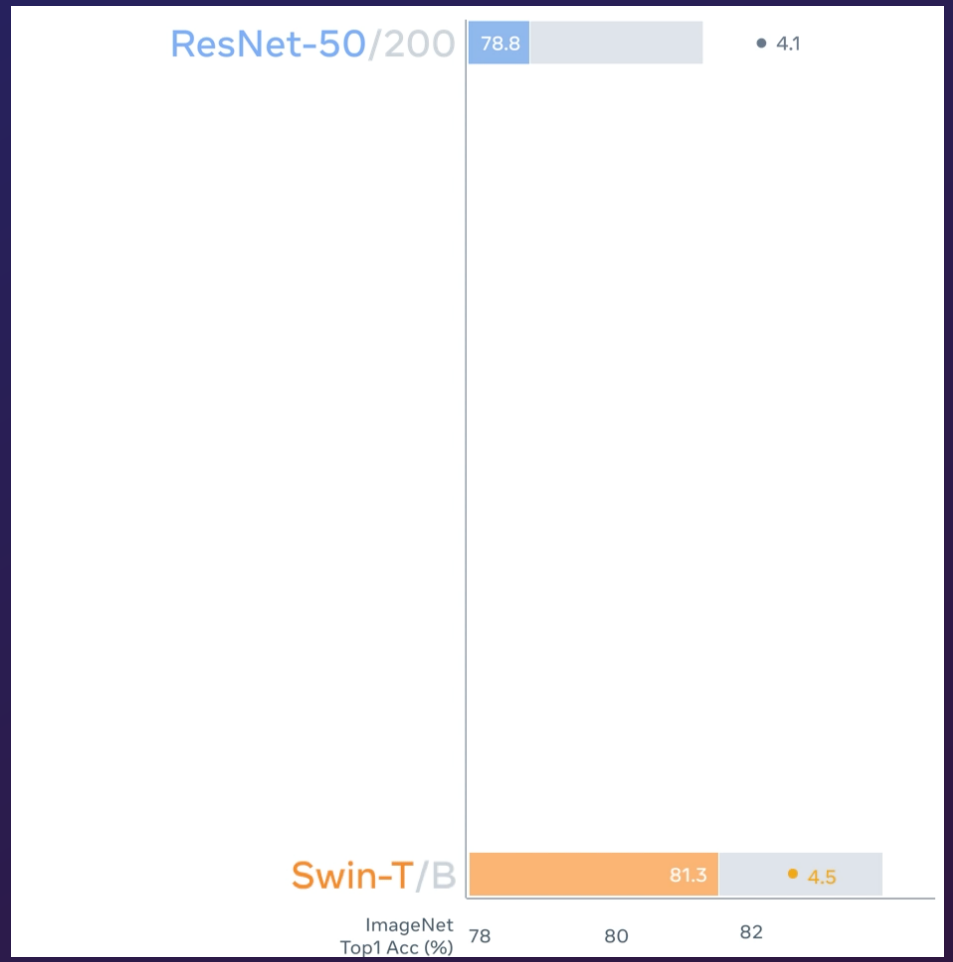
ConvNet vs. Transformer ROUND 2:

Self-Supervised Learning and Diffusion Models

Saining Xie
New York University

Joint work with Sanghyun Woo, Bill Peebles, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon

