Scalable Adaptive Computation for Iterative Generation

Allan Jabri [1 2 †] **David J. Fleet** [1] **Ting Chen** [1]

Abstract

Natural data is redundant yet predominant architectures tile computation uniformly across their input and output space. We propose the *Recurrent Interface Network* (RIN), an attention-based architecture that decouples its core computation from the dimensionality of the data, enabling adaptive computation for more scalable generation of high-dimensional data. RINs focus the bulk of computation (i.e. global self-attention) on a set of *latent* tokens, using cross-attention to read and write (i.e. *route*) information between latent and data tokens. Stacking RIN blocks allows bottom-up (data to latent) and top-down (latent to data) feedback, leading to deeper and more expressive routing. While this routing introduces challenges, this is less problematic in recurrent computation settings where the task (and routing problem) changes gradually, such as iterative generation with diffusion models. We show how to leverage recurrence by conditioning the latent tokens at each forward pass of the reverse diffusion process with those from prior computation, i.e. latent self-conditioning. RINs yield state-of-the-art pixel diffusion models for image and video generation, scaling to 1024×1024 images without cascades or guidance, while being domain-agnostic and up to $10\times$ more efficient than 2D and 3D U-Nets.

1. Introduction

The design of effective neural network architectures has been crucial to the success of deep learning (Krizhevsky et al., 2012; He et al., 2016; Vaswani et al., 2017). Influenced by modern accelerator hardware, predominant architectures, such as convolutional neural networks (Fukushima, 1988; LeCun et al., 1989; He et al., 2016) and Transformers (Vaswani et al., 2017), allocate computation in a fixed, uniform manner over the input data (e.g., over image pixels, image patches, or token sequences). Information in natural data is often distributed unevenly, or exhibits redundancy, so it is important to ask how to allocate computation in an adaptive manner to improve scalability. While prior work has explored more dynamic and input-decoupled computation, e.g., networks with auxiliary memory (Dai et al., 2019; Rae et al., 2019) and global units (Zaheer et al., 2020; Burtsev et al., 2020; Jaegle et al., 2021b;a), general architectures that leverage adaptive computation to effectively scale to tasks with large input and output spaces remain elusive.

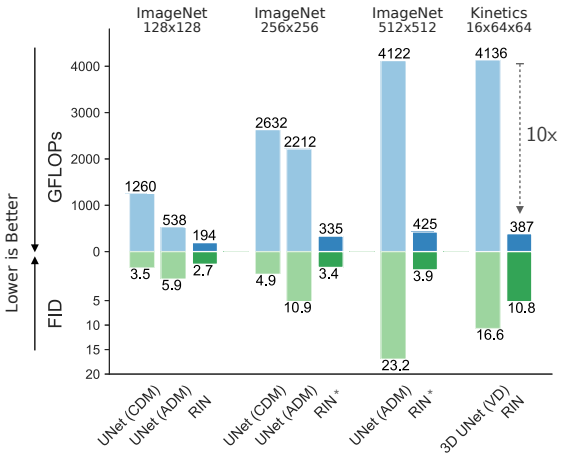


Figure 1. RINs outperform U-Nets widely used in state-of-the-art image and video diffusion models, while being more efficient and domain-agnostic. Our models are simple pixel-level denoising diffusion models without cascades as in (CDM (Ho et al., 2022a)) or guidance (as in ADM (Dhariwal & Nichol, 2022) and VD (Ho et al., 2022b)). $*$: uses input scaling (Chen, 2023).

In this paper, we consider this issue as it manifests in high-dimensional generative modeling tasks, such as image and video generation. When generating an image with a simple background, an adaptive architecture should ideally be able to allocate computation to regions with complex objects and textures, rather than regions with little or no structure (e.g., the sky). When generating video, one should exploit temporal redundancy, allocating less computation to static regions. While such non-uniform computation becomes more crucial in higher-dimensional data, achieving it efficiently is challenging on modern hardware, given the preference for fixed computation graphs with dense matrix multiplication.

[†]Work done as a student researcher at Google. [1]Google Brain, Toronto. [2]Department of EECS, UC Berkeley. Correspondence to: Ting Chen <iamtingchen@google.com>, Allan Jabri <ajabri@berkeley.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

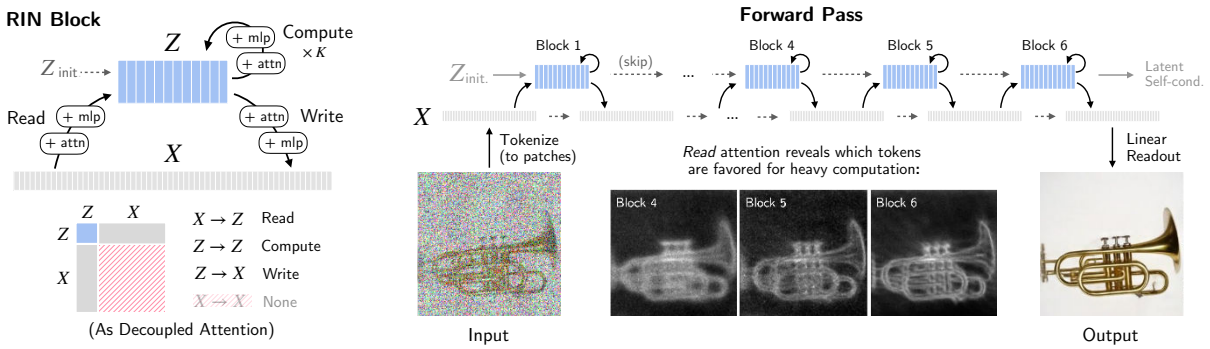


Figure 2. Overview of Recurrent Interface Networks. The input is tokenized to form the interface X. A stack of blocks route information between X and latents Z, avoiding quadratic pairwise interactions between tokens in X (bottom left). Note that $\dim(Z) > \dim(X)$, and most computation is applied to Z, which allows for scaling to large X. The network's read attention maps reveals how tokens are favoured for latent computation (right), when trained for a task like diffusion generative modeling.

To address this challenge, we propose an architecture, dubbed Recurrent Interface Networks (RINs). In RINs (Fig. 2), hidden units are partitioned into the *interface* X and *latents* Z. Interface units are locally connected to the input and grow linearly with input size. In contrast, latents are decoupled from the input space, forming a more compact representation on which the bulk of computation operates. The forward pass proceeds as a stack of blocks that read, compute, and write: in each block, information is routed from interface tokens (with cross-attention) into the latents for high-capacity global processing (with self-attention), and updates are written back to interface tokens (with cross-attention). Alternating computation between latents and interface allows for processing at local and global levels, accumulating context for better routing. As such, RINs allocate computation more dynamically than uniform models, scaling better when information is unevenly distributed across the input and output, as is common in natural data.

This decoupling introduces additional challenges, which can overshadow benefits if the latents are initialized without context in each forward pass, leading to shallow and less expressive routing. We show this cost can be mitigated in scenarios involving recurrent computation, where the task and inputs change gradually and persistent context can be leveraged across iterations to in effect form a deeper network. In particular, we consider iterative generation of images and video with denoising diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2020; 2021). To leverage recurrence, we propose latent self-conditioning as a "warm-start" mechanism for latents to amortize the cost of routing. Instead of reinitializing latents at each forward pass, we use latents from previous iterations as additional context, similar to a recurrent network but without requiring backpropagation through time.

Our experiments with diffusion models show that RINs outperform U-Net architectures for image and video generation, as shown in Figure 1. For class-conditional ImageNet models, from 64×64 up to 1024×1024, RINs outperform leading diffusion models that use cascades (Ho et al., 2022a) or guidance (Dhariwal & Nichol, 2022; Ho & Salimans, 2021), while consuming up to 10× fewer FLOPs per inference step. For video prediction, RINs surpass leading approaches (Ho et al., 2022b) on the Kinetics600 benchmark while reducing the FLOPs of each step by 10×.

Our contributions are summarized as follows:

- We propose RINs, a domain-agnostic architecture capable of adaptive computation for scalable generation of high dimensional data.
- We identify recurrent computation settings in which RINs thrive and advocate latent self-conditioning to amortize the challenge of routing.
- Despite reduced inductive bias, this leads to performance and efficiency gains over U-Net diffusion models for image and video generation.

2. Method

In RINs, the interface is locally connected to the input space and initialized via a form of tokenization (e.g., patch embeddings), while the latents are decoupled from data and initialized as learnable embeddings. The basic RIN block allocates computation by *routing* information between the interface and the latents. By stacking multiple blocks, we can update the interface and latents repeatedly, such that bottom-up and top-down context can inform routing in the next block (see Fig. 3). A linear readout function predicts the network's output from the final interface representation.

