

306 See visual results on anonymous website <https://ttt-done-right.github.io/>,³

307 A Analysis on Design Choices

308 In this section, we analyze several key design choices in our model, focusing on both the novel view
 309 synthesis and language modeling tasks. Specifically, we evaluate the impact of state size (Fig. 7a),
 310 test-time optimizers (Fig. 7b), linear versus nonlinear fast weights (Fig. 8a), and per-token recurrence
 311 versus chunk-wise recurrence (Fig. 8b). Overall, we find that a large state size, advanced optimizers
 312 such as Muon, and nonlinear fast weights significantly improve our model’s performance. In a
 313 controlled NVS experiment, our linear large-chunk recurrence strategy outperforms linear per-token
 314 recurrence with the same state. For language modeling, where chunk structures are not inherent, our
 315 linear large-chunk recurrence variant—while initially underperforming per-token methods like GLA
 316 and DeltaNet—surpasses them when combined with a large nonlinear state and the Muon optimizer.
 317 We refer the readers to each figure and its caption for more detailed analysis.

318 **Experiment details.** The analyses in this section used the following experiment configurations:

319 For the NVS analysis, we utilized an object dataset. For the state size scaling experiments, we train
 320 all compared approaches for 167 billion tokens using model specs of 14 stacked blocks with model
 321 dimension of 768. For the comparison between different optimizers, and the comparison between
 322 linear and nonlinear fast weight function, we train all compared approaches for 671 tokens using
 323 model specs of 24 stacked blocks with model dimension of 768.

324 Language modeling analyses were performed at a 760M parameter scale, training for 40 billion
 325 tokens. Both the sliding window attention (SWA) window size and LaCT chunk size were set to 2048
 326 tokens.

327 For the state size scaling experiments, we keep model dimension fixed ($d = 768$) and increase the
 328 intermediate multiplier of the fast weight SwiGLU MLP. For example, with an intermediate multiplier
 329 of 2 results in a hidden dimension of the fast weight SwiGLU MLP of 1536, and a total state size per
 330 block of $6d^2 \simeq 3.37$ MB.

331 For experiment with different test-time optimizer, our “momentum” variant follows Titans [5]. We
 332 predict a scalar momentum term β_i from each token:

$$\beta_i = \sigma(\text{Linear}(\mathbf{x}_i)), \quad (13)$$

333 where σ is the sigmoid function. This β_i is then averaged over all tokens in the chunk, and the average
 334 momentum is applied as follows:

$$\begin{aligned} g &\leftarrow \sum_i^b \eta_i \nabla_W \mathcal{L}(f_W(k_i), v_i), \\ M &\leftarrow M(\sum_i^b \beta_i / b) + g, \\ W &\leftarrow \text{weight-update}(W, M), \end{aligned} \quad (14)$$

335 where the weight-update can be simple subtraction followed by L2 normalization normalization (as
 336 in Equation 8). or Muon update before subtraction (as in Equation 9).

337 **Experiment details for Large-Chunk v.s. Per-token Recurrence.** Figure 8(b) includes controlled
 338 experiments for a fair comparison between our large-chunk recurrence and per-token recurrence
 339 strategies. In the novel view synthesis (NVS) task, “Our Linear” variant employs a linear fast weight:
 340 $f_W(q) = Wq$ and is benchmarked against a Mamba-2 baseline (a linear per-token recurrence model)
 341 with an identical state size. To accommodate the bidirectional context required by NVS over input
 342 image tokens, the Mamba-2 baseline uses two Mamba-2 layers applied in opposite directions within
 343 each model block. Both our linear variant and this bidirectional Mamba-2 have state size of d^2 per
 344 block. Both of these two approaches employs a per-image window attention within each model

³This anonymous website was not updated after the supplementary material due. The local version is attached in the supplementary folder.

block. Under this fair comparison, our linear large-chunk recurrence achieves significantly better view synthesis performance.

For the language modeling experiments also shown in Figure 8(b), the blue line “Our Linear” variant uses the same state size ($0.25d^2$) as the GLA SWA baseline. It initially underperforms GLA SWA (blue line underperforms yellow line), likely because language data lacks the inherent chunk structures that benefit our basic linear chunk recurrence. However, when LaCT is equipped with a larger nonlinear state ($1.5d^2$) and Muon updates, we significantly outperforms these per-token recurrence baselines.

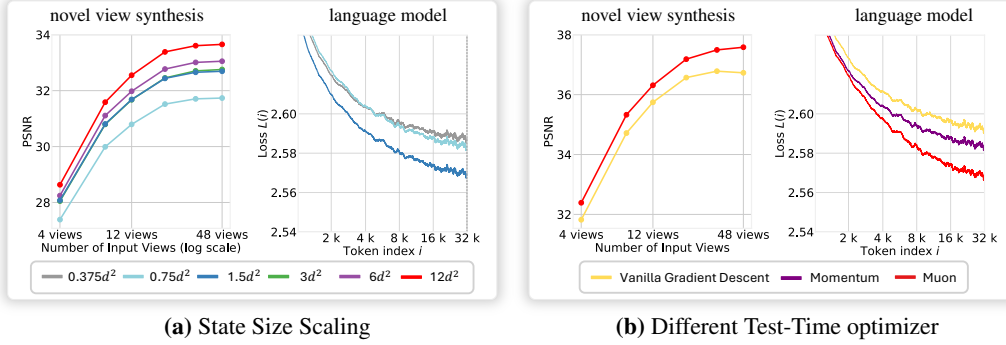


Figure 7: (a) Scaling up the state size consistently improves performance in both novel view synthesis and language modeling tasks. Note, the largest version has state size of $12d^2$ per block, totaling 40% of model weights as fast weights. (b) Comparison of test-time optimizers demonstrates Muon’s surprising effectiveness over Vanilla Gradient Descent and Momentum.

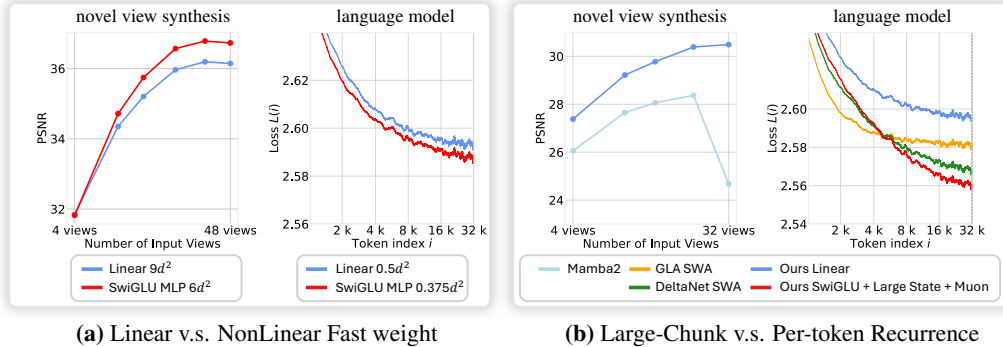


Figure 8: (a) Nonlinear fast weights consistently outperform linear fast weights despite using smaller state sizes. (b) Our linear large-chunk recurrence approach significantly outperforms linear per-token recurrence (bidirectional Mamba2) for view synthesis tasks at the same state sizes. In language tasks, linear large-chunk recurrence of the same state size underperforms the GLA baseline, but when combined with larger nonlinear states and Muon test-time optimizer, it surpasses all per-token recurrence methods.

B Related Work

Test-time training. Test-Time Training (TTT) [2] is an emerging concept in sequence modeling that extends the concept of recurrent states in RNNs to online-adapted neural network components. In TTT models, a subset of weights, termed “fast weights,” are updated online to store in-context information. Existing methods typically employ a self-supervised loss that encourages these fast weights to memorize key-value associations from in-context tokens, using variants of gradient descent for online adaptation.

TTT [2, 4] has opened a vast design space for new recurrent model architectures. For instance, many recent works have developed novel test-time optimizers [5, 7] and online training objectives [40].

However, current TTT approaches often suffer from low hardware utilization and limited state sizes, and consequently have not yet demonstrated their full potential. Our work primarily addresses these challenges by advocating for a new paradigm of using extremely large online minibatch (chunk) sizes for updating the fast weights. This paradigm can achieve orders-of-magnitude higher hardware utilization without relying on error-prone custom kernel implementations. Furthermore, it enables efficient scaling of nonlinear state sizes and offers the flexibility to use diverse fast weight neural networks and optimizers, thereby accelerating research progress in this area.

Combining chunk attention with recurrence. Several recent models combine local chunk attention with linear recurrence, such as Gated Attention Unit (GAU) [41], MEGA [42], MEGALODON [43], and InfiniAttention [44]. Among these, InfiniAttention is conceptually closest to our work, as it incorporates recurrence at the chunk level using the delta rule—interpreted as an online linear regression objective from the perspective of Test-Time Training (TTT). However, this update rule is limited in expressivity. In contrast, we employ a significantly more expressive update mechanism derived from a more general TTT framework, and demonstrate the substantial gains this brings.

Block-Recurrent Transformer [45] also explores large chunk memory updates, where memory tokens act as recurrent states that can self-attend and cross-attend with input tokens during each chunk update via attention mechanisms. The Perceiver-style register-token attention baseline used in our novel view synthesis experiments (Sec. 5.1, Table 2) is conceptually similar to the Block-Recurrent Transformer in its use of register tokens for context compression. As shown in Figure 4, our method significantly outperforms this approach in both speed and quality, with a comparable state size.

Novel view synthesis. Novel view synthesis (NVS) is a long-standing task at the intersection of computer vision, graphics, and computational photography, requiring algorithms to render images of a static scene from previously unobserved viewpoints. Optimization-based approaches, such as NeRF [46] and 3D Gaussian Splatting [9], have achieved significant breakthroughs. These methods optimize a set of parameterized graphics primitives (i.e., explicit or implicit representations of radiance fields) through differentiable volumetric rendering to minimize reconstruction loss on input images. After an optimization process typically lasting tens of minutes, these approaches can render novel views photorealistically, and the optimized parameters form a 3D representation of the input scene.