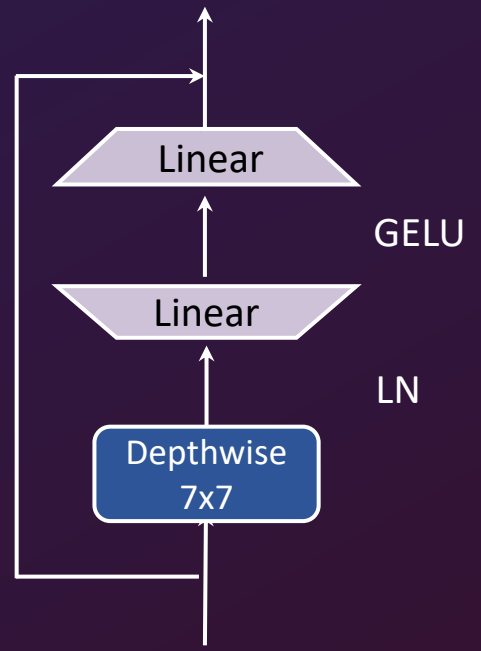
Previously, on T4V …



ResNe
t

Modernize 🚀

ConvNeXt



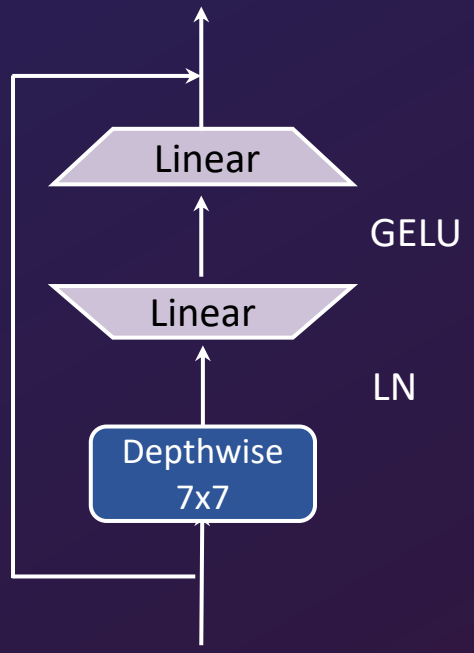
A ConvNet for the 2020s, Liu et al., CVPR 2022