Since the interface is tied to data, it grows linearly with input size and may be large (e.g., thousands of vectors), while the number of latent units can be much smaller (e.g., hundreds of vectors). The computation operating directly on the interface (e.g. tokenization, read, write) is uniform across the input space, but is designed to be relatively light-weight, for

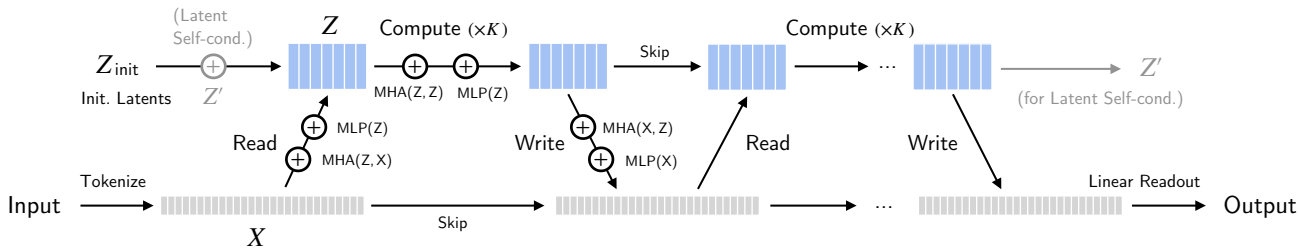


Figure 3. **The computation graph of RIN Blocks.** RINs stack blocks that read, compute, and write. *Read* operations load information into latents with cross-attention. *Compute* operations exchange information across latent tokens with self-attention and across channels with token-wise MLPs. *Write* operations update the interface with information from the latents with cross-attention, and mix information across channels with token-wise MLPs. Latent self-conditioning (gray lines) allows for propagation of latent context between iterations.

minimal uniform computation. The high-capacity processing is reserved for the latents, formed by reading information from the interface selectively, such that most computation can be adapted to the structure and content of the input.

Compared to convolutional nets such as U-Nets (Ronneberger et al., 2015; Ho et al., 2020), RINs do not rely on fixed downsampling or upsampling for global computation. Compared to Transformers (Vaswani et al., 2017), RINs operate on sets of tokens with positional encoding for similar flexibility across input domains, but avoid pairwise attention across tokens to reduce compute and memory requirements per token. Compared to other decoupled architectures such as PerceiverIO (Jaegle et al., 2021b;a), alternating computation between interface and latents enables more expressive routing without a prohibitively large set of latents.

While RINs are versatile, their advantages are more pronounced in recurrent settings, where inputs may change gradually over time such that it is possible to propagate persistent context to further prime the routing of information. Therefore, here we focus on the application of RINs to iterative generation with diffusion models.

2.1. Background: Iterative Generation with Diffusion

We first provide a brief overview of diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2020; 2021; Kingma et al., 2021; Chen et al., 2022c). Diffusion models learn a series of state transitions to map noise $\boldsymbol{\epsilon}$ from a known prior distribution to $\boldsymbol{x}_0$ from the data distribution. To learn this (reverse) transition from noise to data, a forward transition from $\boldsymbol{x}_0$ to $\boldsymbol{x}_t$ is first defined:

$$\boldsymbol{x}_t = \sqrt{\gamma(t)}\,\boldsymbol{x}_0 + \sqrt{1-\gamma(t)}\,\boldsymbol{\epsilon},$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, $t \sim \mathcal{U}(0,1)$, and $\gamma(t)$ is a monotonically decreasing function from 1 to 0. Instead of directly learning a neural net to model the transition from $\boldsymbol{x}_t$ to $\boldsymbol{x}_{t-\Delta}$, one can learn a neural net $f(\boldsymbol{x}_t, t)$ to predict $\boldsymbol{\epsilon}$ from $\boldsymbol{x}_t$, and then estimate $\boldsymbol{x}_{t-\Delta}$ from the estimated $\tilde{\boldsymbol{\epsilon}}$ and $\boldsymbol{x}_t$. The objective for $f(\boldsymbol{x}_t, t)$ is thus the ℓ_2 regression loss:

$$\mathbb{E}_{t\sim\mathcal{U}(0,1),\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{1})}\|f(\sqrt{\gamma(t)}\,\boldsymbol{x}_0 + \sqrt{1-\gamma(t)}\,\boldsymbol{\epsilon}, t) - \boldsymbol{\epsilon}\|^2.$$

To generate samples from a learned model, we follow a series of (reverse) state transition $\boldsymbol{x}_1 \to \boldsymbol{x}_{1-\Delta} \to \cdots \to \boldsymbol{x}_0$. This is done by iteratively applying the denoising function f on each state $\boldsymbol{x}_t$ to estimate $\boldsymbol{\epsilon}$, and hence $\boldsymbol{x}_{t-\Delta}$, using transition rules as in DDPM (Ho et al., 2020) or DDIM (Song et al., 2020). As we will see, the gradual refinement of $\boldsymbol{x}$ through repeated application of the denoising function is a natural fit for RINs. The network takes as input a noisy image $\boldsymbol{x}_t$, a time step t, and an optional conditioning variable e.g. a class label y, and then outputs the estimated noise $\tilde{\boldsymbol{\epsilon}}$.

2.2. Elements of Recurrent Interface Networks

We next describe the major components of RINs (Fig. 3).

Interface Initialization. The interface is initialized from an input $\boldsymbol{x}$, such as an image $\boldsymbol{x}_{\text{image}} \in \mathbb{R}^{h\times w\times 3}$, or video $\boldsymbol{x}_{\text{video}} \in \mathbb{R}^{h\times w\times l\times 3}$ by tokenizing $\boldsymbol{x}$ into a set of n vectors $X \in \mathbb{R}^{n\times d}$. For example, we use a linear patch embedding similar to (Dosovitskiy et al., 2020) to convert an image into a set of patch tokens; for video, we use 3-D patches. To indicate their location, patch embeddings are summed with (learnable) positional encodings. Beyond tokenization, the model is domain-agnostic, as X is simply a set of vectors.

Latent Initialization. The latents $Z \in \mathbb{R}^{m\times d'}$ are (for now) initialized as learned embeddings, independent of the input. Conditioning variables, such as class labels and time step t of diffusion models, are mapped to embeddings; in our experiments, we simply concatenate them to the set of latents, since they only account for two tokens.

Core RIN Block. The RIN blocks routes information between X and Z with key components of Transformers:

$$\begin{aligned}
\textbf{Read:}\quad & Z = Z + \text{MHA}(Z, X)\\
& Z = Z + \text{MLP}(Z)\\
\textbf{Compute:}\quad & Z = Z + \text{MHA}(Z, Z)\\
(\times K)\quad & Z = Z + \text{MLP}(Z)\\
\textbf{Write:}\quad & X = X + \text{MHA}(X, Z)\\
& X = X + \text{MLP}(X)
\end{aligned}$$

MLP denotes a multi-layer perceptron, and MHA(Q, K)

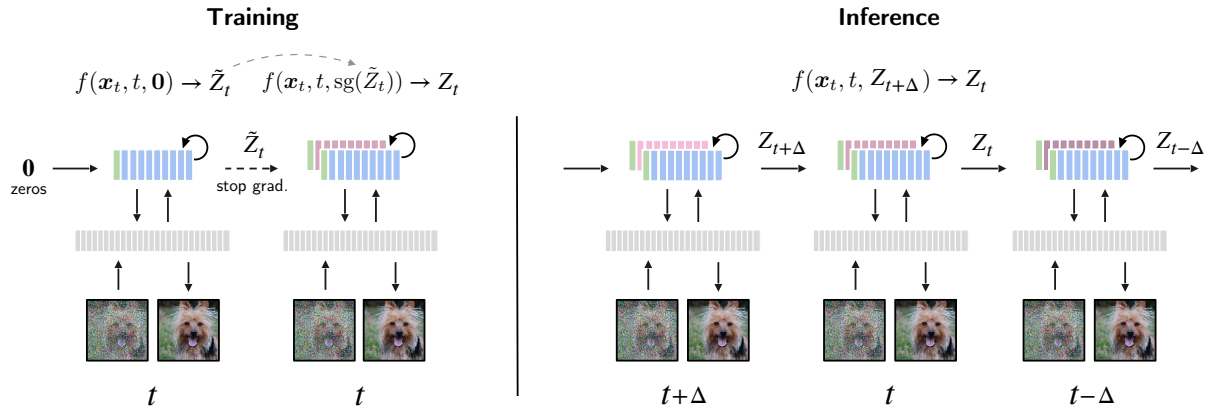


Figure 4. **Latent Self-Conditioning for Diffusion Models with RINs.** (Left) During training, latents for self-conditioning are first estimated with a forward pass of the denoising network (with zeros as previous latents); we then condition the denoising network with these estimated latents by treating them as latents of the previous iteration (without back-propagating through the estimated latents). (Right) During sampling, we start with zero latents, and use computed latents at each time-step to condition the next time-step.

denotes multi-head attention with queries Q, and keys and values K.[1] Note that we always apply LayerNorm (Ba et al., 2016) on the queries of MHA operations. The depth of each block K controls the ratio of computation occurring on the interface and latents. From the perspective of information exchange among hidden units, MHA propagates information across vectors (i.e. between latents, or between latents and interface), while the MLP (applied vector-wise, with shared weights) mixes information across their channels. Note that here computation on the interface is folded into the write operation, as MHA followed by an MLP.

RIN blocks can be stacked to allow latents to accumulate context and write incremental updates to the interface. To produce output predictions, we apply a readout layer (e.g. a linear projection) to the corresponding interface tokens to predict local outputs (such as patches of images or videos). The local outputs are then combined to form the desired output (e.g., patches are simply reshaped into an image). A detailed implementation is given in Appendix A (Alg 3).

2.3. Latent Self-Conditioning

RINs rely on routing information to dynamically allocate compute to parts of the input. Effective routing relies on latents that are specific to the input, and input-specific latents are built by reading interface information. This iterative process can incur additional cost that may overshadow the benefits of adaptive computation, especially if the network begins without context, i.e. from a "cold-start". Intuitively, as humans, we face a similar "cold-start" problem under changes in the environment, requiring gradual familiarization of new state to enhance our ability to infer relevant information. If contexts switch rapidly without sufficient time for "warm-up", we repeatedly face the cost of adapting to context. The "warm-up" cost in RINs can be similarly amortized in recurrent computation settings where inputs gradually change while global context persists. We posit that in such settings, there exists useful context in the latents accumulated in each forward pass.