Recently, data-driven approaches [26, 23, 30, 47] have also shown promising results. These methods can either directly render novel views or predict 3D representations given input images. Although successful on simpler object datasets, these methods often struggle with densely sampled scenes (e.g., scenes with over 100 input images). Our experiments demonstrate that our large-chunk test-time training approach outperforms or achieves comparable performance to 3D Gaussian Splatting on challenging scene datasets with up to 128 input images with 536×960 resolution at challenging scene datasets. We hope our method will inspire further research into effectively scaling data-driven NVS methods to longer and more complex input sequences.

Autoregressive video generation. Current state-of-the-art video generation is dominated by bidirectional diffusion transformers operating in latent space [48, 49, 50, 38]. These models typically factorize the video distribution into a sequence of conditional distributions based on noise levels, following diffusion processes [51] or flow matching [52]. Autoregressive video generation introduces an additional temporal dimension to this factorization, where video chunks are generated iteratively, each conditioned on previously generated (and denoised) chunks.

During training, some autoregressive methods employ teacher forcing, using clean context frames and noisy subsequent frames as input [53], though this can lead to low token utilization, i.e. only a small portion of tokens get supervision. To improve token efficiency, other techniques such as progressive noise injection [54] or the use of frame-independent noises (sometimes in a diffusion-forcing style) [55, 56, 57] have been proposed. When applying our large-chunk design to autoregressive video generation, we format the input sequence with interleaved clean and noisy chunks (see Equation 12). This strategy achieves over 50% token utilization and integrates effectively with our large-chunk TTT implementation, by only changing a few lines to constrain fast-weights are only updated on clean frame chunks.

414 B.1 Limitation

415 We conduct our experiments on three tasks. Although the tasks are diverse and cover different
416 modalities, the effectiveness of our method would request of more tasks. For example, the novel-view
417 synthesis task is essentially a 3D reconstruction with input pose information. The task of unposed
418 reconstruction is more challenging and is not explored in this paper.

419 On the language modeling task, some key aspects are not explored due to computation limitation.
420 These aspects include the reasoning capacity of our LaCT model and also the scalability regarding
421 the parameter size. Previous papers showed that a main weakness of the state-based model (where
422 LaCT belongs to) is its reasoning ability. However, the reasoning ability is only gained with certain
423 amount of training compute thus it is beyond our budget.

424 Lastly, for the autoregressive video diffusion, it is hard to find a reliable and distinguishable metric to
425 measure the model’s scalability. It is in contrast to the language modeling with perplexity (i.e., log
426 likelihood loss) and the novel-view synthesis with PSNR. We show the validation loss in our paper
427 and it is a common choice in evaluating the scalability of video generation. This is a general problem
428 for the video generation evaluation and is not specific to our paper.

Algorithm 1 Large Chunk Test-Time Training Layer Pseudocode

```

def apply_fw(fast_weight, q):
    w1, w2, w3 = fast_weight
    hidden = silu(matmul(q, w1)) * matmul(q, w3) # [b, 1, dh] = [b, 1, d] x [b, d, dh]
    return matmul(hidden, w2)

def update(fast_weight, k, v, lr, use_muon=True):
    """
    Fast-weight update for a SwiGLU MLP using chunk of tensors.
    Args:
        fast_weight : tuple(w1, w2, w3) with shapes: w1, w3: [b, d, dh]; w2: [b, dh, d]
        k, v : key / value tensor of shape [b, 1, d]
        lr: : per-token learning rates of shape [b, 1, 3] -> (lr1, lr2, lr3)
        use_muon : whether to apply Muon to orthogonalize the update
    Note:
        The head dimension for input tensors k, v, lr are assumed to be merged into the batch dimension.
        This is not necessary, but simplifies shape annotation in this pseudocode.
    """
    # Forward with k:
    gate_before_act = matmul(k, w1) # [b, 1, dh] = [b, 1, d] x [b, d, dh]
    hidden_before_gate = matmul(k, w3) # [b, 1, dh] = [b, 1, d] x [b, d, dh]
    hidden = silu(gate_before_act) * hidden_before_gate

    # Backward:
    dhidden = matmul(v, w2.transpose(-1, -2)) # [b, 1, dh] = [b, 1, d] x [b, d, dh]
    dhidden_before_gate = dhidden * silu(gate_before_act)
    dgate = dhidden * hidden_before_gate
    dgate_before_act = silu_backprop(dgate, gate_before_act)

    # Compute gradients:
    w2.grad = -matmul(hidden.transpose(-1, -2), v * lr2) # [b, dh, d] = [b, dh, 1] x [b, 1, d]
    # [b, d, dh] = [b, d, 1] x [b, 1, dh]
    w1.grad = -matmul((k * lr1).transpose(-1, -2), dgate_before_act)
    w3.grad = -matmul((k * lr3).transpose(-1, -2), dhidden_before_gate)

    # Weight update
    if use_muon:
        for w in fast_weight:
            w.grad = zeropower_via_newtonschulz5(w.grad)
        for w in fast_weight:
            w = (w - w.grad) / (w - w.grad).norm(dim=1) * w.norm(dim=1)
    return fast_weight

def silu_backprop(dy, x):
    sigma = sigmoid(x)
    return dy * sigma * (1 + x * (1 - sigma))

##### MultiHead LaCT layer #####
# x: input sequence [b, 1, d], b is the batch dim, 1 is length, d is model dimension
# fast_weight: tuple of initial fast weights-(w1, w2, w3); w1, w3 of shape [nh, d, dh], w2: [nh, dh, d]
# ttt_config: list of (operation, begin, end) tuples
qkv = silu(LinearQKV(x)) # [b, 1, d * 3]
qkv = rearrange(qkv, 'b 1 (nh hd)' -> (b nh) 1 hd', nh=num_heads).split(3) # merge heads into batch dim
q, k = q / q.norm(-1), k / k.norm(-1)
lr = softplus(LinearLR(x) + const_lr_bias) # [b, 1, 3 * num_heads]
lr = rearrange(lr, 'b 1 (nh 3)' -> (b nh) 1 3', nh=num_heads)
fast_weight = repeat(fast_weight, dim=0, repeat=b) # [nh, ...] -> [b * nh, ...]

o = zeros_like(v) # [b * nh, 1, hd]
for mode, begin, end in ttt_config:
    qi, ki, vi, lri = q[:, begin:end], k[:, begin:end], v[:, begin:end], lr[:, begin:end]

    if mode == "update_then_apply": # figure-3(a, b) bidirectional attention
        fast_weight = update(fast_weight, ki, vi, lri, use_muon)
        o[:, begin: end] = apply_fw(fast_weight, qi)

    elif mode == "apply_then_update": # figure-3(b) shifted block-wise causal
        o[:, begin: end] = apply_fw(fast_weight, qi)
        fast_weight = update(fast_weight, ki, vi, lri, use_muon)

    elif mode == "update_only":
        fast_weight = update(fast_weight, ki, vi, lri, use_muon)

    elif mode == "apply_only":
        o[:, begin: end] = apply_fw(fast_weight, qi)

o = RMSNorm(o) # per-head norm
o = LinearOutput(rearrange(o, '(b nh) 1 hd' -> b 1 (nh hd)', nh=num_heads))
return o

```

Algorithm 2 LaCT Layer with In-Layer Hybrid Window Attention Pseudocode

```

# Input:
# x: input sequence [b, l, d], b is the batch dim, l is length, d is model dimension
# fast_weight: tuple of initial fast weights-(w1, w2, w3); w1, w3 of shape [d, dh], w2 of shape [dh, d]
# ttt_config: list of (operation, begin, end) tuples

q, k, v = LinearQKV(x).split(3)

#### Local quadratic-cost window attention
attn_q = q * learnable_q_scale + learnable_q_offset # per-channel rescale and shift
attn_k = k * learnable_k_scale + learnable_k_offset # per-channel rescale and shift
attn_o = local_softmax_multihead_attn(attn_q, attn_k, v, attn_mask)

#### large chunk test-time training for long memory
q, k = rearrange(q, k, 'b l (nh hd) -> (b nh) l hd', nh=num_heads)
q, k = silu(q), silu(k)
q, k = q / q.norm(-1), k / k.norm(-1)
lr = softplus(LinearLR(x)) # [b, l, 3 * num_heads]
lr = rearrange(lr, 'b l (nh 3) -> (b nh) l 3', nh=num_heads)

# Perform update and apply_fw operations iteratively over chunks of tokens.
lact_o = lact(fast_weight, q, k, v, lr, ttt_config)
lact_o = RMSNorm(lact_o)

scale_per_head = rearrange(silu(Linear(x)), 'b l nh -> (b nh) l 1', nh=num_heads)
lact_o = lact_o * scale_per_head
lact_o = rearrange(lact_o, '(b nh) l hd -> b l (nh hd)', nh=num_heads)

#### Merge attention results (shape: [b, l, d])
o = attn_o + lact_o

o = LinearOutput(o)

return o

```

430 **State Size calculation.** Motivated by recent progress in LLM, we adopt SwiGLU-MLP [18]
 431 without bias terms as the fast-weight network. Our fast weights consists of three weight matrix
 432 $W = \{W_1, W_2, W_3\}$, and the forward pass of the fast weight model is:

$$f_W(x) = W_2 [\text{SiLU}(W_1 x) \circ (W_3 x)] \quad (15)$$

433 where \circ is an elementwise multiplication. We define hd as the head dimension, nh as the number
 434 of heads, and the intermediate dimension of the SwiGLU-MLP as $hd \times r$, where r is a scaling
 435 multiplier. When $r > 1$, the intermediate dimension surpasses the input head dimension, which is
 436 the current common practice in LLMs. Thus, matrices $W_1, W_2 \in \mathbb{R}^{hd \times hd}$ and $W_3 \in \mathbb{R}^{hd \times hd \times r}$.
 437 Consequently, the total state size becomes $nh \times hd \times hd \times r$. Given that typically the total head
 438 dimension across all heads equals the model dimension d (i.e., $nh \times hd = d$), the total state size
 439 simplifies to:

$$\text{State Size} = \frac{d^2}{nh} * r. \quad (16)$$

440 Therefore, we can increase the state size either by reducing the number of heads or by increasing the
 441 intermediate dimension multiplier.