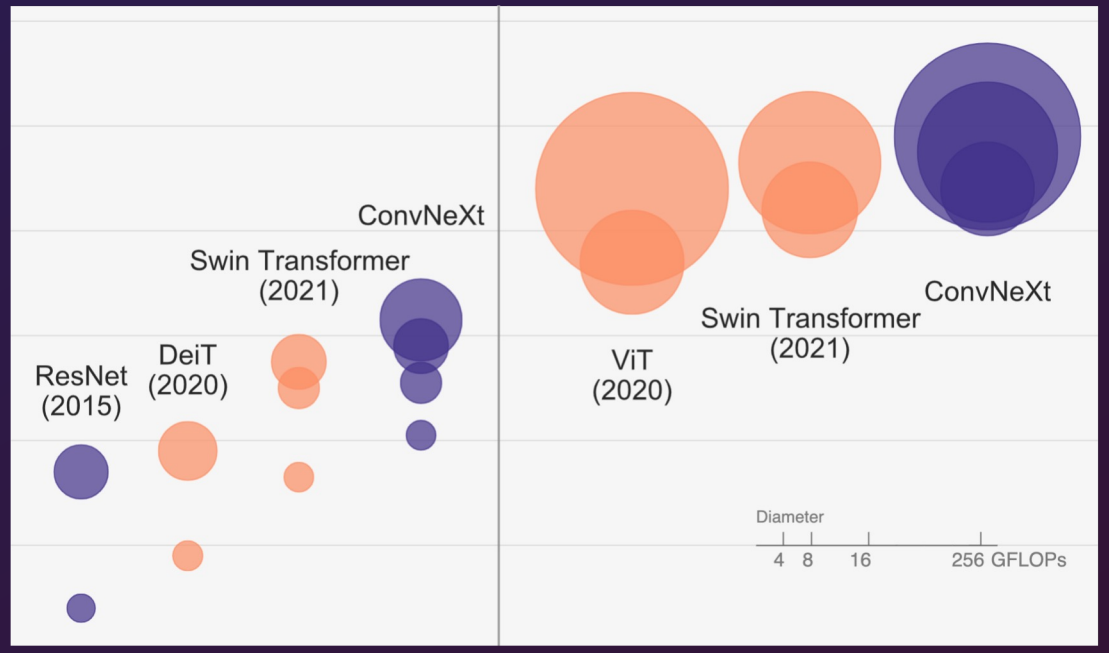
Previously, on T4V …

ResNe
t

Modernize 🚀

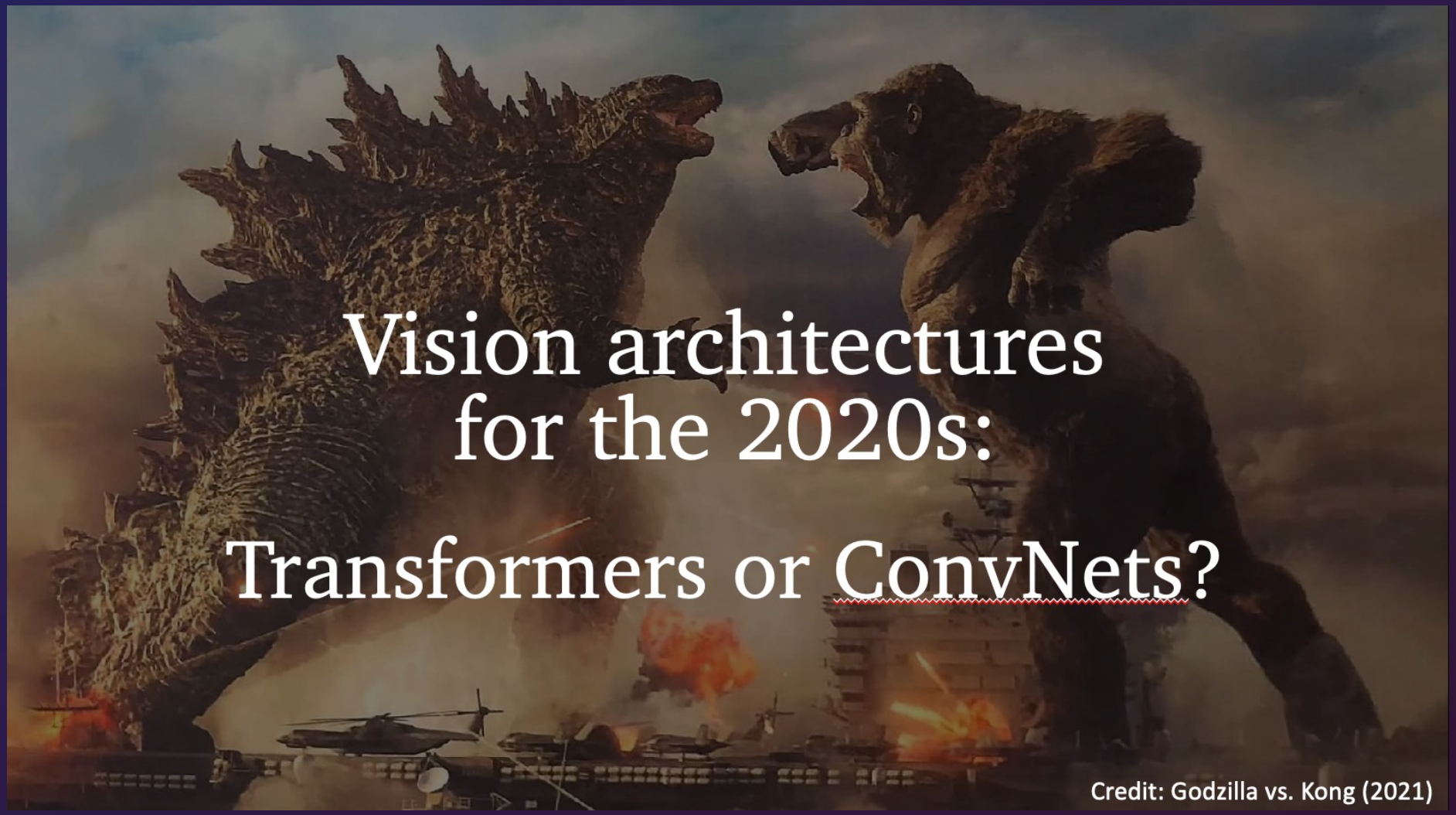
