5}}\tilde{\mathbf{x}}(t) + \tilde{\mathbf{B}}\mathbf{u}(t), \qquad \mathbf{y}(t) = \tilde{\mathbf{C}}\tilde{\mathbf{x}}(t) + \mathbf{D}\mathbf{u}(t). \tag{26}$$

S5 uses learnable timescale parameters $\mathbf{\Delta} \in \mathbb{R}^P$ and the following zero-order hold (ZOH) disretization:

$$\overline{\mathbf{\Lambda}}_{\text{S5}} = \text{DISCRETIZE}_{\text{A}}(\mathbf{\Lambda}_{\text{S5}}, \mathbf{\Delta}) := e^{\mathbf{\Lambda}_{\text{S5}}\mathbf{\Delta}} \tag{27}$$

$$\overline{\mathbf{B}}_{\text{S5}} = \text{DISCRETIZE}_{\text{B}}(\mathbf{\Lambda}_{\text{S5}}, \tilde{\mathbf{B}}, \mathbf{\Delta}) := \mathbf{\Lambda}_{\text{S5}}^{-1}(\overline{\mathbf{\Lambda}}_{\text{S5}} - \mathbf{I})\tilde{\mathbf{B}} \tag{28}$$

S5 Initialization S5 initializes its state matrix by diagonalizing $\mathbf{A}_{\text{S5}}$ as defined here:

$$\mathbf{A}_{\text{S5}_{nk}} = -\begin{cases} (n+\frac{1}{2})^{1/2}(k+\frac{1}{2})^{1/2}, & n > k \\ \frac{1}{2}, & n = k \\ (n+\frac{1}{2})^{1/2}(k+\frac{1}{2})^{1/2}, & n < k \end{cases}. \tag{29}$$

This matrix is the normal part of the normal plus low-rank HiPPO-LegS matrix from the HiPPO framework [53] for online function approximation. S4 originally initialized its single-input, single-output (SISO) SSMs with a representation of the full HiPPO-LegS matrix. This was shown to be approximating long-range dependencies at initialization with respect to an infinitely long, exponentially-decaying measure [105]. Gupta et al. [41] empirically showed that the low-rank terms could be removed without impacting performance. Gu et al. [42] showed that in the limit of infinite state dimension, the linear, single-input ODE with this normal approximation to the HiPPO-LegS matrix produces the same dynamics as the linear, single-input ODE with the full HiPPO-LegS matrix. The S5 work extended these findings to the multi-input SSM setting [20].

Importance of SSM Parameterization, Discretization and Initialization Prior research has highlighted the importance of parameterization, discretization and initialization choices of deep SSM methods through ablations and analysis [56, 19, 42, 20, 57]. Concurrent work from Orvieto et al. [57] provides particular insight into the favorable initial eigenvalue distributions provided by initializing with HiPPO-inspired matrices as well as an important normalization effect provided by the explicit discretization procedure. They also introduce a purely discrete-time parameterization that can perform similarly to the continuous-time discretization of S4 and S5. However, their parameterization practically ends up quite similar to the equations of (27-28). We choose to use the continuous-time parameterization of S5 for the implicit parameterization of ConvS5 since it can also be leveraged for zero-shot resolution changes [19, 20, 45] and processing irregularly sampled time-series in parallel [20]. However, due to Proposition 3, other long-range SSM parameterization strategies can also be used, such as in Orvieto et al. [57] or potential future innovations.

B.2 ConvS5 Diagonalization

We leverage S5's diagonalized parameterization to reduce the cost of the parallel scan of ConvS5.

Concretely, we initialize $\mathbf{A}_{\text{S5}}$ as in (29) and diagonalize as $\mathbf{A}_{\text{S5}} = \mathbf{V}\mathbf{\Lambda}_{\text{S5}}\mathbf{V}^{-1}$. To apply ConvS5, we compute $\overline{\mathbf{\Lambda}}_{\text{S5}}$ and $\overline{\mathbf{B}}_{\text{S5}}$ using (27-28), and then form the ConvS5 state and input kernels:

$$\overline{\mathbf{\Lambda}}_{\text{S5}} \in \mathbb{R}^{P\times P} \xrightarrow{\text{reshape}} \overline{\boldsymbol{\mathcal{A}}}_{\text{ConvS5}} \in \mathbb{R}^{P\times P\times 1\times 1} \tag{30}$$

$$\overline{\mathbf{B}}_{\text{S5}} \in \mathbb{R}^{P\times (Uk_B^2)} \xrightarrow{\text{reshape}} \overline{\boldsymbol{\mathcal{B}}}_{\text{ConvS5}} \in \mathbb{R}^{P\times U\times k_B\times k_B}. \tag{31}$$

See Listing 1 for an example of the core implementation. Note, the state kernel $\overline{\boldsymbol{\mathcal{A}}}_{\text{ConvS5}}$ is "diagonalized" in the sense that all entries in the state kernel are zero except $\overline{\boldsymbol{\mathcal{A}}}_{\text{ConvS5},i,i} = \overline{\mathbf{\Lambda}}_{\text{S5},i,i} \;\; \forall i \in [P]$.

This means that the pointwise convolutions reduce to channel-wise multiplications. However, this does not reduce expressivity compared to a full pointwise convolution. This is because, given the ConvSSM to SSM equivalence of Proposition 3 and the use of complex-valued kernels, the diagonalization maintains expressivity since almost all SSMs are diagonalizable [41, 42], which follows from the well-known fact that almost all square matrices diagonalize over the complex plane.

```python
import jax
import jax.numpy as np
from ConvSSM_helpers import discretize, Conv2D, ResNet_Block
parallel_scan = jax.lax.associative_scan

def apply_ConvS5_layer(A, B, B_shape, C_kernel, log_Delta, resnet_params, us, x0):
    """Compute the outputs of ConvS5 layer given input sequence.
    Args:
        A (complex64): S5 state matrix                  (P,)
        B (complex64): S5 input matrix                  (P,Uk_B^2)
        B_shape (tuple): shape of B_kernel
        C_kernel (complex64) output kernel              (U,P,k_C,k_C)
        log_Delta (float32): learnable timescale params (P,)
        resnet_params (dict): ResNet block params
        us (float32): input sequence of features        (L,bsz,H,W,U)
        x0 (complex64): initial state                   (bsz,H,W,P)
    Returns:
        outputs (float32): the ConvS5 layer outputs     (L,bsz,H,W,U)
        x_L (complex64): the last state of the ConvSSM  (bsz,H,W,P)
    """
    # Discretize and reshape ConvS5 state and input kernels
    P, U, k_B = B_shape
    A_bar, B_bar = discretize(A, B, np.exp(log_Delta))
    A_kernel = A_bar   # already correct shape due to diagonalization
    B_kernel = B_bar.reshape(P, U, k_B, k_B)

    # Apply ConvS5
    ys, xs = apply_ConvS5(A_kernel, B_kernel, C_kernel, us, x0)

    # Apply ResNet activation function
    outputs = jax.vmap(ResNet_Block, axis=(None,0))(resnet_params, ys)
    return outputs, xs[-1]

def apply_ConvS5(A_kernel, B_kernel, C_kernel, us, x0):
    """Compute the output sequence of the convolutional SSM
        given the input sequence using a parallel scan.
        Computes x_k = A * x_{k-1} + B * u_k
                 y_k = C * x_k
        where * is a convolution operator.
    Args:
        A_kernel (complex64): state kernel   (P,)
        B_kernel (complex64): input kernel   (P,U,k_B,k_B)
        C_kernel (complex64): output kernel  (U,P,k_C,k_C)
        us (float32): input sequence         (L,bsz,H,W,U)
        x0 (complex64): initial state        (bsz,H,W,P)
    Returns:
        ys (float32): the convS5 outputs     (L,bsz,H,W,U)
        x_L (complex64): the last state      (bsz,H,W,P)
    """
    # Compute initial scan elements
    As = np.repeat(A_kernel[None, ...], us.shape[0], axis=0)
    Bus = jax.vmap(Conv2D)(B_kernel, np.complex64(us))
    Bus = Bus.at[0].add(np.expand_dims(A_bar, (0, 1, 2)) * x0)

    # Convolutional recurrence with parallel scan
    _, xs = parallel_scan(conv_binary_operator, (As, Bus))

    # Compute ConvS5 outputs
    ys = jax.vmap(Conv2D)(C_kernel, xs).real
    return ys, xs

def conv_binary_operator(q_i, q_j):
    """Binary operator for convolutional recurrence
       with "diagonalized" 1X1 state kernels.
    Args:
       q_i, q_j (tuples): scan elements q_i=(A_i, BU_i) where
                 A_i (complex64) is state kernel     (P,)
                 BU_i (complex64) is effective input (bsz,H,W,P)
    Returns:
        output tuple q_i \circledast q_j
    """
    A_i, BU_i = q_i
    A_j, BU_j = q_j
    # Convolve "diagonal" 1X1 kernels
    AA = A_j * A_i
    # Convolve "diagonal" A_j with BU_i
    A_jBU_i = np.expand_dims(A_j, (0, 1, 2)) * BU_i
    return AA, A_jBU_i + BU_j
```

Listing 1: JAX implementation of core code to apply a single ConvS5 layer to a batch of spatiotemporal input sequences.