442 **FLOPS calculation.** When using then negative dot product loss as the online test-time training
 443 objectives, we don't need to compute the final results of $f_W(v)$. We only need to compute $W_1 v, W_3 v$
 444 when runing forward pass with keys k , thus there are two matmuls in the forward pass with keys.
 445 When computing the gradients, there are four matmuls. And in the final forward pass the queries,
 446 there would be three matmuls. So the total flops with n tokens would be:

$$\text{FLOPS} = 4n \frac{d^2}{nh} r + 8n \frac{d^2}{nh} r + 6n \frac{d^2}{nh} r = 18n \frac{d^2}{nh} r = 6 * \text{State Size} \quad (17)$$

447 **Model initializations.** We randomly initialize the linear layers using a standard deviation of 0.02.
 448 For the learnable initial fast weights, we initialize them with a standard deviation of $1.0/\sqrt{\text{fan-in}}$.
 449 Specifically, in the SwiGLU FFN fast weights, the matrices w_1 and w_3 have their fan-in set as the head
 450 dimension, while the fan-in of w_2 is the intermediate dimension of the SwiGLU FFN fast weights.
 451 Additionally, when local window attention is incorporated within the LaCT layer, we introduce four
 452 extra learnable embeddings: two scales and two reshifts for queries and values. We initialize the
 453 scale embeddings as ones and the reshift embeddings as zeros.

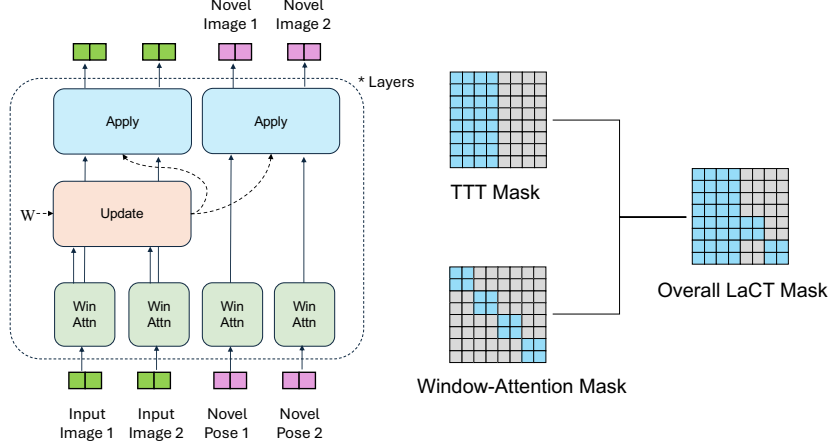


Figure 9: Detailed LaCT model for our Novel-View Synthesis. Dashed line indicates flow of fast weight. Solid line indicates flow of tokens. Window attention is bidirectional within a single image, either the input image or the novel target image. TTT updates over all input tokens and apply to all tokens.

Details of Muon Muon [8] is a recently proposed optimizer that orthogonalizes the matrix gradients during updates of matrix weights. It utilizes Newton-Schulz iterations to achieve orthogonalization. Given a matrix gradient G , Muon first normalizes it as $G_0 = G/|G|F$, then iteratively applies:

$$\mathbf{G}_k = a\mathbf{G}_{k-1} + b(\mathbf{G}_{k-1}\mathbf{G}_{k-1}^T)\mathbf{G}_{k-1} + c(\mathbf{G}_{k-1}\mathbf{G}_{k-1}^T)^2\mathbf{G}_{k-1}, \quad (18)$$

where the constants a, b, c are carefully chosen for optimal convergence. Following the original implementation, we set $a = 3.4445$, $b = -4.7750$, $c = 2.0315$, and perform five iterations.

Each Muon iteration requires three matrix multiplications, resulting in a computational cost per fast weight head of $2hd^3r + 2hd^3 + 2hd^3r = hd^3(4r + 2)$ FLOPS. Hence, the total computation for five iterations across all fast weights is:

$$5 \times nh \times hd^3 \times (4r + 2). \quad (19)$$

For the case where $r = 1$ (head and intermediate dimensions are equal), the total computational cost simplifies to:

$$30 \times nh \times hd^3 = 30 \times hd \times \text{State Size}. \quad (20)$$

This indicates that the computational overhead of Muon becomes less significant than computing token outputs only if the online chunk size exceeds $\frac{5}{3}hd$.

Rotation invariance. Softmax attention and linear attention exhibit rotation invariance: rotating the queries and keys by the same rotation matrix does not alter the output. This property is also used in developing relative positional encodings, like RoPE [37]. In contrast, our SwiGLU and Linear Fast Weight components do not possess this property.

C.1 Pseudocode

See [1] for pseudocode of a full LaCT layer. For details on how to mix local window attention inside each layer with shared query, value, embedding, see [2] for pseudocode.

D Detailed Model Architectures

D.1 LaCT Architecture for Novel View Synthesis

Novel view synthesis (NVS) renders images of a static scene from novel viewpoints. Formally, given a set of N input posed images $\{(I_i, P_i)\}_{i=1}^N$ of a static scene, where $I_i \in \mathbb{R}^{H \times W \times 3}$ is an RGB image

477 and P_i is its corresponding camera pose, the model needs to synthesize new images from novel
 478 camera poses P_{novel} that typically do not overlap with the input views.

479 Traditional methods in 3D vision usually solved the NVS task by reconstruction and rendering. The
 480 reconstruction compresses the posed input into a compact representation and then the render renders
 481 the novel view from it. Our method mimics such pipeline, where we first compress the posed input
 482 images into fast weights by the ‘Update Operation’ (Sec. 3.1) in LaCT. Then we render the novel
 483 view images from the information in the compressed weights with the ‘Applying Operation’.

484 In details, we first convert the input and output into tokens. The camera pose P for each view is
 485 represented in dense ray information for each pixel (usually from the camera intrinsics and extrinsic),
 486 i.e., $P = (\text{rays}_o, \text{rays}_d)$. $\text{rays}_o \in \mathbb{R}^{H \times W \times 3}$ is the 3D coordinate for the origins of the ray, and
 487 $\text{rays}_d \in \mathbb{R}^{H \times W \times 3}$ is the direction of the ray. We follow GS-LRM [26] to use the Plücker ray
 488 embedding for the rays. Plücker ray embedding computes the cross product between the ray origin
 489 and ray direction for a normalization. The final positional embedding is a concatenation of the ray’s
 490 origin, the ray’s direction, and the cross product of the above two: $[\text{rays}_o, \text{rays}_d, \text{rays}_o \times \text{rays}_d]$. We
 491 add ray’s origin into the embedding since different origins in the same ray can results in different
 492 colors due to the occlusions. We then use patchifying and two different Linear layers to convert the
 493 RGB map and ray map (i.e., the positional embedding) into tokens. For the posed RGB input images,
 494 we simply the sum the RGB embedding and pose embedding as model input. For the novel view
 495 cameras, we only use the pose embedding.

496 We illustrate the design of NVS’s LaCT block in Fig. 9. We first apply the attention for each image
 497 (either input or the novel target). The attention is bidirectional for the tokens belonging to the same
 498 image, and is independent among different images. Then, the TTT update operation is applied to all
 499 input tokens, i.e., all tokens that belonging to all input images. The updated weight then is applied to
 500 all tokens. The two updates blocks in Fig. 9 take the same updated fast weight thus can be combined,
 501 and we left two ‘Apply’ block for clearance. Note that the original NVS task definition renders
 502 novel views independently. We here supervise multiple novel image poses and their corresponding
 503 images in a single data point for better training efficiency. Given the design, the novel images are
 504 independent to each other, which is illustrated in the ‘Overall LaCT Mask’ in the right of Fig. 9.
 505 The layer normalization layer, the residual connections, and the feed-forward network is omitted for
 506 clarity. The block is repeated by number of layers times to formulate the full model. The general
 507 model largely follow the design of the encoder-decoder model in LVSM [23], except we use TTT in
 508 replace of transformer for long-context modeling.

509 For actually using this model for NVS task during inference, we first get the updated fast weight
 510 with all input images. Then, we would not change the fast weight during the rendering process (i.e.,
 511 the process to convert novel camera poses to the novel images). The LaCT during rendering would
 512 be similar to a ViT (Vision transformer) architecture despite having two feed-forward networks:
 513 the feed-forward network from the fast weight stores the scene information, and the feed-forward
 514 network from the slow weight (i.e., the FFN in Fig. 2) stored the world knowledge like physical
 515 rendering rules.

516 D.2 LaCT Architecture for Language Models

517 Autoregressive Language Models (LM) predicts the distribution of the next tokens
 518 $p_\theta(x_n|x_1, \dots, x_{n-1})$ from its history context. It is a factorization of the full sequence distribu-
 519 tion $p_\theta(x_1, \dots, x_n)$ through chain rule $p_\theta(x_1, \dots, x_n) = p_\theta(x_1)p_\theta(x_2|x_1) \dots p_\theta(x_n|x_1, \dots, x_{n-1})$.
 520 Thus it requires a token-level causal mask (demonstrated in the topright of Fig. 10) and this is the
 521 main difficulty for the large-chunk design in LaCT. We use a combination of TTT layer with ‘Shifted
 522 Causal Block Mask’ (introduced before in Fig. 3c) and a sliding window attention to facilitate it.
 523 By shifting the mask of TTT, it excludes the information leakage from future tokens. As shown in
 524 the right part of Fig. 10, the overall dependency mask is the union of the TTT mask and the sliding
 525 window attention mask. To achieve a token-level causal mask without bubbles, the only requirement
 526 is that the window size of the sliding window attention is greater or equal to the chunk size from
 527 the TTT. We illustrate two example of such mask with ‘Window Size’ = ‘TTT Chunk Size’ = 2 or
 528 4. The actual chunk size and attention window size is above 2048 in our implementation for better
 529 utilization and state size scaling (discussed in Sec. 2.2).