Warm-starting Latents. With this in mind, we propose to "warm-start" the latents using latents computed at a previous step. The initial latents at current time step t are the sum of the learnable embeddings Z_{init} (independent of the input), and a transformation of previous latents computed in the previous iteration t':

$$Z_t = Z_{init} + \mathrm{LayerNorm}(Z_{t'} + \mathrm{MLP}(Z_{t'}))\ ,$$

where LayerNorm is initialized with zero scaling and bias, so that $Z_t = Z_{init}$ early in training.

In principle, this relies on the existence of latents from a previous time step, $Z_{t'}$, and requires unrolling iterations and learning with backpropagation through time, which can hamper scalability. A key advantage of diffusion models is that the chain of transitions decomposes into conditionally independent steps allowing for highly parallelizable training, an effect we would like to preserve. To this end, we draw inspiration from the self-conditioning technique of (Chen et al., 2022c), which conditions a denoising network at time t with its own unconditional prediction for time t.

Concretely, consider the conditional denoising network $f(\boldsymbol{x}_t, t, Z_{t'})$ that takes as input $\boldsymbol{x}_t$ and t, as well as context latents $Z_{t'}$. During training, with some probability, we use $f(\boldsymbol{x}_t, t, \mathbf{0})$ to directly compute the prediction $\tilde{\epsilon}_t$. Otherwise, we first apply $f(\boldsymbol{x}_t, t, \mathbf{0})$ to obtain latents $\tilde{Z}_t$ as an estimate of $Z_{t'}$, and compute the prediction with $f(\boldsymbol{x}_t, t, \mathrm{sg}(\tilde{Z}_t))$.

[1] See (Vaswani et al., 2017) for details about multi-head attention, which extends single-head attention defined as $\mathrm{Attention}(Z, X) = \mathrm{softmax}(ZW_Q W_K^\top X^\top)XW_V$. $\mathrm{MLP}(Z) = \sigma(ZW_1 + b_1)W_2 + b_2$ where σ is the GELU activation function (Hendrycks & Gimpel, 2016). W are learned linear projections.

Algorithm 1 Training RINs with Latent Self-Cond.

```
def train_loss(x, self_cond_rate, latent_shape):
  # Add noise.
  t = uniform(0, 1)
  eps = normal(mean=0, std=1)
  x_t = sqrt(gamma(t)) * x + sqrt(1-gamma(t)) * eps

  # Compute latent self-cond estimate.
  latents = zeros(latent_shape)
  if uniform(0, 1) < self_cond_rate:
    _, latents = rin((x_t, latents), t)
    latents = stop_gradient(latents)

  # Predict and compute loss.
  eps_pred, _ = rin((x_t, latents), t)
  loss = (eps_pred - eps)**2
  return loss.mean()
```

Algorithm 2 Sampling with Latent Self-Cond.

```
def generate(steps):
  x_t = normal(mean=0, std=1)
  latents = zeros(latent_shape)

  for step in range(steps):
    # Get time for current and next states.
    t = 1 - step / steps
    t_m1 = max(1 - (step + 1) / steps, 0)

    # Predict eps.
    eps_pred, latents = rin((x_t, latents), t)

    # Estimate x at t_m1.
    x_t = ddim_or_ddpm_step(x_t, eps_pred, t, t_m1)

  return x_t
```

Here, sg is the stop-gradient operation, used to avoid back-propagating through the latent estimates. At inference time, we directly use latents from previous time step t' to initialize the latents at current time step t, i.e., $f(\boldsymbol{x}_t, t, Z_{t'})$, in a recurrent fashion. This bootstrapping procedure marginally increases the training time ($< 25\%$ in practice, due to the stop-gradient), but has a negligible cost at inference time. In contrast to self-conditioning at the data level (Chen et al., 2022c), here we condition on the latent activations of the neural network, so we call it *latent self-conditioning*.

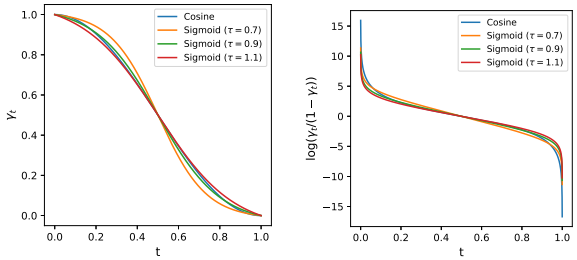
Figure 4 illustrates the training and sampling process with the proposed latent self-conditioning. Algorithms 1 and 2 give the proposed modifications to training and sampling of the standard diffusion process. Details of common functions used in the algorithms can be found in Appendix B.

3. Experiments

We demonstrate that RINs improve state-of-the-art performance on benchmarks for image generation and video prediction with pixel-space diffusion models. In all experiments, we do not use guidance. For each benchmark, we also compare the number of floating point operations (GFLOPs) across methods; as we will see, RINs are also more efficient. Samples and further visualizations are provided in Appendix D and the supplementary material.

3.1. Implementation Details

Noise Schedule. Similar to (Kingma et al., 2021; Chen et al., 2022c), we use a continuous-time noise schedule function $\gamma(t)$. By default we use a cosine schedule, as in previous work (Nichol & Dhariwal, 2021) but find it is sometimes unstable for higher resolution images. We therefore explore schedules based the sigmoid function with different temperature, which shifts weight away from the tails of the noise schedule. We use a default temperature of 0.9, and its effect is ablated in our experiments. Detailed implementation of noise schedules and ablations are provided in Appendix B. For larger images, we also report models trained using input scaling(Chen, 2023; Chen et al., 2022a).



(a) $\gamma(t)$ (b) $\log(\gamma(t)/(1-\gamma(t)))$

Figure 5. Compared to the cosine schedule, sigmoid (with appropriate τ) can place less weight on noise levels on the tails.

Tokenization and Readout. For image generation, we tokenize images by extracting non-overlapping patches followed by a linear projection. We use a patch size of 4 for 64×64 and 128×128 images, and 8 for larger images. To produce the output, we apply a linear projection to interface tokens and unfold each projected token to obtain predicted patches, which we reshape to form an image.

For video, we tokenize and produce predictions in the same manner as images; for 16×64×64 inputs, we use 2×4×4 patches, resulting in 2048 tokens. For conditional generation, during training, the context frames are provided as part of the input, without noise added. During sampling, the context frames are held fixed.

Table 1 compares model configuration across tasks; note in particular the ratio of $|Z|$ and $|X|$. See Appendix C for detailed model and training hyper-parameters, and Appendix A for detailed pseudo-code of the full model.

3.2. Experimental Setup

For image generation, we mainly use the ImageNet dataset (Russakovsky et al., 2015). For data augmentation, we only use center crops and random left-right flipping. We also use CIFAR-10 (Krizhevsky et al.) to show the model can be trained with small datasets. For evaluation, we follow common practice, using FID (Heusel et al., 2017) and Inception Score (Salimans et al., 2016) as metrics computed on 50K samples, generated with 1000 steps of DDPM.