C Supplementary Results

We include expanded tables and sample trajectories from the experiments in the main paper. Sample videos can be found at the anonymized website:

https://sites.google.com/view/convssm.

C.1 Moving-MNIST

Table 5: Full results on the Moving-Mnist dataset [54]. For the top table, all models are trained on 300 frames. For the bottom table, all models are trained on 600 frames. The evaluation task is to condition on 100 frames, and then generate forward 400, 800 and 1200 frames. ConvSSM (ablation) is performed by randomly initializing the state kernel (see Section 5.3 and Appendix D.4).

Trained on 300 frames

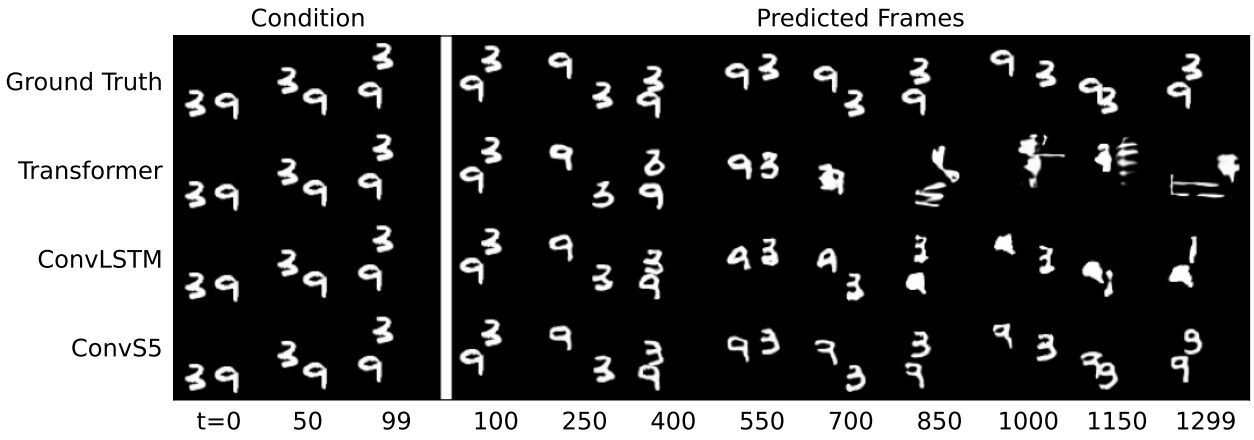
		$100 \rightarrow 400$				$100 \rightarrow 800$				$100 \rightarrow 1200$			
Method	Params	FVD ↓	PSNR ↑	SSIM ↑	LPIPS ↓	FVD ↓	PSNR ↑	SSIM ↑	LPIPS ↓	FVD ↓	PSNR ↑	SSIM ↑	LPIPS ↓
Transformer	164M	73 ± 3	13.5 ± 0.1	0.669 ± 0.002	0.213 ± 0.003	159 ± 7	12.6 ± 0.1	0.609 ± 0.002	0.287 ± 0.001	265 ± 8	12.4 ± 0.1	0.591 ± 0.002	0.321 ± 0.002
ConvLSTM	43M	57 ± 3	16.9 ± 0.2	0.796 ± 0.004	0.113 ± 0.002	128 ± 4	15.0 ± 0.1	0.737 ± 0.003	0.169 ± 0.001	$\mathbf{187 \pm 6}$	14.1 ± 0.1	$\mathbf{0.706 \pm 0.003}$	$\mathbf{0.203 \pm 0.001}$
ConvSSM (ablation)	41M	67 ± 3	15.5 ± 0.1	0.742 ± 0.001	0.168 ± 0.001	287 ± 5	13.6 ± 0.1	0.577 ± 0.001	0.293 ± 0.001	511 ± 8	13.3 ± 0.1	0.515 ± 0.001	0.348 ± 0.001
ConvS5	41M	$\mathbf{26 \pm 1}$	$\mathbf{18.1 \pm 0.1}$	$\mathbf{0.830 \pm 0.003}$	$\mathbf{0.094 \pm 0.002}$	$\mathbf{72 \pm 3}$	$\mathbf{16.0 \pm 0.1}$	$\mathbf{0.761 \pm 0.005}$	$\mathbf{0.156 \pm 0.003}$	$\mathbf{187 \pm 5}$	$\mathbf{14.5 \pm 0.1}$	0.678 ± 0.003	0.230 ± 0.004

Trained on 600 frames

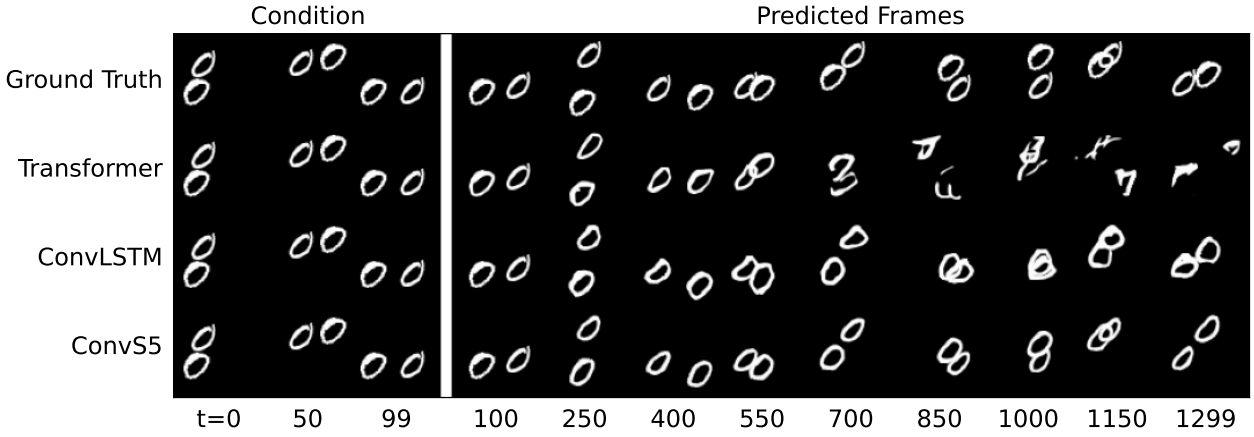
		$100 \rightarrow 400$				$100 \rightarrow 800$				$100 \rightarrow 1200$			
Method	Params	FVD ↓	PSNR ↑	SSIM ↑	LPIPS ↓	FVD ↓	PSNR ↑	SSIM ↑	LPIPS ↓	FVD ↓	PSNR ↑	SSIM ↑	LPIPS ↓
Transformer	164M	$\mathbf{21 \pm 1}$	15.0 ± 0.1	0.741 ± 0.002	0.138 ± 0.001	$\mathbf{42 \pm 2}$	13.7 ± 0.1	0.672 ± 0.002	0.207 ± 0.003	91 ± 6	13.1 ± 0.1	0.631 ± 0.004	0.252 ± 0.002
ConvLSTM	43M	39 ± 5	17.3 ± 0.2	0.812 ± 0.005	0.100 ± 0.003	91 ± 7	15.5 ± 0.2	0.757 ± 0.005	0.149 ± 0.003	137 ± 9	14.6 ± 0.1	0.727 ± 0.004	0.180 ± 0.003
ConvSSM (ablation)	41M	81 ± 6	15.5 ± 0.1	0.743 ± 0.002	0.163 ± 0.003	145 ± 8	14.3 ± 0.1	0.696 ± 0.002	0.218 ± 0.002	215 ± 9	13.4 ± 0.1	0.614 ± 0.001	0.287 ± 0.001
ConvS5	41M	23 ± 3	$\mathbf{18.1 \pm 0.1}$	$\mathbf{0.832 \pm 0.003}$	$\mathbf{0.092 \pm 0.003}$	47 ± 7	$\mathbf{16.4 \pm 0.1}$	$\mathbf{0.788 \pm 0.002}$	$\mathbf{0.134 \pm 0.003}$	$\mathbf{71 \pm 9}$	$\mathbf{15.6 \pm 0.1}$	$\mathbf{0.763 \pm 0.002}$	$\mathbf{0.162 \pm 0.003}$

Table 6: Model runtime comparison for Moving-MNIST results in Table 5. ConvS5 can be parallelized like a Transformer but maintains the constant cost-per-step autoregressive generation of ConvRNNs.