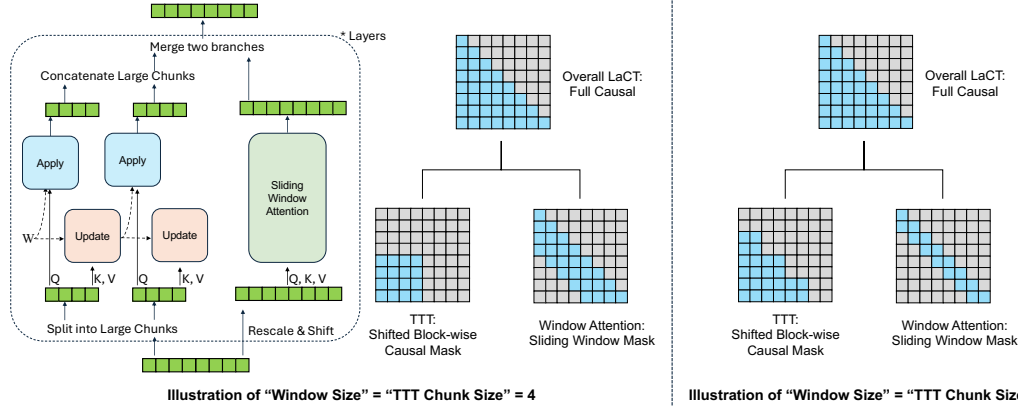


Figure 10: Detailed LaCT model for language models. Dashed line indicates flow of fast weight. Solid line indicates flow of tokens. We illustrate with TTT chunk size 4 or 2, and the actual chunk size is over 2048 in LaCT. We take the parallel design for the window attention and TTT block with shared QKV. The overall mask is the causal mask.

As illustrated in the left most of Fig. 10, we employ a parallel design of the TTT layer and sliding window attention to save the number of model parameters and computation FLOPs. In details, the query (Q), key (K) and value (V) are shared between the TTT layer and window attention. Sliding window attention is an attention with constant window size over the past history, starting from the target tokens. For the TTT layer, we start with an apply operation over the first chunk using the initialized fast weight (i.e., unupdated yet). The ‘apply’ operation is followed by the ‘update’ operation over the first chunk. In this way, the ‘apply’ operation would not see information inside the current chunk to avoid leaking future token information inside the chunk. Alternatively using ‘apply’ followed by ‘update’ over subsequent blocks completes the desired ‘shifted block-wise causal mask’ illustrated in Fig. 10. For details of the parallel design, please refer to the Pseudocode Algorithm 2.

We use the multi-head design for both TTT layer and window attention, although their number of heads are different. We empirically take less number of heads for TTT layer (i.e., larger head dimension) to enable larger state size, as state size is proportional to the head dimension in our design (Sec. C and Equation 16). By default, we use four heads in language model experiments. For positional encoding, we use the same RoPE as the window attention branch.

D.3 LaCT Architecture for Autoregressive Video Diffusion

Chunkwise autoregressive video diffusion generates videos by iteratively denoising sequential chunks of video frames, conditioned on previously generated clean frames. Each chunk can contain several video frames and span thousands of visual tokens. We use teacher-forcing training by interleaving noisy and clean frame chunks. Specifically, a video of N frame chunks is structured as:

$$S = [X_1^{\text{noise}}, X_1, X_2^{\text{noise}}, X_2, \dots, X_N^{\text{noise}}] \quad (21)$$

where each noisy chunk X_i^{noise} is produced by adding unit Gaussian noise ϵ to the i -th clean video chunk as $X_i^{\text{noise}} = X_i(1 - t_i) + \epsilon t_i$ and $t_i \in [0, 1]$ denotes the strength of chunk-independent noise. However, compared to previous methods that employs progressive [54] or frame-independent noise strategies [55] like diffusion forcing [56], our teacher-forcing formulation in Equation 21 only uses around 50% of the tokens of the entire sequence to compute the denoising loss. To improve token utilization, we consider an alternate approach by repeating each video chunk with two noise levels in the training sequence as:

$$S = [X_1^{\text{noise}}, X_1^{\text{noise}*}, X_1, X_2^{\text{noise}}, X_2^{\text{noise}*}, X_2, \dots, X_N^{\text{noise}}, X_N^{\text{noise}*}], \quad (22)$$

where X_i^{noise} and $X_i^{\text{noise}*}$ represent two different noise levels applied to each clean video chunk X_i . This increases token utilization from 50% to about 67%. While more repetition could further increase token utilization, it would also reduce training sample diversity; thus, we limit the repetition to twice. We use such repeating strategy when training the 1.3 billion parameter video diffusion model on five seconds videos.

fast-weight SwiGLU-MLP with a hidden dimension of 1536. The window attention has 12 heads with head dimension 64, and is equipped with QK-normalization [59]. The Feed-forward Network has 3072 as its intermediate hidden dimension. The model has a total of 312M parameters, of which 84M are fast weights (i.e., $6d^2$ per block). We use an fast-weight lr initialization of 0.01 by setting ‘const_lr_bias’ in Algorithm 1 to $\text{softplus}(\text{const_lr_bias}) = 0.01$. As we used Muon in fast weight update for NVS, LaCT is not sensitive to lr scale as discussed in Sec. 3.2.

Baselines. For object-level evaluations, we compare against two baselines, including a full-attention model, and a register-attention model in a Perceiver style [29]. In the full-attention baseline, we replace the TTT layer in our model with a block-wise causal attention layer, where the input tokens interact bidirectionally and the novel view tokens cross-attend to the input tokens. Such a design resembles our method’s prefill and parallel decoding strategy described in Section 4.1, and the key-value caches of the input tokens server as scene representations for novel view renderings. In the Perceiver-style model, we replace half of the TTT layers with input-to-register full-attention layers and the remaining half with register-to-novel-view cross-attention layers. Such a model first compresses the input tokens into a constant set of register tokens and then decoding the novel view tokens by attending to the registers. For scene-level evaluations, we compare against a state-of-the-art long-sequence 3D reconstruction work LongLRM [30] that applies Mamba [12] hybrid with full attention to predict 3D Gaussian splats [9]. We also include comparisons with pure optimization-based 3D Gaussian splatting methods. Tab. 2 compares the computational complexity of the baseline models and our models.

Training details. For object-level experiments, we first train all the model with 671B tokens at 8 input view and 8 novel view setting at a resolution of 256×256 . We then finetune them with 512×512 resolution for an additional 587B tokens. For scene dataset, we first pre-train our model first with 32 input views and 32 novel views at 128×128 resolution for 1.5T tokens, then progressively finetune at larger resolutions, larger field-of-views, and more input views. The finetuning is always go with a non-squared FoV to match the raw data. Non-squared FoV has larger view range than the squared FoV, thus is harder. The input and novel views in fine-tuning are both 64 to support better view coverage. The curriculum of the fine-tuning resolution is set as 72×128 , 144×256 , 288×512 , and 536×960 . The training tokens for each stage is around 100B. High-resolution models (starting from 288×512) are trained with inner-chunk context parallelism (Sec. 3.4).

At each training stage, we always use AdamW with linear learning rate warmup and weight decay of 0.05. The peak learning rate of the pre-training is $4e - 4$. During fine-tuning, we use smaller learning rate (usually $1e - 5$ to $5e - 5$).

The training is completed with 64 A100 GPUs. The pre-training takes 8 days, and each fine-tuning stage is about 12hours (thus 2 days in total).

Detailed Result Numbers We here provided the detailed number for object-level results on the GSO dataset (at resolution 256×256 in Table 4, 512×512 in Table 5) and DL3DV evaluations (at resolution 960×536 in Table 6). The DL3DV number is slightly higher than the main paper as it continues training for about 4 hours.

Table 4: 256-Res object-level novel view synthesis results on GSO. Both the input and output are with resolution 256×256 comparison across methods. \uparrow : higher is better, \downarrow : lower is better.

Views	LaCT			Full Attention			Perceiver Attention		
	PSNR (\uparrow)	LPIPS (\downarrow)	SSIM (\uparrow)	PSNR (\uparrow)	LPIPS (\downarrow)	SSIM (\uparrow)	PSNR (\uparrow)	LPIPS (\downarrow)	SSIM (\uparrow)
4	32.4	0.030	0.962	32.6	0.029	0.964	30.3	0.039	0.950
8	35.3	0.019	0.976	35.6	0.018	0.978	32.8	0.026	0.967
12	36.3	0.017	0.980	36.6	0.015	0.982	33.6	0.023	0.971
20	37.2	0.015	0.982	37.5	0.014	0.984	34.3	0.021	0.974
32	37.5	0.014	0.982	37.9	0.013	0.985	34.2	0.021	0.974
48	37.6	0.014	0.983	37.9	0.013	0.985	33.7	0.022	0.972

E.2 Language Modeling

Datasets & Metrics. We train our models on the Long-Data-Collections dataset [31], containing approximately 68.8B tokens tokenized using Mixstra tokenizer (32,000 codebook size). The dataset is a mix of 41.4% General Data (e.g., RedPajama-Book, RedPajama-ArXiv, 1B tokens from RedPajama,

Table 5: 512-Res object-level novel view synthesis results on GSO. Both the input and output are with resolution 512×512 comparison across methods. \uparrow : higher is better, \downarrow : lower is better.

Views	# Input Tokens	LaCT			Full Attention		
		PSNR (\uparrow)	LPIPS (\downarrow)	SSIM (\uparrow)	PSNR (\uparrow)	LPIPS (\downarrow)	SSIM (\uparrow)
4	16,384	33.4	0.029	0.969	33.6	0.027	0.971
8	32,768	36.6	0.020	0.979	36.7	0.017	0.982
12	49,152	37.7	0.017	0.983	37.9	0.015	0.985
20	81,920	38.6	0.016	0.984	38.9	0.013	0.987
32	131,072	39.0	0.015	0.985	39.3	0.013	0.988
48	196,608	39.1	0.015	0.985	39.3	0.012	0.988

Table 6: 960P scene-level novel view synthesis results for LaCT on DL3DV. Both the input and output are with resolution 960×536 (width x height). \uparrow : higher is better, \downarrow : lower is better.