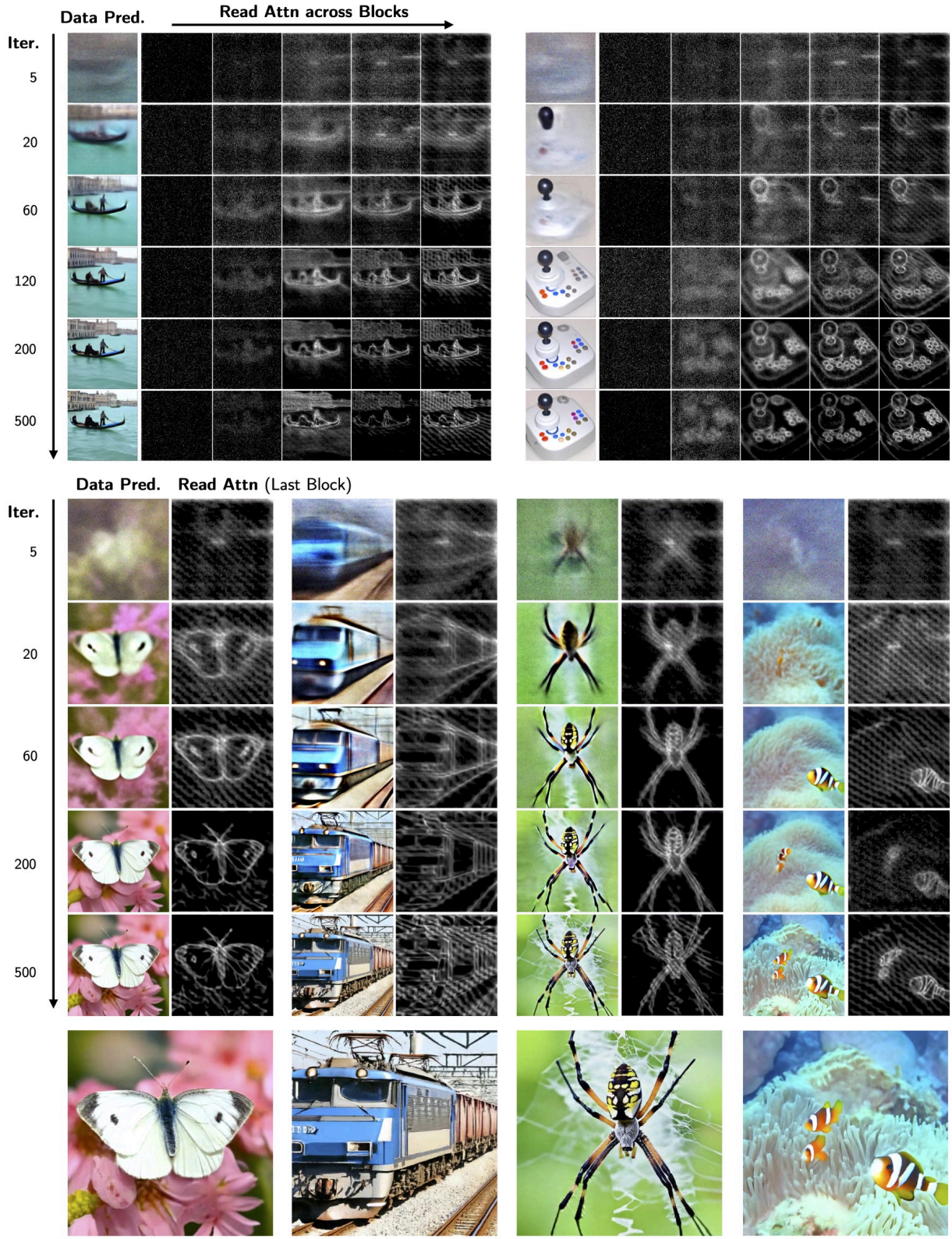


Figure 6. **Visualizing Adaptive Computation.** The read attention reveals which information is routed into latents for heavy computation. We visualize the read attention (averaged across latents) at each block (top) or the last block (bottom), at selected steps of the reverse process when generating ImageNet 512×512 samples. While it is similar across samples in early iterations, it becomes more sparse and data-specific, focusing latent computation on more complex regions.

Table 1. RIN configurations for each task.

	128px	256px	512px	1024px	Kinetics
$\lvert Z \rvert$	128	256	256	256	256
$\dim(Z)$	1024	1024	768	768	1024
$\lvert X \rvert$	1024	1024	4096	16384	2048
$\dim(X)$	512	512	512	512	512
Blocks	6	6	6	6	6
Depth K	4	4	6	8	4
Tokens	4×4	8×8	8×8	8×8	2×4×4

Table 2. Comparison to leading approaches for Class-Conditional Generation on ImageNet. †: use of class guidance, 1: (Dhariwal & Nichol, 2022), 2: (Ho & Salimans, 2021), 3: (Ho et al., 2022a).

Method	FID ↓	IS ↑	GFLOPs	Param(M)
IN 64×64				
ADM [1]	–	2.07	210	297
CF-guidance [2†]	1.55	66.0	–	–
CDM [3]	1.48	66.0	–	–
RIN	**1.23**	**66.5**	106	281
IN 128×128				
ADM [1]	5.91	–	538	386
ADM + guid. [1†]	2.97	–	>538	>386
CF-guidance [2†]	**2.43**	**156.0**	–	–
CDM [3]	3.51	128.0	1268	1058
RIN	2.75	144.1	194	410
IN 256×256				
ADM [1]	10.94	100.9	2212	553
ADM + guid.[1†]	4.59	186.7	>2212	>553
CDM[3]	4.88	158.7	2620	1953
RIN	4.51	161.0	334	410
RIN + inp. scale	**3.42**	**182.0**	334	410
IN 512×512				
ADM [1]	23.2	58.1	4122	559
ADM + guid.[1†]	7.72	172.7	>4122	>559
RIN + inp. scale	**3.95**	**216.0**	415	320
IN 1024×1024				
RIN + inp. scale	**8.72**	**163.9**	1120	412

For video prediction, we use the Kinetics-600 dataset (Carreira et al., 2018) at 16×64×64 resolution. For evaluation, we follow common practice (Ho et al., 2022b) and use FVD (Unterthiner et al., 2018) and Inception Scores computed on 50K samples, with 400 or 1000 steps of DDPM.

3.3. Comparison to SOTA

Image Generation. Table 2 compares our architectures against existing state-of-the-art pixel-space diffusion models on ImageNet. Despite being fully attention-based and single-scale, our model attains superior generation quality (in both FID and IS) compared to existing models that rely on specialized convolutional architectures, cascaded generation, and/or class-guidance. Both the parameter count and FLOPs are significantly reduced in our model compared to baselines, which is useful for training performant models at higher resolutions without relying on cascades (see samples in Appendix Fig. D.1 & D.2). For large images (512 and 1024), we report performance of RINs trained with input scaling (Chen, 2023). We find that 256 latents are sufficient for strong performance even for 1024×1024 images, which produce 16384 tokens; this is 2× more efficient than the 256×256 ADM UNet, despite operating at 4× higher resolution.

Table 3. Video Prediction on Kinetics. †: reconstruction guidance. 1: (Clark et al., 2019), 2: (Walker et al., 2021), 3: (Luc et al., 2020), 4: (Nash et al., 2022), 5: (Ho et al., 2022b).

Method	FVD	IS	GFLOPs	Param (M)
DVD-GAN-FP[1]	69.1	–	–	–
Video VQ-VAE[2]	64.3	–	–	–
TrIVD-GAN-FP[3]	25.7	12.54	–	–
Transframer[4]	25.4	–	–	–
Video Diffusion[5]†	16.6	15.64	4136	1100
RIN – 400 steps	11.5	17.7	386	411
RIN – 1000 steps	**10.8**	**17.7**	386	411

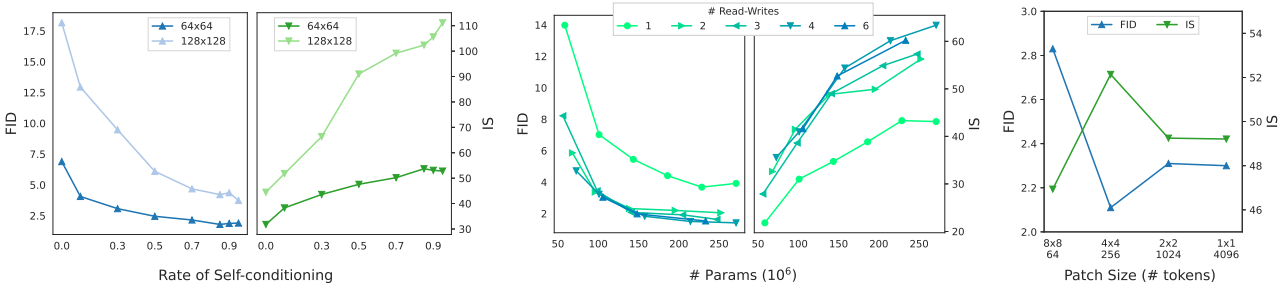
Despite the lack of inductive bias, the model also works well with small datasets such as CIFAR-10. Compared to state-of-the-art FID of 1.79 EDM (Karras et al., 2022), we obtain 1.81 FID without using their improved sampling procedure. We used a model with 31M params (2x smaller) and trained in 3 hours (10x less) using comparable compute.

Video Generation. Table 3 compares our model to existing methods on the Kinetics-600 Video Prediction benchmark. We follow common practice and use 5 conditioning frames. Despite the architecture's simplicity, RINs attain superior quality and are more efficient (up to 10× per step), without using guidance. Beyond using 3D patches instead of 2D patches, the architecture is identical to that used in 256×256 image generation; while the number of tokens is 2048, the model can attain strong performance with 256 latents. The model is especially suitable for video given the intense temporal redundancy, and learns to copy information and dedicate computation to regions of change, as discussed in Section 3.5. Samples can be found in Appendix Fig. D.5.

3.4. Ablations

For efficiency, we ablate using smaller architectures (latent dimension of 768 instead of 1024) on the ImageNet 64×64 and 128×128 tasks with higher learning rate (2×10^{-3}) and fewer updates (150k and 220k, respectively). While this performs worse than our best models, it is sufficient for demonstrating the effect of different design choices.

Latent Self-conditioning. We study the effect of the rate of self-conditioning at training time. A rate of 0 denotes the



(a) Latent self-condition (b) Number of Blocks (c) Tokenization

Figure 7. **Ablations.** (a) Effect of the self-conditioning rate for training: self-conditioning is crucial; a rate of 0 is the special case of no self-conditioning. (b) Effect of the read-write/routing frequency: multiple rounds of read-writes are important to obtain the best result. (c) Effect of tokenization: the model can handle a large number (4096, with 1×1 patches in this case) of tokens on the inferface.

special case where no self-conditioning is used (for training nor inference), while a rate > 0 e.g. 0.9 means that self-conditioning is used for 90% of each batch of training tasks (and always used at inference). As demonstrated in Figure 7a, there is a clear correlation between self-conditioning rate and sample quality (i.e., FID/IS), validating the importance using latent self-conditioning to provide context for enhanced routing. We use a rate of 0.9 for our best results reported.

Stacking Blocks. An important design choice in our architecture is the stacking of read-process-write blocks to enhance global and local processing. For a fair comparison, we analyze the effect of model size on generation quality for a variety of read-write frequencies (Fig. 7b) obtained by stacking blocks with varying number of processing layers per block. Note that a *single* read-write operation *without latent self-conditioning* is similar to architectures such as PerceiverIO (Jaegle et al., 2021a). With a single read-write, the performance saturates earlier as we increase model size. With more frequent read-writes, the model saturates later and with significantly better sample quality, validating the importance of iterative routing.

Tokenization. Recall that images are split into patches to form tokens on the interface. Fig. 7c shows that RINs can handle a wide range of patch sizes. For instance, it can scale to a large number of tokens (4096, for 1×1). While larger patch sizes force tokens to represent more information (i.e., with 8×8 patches), performance remains reasonable.