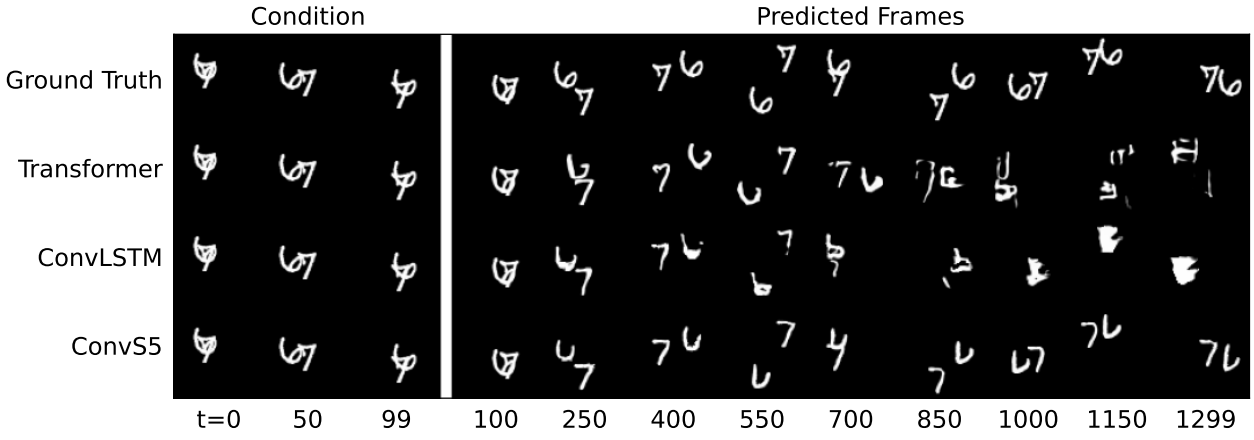
Method	Train Step Time (s) ↓	$100 \rightarrow 400$ Sampling Speed (frames/s) ↑	$100 \rightarrow 800$ Sampling Speed (frames/s) ↑	$100 \rightarrow 1200$ Sampling Speed (frames/s) ↑
Transformer	$\mathbf{0.77}$ $(\mathbf{1.0}\times)$	1.1 $(1.0\times)$	0.34 $(1.0\times)$	0.21 $(1.0\times)$
ConvLSTM	3.0 $(3.9\times)$	$\mathbf{117}$ $(\mathbf{106}\times)$	$\mathbf{117}$ $(\mathbf{345}\times)$	$\mathbf{117}$ $(\mathbf{557}\times)$
ConvS5	0.93 $(1.2\times)$	90 $(82\times)$	90 $(265\times)$	90 $(429\times)$



(a) Example 1



(b) Example 2



(c) Example 3

Figure 3: Moving-MNIST Samples: 1200 frames generated conditioned on 100.

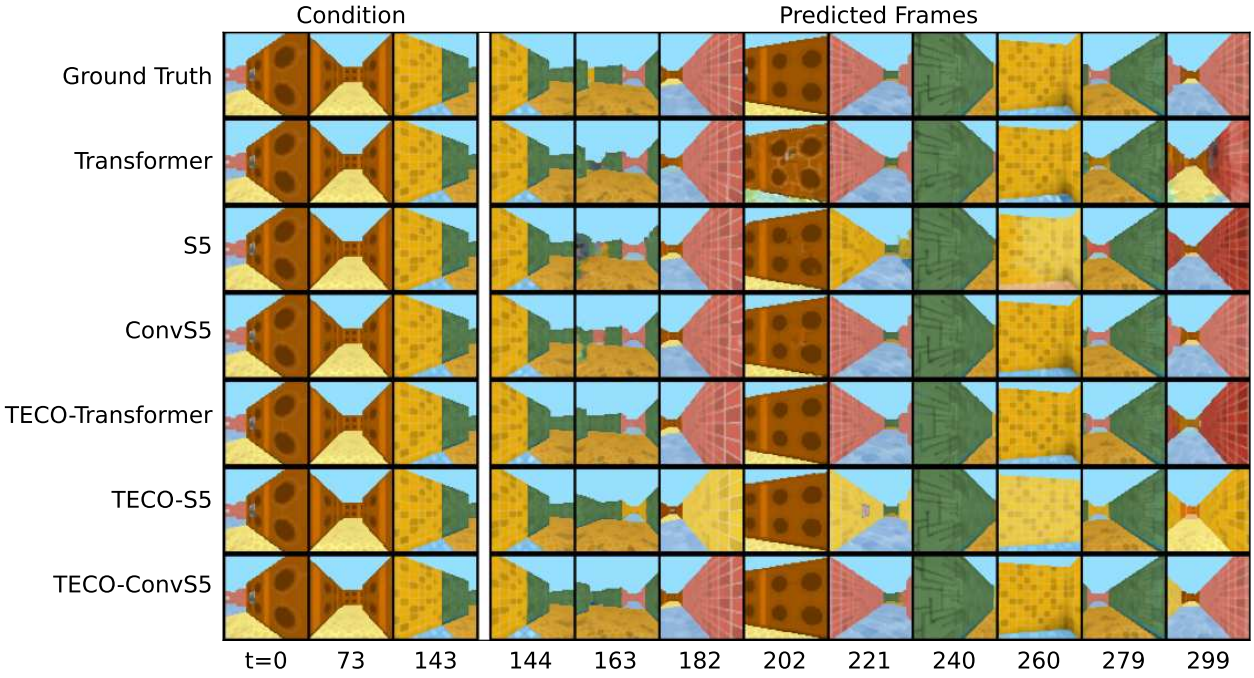
C.2 3D Environments

Table 7: Full results for DMLab long-range benchmark dataset [13]. Results from Yan et al. [13] are indicated with $*$. We separate out the methods trained using the TECO [13] training framework in the bottom of the table. TECO-ConvSSM (ablation) refers to the ablation performed by randomly initializing the state kernel (see Section 5.3 and Appendix D.5).

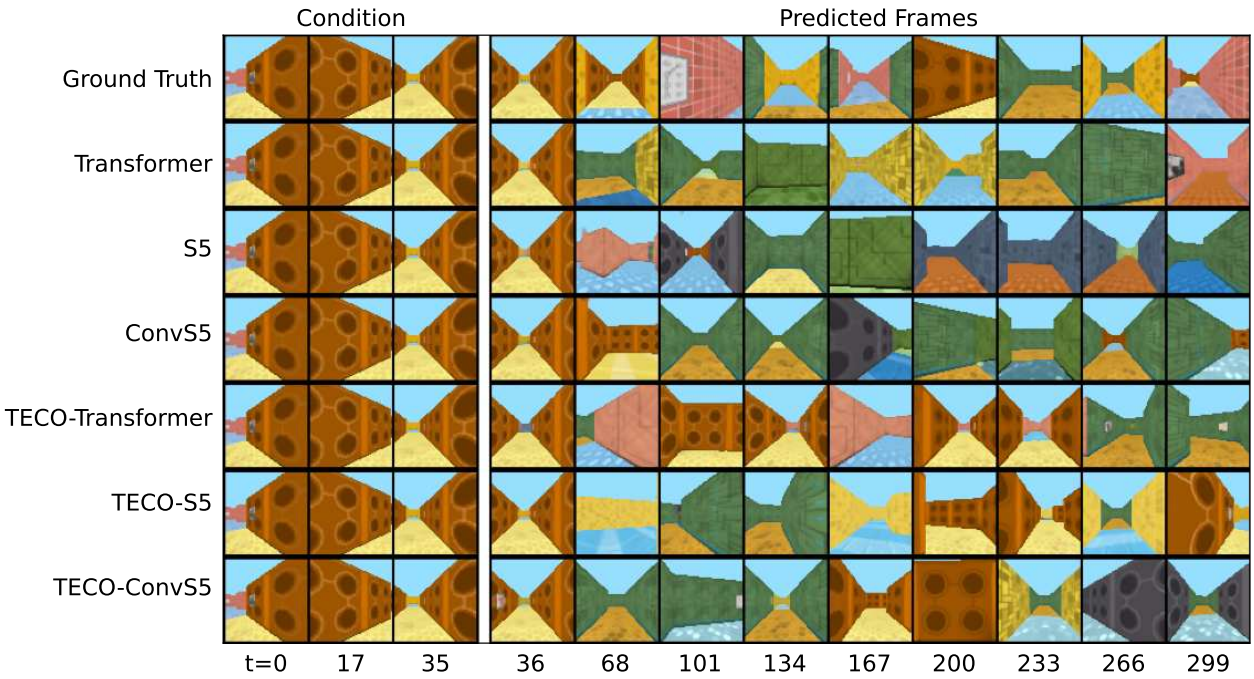
		DMLab			
Method	Params	FVD ↓	PSNR ↑	SSIM ↑	LPIPS ↓
FitVid*	165M	176 ± 4.86	12.0 ± 0.013	0.356 ± 0.00171	0.491 ± 0.00108
CW-VAE*	111M	125 ± 7.95	12.6 ± 0.059	0.372 ± 0.00033	0.465 ± 0.00156
Perceiver AR*	30M	96.3 ± 3.64	11.2 ± 0.004	0.304 ± 0.00004	0.487 ± 0.00123
Latent FDM*	31M	181 ± 2.20	17.8 ± 0.111	0.588 ± 0.00453	0.222 ± 0.00493
Transformer	152M	97.0 ± 5.98	19.9 ± 0.108	0.619 ± 0.00506	0.123 ± 0.00191
S5	140M	221 ± 13.1	19.3 ± 0.128	0.641 ± 0.00400	0.162 ± 0.04510
ConvS5	101M	$\mathbf{53.4 \pm 4.51}$	$\mathbf{23.6 \pm 0.015}$	$\mathbf{0.782 \pm 0.01020}$	$\mathbf{0.074 \pm 0.00979}$
TECO-Transformer*	173M	$\mathbf{27.5 \pm 1.77}$	22.4 ± 0.368	0.709 ± 0.0119	0.155 ± 0.00958
TECO-Transformer (our run)	173M	$\mathbf{28.2 \pm 0.66}$	21.6 ± 0.079	0.696 ± 0.02640	$\mathbf{0.082 \pm 0.00119}$
TECO-S5	180M	34.6 ± 0.26	20.1 ± 0.037	0.687 ± 0.00132	0.143 ± 0.00049
TECO-ConvSSM (ablation)	175M	44.3 ± 2.69	21.0 ± 0.106	0.691 ± 0.00004	0.010 ± 0.00267
TECO-ConvS5	175M	31.2 ± 0.23	$\mathbf{23.8 \pm 0.056}$	$\mathbf{0.803 \pm 0.0020}$	0.085 ± 0.00179

Table 8: Full results on the Minecraft and Habitat long-range benchmark datasets [13]. Results from Yan et al. [13] are indicated with $*$. Note that Yan et al. [13] did not evaluate FitVid or CW-VAE on Habitat due to cost.