Views	# Input Tokens	PSNR (\uparrow)	LPIPS (\downarrow)	SSIM (\uparrow)
16	128,640	24.7	0.224	0.793
32	257,280	26.9	0.185	0.837
64	515,520	28.3	0.169	0.857
128	1,031,520	28.9	0.166	0.861

and a Pile subsample) and 58.6% Instruction Data (e.g., UL2 Oscar, NI, and P3). To evaluate long-context capabilities, we utilized the per-token loss metric from [32]. A consistently decreasing per-token loss across the input sequence indicates effective use of the entire context, while a plateau suggests an inability to leverage information beyond that point. Specifically, we evaluated next-token prediction loss on 2.5B tokens from the Book-3 dataset [60] for our 760M-parameter model, and on 5B tokens for our 3B-parameter model. Additionally, we measured retrieval accuracy using the RULER benchmark [33] across various sequence lengths to assess context memorization and information retrieval, evaluating up to the trained sequence length.

Model details. We modified the original LaCT block by removing its window-attention layer. Instead, we incorporate a causal sliding window-attention(SWA) layer directly into the Large-Chunk TTT layer. The SWA layer shares the same Q, K, and V vectors as the fast-weight network, except that a per-channel learnable scale and shift is applied to Q and K before they are fed to the SWA layer (as done in GAU [34]). We sum up the output of the SWA layer and that of the TTT layer, where the output of the TTT layer is scaled by another per-head learnable scalar. We use an fast-weight lr initialization of 0.001 by setting ‘const_lr_bias’ in Algorithm 1 to $\text{softplus}(\text{const_lr_bias}) = 0.001$. We illustrate this architecture in Figure 10.

To ensure a fair comparison with baselines in terms of trainable parameters, we adjusted the LaCT block’s extra learnable initial fast weights $W = [W_1, W_2, W_3]$. To reduce parameters, we employed a low-rank version for W_1, W_3 with a rank of 32. For instance, if $W_1 \in \mathcal{R}^{d \times d}$, its low-rank initial fast weight is $W_1 = L \cdot R + 0.5 \cdot I_d$, where $L \in \mathcal{R}^{d \times 32}$, $R \in \mathcal{R}^{32 \times d}$, and I_d is identity matrix. This reduces the extra trainable parameters for the fast weights in each block to $128 * \text{model-dim} + \frac{1}{\text{num-heads}}(\text{model-dim}^2)$. Additional minor parameters for learning rate projection, per-head scalars, an extra RMSNorm, and the SWA’s learnable scale and shift are of order $O(\text{model-dim})$. Standard blocks typically have $12 * \text{model-dim}^2$ parameters. Our approach adds approximately $\frac{1}{\text{num-heads}}(\text{model-dim}^2)$ extra parameters, which is less than 3% of total trainable weights with four heads, and below 4.5% with two heads. By default, LaCT use four heads in the experiments, unless noted otherwise, which means that the default state size per block is $\frac{3}{4}(\text{model-dim}^2)$.

Baselines. We compare our approach with full attention, Gated Linear Attention (GLA) [13], DeltaNet [3, 15]. To ensure fairness, we enhance both GLA and DeltaNet with the same sliding window attention. As pointed out in previous work [32, 35, 36], a large RoPE [37] base is critical for transformers in long-context training, thus we adopt a large RoPE base of 1 million for training with 32K token contexts whenever softmax attention is used. Tab. 7 compares the mechanism and computing complexity of the baseline methods and our method. Training throughput (tokens per second per GPU, TPS) was using a 3B-parameter model on eight A100-40GB SXM4 GPUs with activation checkpointing and FSDP.

Training Details. We trained models at two scales using a sequence length of 32,768 tokens:

Table 7: Comparison of baseline methods in terms of state size, training throughput (measured in tokens per second, TPS), update rules, and memory read-out mechanisms. Training throughput is evaluated using a 3B-parameter model with 32K-sequence length on A100-40GB GPUs.

	State size	Train TPS	Update Rule	Memory read-out
Transformer	–	4.1K	–	–
Transformer SWA	–	6.4K	–	–
<i>Per-token recurrence</i>				
GLA SWA	384d	5.0K	$\mathbf{S}_t \leftarrow \mathbf{S}_{t-1} \text{Diag}(\boldsymbol{\alpha}_t) + \mathbf{v}_t \mathbf{k}_t^\top$	$\mathbf{o}_t = \mathbf{S}_t \mathbf{q}_t$
DeltaNet SWA	128d	5.1K	$\mathbf{S}_t \leftarrow \mathbf{S}_{t-1}(\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top) + \beta_t \mathbf{v}_t \mathbf{k}_t^\top$	$\mathbf{o}_t = \mathbf{S}_t \mathbf{q}_t$
<i>Large-chunk recurrence</i>				
Ours GD	2304d	5.0K	$W \leftarrow \text{L2norm}(W - \sum_i \eta_i \nabla_W \mathcal{L}_i)$	$\mathbf{o}_t = f_W(\mathbf{q}_t)$
Ours Momentum	2304d	4.9K	$M \leftarrow \beta M + \sum_i \eta_i \nabla_W \mathcal{L}_i; W \leftarrow \text{L2norm}(W - M)$	$\mathbf{o}_t = f_W(\mathbf{q}_t)$
Ours Muon	2304d	4.3K	$M \leftarrow \beta M + \sum_i \eta_i \nabla_W \mathcal{L}_i; W \leftarrow \text{L2norm}(W - \text{Muon}(M))$	$\mathbf{o}_t = f_W(\mathbf{q}_t)$

- 760M parameters: We use 24 stack blocks, with model dimension as 1536. All models are trained for 40B tokens (40,960 steps) with a sliding window of 2048 tokens and a batch size of 1 million tokens. Each experiment ran on 32 A100-40GB SXM GPUs for approximately 20 hours.
- 3B parameters: We use 25 stack blocks, with model dimension as 3072. All models are trained for 60B tokens (30,000 steps) with a sliding window of 4096 tokens and a batch size of 2 million tokens. Each experiment ran on 64 A100-40GB SXM GPUs for approximately 50-60 hours.

For both scales, we used a base learning rate of 1×10^{-3} with a cosine decay scheduler and 1024 warmup steps. All models were randomly initialized with a standard deviation of 0.02.

Results. Detailed results on the RULER benchmark [33] are presented in Tables 8 and 9. We evaluated models on S-NIAH-1, S-NIAH-2, and S-NIAH-3 tasks, which represent varying difficulties of the single "needle in a haystack" retrieval. We also report performance on NIAH-MultiKey-1, NIAH-MultiQuery, and NIAH-MultiValue. Other RULER tasks are not reported as the full attention baseline also achieved trivial results beyond a 16K sequence length.

E.3 Autoregressive Video Diffusion

We fine-tune the pretrained Wan 2.1 [38] text-to-video diffusion model into an autoregressive video diffusion model, that generates videos by iteratively denoising successive chunks of video frames.

Model details. The original Wan 2.1 is a bidirectional diffusion transformer operating on the latent space of a causal video VAE, which performs 8x spatial and 4x temporal downsampling. The diffusion transformer uses a $2 \times 2 \times 1$ patchification layer to convert VAE video latents to tokens. Each block of the diffusion transformer comprises an MLP layer, a bidirectional self-attention layer for visual tokens, and a cross-attention layer for visual and text tokens.

Our primary modification is to the bidirectional self-attention. We first replace it with block-causal sliding window attention (SWA), using a window size of two chunks of video frames. We then integrate our LaCT into the same layer. We initialize learnable fast weights for LaCT. Consistent with our language modeling experiments, SWA and our test-time training mechanism are combined within each layer: Q and K vectors are rescaled and shifted before input to the test-time training operation. The outputs of SWA and the test-time training layer are summed, with a per-head learnable scalar (from a zero-initialized linear projection) applied to the latter. We do not use Muon in the fast-weight update, as it showed no significant difference in validation loss empirically. We use an fast-weight lr initialization of 0.001 by setting 'const_lr_bias' in Algorithm 1 to $\text{softplus}(\text{const_lr_bias}) = 0.001$. This allows small update to the fast weight in the beginning of the fine-tuning. To maintain minimal changes to the original Wan architecture, LaCT layers utilize the original RoPE from the Wan model, and we remove the SiLU activation function previously applied to queries and values.

Datasets. We fine-tune the model using an internal, filtered proprietary collection of videos, each accompanied by a short text prompt generated by a visual language model [61].

Training details. Following [39, 38], we use time-step shifting (scale factor 3.0) and logit-normal denoising loss weighting (mean=0.5, std=1.0). We also apply an exponential moving average with

a decay rate of 0.995 to the model weights. Each 5-second video (16 FPS, 480×832 resolution) is encoded by the Wan VAE into a [21,60,104] latent representation. Denoising is performed autoregressively in chunks of three latent frames (4680 visual tokens each). We employ teacher-forcing with an interleaved noisy-clean chunk sequence (see Section 4.3).

- **1.3B Parameter Model:** For initial training on 5-second videos, noisy chunks are repeated twice. This results in sequences of 60 latent frames (14 noisy, 6 clean chunks), totaling 93,600 tokens. We finetune the model with a batch size of 64 for 5000 iterations. The base learning rate is set to 2×10^{-5} with a linear warm-up of 1000 iterations and linear decay. Subsequently, the model is fine-tuned on 10-second video clips for 1,000 iterations. These clips correspond to 42 latent frames for the clean video portion, forming an interleaved sequence of 81 latent frames (approximately 126K tokens including noisy chunks). Training for 5-second videos takes ~20 seconds per iteration on 64 A100 80GB SXM GPUs (or ~10 seconds on 64 H100 80GB SXM GPUs).