Effect of Noise Schedule. We find that the sigmoid schedule with an appropriate temperature is more stable training than the cosine schedule, particularly for larger images. For sampling, the noise schedule has less impact and the default cosine schedule can suffice (see Appendix Figure B.1).

3.5. Visualizing Adaptive Computation

To better understand the network's emergent adaptive computation, we analyze how information is routed by visualizing the attention distribution of read operations. For image generation, this reveals which parts of the image are most attended to for latent computation. Figure 6 shows the progression of two samples across the reverse process and the read attention (averaged over latents) through the blocks of the corresponding forward pass. As the generation progresses, the first read (guided by latent self-conditioning) is increasingly adapted to the sample. The read attention distribution becomes more sparse and favour regions of high information. Since the read attention loads information into the latents for high capacity computation, this suggests that the model learns to dynamically allocate computation on information as needed. More examples for ImageNet can be found in Appendix Fig. D.6. Appendix Fig. D.7 further shows similar phenomena in the video prediction setting, with the added effect of reading favouring information that cannot merely be copied from conditioning frames, such as object motion and panning.

4. Related Work

Neural architectures. Recurrent Interface Networks bear resemblance to architectures that leverage auxiliary memory to decouple computation from the input structure such as Memory Networks (Weston et al., 2014; Sukhbaatar et al., 2015), Neural Turing Machines (Graves et al., 2014), Stack-RNN (Joulin & Mikolov, 2015), Set Transformer (Lee et al., 2019), Memory Transformers (Burtsev et al., 2020), Slot Attention (Locatello et al., 2020), BigBird (Zaheer et al., 2020), and Workspace models (Goyal et al., 2021). While latents in our work are similar to auxiliary memory in prior work, we allocate the bulk of computation to latents and iteratively write back updates to the interface, rather than treating them simply as auxiliary memory. Recurrent Interface Networks are perhaps most similar to Set Transformers (Lee et al., 2018) and Perceivers (Jaegle et al., 2021b;a), which also leverage a set of latents for input-agnostic computation. Unlike these approaches, RINs alternate computation between the interface and latents, which is important for processing of information at both local and global levels without resort-

ing to prohibitively many latents. Moreover, in contrast to existing architectures, latent self-conditioning allows RINs to leverage recurrence; this allows for propagation of routing context along very deep computation graphs to amortize the cost of iterative routing, which is crucial for achieving strong performance.

Other approaches for adaptive computation have mainly explored models with dynamic depth with recurrent networks (Graves, 2016; Figurnov et al., 2017) or sparse computation (Yin et al., 2021), facing the challenges non-differentiability and dynamic or masked computation graphs. RINs are able to allocate compute non-uniformly despite having fixed computation graphs and being differentiable. RINs are closely related to recurrent models with input attention such as (Gregor et al., 2015), but scale better by leveraging piecewise optimization enabled by diffusion models.

Diffusion Models. Common diffusion models for images and videos can be roughly divided into pixel diffusion models (Sohl-Dickstein et al., 2015; Ho et al., 2020; Song et al., 2020; Dhariwal & Nichol, 2022; Ho et al., 2022a; Karras et al., 2022) and latent diffusion models (Rombach et al., 2022). In this work we focus on pixel diffusion models due to their relative simplicity. It is known to be challenging to train pixel diffusion models for high resolution images on ImageNet without guidance (Dhariwal & Nichol, 2022; Ho & Salimans, 2021) or cascades (Ho et al., 2022a). We show how improved architectures can allow for scaling pixel-level diffusion models to such large inputs without guidance and cascades, and we expect some insights to transfer to latent diffusion models (Rombach et al., 2022).

The U-Net (Ronneberger et al., 2015; Ho et al., 2020) is the predominant architecture for image and video diffusion models (Dhariwal & Nichol, 2022; Ho et al., 2022a;b). While recent work (Luhman & Luhman, 2022) has explored pixel-level diffusion with Transformers, they have not been shown to attain strong performance or scale to large inputs. Concurrent work (Peebles & Xie, 2022) has shown Transformers may be more tenable when combined with latent diffusion i.e. by downsampling inputs with large-scale pretrained VAEs, but reliance on uniform computation limits gracefully scaling to larger data. Our model suggests a path forward for simple performant and scalable iterative generation of images and video, comparing favourably to U-Nets in sample quality and efficiency, while based on domain-agnostic operations such as attention and fully-connected MLPs, and therefore more universal.

Self-conditioning for diffusion models was originally proposed in (Chen et al., 2022c). It bears similarity to step-unrolled autoencoders (Savinov et al., 2021) and has been adopted in several existing work (Strudel et al., 2022; Dieleman et al., 2022; Chen et al., 2022a). While these works condition on predictions of data, latent self-conditioning conditions a neural network on its own hidden activations, akin to recurrent neural network at inference while training without backpropagation through time.

5. Conclusion

Recurrent Interface Networks are neural networks that explicitly partition hidden units into interface and latent tokens. The interface links the input space to the core computation units operating on the latents, decoupling computation from data layout and allowing adaptive allocation of capacity to different parts of the input. We show how the challenge of building latents can be amortized in recurrent computation settings – where the effective network is deep and persistent context can be leveraged – while still allowing for efficient training. While RINs are domain-agnostic, we found them to be performant and efficient for image and video generation tasks. As we look towards building more powerful generative models, we hope RINs can serve as a simple and unified architecture that scales to high-dimensional data across a range of modalities. To further improve RINs, we hope to better understand and enhance the effect of latent self-conditioning. Moreover, we hope to combine the advantages of RINs with orthogonal techniques, such as guidance and latent diffusion.

Acknowledgements

We thank Geoffrey Hinton, Thomas Kipf, Sara Sabour, and Ilija Radosavovic for helpful suggestions, and Ben Poole for his invaluable feedback on our draft. We also thank Lala Li, Saurabh Saxena, Ruixiang Zhang for helpful discussions and their contributions to the Pix2Seq (Chen et al., 2021; 2022b) codebase used in this project.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pp. 265–283, 2016.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Burtsev, M. S., Kuratov, Y., Peganov, A., and Sapunov, G. V. Memory transformer. *arXiv preprint arXiv:2006.11527*, 2020.

Carreira, J., Noland, E., Banki-Horvath, A., Hillier, C., and Zisserman, A. A short note about kinetics-600. *arXiv preprint arXiv:1808.01340*, 2018.

Chen, T. On the importance of noise schedules for diffusion models. *arXiv preprint arXiv:2301.10972*, 2023.

Chen, T., Saxena, S., Li, L., Fleet, D. J., and Hinton, G. Pix2seq: A language modeling framework for object detection. *arXiv preprint arXiv:2109.10852*, 2021.

Chen, T., Li, L., Saxena, S., Hinton, G., and Fleet, D. J. A generalist framework for panoptic segmentation of images and videos. *arXiv preprint arXiv:2210.06366*, 2022a.

Chen, T., Saxena, S., Li, L., Lin, T.-Y., Fleet, D. J., and Hinton, G. A unified sequence interface for vision tasks. *arXiv preprint arXiv:2206.07669*, 2022b.

Chen, T., Zhang, R., and Hinton, G. Analog bits: Generating discrete data using diffusion models with self-conditioning. *arXiv preprint arXiv:2208.04202*, 2022c.

Clark, A., Donahue, J., and Simonyan, K. Adversarial video generation on complex datasets. *arXiv preprint arXiv:1907.06571*, 2019.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Dhariwal, P. and Nichol, A. Diffusion models beat GANs on image synthesis. In *NeurIPS*, 2022.

Dieleman, S., Sartran, L., Roshannai, A., Savinov, N., Ganin, Y., Richemond, P. H., Doucet, A., Strudel, R., Dyer, C., Durkan, C., et al. Continuous diffusion for categorical data. *arXiv preprint arXiv:2211.15089*, 2022.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Figurnov, M., Collins, M. D., Zhu, Y., Zhang, L., Huang, J., Vetrov, D., and Salakhutdinov, R. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1039–1048, 2017.

Fukushima, K. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, 1(2): 119–130, 1988.

Goyal, A., Didolkar, A., Lamb, A., Badola, K., Ke, N. R., Rahaman, N., Binas, J., Blundell, C., Mozer, M., and Bengio, Y. Coordination among neural modules through a shared global workspace. *arXiv preprint arXiv:2103.01197*, 2021.

Graves, A. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Graves, A., Wayne, G., and Danihelka, I. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Gregor, K., Danihelka, I., Graves, A., Rezende, D., and Wierstra, D. Draw: A recurrent neural network for image generation. In *International conference on machine learning*, pp. 1462–1471. PMLR, 2015.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Ho, J. and Salimans, T. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

Ho, J., Jain, A., and Abbeel, P. Denoising Diffusion Probabilistic Models. *NeurIPS*, 2020.

Ho, J., Saharia, C., Chan, W., Fleet, D. J., Norouzi, M., and Salimans, T. Cascaded diffusion models for high fidelity image generation. *JMLR*, 2022a.

Ho, J., Salimans, T., Gritsenko, A., Chan, W., Norouzi, M., and Fleet, D. J. Video Diffusion Models. In *NeurIPS*, 2022b.

Jaegle, A., Borgeaud, S., Alayrac, J.-B., Doersch, C., Ionescu, C., Ding, D., Koppula, S., Zoran, D., Brock, A., Shelhamer, E., et al. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021a.