ConvNeXt



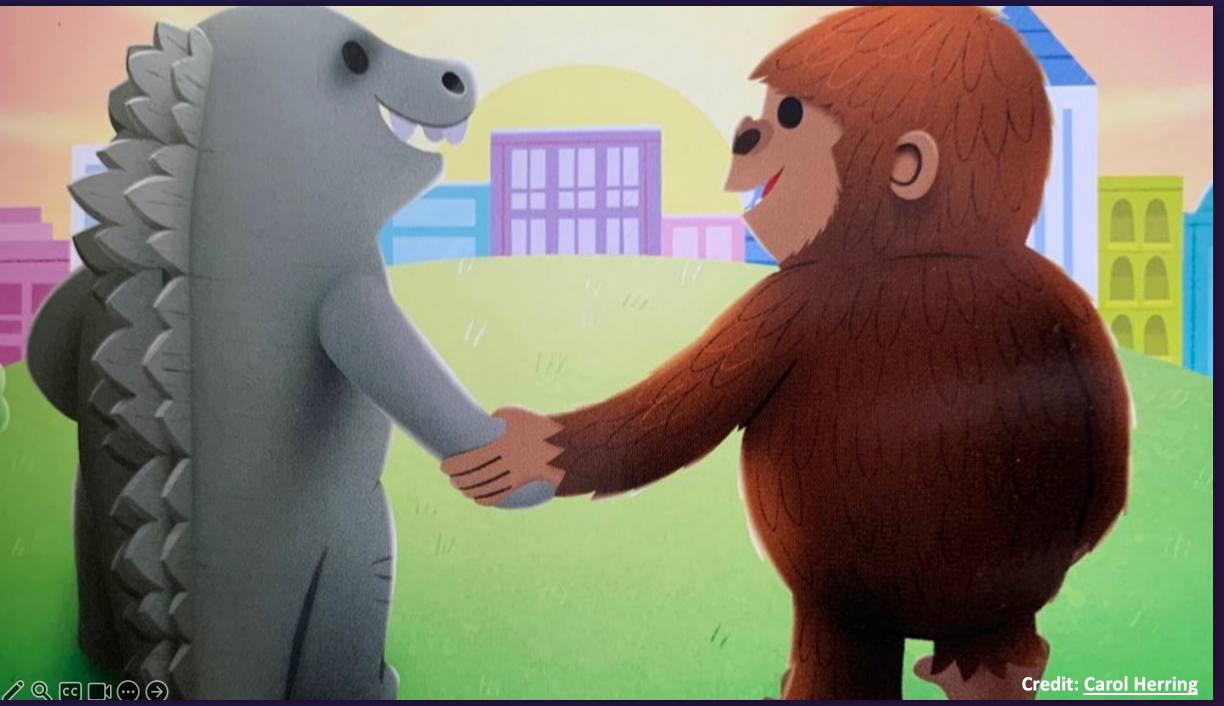


Attention is not essential for scalability.