- **14B Parameter Model:** To manage GPU memory usage, noisy chunks are not repeated in this setting. We train the model on five-second videos with a batch size of 64 for 5000 iterations with a base learning rate of 5×10^{-6} , and use a sequence parallel size of 2 GPUs. This phase takes ~80 seconds per iteration on 64 A100 GPUs. The model is then fine-tuned on 8.8-second video clips (36 latent frames for the clean portion) for an additional 600 iterations, using sequence parallelism (4 GPUs). This fine-tuning takes ~80 seconds per iteration on 64 H100 GPUs.

Baselines. We compare our method with three baselines: sliding window attention, Mamba2 [20] with sliding window attention, and full block-wise causal attention, where the window attention in the baselines is implemented the same as in our model. For the Mamba2 layer, we follow [62] to apply the original projected k , q , and v as B , C , and x , respectively. The Mamba2’s state is updated token-by-token, we revert the state after processing a noise chunk of frames to ensure only clean chunk state updates propagate. The full block-wise causal attention baseline is implemented with FlexAttention [63].

Evaluation. We validate all models on a collection of 2,000 videos after 5,000 training iterations by computing the denoising loss at five timesteps (550, 650, 750, 850, 950). The denoising losses are measured with respect to each video frame chunk and plotted in Figure 6. Figure 6(a) compares validation loss (up to 5s videos) of LaCT against SWA, Mamba2 with SWA, and full block-wise causal attention. Our LaCT is comparable to full attention and outperforms other baselines. Figure 6(b) shows comparisons with the SWA baseline using different window sizes for both our method and the baseline (up to 5s videos). The default window covers six latent frames (two chunks). An additional experiment used a four-frame window. Results indicate that increasing window size from four to six frames improves validation loss, but this improvement is smaller than that achieved by incorporating LaCT. Figure 6(c) presents validation loss (up to 10s videos) after fine-tuning LaCT and the SWA baseline on 10-second videos for 1,000 iterations.

Generated video samples from our model are provided in an appended folder. Each video chunk is sampled following the original Wan method, using a UniPC [64] sampler with 50 steps, classifier-free guidance of 5.0, and a timestep shift of 3.0.

F Details for Mamba Baselines

Mamba is an efficient model architecture, it is logically similar to a linear TTT taking per-token linear update rule of the fast weight (i.e., state in Mamba’s context). Thus it serves as a baseline to understand the gap between the chunk-wise update and per-wise update in Fig. 8 and Fig. 6. In this section, we detailed the experimental setup.

We take the official Mamba-2 implementation⁵ in all our experiment. The original Mamba-2 has multiple components, and we largely simplify its implementation to keep a measurable architecture

⁵<https://github.com/state-spaces/mamba>

while still maintaining the performance. In detail, our Mamba-2’s formulation in experiment is:

$$\begin{aligned}
X, B, C, \delta &= \text{Linear}(u) \\
\delta &= \text{softplus}(\delta + \delta_{init}) \\
H_t &= \exp(-\delta_t) H_{t-1} + \delta_t B_t^T X_t \\
y_t &= C_t H_t
\end{aligned} \tag{23}$$

where X, B, C is of shape (L, d) , and δ is of shape $(L, 1)$. H_t is a matrix state of shape (d, d) .

Transferring the above formula to a standard linear-attention / TTT / DeltaNet notations, it is equivalent to:

$$\begin{aligned}
V, K, Q, lr &= \text{Linear}(input) \\
lr &= \text{softplus}(lr + lr_{init}) \\
W_t &= \exp(-lr_t) W_{t-1} + lr_t K_t^T V_t \\
O_t &= Q_t W_t
\end{aligned} \tag{24}$$

We will denote the above equations as $O = \text{Mamba}(input)$.

We use the multi-head design as in Transformer’s multi-head attention. Multiple independent Mamba-2 layer are run in parallel and their outputs are concatenated. Suppose the number of heads is nh , the formula is:

$$\begin{aligned}
O^k &= \text{Mamba}^k(input) \\
O &= [O^1, \dots, O^{nh}]
\end{aligned} \tag{25}$$

where each Mamba^k is a Mamba with its own parameters. In Mamba-2’s terminology, this design is equivalent to setting the number of ‘groups’ to be the same as the number of heads.

For the novel view synthesis task, we take a bidirectional Mamba over the input image tokens. In detail, we take two independent multi-head Mamba with one reading from left to right and the other reading from right to left. The bidirectional model builds a better connection among input tokens and also doubles the state size. We use a similar ‘apply’ operation as in LaCT that only updates the state for input tokens, and the state is static for the target tokens. We also tested with ‘update’ for the target image tokens, but it empirically leads to worse results. We use a head dimension of 192 and 8 heads. The overall state size, $8 (\text{num heads}) \times 192^2 (\text{head dim}) \times 2 (\text{bidir})$, matches LaCT with a standard large-chunk large-weight linear attention of dimension 768 ($768 \text{ input dim} \times 768 \text{ intermediate dim}$ in Fig. 8). We take $lr_{init} = -4.6$, which corresponds to a 0.01 initialized learning rate (i.e., $\text{softplus}(-4.6) = 0.01$).

For the autoregressive video diffusion task, we apply a unidirectional Mamba over the flattened video tokens. As mentioned in Sec 5.3, we follow [62] to inherit the Wan’s self-attention projected k, q , and v as B, C , and x in the Mamba layer, respectively. Unlike in the NVS task, each token will ‘update’ the state, which will be ‘applied’ to the current output and future tokens. Our Mamba uses 12 heads, each of dimension 128, matching the original multi-head self-attention in Wan. The overall state size is $12 (\text{num heads}) \times 128^2 (\text{head dim}) \times 19$. We take $lr_{init} = -4.6$, which corresponds to a 0.01 initialized learning rate.

G Details of LaCT Context Parallelism Implementation

Context Parallelism(CP) partitions the input sequence along its sequence length dimension and distributed the shards across multiple devices for parallel computing. The feed-forward layer and window attentions are local operations thus support CP naively. Our large-chunk Test-Time Training (TTT) approach facilitates CP by sharding tokens within each large chunk.

Within our large-chunk TTT mechanism, the per-token *apply* operation naively supports CP due to its independent nature. The *update* allows CP by shading tokens within a chunk over multiple devices. This CP can be easily implemented by adding a few lines of distributed all-reduce-sum after computing the local fast weight gradients on each device, logically the same as the Distributed Data Parallelism. Note that the distributed all-reduce-sum is a differentiable operator and its backward is all-reduce-sum over the gradient, thus the network can be trained end-to-end. Algorithm 3 presents the

792 pseudocode detailing this intra-chunk context parallelism specifically for the large-chunk TTT *update*
793 operation. We employed this parallelism in our view synthesis experiments, handling maximum
794 chunk sizes exceeding half a million tokens and maximum sequence lengths over one million tokens
795 during training.

Algorithm 3 Large Chunk Test-Time Training Layer with Context Parallel Sharded inside chunk Pseudocode

```
def update(fast_weight, k, v, lr, cp_group, use_muon=True):
    """
    Fast-weight update for a SwiGLU MLP using a context-parallel chunk.

    Args:
        fast_weight : tuple(w1, w2, w3) with shapes: w1, w3: [b, d, dh]; w2: [b, dh, d]
        k, v : key / value tensor of shape [b, l, d]
        lr : per-token learning rates of shape [b, l, 3] -> (lr1, lr2, lr3)
        cp_group : process group metadata for context parallelism
        use_muon : weather to apply Muon to orthogonalize the update

    Note:
        The input tensors k, v, lr are assumed to be already partitioned (sharded) along the sequence
        dimension over multiple devices. l represents the local sharded sequence length on each device.
        The total effective chunk size processed is l * cp_group.size.
    """

    # Forward with k:
    gate_before_act = matmul(k, w1) # [b, l, dh] = [b, l, d] x [b, d, dh]
    hidden_before_gate = matmul(k, w3) # [b, l, dh] = [b, l, d] x [b, d, dh]
    hidden = silu(gate_before_act) * hidden_before_gate

    # Backward:
    dhidden = matmul(v, w2.transpose(-1, -2)) # [b, l, dh] = [b, l, d] x [b, d, dh]
    dhidden_before_gate = dhidden * silu(gate_before_act)
    dgate = dhidden * hidden_before_gate
    dgate_before_act = silu_backprop(dgate, gate_before_act)

    # Compute gradients:
    w2.grad = -matmul(hidden.transpose(-1, -2), v * lr2) # [b, dh, d] = [b, dh, l] x [b, l, d]
    # [b, d, dh] = [b, d, l] x [b, l, dh]
    w1.grad = -matmul((k * lr1).transpose(-1, -2), dgate_before_act)
    w3.grad = -matmul((k * lr3).transpose(-1, -2), dhidden_before_gate)

    # [Standard forward pass and local backward gradient computations are performed above,
    # resulting in local w.grad for each device.]

    #####
    # BEGIN CONTEXT PARALLELISM SPECIFIC MODIFICATION: Global Gradient Aggregation
    # The following AllReduce operation is the key step introduced for context
    # parallelism. Operations before this point compute local gradients; operations
    # after this point use the globally aggregated gradients.
    #####
    for w in fast_weight:
        w.grad = distributed_all_reduce(w.grad, cp_group, op="SUM")
    #####
    # END CONTEXT PARALLELISM SPECIFIC MODIFICATION.
    # Subsequent operations (Muon, weight updates) now use the globally summed w.grad.
    # The formulas for these subsequent operations remain the same as in a
    # non-parallel version, but they act upon these aggregated gradients.
    #####

    # Weight update
    if use_muon:
        for w in fast_weight:
            w.grad = zeropower_via_newtonschulz5(w.grad)
    for w in fast_weight:
        w = (w - w.grad) / (w - w.grad).norm(dim=1) * w.norm(dim=1)

    return fast_weight
```

796 H Details of LaCT Tensor Parallelism Implementation

797 Beyond Context Parallelism, our large-chunk Test-Time Training (TTT) mechanism also supports
798 Tensor Parallelism (TP). This is primarily achieved by sharding the TTT heads across multiple
799 devices, a strategy similar to that employed in methods like DeepSpeed Ulysses [65].