Jaegle, A., Gimeno, F., Brock, A., Vinyals, O., Zisserman, A., and Carreira, J. Perceiver: General perception with iterative attention. In *International conference on machine learning*, pp. 4651–4664. PMLR, 2021b.

Joulin, A. and Mikolov, T. Inferring algorithmic patterns with stack-augmented recurrent nets. *Advances in neural information processing systems*, 28, 2015.

Karras, T., Aittala, M., Aila, T., and Laine, S. Elucidating the design space of diffusion-based generative models. *arXiv preprint arXiv:2206.00364*, 2022.

Kingma, D., Salimans, T., Poole, B., and Ho, J. Variational diffusion models. *Advances in neural information processing systems*, 34:21696–21707, 2021.

Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 (canadian institute for advanced research). URL http://www.cs.toronto.edu/~kriz/cifar.html.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4): 541–551, 1989.

Lee, J., Lee, Y., Kim, J., Kosiorek, A. R., Choi, S., and Teh, Y. W. Set transformer. *CoRR*, abs/1810.00825, 2018. URL http://arxiv.org/abs/1810.00825.

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pp. 3744–3753. PMLR, 2019.

Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. Object-centric learning with slot attention. *Advances in Neural Information Processing Systems*, 33:11525–11538, 2020.

Luc, P., Clark, A., Dieleman, S., Casas, D. d. L., Doron, Y., Cassirer, A., and Simonyan, K. Transformation-based adversarial video prediction on large-scale data. *arXiv preprint arXiv:2003.04035*, 2020.

Luhman, T. and Luhman, E. Improving diffusion model efficiency through patching. *arXiv preprint arXiv:2207.04316*, 2022.

Nash, C., Carreira, J., Walker, J., Barr, I., Jaegle, A., Malinowski, M., and Battaglia, P. Transframer: Arbitrary frame prediction with generative models. *arXiv preprint arXiv:2203.09494*, 2022.

Nichol, A. and Dhariwal, P. Improved denoising diffusion probabilistic models. *arXiv preprint arXiv:2102.09672*, 2021.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Peebles, W. and Xie, S. Scalable diffusion models with transformers. *arXiv preprint arXiv:2212.09748*, 2022.

Rae, J. W., Potapenko, A., Jayakumar, S. M., and Lillicrap, T. P. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.

Savinov, N., Chung, J., Binkowski, M., Elsen, E., and Oord, A. v. d. Step-unrolled denoising autoencoders for text generation. *arXiv preprint arXiv:2112.06749*, 2021.

Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., and Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pp. 2256–2265. PMLR, 2015.

Song, J., Meng, C., and Ermon, S. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

Strudel, R., Tallec, C., Altché, F., Du, Y., Ganin, Y., Mensch, A., Grathwohl, W., Savinov, N., Dieleman, S., Sifre, L., et al. Self-conditioned embedding diffusion for text generation. *arXiv preprint arXiv:2211.04236*, 2022.

Sukhbaatar, S., Weston, J., Fergus, R., et al. End-to-end memory networks. *Advances in neural information processing systems*, 28, 2015.

Unterthiner, T., van Steenkiste, S., Kurach, K., Marinier, R., Michalski, M., and Gelly, S. Towards accurate generative models of video: A new metric & challenges. *arXiv preprint arXiv:1812.01717*, 2018.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Walker, J., Razavi, A., and Oord, A. v. d. Predicting video with vqvae. *arXiv preprint arXiv:2103.01950*, 2021.

Weston, J., Chopra, S., and Bordes, A. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

Yin, H., Vahdat, A., Alvarez, J., Mallya, A., Kautz, J., and Molchanov, P. Adavit: Adaptive tokens for efficient vision transformer. *arXiv preprint arXiv:2112.07658*, 2021.

You, Y., Li, J., Reddi, S., Hseu, J., Kumar, S., Bhojanapalli, S., Song, X., Demmel, J., Keutzer, K., and Hsieh, C.-J. Large batch optimization for deep learning: Training bert in 76 minutes. *arXiv preprint arXiv:1904.00962*, 2019.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., et al. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*, 33:17283–17297, 2020.

A. Architecture Implementation Pseudo-code

Algorithm 3 provides a more detailed implementation of RINs. Note that for clarity, we only show it for image generation task, but for other tasks or data modalities, we only need to change the interface initialization, i.e. the tokenization of the input. We also omit some functions, such as "multihead_attention" and "ffn" (i.e. feed-forward network), which are specified in Transformers (Vaswani et al., 2017) and available as APIs in major deep learning frameworks, such as Tensorflow (Abadi et al., 2016) and PyTorch (Paszke et al., 2019).

Algorithm 3 RINs Implementation Pseudo-code.

```
def block(z, x, num_layers):
  """Core computation block."""
  z = z + multihead_attention(q=layer_norm(z), kv=x, n_heads=16)
  z = z + ffn(layer_norm(z), expansion=4)

  for _ in range(num_layers):
    zn = layer_norm(z)
    z = z + multihead_attention(q=zn, kv=zn, n_heads=16)
    z = z + ffn(layer_norm(z), expansion=4)

  x = x + multihead_attention(q=layer_norm(x), kv=z, n_heads=16)
  x = x + ffn(layer_norm(x), expansion=4)

  return z, x


def rin(x, patch_size, num_latents, latent_dim, interface_dim,
                  num_blocks, num_layers_per_block, prev_latents=None):
  """Forward pass of Network."""
  bsz, image_size, _, _ = x.shape
  size = image_size // patch_size

  # Initialize interface (with image tokenization as an example)
  x = conv(x, kernel_size=patch_size, stride=patch_size, padding='SAME')
  pos_emb = truncated_normal((1, size, size, dim), scale=0.02)
  x = layer_norm(x) + pos_emb


  # Initialize latents
  z = truncated_normal((num_latents, latent_dim), scale=0.02)

  # Latent self-conditioning
  if prev_latents is not None:
    prev_latents = prev_latents + ffn(stop_grad(prev_latents), expansion=4)
    z = z + layer_norm(prev_latents, init_scale=0, init_bias=0)

  # Compute
  for _ in range(num_blocks):
    z, x = block(z, x, num_layers_per_block)

  # Readout
  x = linear(layer_norm(x), dim=3*patch_size**2)
  x = depth_to_space(reshape(x, [bsz, size, size, -1]), patch_size)

  return z, x
```

B. More Details of Training / Sampling Algorithms, and Noise schedules

Algorithm 4 contains different choices of $\gamma(t)$, the continuous time noise schedule function.

Algorithm 4 Continuous time noise scheduling function.

```
def gamma_cosine_schedule(t, ns=0.0002, ds=0.00025):
  # A scheduling function based on cosine function.
  return numpy.cos(((t + ns) / (1 + ds)) * numpy.pi / 2)**2

def gamma_sigmoid_schedule(t, start=-3, end=3, tau=1.0, clip_min=1e-9):
  # A scheduling function based on sigmoid function.
  v_start = sigmoid(start / tau)
  v_end = sigmoid(end / tau)
  output = (-sigmoid((t * (end - start) + start) / tau) + v_end) / (v_end - v_start)
  return np.clip(output, clip_min, 1.)
```

Algorithm 5 contains DDIM (Song et al., 2020) and DDPM (Ho et al., 2020) updating rules, as specified in (Chen et al., 2022c).

Algorithm 5 x_t estimation with DDIM / DDPM updating rules.

```
def ddim_step(x_t, x_pred, t_now, t_next):
  # Estimate x at t_next with DDIM updating rule.
  γ_now = gamma(t_now)
  γ_next = gamma(t_next)
  x_pred = clip(x_pred, -scale, scale)
  eps = 1/√(1−γ_now) * (x_t − √γ_now * x_pred)
  x_next = √γ_next * x_pred + √(1 − γ_next) * eps
  return x_next

def ddpm_step(x_t, x_pred, t_now, t_next):
  # Estimate x at t_next with DDPM updating rule.
  γ_now = gamma(t_now)
  α_now = gamma(t_now) / gamma(t_next)
  σ_now = sqrt(1 − α_now)
  z = normal(mean=0, std=1)
  x_pred = clip(x_pred, -scale, scale)
  eps = 1/√(1−γ_now) * (x_t − √γ_now * x_pred)
  x_next = 1/√α_now * (x_t − (1−α_now)/√(1−γ_now) * eps) + σ_now * z
  return x_next
```

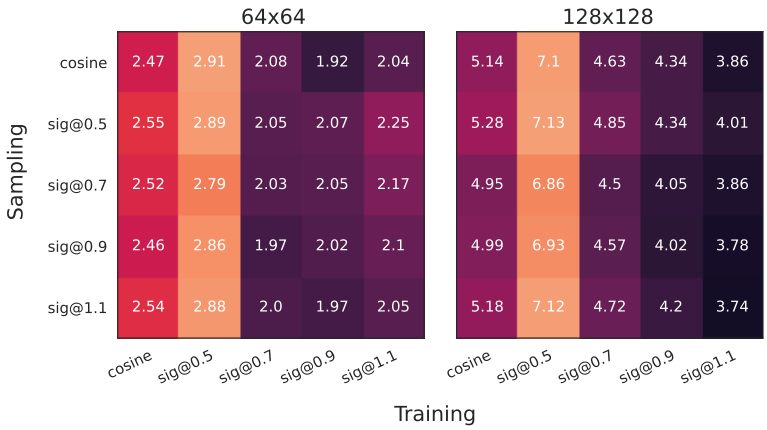


Figure B.1. **Effect of noise schedule.** Comparing noise schedules for training and sampling, with corresponding FID score. The sigmoid schedule with an appropriate temperature is more stable during training than the widely used cosine schedule, particularly for larger images. For sampling, the noise schedule has less impact and the default cosine schedule can suffice.