		Minecraft			
Method	Params	FVD ↓	PSNR ↑	SSIM ↑	LPIPS ↓
FitVid*	176M	956 ± 15.8	13.0 ± 0.0089	0.343 ± 0.00380	0.519 ± 0.00367
CW-VAE*	140M	397 ± 15.5	13.4 ± 0.0610	0.338 ± 0.00274	0.441 ± 0.00367
Perceiver AR*	166M	76.3 ± 1.72	13.2 ± 0.0711	0.323 ± 0.00336	0.441 ± 0.00207
Latent FDM*	33M	167 ± 6.26	13.4 ± 0.0904	0.349 ± 0.00327	0.429 ± 0.00284
TECO-Transformer*	274M	116 ± 5.08	$\mathbf{15.4 \pm 0.0603}$	$\mathbf{0.381 \pm 0.00192}$	$\mathbf{0.340 \pm 0.00264}$
TECO-ConvS5	214M	$\mathbf{70.7 \pm 3.05}$	14.8 ± 0.0984	0.374 ± 0.00414	0.355 ± 0.00467
		Habitat			
Method	Params	FVD ↓	PSNR ↑	SSIM ↑	LPIPS ↓
Perceiver AR*	200M	164 ± 12.6	12.8 ± 0.0423	$\mathbf{0.405 \pm 0.00248}$	0.676 ± 0.00282
Latent FDM*	87M	433 ± 2.67	12.5 ± 0.0121	0.311 ± 0.00083	$\mathbf{0.582 \pm 0.00049}$
TECO-Transformer*	386M	$\mathbf{76.3 \pm 1.72}$	12.8 ± 0.0139	0.363 ± 0.00122	0.604 ± 0.00451
TECO-ConvS5	351M	95.1 ± 3.74	$\mathbf{12.9 \pm 0.212}$	0.390 ± 0.01238	0.632 ± 0.00823

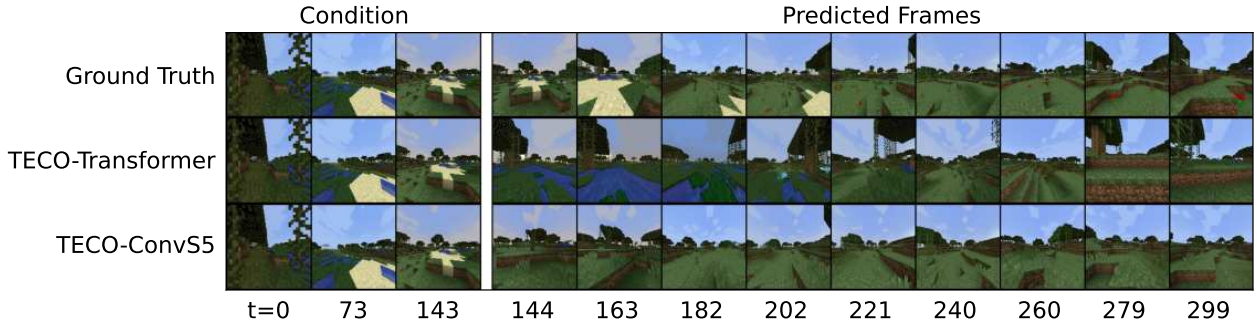


(a) 156 frames generated conditioned on 144 (action-conditioned).

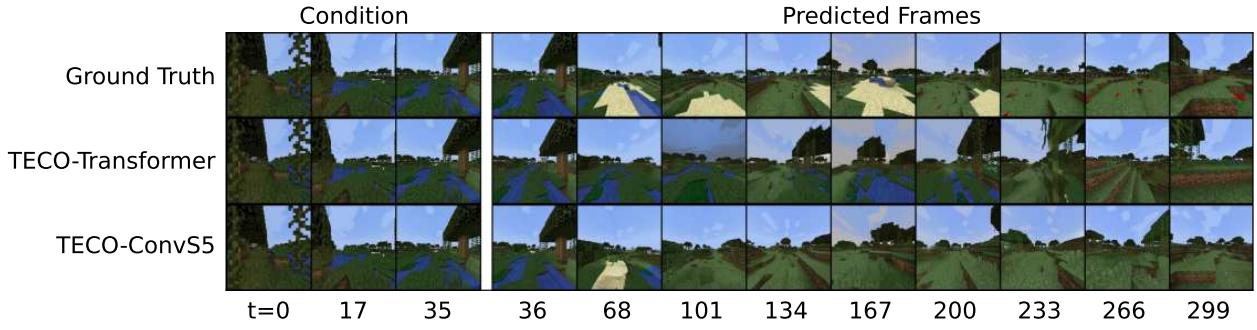


(b) 264 frames generated conditioned on 36 (**no** action-conditioning).

Figure 4: DMLab Samples



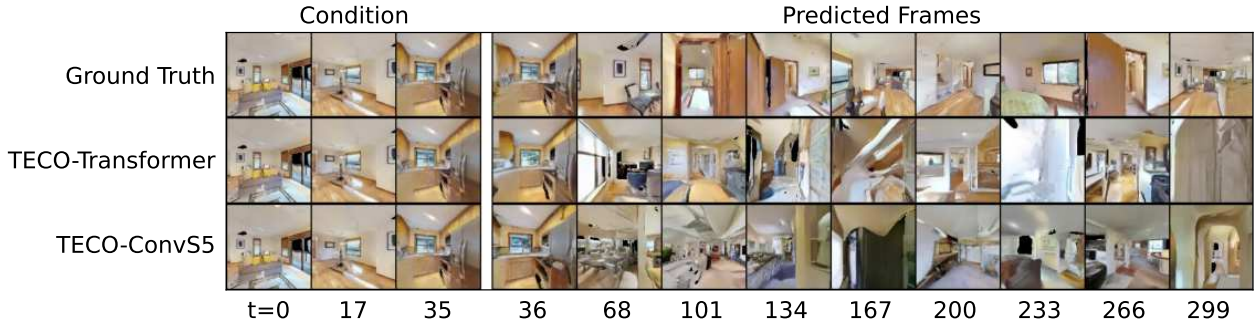
(a) 156 frames generated conditioned on 144 (action-conditioned).



(b) 264 frames generated conditioned on 36 (action-conditioned).

Figure 5: Minecraft Samples

(a) 156 frames generated conditioned on 144 (action-conditioned).



(b) 264 frames generated conditioned on 36 (**no** action-conditioning).

Figure 6: Habitat Samples

Table 9: Model runtime comparison for 3D Environment results in Tables 7-8. The implementations of the baselines FitVid, CW-VAE, Perceiver AR and Latent FDM used in the TECO work [13] are not publicly available in the TECO repository, so we were unable to include direct runtime comparisons for those methods.

	DMLab	
Method	Train Step Time (s) ↓	Sampling Speed (frames/s)
Transformer	**1.25** (**1.0**×)	9.1 (1.0×)
S5	1.34 (1.1×)	28 (3.1×)
ConvS5	2.31 (1.8×)	**56** (**6.2**×)
TECO-Transformer	**0.75** (**0.6**×)	16 (1.8×)
TECO-S5	0.81 (0.7×)	**21** (**2.3**×)
TECO-ConvS5	1.17 (0.9×)	18 (2.0×)
	Minecraft	
Method	Train Step Time (s) ↓	Sampling Speed (frames/s)
TECO-Transformer	**1.91** (**1.0**×)	8.1 (1.0×)
TECO-ConvS5	2.53 (1.3×)	**14** (**1.7**×)
	Habitat	
Method	Train Step Time (s) ↓	Sampling Speed (frames/s)
TECO-Transformer	**2.71** (**1.0**×)	6.8 (1.0×)
TECO-ConvS5	3.10 (1.1×)	**11** (**1.6**×)

D Experiment Configurations

Our codebase modifies the TECO codebase from Yan et al. [13] and we reuse their core Transformer and TECO framework implementations. More architectural details and dataset-specific details are described below.