800 Specifically, while static feed-forward layers in the model might process inputs sharded along the
801 sequence dimension (Context Parallelism), for the TTT operations within our LaCT layer, the data

undergoes a gather-then-scatter transformation. Input tensors (Q, K, V, and learning rates for TTT) that are initially sharded by sequence length are first gathered along the sequence dimension to reconstruct the full sequence context on each device within the tensor-parallel group. Then, these full-sequence tensors are scattered along the head dimension. As a result, each device processes the complete sequence but operates on only its assigned subset of TTT heads during the TTT *update* and *apply* iterations. The reverse transformation (gather heads, scatter sequence) is applied to the output of TTT operation. Algorithm 4 provides pseudocode detailing this tensor parallelism implementation, omitting minor details like padding. While this gather-then-scatter method effectively enables head-sharded tensor parallelism, more sophisticated communication strategies [66, 65] could potentially be employed to further optimize communication overhead.

We utilized this tensor parallelism strategy in our autoregressive video generation experiments, sharding, for example, four TTT heads across four local GPUs. This enabled us to train 14-billion-parameter diffusion models with sequence lengths exceeding 100K tokens.

Algorithm 4 Large Chunk Test-Time Training Layer with Tensor Parallelism by sharding heads Pseudocode

```
def gather_scatter(x, gather_dim, scatter_dim, process_group=cp_group):
    """
    Gathers tensor x along gather_dim across process_group,
    then scatters the result along scatter_dim to each device locally.
    Example: Transform [B, N_full, L_local, D] with gather_dim=2, scatter_dim=1
             to [B, N_local, L_full, D] on each device.
    """
    x = all_gather(x, gather_dim, process_group)

    # Calculate slicing indices for the scatter operation
    local_rank, group_size = process_group.rank, process_group.size
    scatter_stride = x.size(scatter_dim) // group_size
    start_idx = local_rank * scatter_stride
    end_idx = (local_rank + 1) * scatter_stride

    # Slice the tensor to get the local shard for the current device
    x = slice_tensor(x, scatter_dim, start_idx, end_idx)
    return x

##### MultiHead LaCT Layer with Tensor Parallelism (sharding TTT heads) #####
# Input:
# x: input sequence sharded by sequence length (CP). Shape [b, l, d], b is the batch dim, l is local
#     sequence length, d is model dimension.
# fast_weight: tuple of sharded initial fast weights, shared among heads. (w1, w2, w3); w1, w3 of
#     shape [nh, d, dh], w2 of shape [nh, dh, d]. nh: number of local heads.

qkv = silu(LinearQKV(x)) # [b, l, d * 3]
qkv = rearrange(qkv, 'b l (nh hd) -> b nh l hd', nh=num_heads).split(3, dim=-1)
q, k = q / q.norm(-1), k / k.norm(-1)
lr = softplus(LinearLR(x) + const_lr_bias) # [b, l, 3 * num_heads]
lr = rearrange(lr, 'b l (nh 3) -> b nh l 3', nh=num_heads)

#####
# BEGIN TENSOR PARALLELISM SPECIFIC TRANSFORMATION
# Gather along Sequence Length (dim 2), then Scatter along Head Dimension (dim 1).
# Transforms [b, nh_full, l_local, X] -> [b, nh_local, l_full, X]
# Each device now has the full sequence for a subset of heads.
q, k, v, lr = map(lambda x: gather_scatter(x, gather_dim=2, scatter_dim=1), (q, k, v, lr))
# END TENSOR PARALLELISM SPECIFIC TRANSFORMATION
#####

# [b, nh_local, l_full, X]
o_local_heads = ... # Placeholder for actual TTT computation on sharded heads
o_local_heads = RMSNorm(o_local_heads) # per-head norm

#####
# BEGIN TENSOR PARALLELISM SPECIFIC REVERSE TRANSFORMATION
# Gather along Head Dimension (dim 1), then Scatter along Sequence Dimension (dim 2).
# Transforms [b, nh_local, l_full, X] -> [b, nh_full, l_local, X]
# This reconstructs the full head dimension but shards sequence back.
o = gather_scatter(o_local_heads, gather_dim=1, scatter_dim=2)
# END TENSOR PARALLELISM SPECIFIC REVERSE TRANSFORMATION
#####
o = rearrange(o, 'b nh l hd -> b l (nh hd)', nh=num_heads)
o = LinearOutput(o)

return o
```

Table 8: RULER benchmark results for Single Needle in a Haystack (S-NIAH) tasks. * Our method with two heads (default is four).

Model	S-NIAH-1				S-NIAH-2				S-NIAH-3				Average			
	4K	8K	16K	32K	4K	8K	16K	32K	4K	8K	16K	32K	4K	8K	16K	32K
<i>760M parameters</i>																
Transformer	99.2	96.6	85.2	68.0	100	100	85.8	82.2	81.0	73.8	74.8	36.8	93.4	90.1	81.9	62.3
DeltaNet + SWA	84.0	85.2	87.8	86.8	62.8	29.4	14.2	7.8	53.8	21.8	11.2	5.8	66.9	45.5	37.7	33.5
GLA + SWA	51.8	26.2	14.4	8.6	55.8	26.4	15.8	7.8	58.0	23.8	16.2	5.0	55.2	25.5	15.5	7.1
Ours	94.8	53.2	26.0	14.8	74.0	28.0	14.2	7.8	42.8	26.6	14.4	6.8	70.5	35.9	18.2	9.8
Ours Momentum	95.6	84.8	83.4	84.8	91.4	73.4	22.8	7.8	82.6	34.8	16.6	6.6	89.9	64.3	40.9	33.1
Ours Momentum*	59.0	30.0	12.4	8.4	93.4	50.0	18.2	7.8	60.2	25.6	14.2	6.8	70.9	35.2	14.9	7.7
Ours Muon	98.0	95.0	92.2	92.4	86.6	60.2	17.0	7.8	49.2	26.2	10.9	5.2	77.9	60.5	40.0	35.1
<i>3B parameters</i>																
Transformer	100	100	100	100	100	99.8	100	98.6	98.6	95.8	90.8	75.0	99.5	98.5	96.9	91.2
GLA SWA	100	52.8	26.0	13.2	100	51.8	29.6	14.4	98.0	54.4	27.6	12.4	99.3	53.0	27.7	13.3
DeltaNet SWA	100	89.6	76.2	54.8	100	76.4	42.2	17.0	90.6	57.6	27.4	13.4	96.9	74.5	48.6	28.4
Ours Momentum	99.4	97.0	98.6	93.4	100	75.6	39.6	15.0	91.8	63.0	27.8	13.4	97.1	78.5	55.3	40.6
Ours Muon	98.8	99.2	98.6	93.4	100	99.0	83.2	30.8	95.4	90.8	55.6	19.8	98.1	96.3	79.1	48.0

Table 9: Performance on Multi-Key (MK-NIAH), Multi-Query (MQ-NIAH), and Multi-Value (MV-NIAH) Needle in a Haystack tasks from the RULER benchmark. * Our method with two heads (default is four).

Model	MK-NIAH				MQ-NIAH				MV-NIAH				Average			
	4K	8K	16K	32K	4K	8K	16K	32K	4K	8K	16K	32K	4K	8K	16K	32K
<i>760M parameters</i>																
Transformer	63.8	72	71.4	54	33.4	28.9	24	23.1	27.95	24	20.5	27.35	41.7	41.6	38.6	34.8
DeltaNet+SWA	41.2	30	14.6	8.2	33	22.45	7.5	4.3	32.4	22.8	9.15	6.6	35.5	25.1	10.4	6.4
GLA + SWA	45.4	28.4	15.8	6.6	26.1	17.75	10.2	5.85	25.4	16.85	10.1	6.6	32.3	21.0	12.0	6.3
Ours	60.8	34.6	16.8	7	35	23.65	14.1	7.45	20.7	22.05	12.7	6.85	38.8	26.8	14.5	7.1
Ours Momentum	62	41	21.2	10.4	35.3	24.95	17.7	8.6	27.9	23.15	16.65	8.2	41.7	29.7	18.5	9.1
Ours Momentum*	59.8	37.8	19.2	8.8	36.65	20.45	12.5	7.4	24.45	16.95	11.6	6.8	40.3	25.1	14.4	7.7
Ours Muon	62.8	46.6	22	8.6	37.7	26.55	15.7	7.1	28.35	23.15	13.6	6.85	42.9	32.1	17.1	7.5
<i>3B parameters</i>																
Transformer	95	90.4	81.6	65.2	86.45	81.55	71.70	40.85	61.8	42.8	30.75	22.9	81.1	71.6	61.4	43.0
GLA 3B	78	45.8	28.6	14.4	50.05	28.05	19	10.7	29.4	21.4	16.75	9.9	52.5	31.8	21.4	11.7
DeltaNet SWA	75.8	57.4	34.2	17.8	66.25	33.05	21.45	13.45	43.7	23.2	18.85	13.2	61.9	37.9	24.8	14.8
Ours Momentum	96.2	59.6	35	17.2	87.05	40.25	25.6	13.2	88.08	30.65	21.9	12.3	90.4	43.5	27.5	14.2
Ours Muon	75.2	69.2	46.2	25.2	44.75	39.1	24.9	19	26.55	29.1	25.05	19.3	48.8	45.8	32.0	21.2

References

- [1] 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.
- [2] Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.
- [3] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, pages 9355–9366. PMLR, 2021.
- [4] Ke Alexander Wang, Jiaxin Shi, and Emily B. Fox. Test-time regression: a unifying framework for designing sequence models with associative memory, 2025.
- [5] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024.
- [6] Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. It’s all connected: A journey through test-time memorization, attentional bias, retention, and online optimization, 2025.
- [7] Mahdi Karami and Vahab Mirrokni. Lattice: Learning to efficiently compress the memory. *arXiv preprint arXiv:2504.05646*, 2025.
- [8] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024.
- [9] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023.