Previously, on T4V …

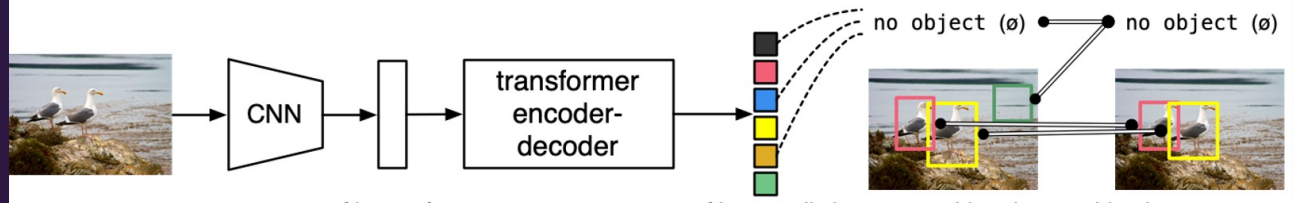


Previously, on T4V …



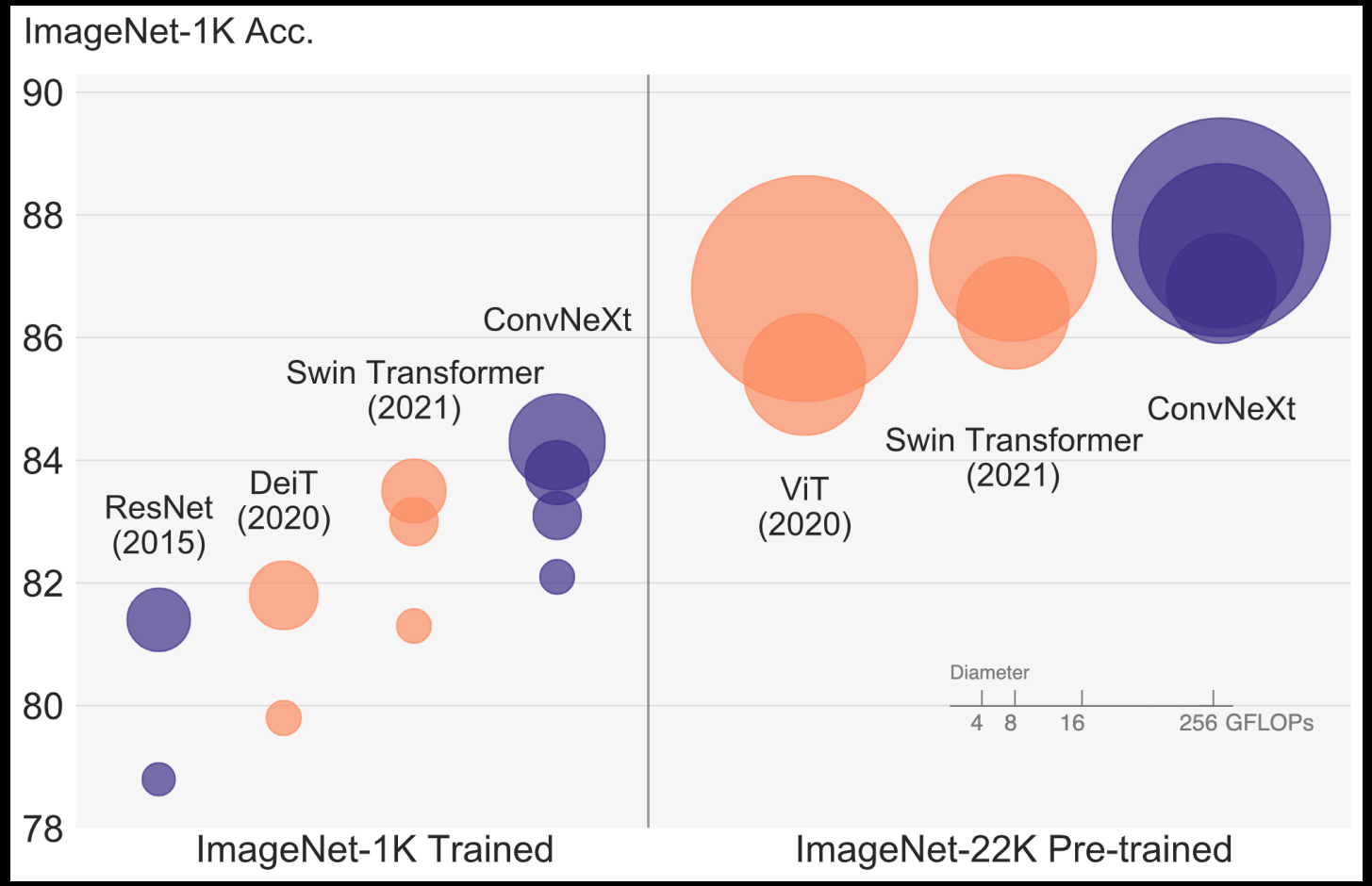
For practitioners, a hybrid system is all you need