C. Hyper-parameters and Other Training Details

We train most models on 32 TPUv3 chips with a batch size of 1024. Models for 512×512 and 1024×1024 are trained on 64 TPUv3 chips and 256 TPUv4 chips, respectively. All models are trained with the LAMB optimizer (You et al., 2019).

Table C.1. Model Hyper-parameters.

Task	Input/Output	Blocks	Depth	Latents	dim(Z)	dim(X)	Tokens (patch size)	Heads	Params	GFLOPs
IN	64×64×3	4	4	128	1024	256	256 (4×4)	16	280M	106
IN	128×128×3	6	4	128	1024	512	1024 (4×4)	16	410M	194
IN	256×256×3	6	4	256	1024	512	1024 (8×8)	16	410M	334
IN	512×512×3	6	6	256	768	512	4096 (8×8)	16	320M	415
IN	1024×1024×3	6	8	256	768	512	16384 (8×8)	16	415M	1120
K-600	16×64×64×3	6	4	256	1024	512	2048 (2×4×4)	16	411M	386

Table C.2. Training Hyper-parameters.

Task	Input/Output	Updates	Batch Size	LR	LR-decay	Optim β_2	Weight Dec.	Self-cond. Rate	EMA β
IN	64×64×3	300K	1024	1e-3	cosine	0.999	0.01	0.9	0.9999
IN	128×128×3	600K	1024	1e-3	cosine	0.999	0.001	0.9	0.9999
IN	256×256×3	600K	1024	1e-3	cosine	0.999	0.001	0.9	0.9999
IN	512×512×3	1M	1024	1e-3	cosine	0.999	0.01	0.9	0.9999
IN	1024×1024×3	1M	512	1e-3	None	0.999	0.01	0.9	0.9999
K-600	16×64×64×3	500K	1024	1e-3	cosine	0.999	0.001	0.85	0.99

D. Sample Visualizations

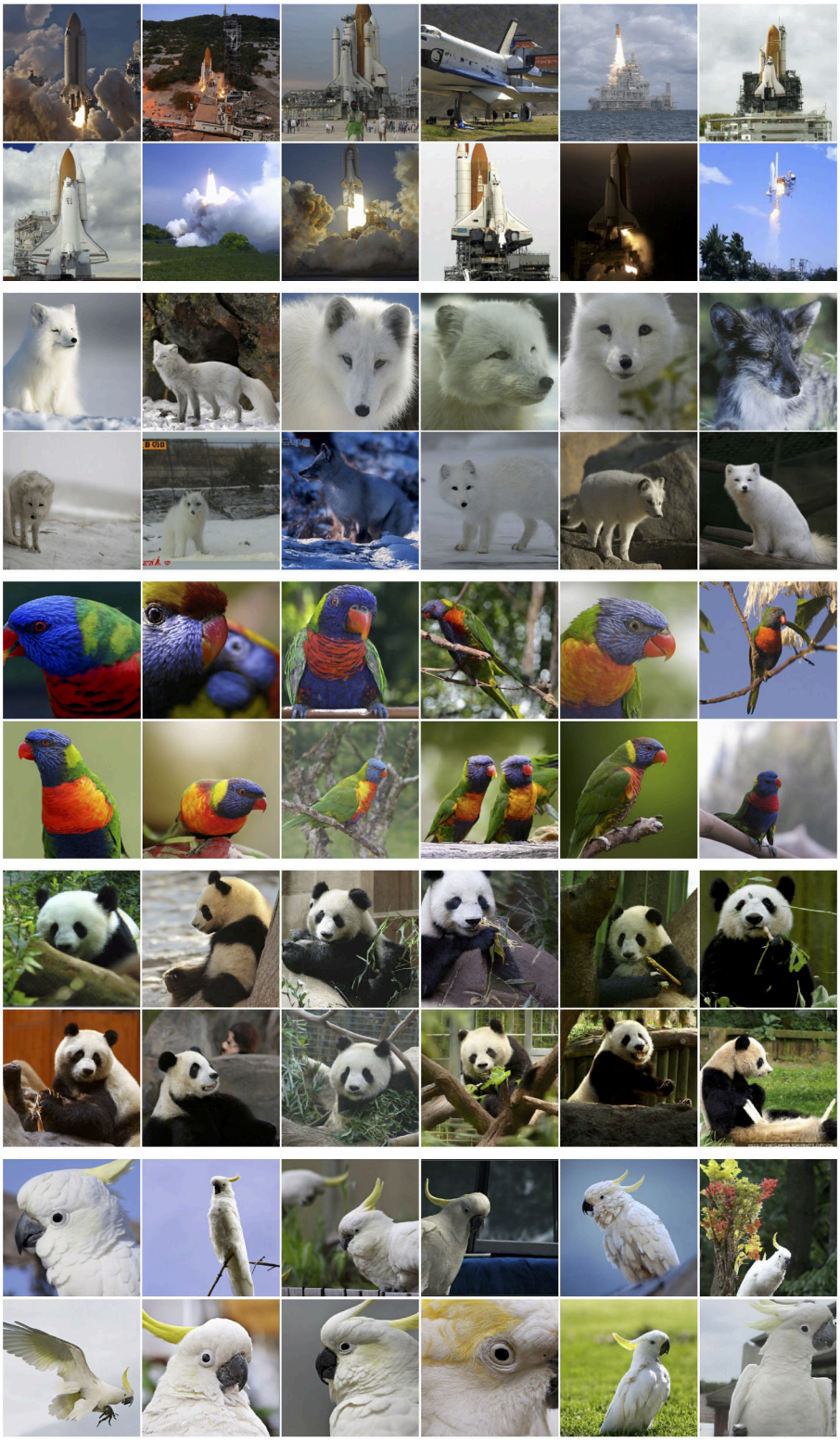


Figure D.1. Selected class-conditional samples from a model trained on ImageNet 256×256. Classes from the top: space shuttle (812), arctic fox (279), lorikeet (90), giant panda (388), cockatoo (89).

Figure D.2. Selected class-conditional samples from a model trained on ImageNet 256×256. Classes from the top: go-kart (573), macaw (88), white wolf (270), lion (291), siberian husky (250).

Figure D.3. Random class-conditional ImageNet 768×768 samples generated by RINs trained with input scaling (Chen, 2023). Note that these samples are uncurated but generated using classifier-free guidance. These demonstrate the architecture can scale to higher-resolutions despite being single-scale and operating directly on pixels.

Figure D.4. Random class-conditional ImageNet 1024×1024 samples generated by RINs trained with input scaling (Chen, 2023). Note that these samples are uncurated but generated using classifier-free guidance.

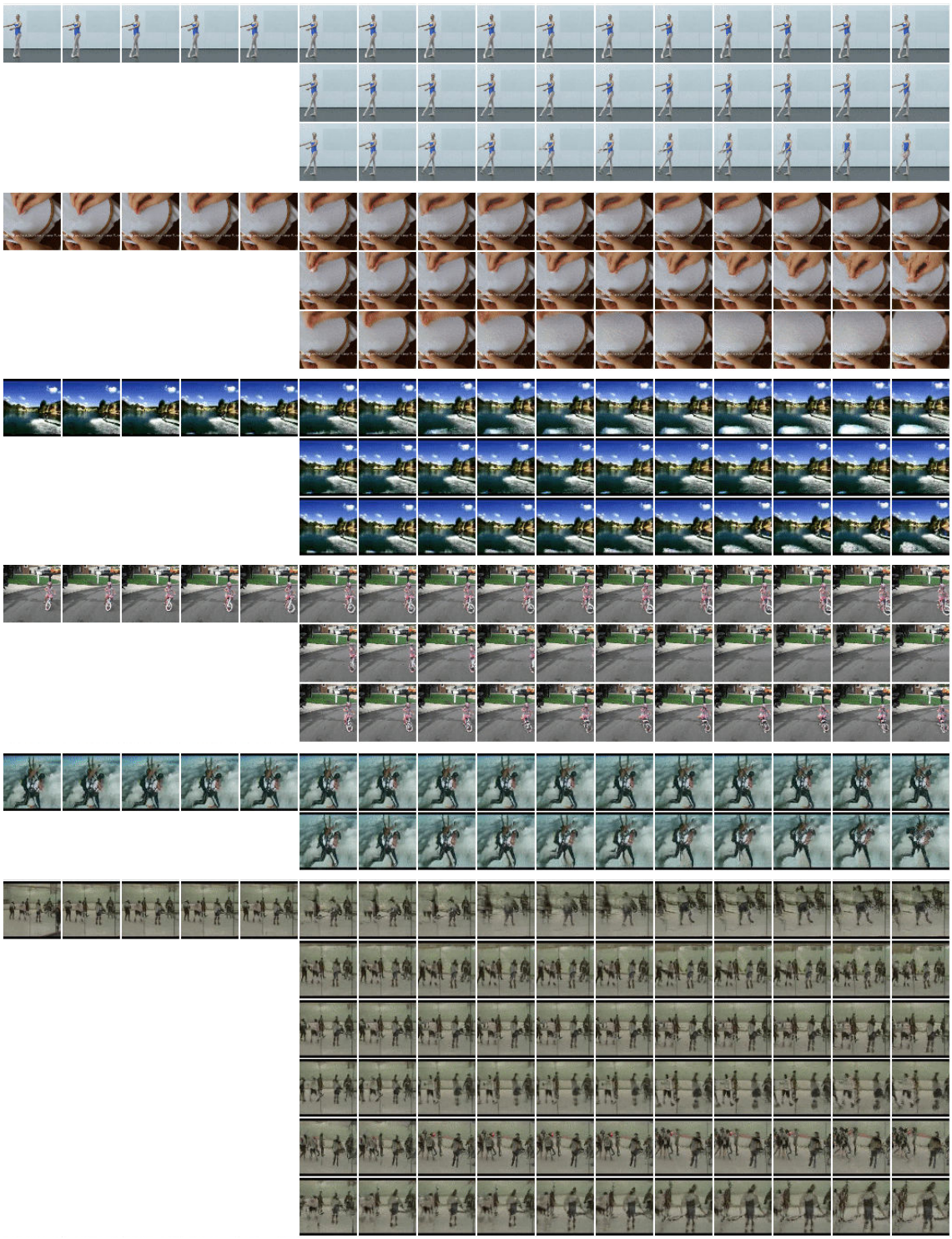


Figure D.5. Selected samples of video prediction on Kinetics-600 at 16×64×64 showing examples of multi-modality across different future predictions, with conditioning frames from the test set. For example, the ballerina's arm and leg movements vary (first); the hand moves in different ways while sewing (second); the wakeboarder faces different waves (third); the bicyclist takes different turns; the sky-divers face different fates; the hockey scene (last) is zoomed and panned in different ways.