D.1 Spatiotemporal Sequence Model Architectures

ConvS5, ConvLSTM and S5 models are formed by stacking multiple ConvS5, ConvLSTM or S5 layers, respectively. For each of these models, layer normalization [106] with a post-norm setup is used along with residual connections. For the Transformer, we use the Transformer implementation from Yan et al. [13] which consists of a stack of multi-head attention layers.

ConvS5 and ConvLSTM are applied directly to sequences of frames of shape [sequence length, latent height, latent width, latent features], where the original data has been convolved to a latent resolution and latent number of features. Since S5 and Transformer act on vector-valued sequences, these models require an additional downsampling convolution operation to project the latent frames into a token and an upsampling transposed convolution operation to project the Transformer output tokens back into latent frames. We use the same sequence of compression operations for this as in Yan et al. [13]. The Encoder and Decoder referred to for all models in the hyperparameter tables below consist of ResNet Blocks with 3×3 kernels as implemented in Yan et al. [13].

D.2 Evaluation Metrics

We evaluate methods by computing Fréchet Video Distance (FVD) [107], peak signal-to-noise ratio (PSNR), structural similarity index measure [108] and Learned Perceptual Image Patch Similarity (LPIPS) [109] between sampled trajectories and ground truth trajectories.

D.3 Compute

All models were trained with 32GB NVIDIA V100 GPUs. For Moving-MNIST, models were trained with 8 V100s. For all other experiments, models were trained with 16 V100s. We list V100 days in the hyperparameters, which denotes the number of days it would take to train on a single V100.

D.4 Moving-MNIST

All models were trained to minimize L1+L2 loss over the frames directly in pixel space, as in Su et al. [85]. We trained models on 300 frames. We then repeated the experiment and trained models on 600 frames. For ConvS5 and ConvLSTM, we fixed the hidden dimensions (layer input/output features) and state sizes to be 256, and we swept over the following learning rates [1×10^{-4}, 5×10^{-4}, 1×10^{-3}] and chose the best model. For Transformer, we swept over model size, considering hidden dimensions of [512, 2014] and learning rates [1×10^{-4}, 5×10^{-4}, 1×10^{-3}] and chose the best model. We also observed better performance for the Transformer by convolving frames down to an 8×8 latent resolution (rather than the 16×16 used by ConvS5 and ConvLSTM) before downsampling to a token. All other relevant training parameters were kept the same between the three methods. See Tables 10-12 for detailed experiment configurations.

Each model was evaluated by collecting 1024 trajectories using the following procedure: condition on 100 frames from the ground truth test set, then generate forward 1200 frames. These samples were compared with the ground truth to compute FVD, PSNR, SSIM and LPIPS.

The ConvSSM ablation was performed using the exact settings as ConvS5, except the state kernel was initialized with a Gaussian and we swept over the following learning rates [1×10^{-4}, 5×10^{-4}, 1×10^{-3}].

Table 10: Experiment Configuration for ConvS5 on Moving-MNIST experiments

Hyperparameters		Moving-MNIST-300	Moving-MNIST-600
	V100 Days	25	50
	Params	41M	41M
	Input Resolution	64×64	64×64
	Latent Resolution	16×16	16×16
	Batch Size	8	8
	Sequence Length	300	600
	LR	1×10^{-3}	1×10^{-3}
	LR Schedule	cosine	cosine
	Warmup Steps	5k	5k
	Max Training Steps	300K	300K
	Weight Decay	1×10^{-5}	1×10^{-5}
Encoder	Depths	64, 128, 256	64, 128, 256
	Blocks	1	1
Decoder	Depths	64, 128, 256	64, 128, 256
	Blocks	1	1
ConvS5	Hidden Dim (U)	256	256
	State Size (P)	256	256
	$\boldsymbol{\mathcal{B}}$ Kernel Size	3×3	3×3
	$\boldsymbol{\mathcal{C}}$ Kernel Size	3×3	3×3
	Layers	8	8
	Dropout	0	0
	Activation	ResNet	ResNet

Table 11: Experiment Configuration for ConvLSTM on Moving-MNIST experiments

Hyperparameters		Moving-MNIST-300	Moving-MNIST-600
	V100 Days	75	150
	Params	43M	43M
	Input Resolution	64×64	64×64
	Latent Resolution	16×16	16×16
	Batch Size	8	8
	Sequence Length	300	600
	LR	5×10^{-4}	5×10^{-4}
	LR Schedule	cosine	cosine
	Warmup Steps	5k	5k
	Max Training Steps	300K	300K
	Weight Decay	1×10^{-5}	1×10^{-5}
Encoder	Depths	64, 128, 256	64, 128, 256
	Blocks	1	1
Decoder	Depths	64, 128, 256	64, 128, 256
	Blocks	1	1
ConvLSTM	Hidden Dim	256	256
	State Size	256	256
	Kernel Size	3×3	3×3
	Layers	8	8
	Dropout	0	0

Table 12: Experiment Configuration for Transformer on Moving-MNIST experiments

Hyperparameters		Moving-MNIST-300	Moving-MNIST-600
	V100 Days	25	50
	Params	164M	164M
	Input Resolution	64×64	64×64
	Latent Resolution	8×8	8×8
	Batch Size	8	8
	Sequence Length	300	600
	LR	5×10^{-4}	1×10^{-4}
	LR Schedule	cosine	cosine
	Warmup Steps	5k	5k
	Max Training Steps	300K	300K
	Weight Decay	1×10^{-5}	1×10^{-5}
Encoder	Depths	64, 128, 256, 512	64, 128, 256, 512
	Blocks	1	1
Decoder	Depths	64, 128, 256, 512	64, 128, 256, 512
	Blocks	1	1
	Downsample Factor	8	8
	Hidden Dim	1024	1024
Temporal	Feedforward Dim	4096	4096
Transformer	Heads	16	16
	Layers	8	8
	Dropout	0	0

D.5 Long-Range 3D Environment Benchmarks

We follow the procedures from Yan et al. [13] and train models on the same pre-trained vector-quantized (VQ) 16×16 codes used by the baselines evaluated in that work. Models were trained to optimize a cross-entropy reconstruction loss between the predictions and true VQ codes. The evaluation of DMLab and Habitat involves both an action-conditioned and unconditioned setting. Therefore, as in Yan et al. [13], the actions were randomly dropped out half the time during training on these datasets.

After training, we follow the procedure from Yan et al. [13] for evaluation in two different settings. The first setting involves computing PSNR, SSIM and LPIPS from 1024 samples generated by conditioning on the first 144 frames and then generating the next 156 frames while providing the model with past and future actions. The second setting does not provide actions as input (with the exception of Minecraft, which also provides actions in this setting). It involves computing FVD using 1024 samples generated by conditioning on the first 36 frames and then predicting the remaining 264 frames.

All sequence models we trained used the same number of layers as the Transformer used in the TECO-Transformer trained by Yan et al. [13]. In addition, the TECO-Transformer, TECO-S5 and TECO-ConvS5 models we trained used the exact encoder/decoder configuration and MaskGit configuration as in Yan et al. [13]. The Transformer, S5 and ConvS5 models we trained without the TECO framework were all trained using the same encoder/decoder configuration. See Tables 13-19 for more detailed experimental configuration details. See dataset-specific paragraphs below for hyperparameter tuning information.

TECO Training Framework Yan et al. [13] proposed the TECO training framework to train Transformers on long video data. For some of our experiments, we use ConvS5 layers and S5 layers as a drop-in replacement for the Transformer in this framework. We refer the reader to Yan et al. [13] for full details. Briefly, given the original VQ codes, TECO trains an additional encoder/decoder that compresses the frames to a lower latent resolution (e.g., from 16×16 to 8×8) by training an additional encoder/decoder with a codebook loss, $\mathcal{L}_{\text{VQ}}$. In addition, a MaskGit [98] dynamics prior loss, $\mathcal{L}_{\text{prior}}$, is used for the latent transitions. The sequence model (e.g. Transformer, S5, ConvS5) takes the latent frames (compressed into tokens in the case of Transformer and S5) and produces an output which is used along with the latents by the decoder to produce predictions and a reconstruction loss, $\mathcal{L}_{\text{recon}}$. Models are trained to minimize the following total loss:

$$\mathcal{L}_{\text{TECO}} = \mathcal{L}_{\text{VQ}} + \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{prior}}. \tag{32}$$

In addition, TECO includes the use of DropLoss [13], which drops out a percentage of random timesteps that are not decoded and therefore do not require computing the expensive $\mathcal{L}_{\text{recon}}$ and $\mathcal{L}_{\text{prior}}$ terms.