- [10] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *arXiv preprint arXiv:2406.06484*, 2024.
- [11] Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.
- [12] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.
- [13] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *International Conference on Machine Learning*, pages 56501–56523. PMLR, 2024.
- [14] Zhen Qin, Weigao Sun, Dong Li, Xuyang Shen, Weixuan Sun, and Yiran Zhong. Various lengths, constant speed: Efficient language modeling with lightning attention. In *Forty-first International Conference on Machine Learning*, 2024.
- [15] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.
- [16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. arxiv e-prints. *arXiv preprint arXiv:1512.03385*, 10:9, 2015.
- [17] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.
- [18] Noam Shazeer. Glu variants improve transformer, 2020.
- [19] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016.
- [20] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.
- [21] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 433–440. 2023.
- [22] Marc Levoy and Pat Hanrahan. Light field rendering. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 441–452. 2023.
- [23] Haian Jin, Hanwen Jiang, Hao Tan, Kai Zhang, Sai Bi, Tianyuan Zhang, Fujun Luan, Noah Snavely, and Zexiang Xu. Lvsm: A large view synthesis model with minimal 3d inductive bias. *arXiv preprint arXiv:2410.17242*, 2024.
- [24] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- [25] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13142–13153, 2023.
- [26] Kai Zhang, Sai Bi, Hao Tan, Yuanbo Xiangli, Nanxuan Zhao, Kalyan Sunkavalli, and Zexiang Xu. Gs-irm: Large reconstruction model for 3d gaussian splatting. In *European Conference on Computer Vision*, pages 1–19. Springer, 2024.

- [27] Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Hickman, Krista Reymann, Thomas B McHugh, and Vincent Vanhoucke. Google scanned objects: A high-quality dataset of 3d scanned household items. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2553–2560. IEEE, 2022.
- [28] Lu Ling, Yichen Sheng, Zhi Tu, Wentian Zhao, Cheng Xin, Kun Wan, Lantao Yu, Qianyu Guo, Zixun Yu, Yawen Lu, et al. DI3dv-10k: A large-scale scene dataset for deep learning-based 3d vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 22160–22169, 2024.
- [29] 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.
- [30] Chen Ziwen, Hao Tan, Kai Zhang, Sai Bi, Fujun Luan, Yicong Hong, Li Fuxin, and Zexiang Xu. Long-irm: Long-sequence large reconstruction model for wide-coverage gaussian splats. *arXiv preprint arXiv:2410.12781*, 2024.
- [31] Together AI. Long data collections database, 2024.
- [32] Zhixuan Lin, Evgenii Nikishin, Xu He, and Aaron Courville. Forgetting transformer: Softmax attention with a forget gate. In *The Thirteenth International Conference on Learning Representations*, 2025.
- [33] Cheng-Ping Hsieh, Simeng Sun, Samuel Krirman, Shantanu Acharya, Dima Rekish, Fei Jia, and Boris Ginsburg. Ruler: What’s the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024.
- [34] Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. Transformer quality in linear time. In *International conference on machine learning*, pages 9099–9117. PMLR, 2022.
- [35] Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, Madian Khabsa, Han Fang, Yashar Mehdad, Sharan Narang, Kshitiz Malik, Angela Fan, Shruti Bhosale, Sergey Edunov, Mike Lewis, Sinong Wang, and Hao Ma. Effective long-context scaling of foundation models, 2023.
- [36] Xin Men, Mingyu Xu, Bingning Wang, Qingyu Zhang, Hongyu Lin, Xianpei Han, and Weipeng Chen. Base of rope bounds context length. *arXiv preprint arXiv:2405.14591*, 2024.
- [37] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023.
- [38] Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Fei Wu, Haiming Zhao, Jianxiao Yang, Jianyuan Zeng, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.
- [39] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis, 2024. URL <https://arxiv.org/abs/2403.03206>, 2, 2024.
- [40] Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. It’s all connected: A journey through test-time memorization, attentional bias, retention, and online optimization. *arXiv preprint arXiv:2504.13173*, 2025.
- [41] Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc V. Le. Transformer quality in linear time, 2022.
- [42] Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. Mega: Moving average equipped gated attention. In *The Eleventh International Conference on Learning Representations*, 2023.
- [43] Xuezhe Ma, Xiaomeng Yang, Wenhan Xiong, Beidi Chen, LILI YU, Hao Zhang, Jonathan May, Luke Zettlemoyer, Omer Levy, and Chunting Zhou. Megalodon: Efficient LLM pretraining and inference with unlimited context length. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.
- [44] Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention, 2024.
- [45] DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. Block-recurrent transformers. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

- [46] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.
- [47] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3d object. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9298–9309, 2023.
- [48] Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, Clarence Ng, Ricky Wang, and Aditya Ramesh. Video generation models as world simulators. 2024.
- [49] Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. Cogvideox: Text-to-video diffusion models with an expert transformer. *arXiv preprint arXiv:2408.06072*, 2024.
- [50] A Polyak, A Zohar, A Brown, A Tjandra, A Sinha, A Lee, A Vyas, B Shi, CY Ma, CY Chuang, et al. Movie gen: A cast of media foundation models, 2025. URL <https://arxiv.org/abs/2410.13720>, page 51, 2024.
- [51] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.
- [52] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.
- [53] Yang Jin, Zhicheng Sun, Ningyuan Li, Kun Xu, Hao Jiang, Nan Zhuang, Quzhe Huang, Yang Song, Yadong Mu, and Zhouchen Lin. Pyramidal flow matching for efficient video generative modeling. *arXiv preprint arXiv:2410.05954*, 2024.
- [54] David Ruhe, Jonathan Heek, Tim Salimans, and Emiel Hoogetboom. Rolling diffusion models. In *International Conference on Machine Learning*, pages 42818–42835. PMLR, 2024.
- [55] Tianwei Yin, Qiang Zhang, Richard Zhang, William T Freeman, Fredo Durand, Eli Shechtman, and Xun Huang. From slow bidirectional to fast causal video generators. *arXiv preprint arXiv:2412.07772*, 2024.
- [56] Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37:24081–24125, 2024.
- [57] Sand-AI. Magi-1: Autoregressive video generation at scale, 2025.
- [58] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.
- [59] Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4246–4253, 2020.
- [60] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.
- [61] Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, et al. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 24185–24198, 2024.
- [62] Junxiong Wang, Daniele Paliotta, Avner May, Alexander Rush, and Tri Dao. The mamba in the llama: Distilling and accelerating hybrid models. *Advances in Neural Information Processing Systems*, 37:62432–62457, 2024.
- [63] Horace He, Driss Guessous, Yanbo Liang, and Joy Dong. Flexattention: The flexibility of pytorch with the performance of flashattention. *PyTorch Blog*, 8, 2024.
- [64] Wenliang Zhao, Lujia Bai, Yongming Rao, Jie Zhou, and Jiwen Lu. Unipc: A unified predictor-corrector framework for fast sampling of diffusion models. *Advances in Neural Information Processing Systems*, 36:49842–49869, 2023.

- 991 [65] Sam Ade Jacobs, Masahiro Tanaka, Chengming Zhang, Minjia Zhang, Shuaiwen Leon Song, Samyam
992 Rajbhandari, and Yuxiong He. Deepspeed ulysses: System optimizations for enabling training of extreme
993 long sequence transformer models. *arXiv preprint arXiv:2309.14509*, 2023.
- 994 [66] Jiarui Fang and Shangchun Zhao. Usp: A unified sequence parallelism approach for long context generative
995 ai. *arXiv preprint arXiv:2405.07719*, 2024.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [NA].
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes]" is generally preferable to "[No]", it is perfectly acceptable to answer "[No]" provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No]" or "[NA]" is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS paper checklist",**
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers.**

1. Claims

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

Answer: [Yes]

Justification: We discuss how to improve GPU utilization and achieve better performance with large-chunk TTT layer.

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 have a limitation section.

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: [Yes]

Justification: Yes, we do have. For most of the results in our papers, they are empirical. The formulas are mostly used to present the implementation concisely.

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: Yes, we provided. We will also release the code and model checkpoints for some of the tasks.

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 present pseudo code in supp and will release the code once acceptance.

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, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: Yes, please see the experimental section.

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: No, most of the experiments are too expensive to repeat.

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: We provided such information in the Appendix.

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: Yes, we do.

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: This is a general method paper and it does not have specific concerns regarding societal impacts comparing with other papers.

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: The NVS model is a non-generative model and only repeat the input information. The LM model is with small size.

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: Yes, we give credit to the data, model, and code that we 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: No new assets released.

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)?

1305 Answer: [NA]
 1306 Justification: NA
 1307 Guidelines:

- 1308 • The answer NA means that the paper does not involve crowdsourcing nor research with
- 1309 human subjects.
- 1310 • Including this information in the supplemental material is fine, but if the main contribu-
- 1311 tion of the paper involves human subjects, then as much detail as possible should be
- 1312 included in the main paper.
- 1313 • According to the NeurIPS Code of Ethics, workers involved in data collection, curation,
- 1314 or other labor should be paid at least the minimum wage in the country of the data
- 1315 collector.

1316 **15. Institutional Review Board (IRB) Approvals or Equivalent for Research with Human**
 1317 **Subjects**

1318 Question: Does the paper describe potential risks incurred by study participants, whether
 1319 such risks were disclosed to the subjects, and whether Institutional Review Board (IRB)
 1320 approvals (or an equivalent approval/review based on the requirements of your country or
 1321 institution) were obtained?

1322 Answer: [NA]
 1323 Justification: NA
 1324 Guidelines:

- 1325 • The answer NA means that the paper does not involve crowdsourcing nor research with
- 1326 human subjects.
- 1327 • Depending on the country in which research is conducted, IRB approval (or equivalent)
- 1328 may be required for any human subjects research. If you obtained IRB approval, you
- 1329 should clearly state this in the paper.
- 1330 • We recognize that the procedures for this may vary significantly between institutions
- 1331 and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the
- 1332 guidelines for their institution.
- 1333 • For initial submissions, do not include any information that would break anonymity (if
- 1334 applicable), such as the institution conducting the review.