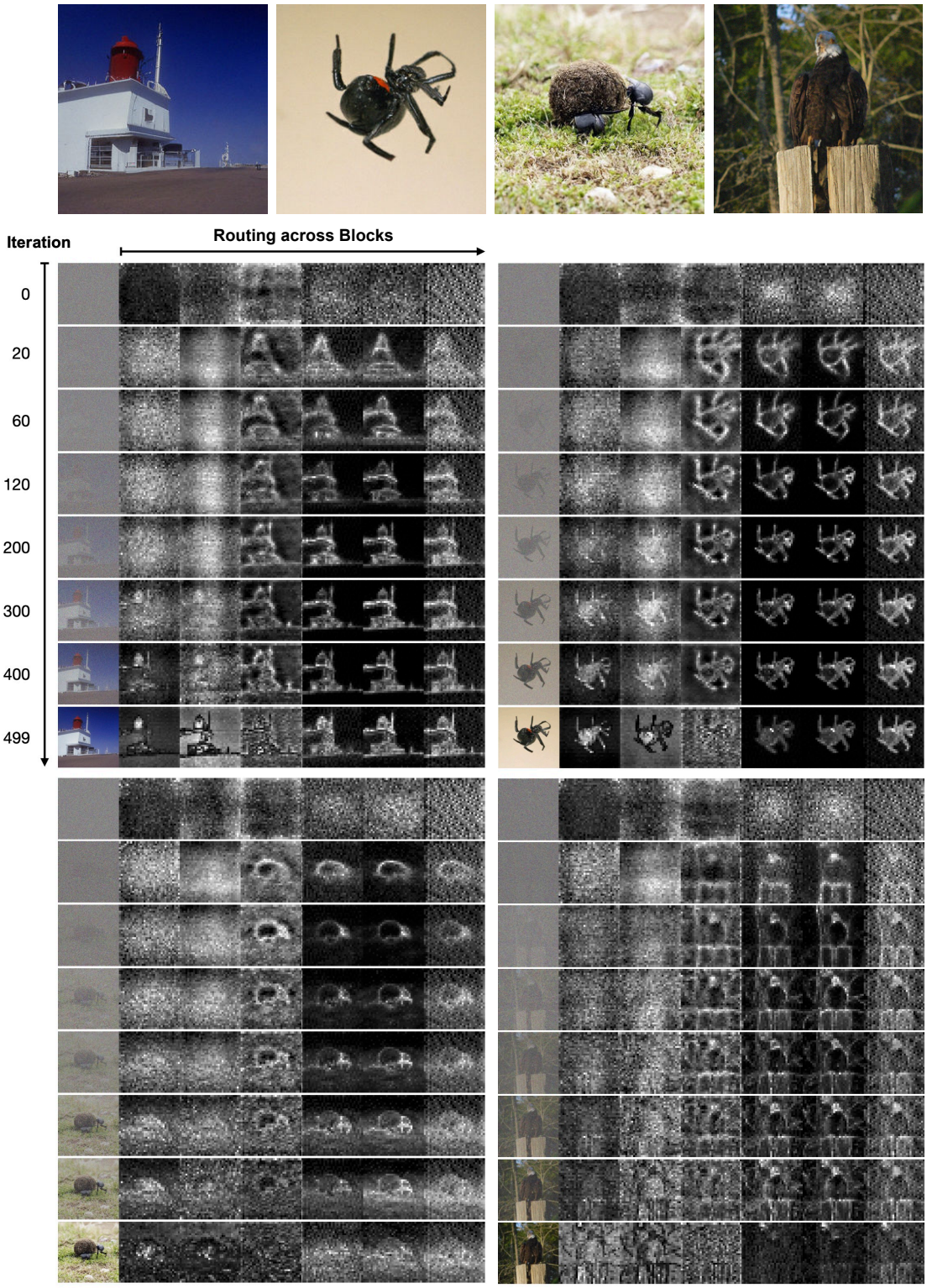


Figure D.6. Visualization of emergent adaptive computation for ImageNet 256×256 samples.

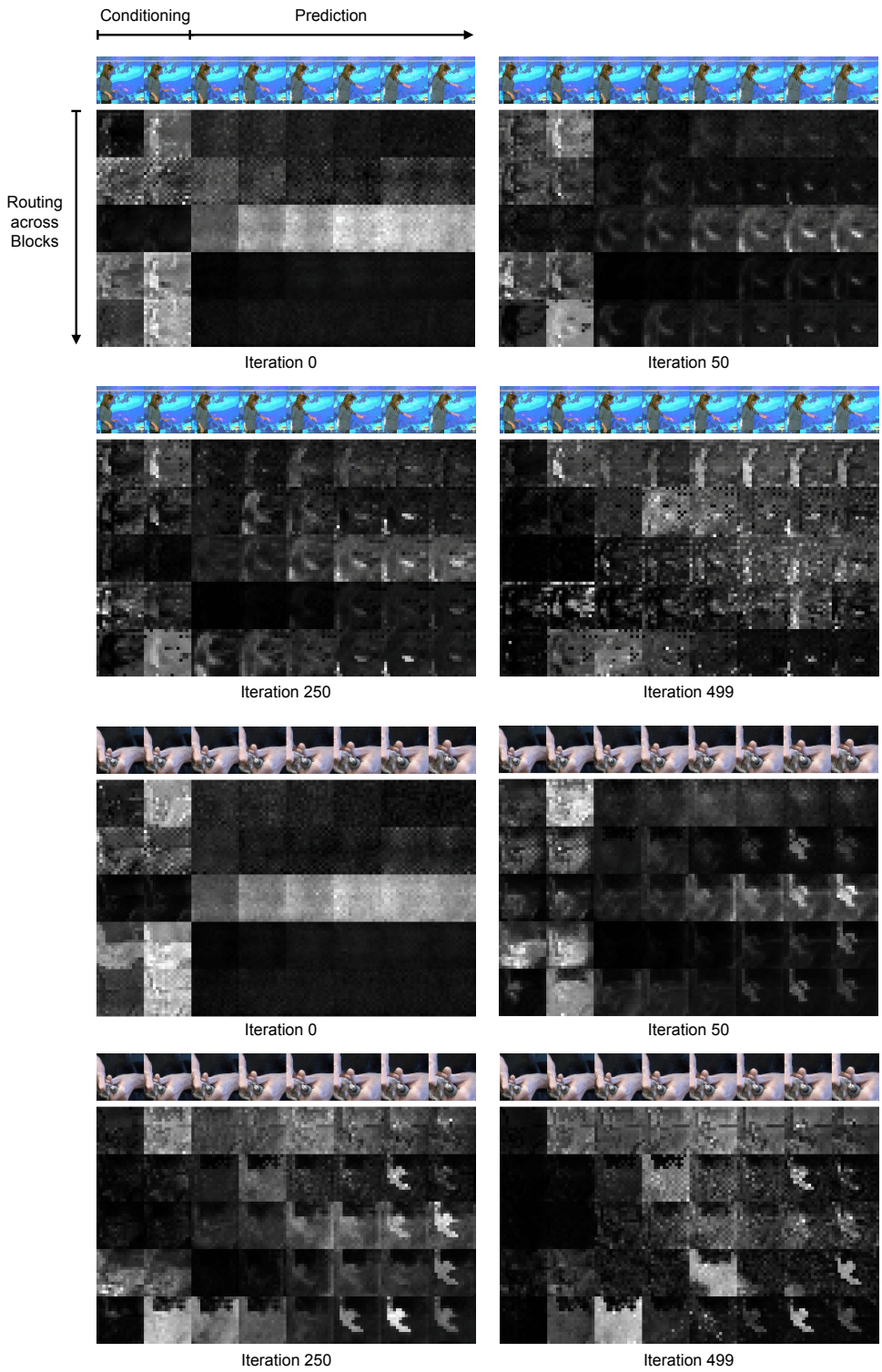


Figure D.7. Visualization of emergent adaptive computation for video prediction on Kinetics-600. The samples are subsampled $2\times$ in time to align with the attention visualization. In each column of the attention visualization, the first two columns are read attention on conditioning frames. We observe that read attention and hence computation is focused on regions of motion, that cannot be generated by simply copying from the conditioning frames.