DMLab As mentioned above, the actions were randomly dropped out of sequences half the time (due to the two evaluation scenarios, action-conditioned and unconditioned). We observed that for DMLab, when provided past and future actions, models converged faster using the simple masking strategy discussed in Gu et al. [19] that masks the future inputs rather than feeding the predicted inputs (or true inputs during training) autoregressively. Therefore we trained all models (Transformer, S5, ConvS5, Teco-Transformer, TECO-S5, TECO-ConvS5) by using this strategy when the actions were provided, and using the autoregressive strategy when actions were not provided. We observed this significantly improved the LPIPS of the Transformer baselines. Note, for Minecraft and Habitat, we observed this strategy led to lower-quality frames and did not use it for the reported results for those datasets.

We trained each model, Transformer, S5, ConvS5, Teco-Transformer, TECO-S5, TECO-ConvS5, with three different learning rates [1×10^{-4}, 5×10^{-4}, 1×10^{-3}] and selected the best run for each model. See Tables 13-18 for more experiment configuration details.

The TECO-ConvSSM ablation used the exact same settings as TECO-ConvS5, except the state kernel was initialized with a random Gaussian and a lower learning rate of 1×10^{-5} was required for stable training.

Table 13: Experiment Configuration for ConvS5 on DMLab

Hyperparameters		DMLab
	V100 Days	150
	Params	101M
	Input Resolution	64×64
	Latent Resolution	16×16
	Batch Size	16
	Sequence Length	300
	LR	5×10^{-4}
	LR Schedule	cosine
	Warmup Steps	5k
	Max Training Steps	500K
	Weight Decay	1×10^{-5}
Encoder	Depths	256
	Blocks	1
Decoder	Depths	256
	Blocks	4
ConvS5	Hidden Dim (U)	512
	State Size (P)	512
	$\boldsymbol{\mathcal{B}}$ Kernel Size	3×3
	$\boldsymbol{\mathcal{C}}$ Kernel Size	3×3
	Layers	8
	Dropout	0
	Activation	ResNet

Table 14: Experiment Configuration for S5 on DMLab

Hyperparameters		DMLab
	V100 Days	125
	Params	140M
	Input Resolution	64×64
	Latent Resolution	16×16
	Batch Size	16
	Sequence Length	300
	LR	1×10^{-3}
	LR Schedule	cosine
	Warmup Steps	5k
	Max Training Steps	500K
	Weight Decay	1×10^{-5}
Encoder	Depths	256
	Blocks	1
Decoder	Depths	256
	Blocks	4
S5	Downsample Factor	16
	Hidden Dim (U)	1024
	State Size (P)	1024
	Layers	8
	Dropout	0
	Activation	GLU (half)

Table 15: Experiment Configuration for Transformer on DMLab

Hyperparameters		DMLab
	V100 Days	125
	Params	152M
	Input Resolution	64×64
	Latent Resolution	16×16
	Batch Size	16
	Sequence Length	300
	LR	5×10^{-4}
	LR Schedule	cosine
	Warmup Steps	5k
	Max Training Steps	500K
	Weight Decay	1×10^{-5}
Encoder	Depths	256
	Blocks	1
Decoder	Depths	256
	Blocks	4
Temporal Transformer	Downsample Factor	16
	Hidden Dim	512
	Feedforward Dim	2048
	Heads	16
	Layers	8
	Dropout	0

Table 16: Experiment Configuration for TECO-ConvS5 on DMLab

Hyperparameters	DMLab	
	V100 Days	110
	Params	175M
	Input Resolution	64×64
	Latent Resolution	8×8
	Batch Size	16
	Sequence Length	300
	LR	5×10^{-4}
	LR Schedule	cosine
	Warmup Steps	5k
	Max Training Steps	500K
	Weight Decay	1×10^{-5}
	DropLoss Rate	0.9
Encoder	Depths	256, 512
	Blocks	2
Codebook	Size	1024
	Embedding Dim	32
Decoder	Depths	256, 512
	Blocks	4
ConvS5	Hidden Dim (U)	512
	State Size (P)	1024
	$\boldsymbol{\mathcal{B}}$ Kernel Size	3×3
	$\boldsymbol{\mathcal{C}}$ Kernel Size	3×3
	Layers	8
	Dropout	0
	Activation	ResNet
MaskGit	Mask Schedule	cosine
	Hidden Dim	512
	Feedforward Dim	2048
	Heads	8
	Layers	8
	Dropout	0

Table 17: Experiment Configuration for TECO-S5 on DMLab

Hyperparameters	DMLab	
	V100 Days	80
	Params	180M
	Input Resolution	64×64
	Latent Resolution	8×8
	Batch Size	16
	Sequence Length	300
	LR	1×10^{-3}
	LR Schedule	cosine
	Warmup Steps	5k
	Max Training Steps	500K
	Weight Decay	1×10^{-5}
	DropLoss Rate	0.9
Encoder	Depths	256, 512
	Blocks	2
Codebook	Size	1024
	Embedding Dim	32
Decoder	Depths	256, 512
	Blocks	4
S5	Downsample Factor	8
	Hidden Dim (U)	2048
	State Size (P)	2048
	Layers	8
	Dropout	0
	Activation	GLU (half)
MaskGit	Mask Schedule	cosine
	Hidden Dim	512
	Feedforward Dim	2048
	Heads	8
	Layers	8
	Dropout	0

Table 18: Experiment Configuration for TECO-Transformer on DMLab

Hyperparameters	DMLab	
	V100 Days	80
	Params	173M
	Input Resolution	64×64
	Latent Resolution	8×8
	Batch Size	16
	Sequence Length	300
	LR	1×10^{-4}
	LR Schedule	cosine
	Warmup Steps	5k
	Max Training Steps	500K
	Weight Decay	1×10^{-5}
	DropLoss Rate	0.9
Encoder	Depths	256, 512
	Blocks	2
Codebook	Size	1024
	Embedding Dim	32
Decoder	Depths	256, 512
	Blocks	4
Temporal Transformer	Downsample Factor	8
	Hidden Dim	1024
	Feedforward Dim	4096
	Heads	16
	Layers	8
	Dropout	0
MaskGit	Mask Schedule	cosine
	Hidden Dim	512
	Feedforward Dim	2048
	Heads	8
	Layers	8
	Dropout	0

Minecraft and Habitat For Minecraft and Habitat, we only trained TECO-ConvS5 due to the costs of training on these datasets. See dataset details in Appendix E and reported compute costs in Yan et al. [13]. For Minecraft, we evaluated two different learning rates [1×10^{-4}, 5×10^{-4}] and chose the best. For Habitat, we only performed one run with no further tuning. See Table 19 for further experiment configuration details.

Table 19: Experiment Configuration for TECO-ConvS5 on Minecraft and Habitat

Hyperparameters		Minecraft	Habitat
	V100 Days	470	575
	Params	214M	351M
	Input Resolution	128×128	128×128
	Latent Resolution	8×8	8×8
	Batch Size	16	16
	Sequence Length	300	300
	LR	5×10^{-4}	1×10^{-4}
	LR Schedule	cosine	cosine
	Warmup Steps	5k	5k
	Max Training Steps	1M	1M
	DropLoss Rate	0.9	0.9
Encoder	Depths	256, 512	256, 512
	Blocks	4	4
Codebook	Size	1024	1024
	Embedding Dim	32	32
Decoder	Depths	256, 512	256, 512
	Blocks	8	8
ConvS5	Hidden Dim (U)	512	512
	State Size (P)	512	512
	$\boldsymbol{\mathcal{B}}$ Kernel Size	3×3	3×3
	$\boldsymbol{\mathcal{C}}$ Kernel Size	3×3	3×3
	Layers	12	8
	Dropout	0	0
	Activation	ResNet	ResNet
MaskGit	Mask Schedule	cosine	cosine
	Hidden Dim	768	1024
	Feedforward Dim	3072	4096
	Heads	12	16
	Layers	6	16
	Dropout	0	0

E Datasets

E.1 Moving-MNIST

The Moving-MNIST [54] dataset is generated by moving two 28×28 size MNIST digits from the MNIST dataset [110] inside a 64×64 black background. The digits begin at a random initial location, and move with constant velocity, bouncing when they reach the boundary. For each of the sequence lengths we consider, 300 and 600, we follow Wang et al. [82] and Su et al. [85] and generate 10,000 sequences for training.

E.2 DMLab

We use the DMLab long-range benchmark designed by Yan et al. [13] using the DeepMind Lab (DMLab) [99] simulator. The simulator generates random 3D mazes with random floor and wall textures. The benchmark consists of 40K action-conditioned, 300 frame videos at a 64×64 resolution. The videos are of an agent randomly navigating 7×7 mazes by choosing random points in the maze and navigating to them through the shortest path.

E.3 Minecraft

We use the Minecraft [100] long-range benchmark designed by Yan et al. [13]. The game features 3D worlds that contain complex terrains such as hills, forests, rivers and lakes. The benchmark was constructed by collecting 200K action-conditioned 300 frame videos at a 128×128 resolution. The videos are in Minecraft's marsh biome and the agent iterates walking forward for a random number of steps and randomly rotating left or right. This results in parts of the scene going out of view and coming back into view later.

E.4 Habitat

We use the Habitat long-range benchmark designed by Yan et al. [13] using the Habitat simulator [101]. The simulator renders trajectories using scans of real 3D scenes. Yan et al. [13] compiled 1400 indoor scans from HM3D [111], Matterport3D [112] and Gibson [113] to generate 200K action-conditioned, 300 frame videos with a 128×128 resolution. Yan et al. [13] used Habitat's in-built path traversal algorithm to construct action trajectories that move the agent between randomly sampled locations.

References

[1] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. *Advances in neural information processing systems*, 29, 2016.

[2] Yi Xu, Longwen Gao, Kai Tian, Shuigeng Zhou, and Huyang Sun. Non-local ConvLSTM for video compression artifact reduction. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 7043–7052, 2019.

[3] He Huang, Zheni Zeng, Danya Yao, Xin Pei, and Yi Zhang. Spatial-temporal ConvLSTM for vehicle driving intention prediction. *Tsinghua Science and Technology*, 27(3):599–609, 2021.

[4] Xiaoyu Chen, Xingsheng Xie, and Da Teng. Short-term traffic flow prediction based on ConvLSTM model. In *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, pages 846–850. IEEE, 2020.

[5] Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, et al. Fourcastnet: A global data-driven high-resolution weather model using adaptive Fourier neural operators. *arXiv preprint arXiv:2202.11214*, 2022.

[6] Jonathan A Weyn, Dale R Durran, Rich Caruana, and Nathaniel Cresswell-Clay. Sub-seasonal forecasting with a large ensemble of deep-learning weather prediction models. *Journal of Advances in Modeling Earth Systems*, 13(7):e2021MS002502, 2021.

[7] Jonathan Ho, Tim Salimans, Alexey A. Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video diffusion models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[8] Aidan Clark, Jeff Donahue, and Karen Simonyan. Adversarial video generation on complex datasets. *arXiv preprint arXiv:1907.06571*, 2019.

[9] Wilson Yan, Yunzhi Zhang, Pieter Abbeel, and Aravind Srinivas. Videogpt: Video generation using VQ-VAE and Transformers. *arXiv preprint arXiv:2104.10157*, 2021.

[10] Guillaume Le Moing, Jean Ponce, and Cordelia Schmid. CCVS: context-aware controllable video synthesis. *Advances in Neural Information Processing Systems*, 34:14042–14055, 2021.

[11] Yu Tian, Jian Ren, Menglei Chai, Kyle Olszewski, Xi Peng, Dimitris N. Metaxas, and Sergey Tulyakov. A good image generator is what you need for high-resolution video synthesis. In *International Conference on Learning Representations*, 2021.

[12] Pauline Luc, Aidan Clark, Sander Dieleman, Diego de Las Casas, Yotam Doron, Albin Cassirer, and Karen Simonyan. Transformation-based adversarial video prediction on large-scale data. *arXiv preprint arXiv:2003.04035*, 2020.

[13] Wilson Yan, Danijar Hafner, Stephen James, and Pieter Abbeel. Temporally consistent video Transformer for long-term video prediction. *arXiv preprint arXiv:2210.02396*, 2022.

[14] Songwei Ge, Thomas Hayes, Harry Yang, Xi Yin, Guan Pang, David Jacobs, Jia-Bin Huang, and Devi Parikh. Long video generation with time-agnostic VQGAN and time-sensitive Transformer. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XVII*, pages 102–118. Springer, 2022.

[15] William Harvey, Saeid Naderiparizi, Vaden Masrani, Christian Dietrich Weilbach, and Frank Wood. Flexible diffusion modeling of long videos. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[16] Vaibhav Saxena, Jimmy Ba, and Danijar Hafner. Clockwork variational autoencoders. *Advances in Neural Information Processing Systems*, 34:29246–29257, 2021.

[17] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*, 28, 2015.

[18] Nicolas Ballas, Li Yao, Chris Pal, and Aaron Courville. Delving deeper into convolutional networks for learning video representations. *arXiv preprint arXiv:1511.06432*, 2015.

[19] Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2021.

[20] Jimmy T.H. Smith, Andrew Warrington, and Scott Linderman. Simplified state space layers for sequence modeling. In *The Eleventh International Conference on Learning Representations*, 2023.

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9 (8):1735–1780, 1997.

[22] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[23] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318. PMLR, 2013.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

[25] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[26] Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. ViViT: A video vision Transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6836–6846, 2021.

[27] Agrim Gupta, Stephen Tian, Yunzhi Zhang, Jiajun Wu, Roberto Martín-Martín, and Li Fei-Fei. Maskvit: Masked visual pre-training for video prediction. *arXiv preprint arXiv:2206.11894*, 2022.

[28] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[29] Jacob Walker, Ali Razavi, and Aäron van den Oord. Predicting video with VQVAE. *arXiv preprint arXiv:2103.01950*, 2021.

[30] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming Transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021.

[31] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Anselm Levskaya, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling Transformer inference. *arXiv preprint arXiv:2211.05102*, 2022.

[32] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[33] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with Performers. In *International Conference on Learning Representations*, 2021.

[34] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive Transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

[35] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient Transformer. In *International Conference on Learning Representations*, 2020.

[36] Iz Beltagy, Matthew Peters, and Arman Cohan. Longformer: The long-document Transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[37] Ankit Gupta and Jonathan Berant. Gmat: Global memory augmentation for Transformers, 2020.

[38] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[39] Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pages 4651–4664. PMLR, 2021.

[40] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long Range Arena: A benchmark for efficient Transformers. In *International Conference on Learning Representations*, 2021.

[41] Ankit Gupta, Albert Gu, and Jonathan Berant. Diagonal state spaces are as effective as structured state spaces. In *Advances in Neural Information Processing Systems*, 2022.

[42] Albert Gu, Karan Goel, Ankit Gupta, and Christopher Ré. On the parameterization and initialization of diagonal state space models. In *Advances in Neural Information Processing Systems*, 2022.

[43] Ramin Hasani, Mathias Lechner, Tsun-Hsuan Wang, Makram Chahine, Alexander Amini, and Daniela Rus. Liquid structural state-space models. In *International Conference on Learning Representations*, 2023.

[44] Karan Goel, Albert Gu, Chris Donahue, and Christopher Re. It's raw! Audio generation with state-space models. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 7616–7633. PMLR, 17–23 Jul 2022.

[45] Eric Nguyen, Karan Goel, Albert Gu, Gordon Downs, Preey Shah, Tri Dao, Stephen Baccus, and Christopher Ré. S4ND: Modeling images and videos as multidimensional signals with state spaces. In *Advances in Neural Information Processing Systems*, 2022.

[46] Md Mohaiminul Islam and Gedas Bertasius. Long movie clip classification with state-space video models. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXV*, pages 87–104, 2022.

[47] Shmuel Bar David, Itamar Zimerman, Eliya Nachmani, and Lior Wolf. Decision S4: Efficient sequence-based RL via state spaces layers. In *The Eleventh International Conference on Learning Representations*, 2023.

[48] Chris Lu, Yannick Schroecker, Albert Gu, Emilio Parisotto, Jakob Foerster, Satinder Singh, and Feryal Behbahani. Structured state space models for in-context reinforcement learning. *arXiv preprint arXiv:2303.03982*, 2023.

[49] Linqi Zhou, Michael Poli, Winnie Xu, Stefano Massaroli, and Stefano Ermon. Deep latent state space models for time-series generation. *arXiv preprint arXiv:2212.12749*, 2022.

[50] Daniel Y Fu, Tri Dao, Khaled Kamal Saab, Armin W Thomas, Atri Rudra, and Christopher Re. Hungry hungry hippos: Towards language modeling with state space models. In *The Eleventh International Conference on Learning Representations*, 2023.

[51] Harsh Mehta, Ankit Gupta, Ashok Cutkosky, and Behnam Neyshabur. Long range language modeling via gated state spaces. In *The Eleventh International Conference on Learning Representations*, 2023.

[52] Junxiong Wang, Jing Nathan Yan, Albert Gu, and Alexander M Rush. Pretraining without attention. *arXiv preprint arXiv:2212.10544*, 2022.

[53] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. HiPPO: Recurrent memory with optimal polynomial projections. *Advances in Neural Information Processing Systems*, 33:1474–1487, 2020.

[54] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhudinov. Unsupervised learning of video representations using LSTMs. In *International conference on machine learning*, pages 843–852. PMLR, 2015.

[55] Arieh Iserles. *A first course in the numerical analysis of differential equations*. 44. Cambridge university press, 2009.

[56] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in Neural Information Processing Systems*, 34, 2021.

[57] Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. *arXiv preprint arXiv:2303.06349*, 2023.

[58] Eric Martin and Chris Cundy. Parallelizing linear recurrent neural nets over sequence length. In *International Conference on Learning Representations*, 2018.

[59] James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. Quasi-recurrent neural networks. In *International Conference on Learning Representations*, 2017.

[60] Tao Lei, Yu Zhang, Sida Wang, Hui Dai, and Yoav Artzi. Simple recurrent units for highly parallelizable recurrence. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4470–4481, 2018.

[61] Guy Blelloch. Prefix sums and their applications. Technical report, Tech. rept. CMU-CS-90-190. School of Computer Science, Carnegie Mellon, 1990.

[62] Wonmin Byeon, Qin Wang, Rupesh Kumar Srivastava, and Petros Koumoutsakos. ContextVP: Fully context-aware video prediction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 753–769, 2018.

[63] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[64] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *Tenth international workshop on frontiers in handwriting recognition*. Suvisoft, 2006.

[65] Yangqing Jia. Learning semantic image representations at a large scale. 2014.

[66] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[67] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[68] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[69] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[70] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A Convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022.

[71] Marijn F Stollenga, Wonmin Byeon, Marcus Liwicki, and Juergen Schmidhuber. Parallel multi-dimensional LSTM, with application to fast biomedical volumetric image segmentation. *Advances in neural information processing systems*, 28, 2015.

[72] Reza Azad, Maryam Asadi-Aghbolaghi, Mahmood Fathy, and Sergio Escalera. Bi-directional ConvLSTM U-net with densely connected convolutions. In *Proceedings of the IEEE/CVF international conference on computer vision workshops*, pages 0–0, 2019.

[73] Si Woon Lee and Ha Young Kim. Stock market forecasting with super-high dimensional time-series data using ConvLSTM, trend sampling, and specialized data augmentation. *expert systems with applications*, 161:113704, 2020.

[74] Qingqing Wang, Ye Huang, Wenjing Jia, Xiangjian He, Michael Blumenstein, Shujing Lyu, and Yue Lu. FACLSTM: ConvLSTM with focused attention for scene text recognition. *Science China Information Sciences*, 63:1–14, 2020.

[75] Mohamadreza Bakhtyari and Sayeh Mirzaei. ADHD detection using dynamic connectivity patterns of EEG data and ConvLSTM with attention framework. *Biomedical Signal Processing and Control*, 76:103708, 2022.

[76] Li Kang, Ziqi Zhou, Jianjun Huang, Wenzhong Han, and IEEE Member. Renal tumors segmentation in abdomen CT images using 3D-CNN and ConvLSTM. *Biomedical Signal Processing and Control*, 72:103334, 2022.

[77] Tie Liu, Mai Xu, and Zulin Wang. Removing rain in videos: a large-scale database and a two-stream ConvLSTM approach. In *2019 IEEE International Conference on Multimedia and Expo (ICME)*, pages 664–669. IEEE, 2019.

[78] Xiaofang Xia, Jian Lin, Qiannan Jia, Xiaoluan Wang, Chaofan Ma, Jiangtao Cui, and Wei Liang. ETD-ConvLSTM: A deep learning approach for electricity theft detection in smart grids. *IEEE Transactions on Information Forensics and Security*, 2023.

[79] Xingjian Shi, Zhihan Gao, Leonard Lausen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Deep learning for precipitation nowcasting: A benchmark and a new model. *Advances in neural information processing systems*, 30, 2017.

[80] Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. Stochastic variational video prediction. In *International Conference on Learning Representations*, 2018.

[81] Yunbo Wang, Haixu Wu, Jianjin Zhang, Zhifeng Gao, Jianmin Wang, S Yu Philip, and Mingsheng Long. PredRNN: A recurrent neural network for spatiotemporal predictive learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(2):2208–2225, 2022.

[82] Yunbo Wang, Zhifeng Gao, Mingsheng Long, Jianmin Wang, and S Yu Philip. PredRNN++: Towards a resolution of the deep-in-time dilemma in spatiotemporal predictive learning. In *International Conference on Machine Learning*, pages 5123–5132. PMLR, 2018.

[83] Yunbo Wang, Lu Jiang, Ming-Hsuan Yang, Li-Jia Li, Mingsheng Long, and Li Fei-Fei. Eidetic 3D LSTM: A model for video prediction and beyond. In *International conference on learning representations*, 2019.

[84] Wei Yu, Yichao Lu, Steve Easterbrook, and Sanja Fidler. Efficient and information-preserving future frame prediction and beyond. In *International Conference on Learning Representations*, 2020.

[85] Jiahao Su, Wonmin Byeon, Jean Kossaifi, Furong Huang, Jan Kautz, and Anima Anandkumar. Convolutional tensor-train LSTM for spatio-temporal learning. *Advances in Neural Information Processing Systems*, 33:13714–13726, 2020.

[86] Jue Wang, Wentao Zhu, Pichao Wang, Xiang Yu, Linda Liu, Mohamed Omar, and Raffay Hamid. Selective structured state-spaces for long-form video understanding. *arXiv preprint arXiv:2303.14526*, 2023.

[87] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[88] Özgün Çiçek, Ahmed Abdulkadir, Soeren S Lienkamp, Thomas Brox, and Olaf Ronneberger. 3D U-net: learning dense volumetric segmentation from sparse annotation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2016: 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part II 19*, pages 424–432. Springer, 2016.

[89] Tobias Höppe, Arash Mehrjou, Stefan Bauer, Didrik Nielsen, and Andrea Dittadi. Diffusion models for video prediction and infilling. *Transactions on Machine Learning Research*.

[90] Mohammad Babaeizadeh, Mohammad Taghi Saffar, Suraj Nair, Sergey Levine, Chelsea Finn, and Dumitru Erhan. FitVid: Overfitting in pixel-level video prediction. *arXiv preprint arXiv:2106.13195*, 2021.

[91] Ruslan Rakhimov, Denis Volkhonskiy, Alexey Artemov, Denis Zorin, and Evgeny Burnaev. Latent video Transformer. *arXiv preprint arXiv:2006.10704*, 2020.

[92] Younggyo Seo, Kimin Lee, Fangchen Liu, Stephen James, and Pieter Abbeel. HARP: Autoregressive latent video prediction with high-fidelity image generator. In *2022 IEEE International Conference on Image Processing (ICIP)*, pages 3943–3947. IEEE, 2022.

[93] Ivan Skorokhodov, Sergey Tulyakov, and Mohamed Elhoseiny. StyleGAN-V: A continuous video generator with the price, image quality and perks of StyleGAN2. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3626–3636, 2022.

[94] Masaki Saito and Shunta Saito. TGANv2: Efficient training of large models for video generation with multiple subsampling layers. *arXiv preprint arXiv:1811.09245*, 2(6), 2018.

[95] Sihyun Yu, Jihoon Tack, Sangwoo Mo, Hyunsu Kim, Junho Kim, Jung-Woo Ha, and Jinwoo Shin. Generating videos with dynamics-aware implicit generative adversarial networks. In *International Conference on Learning Representations*, 2022.

[96] Tim Brooks, Janne Hellsten, Miika Aittala, Ting-Chun Wang, Timo Aila, Jaakko Lehtinen, Ming-Yu Liu, Alexei Efros, and Tero Karras. Generating long videos of dynamic scenes. *Advances in Neural Information Processing Systems*, 35:31769–31781, 2022.

[97] Wenyi Hong, Ming Ding, Wendi Zheng, Xinghan Liu, and Jie Tang. Cogvideo: Large-scale pretraining for text-to-video generation via Transformers. *arXiv preprint arXiv:2205.15868*, 2022.

[98] Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. MaskGit: Masked generative image Transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11315–11325, 2022.

[99] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind Lab. *arXiv preprint arXiv:1612.03801*, 2016.

[100] William H Guss, Brandon Houghton, Nicholay Topin, Phillip Wang, Cayden Codel, Manuela Veloso, and Ruslan Salakhutdinov. MineRL: A large-scale dataset of Minecraft demonstrations. *arXiv preprint arXiv:1907.13440*, 2019.

[101] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied AI research. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9339–9347, 2019.

[102] Keyu Tian, Yi Jiang, Qishuai Diao, Chen Lin, Liwei Wang, and Zehuan Yuan. Designing BERT for convolutional networks: Sparse and hierarchical masked modeling. *arXiv preprint arXiv:2301.03580*, 2023.

[103] Sunghyun Park, Kangyeol Kim, Junsoo Lee, Jaegul Choo, Joonseok Lee, Sookyung Kim, and Edward Choi. Vid-ODE: Continuous-time video generation with neural ordinary differential equation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 2412–2422, 2021.

[104] Mark Harris, Shubhabrata Sengupta, and John D Owens. Parallel prefix sum (scan) with CUDA. *GPU gems*, 3(39):851–876, 2007.

[105] Albert Gu, Isys Johnson, Aman Timalsina, Atri Rudra, and Christopher Re. How to train your HIPPO: State space models with generalized orthogonal basis projections. In *International Conference on Learning Representations*, 2023.

[106] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[107] Thomas Unterthiner, Sjoerd Van Steenkiste, Karol Kurach, Raphael Marinier, Marcin Michalski, and Sylvain Gelly. Towards accurate generative models of video: A new metric & challenges. *arXiv preprint arXiv:1812.01717*, 2018.

[108] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4): 600–612, 2004.

[109] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

[110] Yann LeCun. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[111] Santhosh K Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alex Clegg, John Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X Chang, et al. Habitat-Matterport 3D dataset (HM3D): 1000 large-scale 3D environments for embodied AI. *arXiv preprint arXiv:2109.08238*, 2021.

[112] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017.

[113] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9068–9079, 2018.