- A Conv + self-attention network can work better
  e.g. LeViT [Graham et al., 2021], CoAtNet [Dai et al., 2021], CoaT [Xu et al., 2021], …
- For images, you might not need a full Transformer;
  You need self-attention blocks at the end!

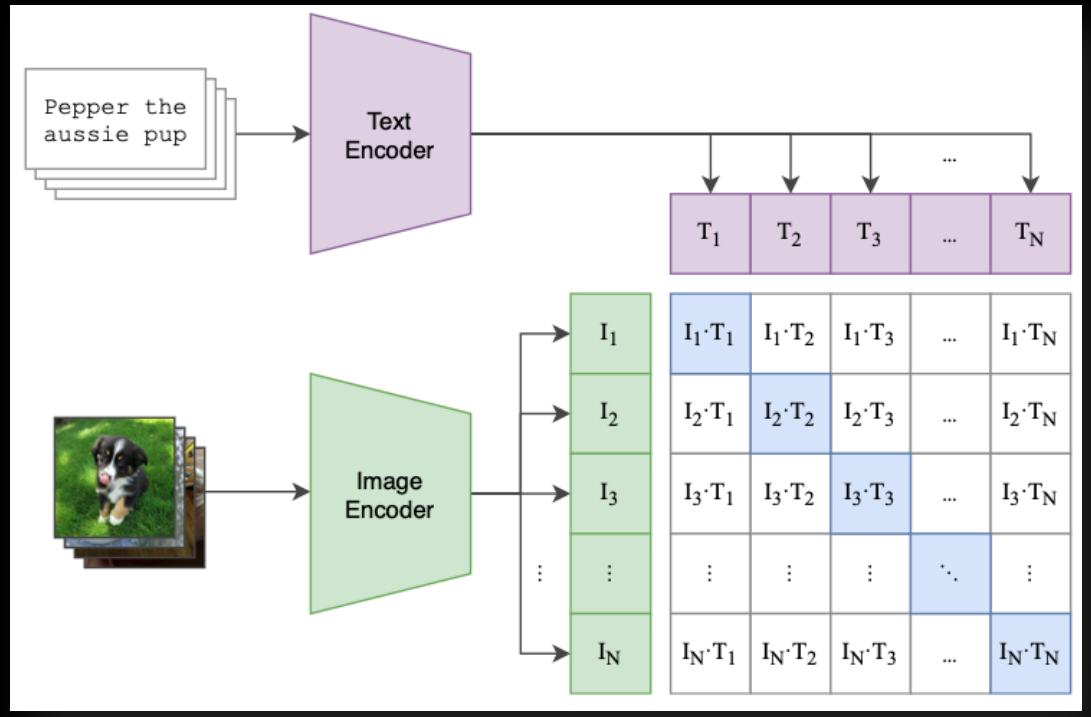


End-to-End Object Detection with Transformers [Carion et al, 2020]

ConvNet vs Transformer:
Supervised Learning



ConvNet vs Transformer:
CLIP Training



Ross Wightman
@wightmanr

Over the past month I've been working on training ConvNeXt CLIP models with OpenCLIP. To date training efforts have focused on the ViT arch. The original OpenAI paper included modified ResNet models but the results weren't inspiring. ConvNeXt is working amazingly (no mods)!

4:44 PM · Jan 29, 2023 · **79.8K** Views

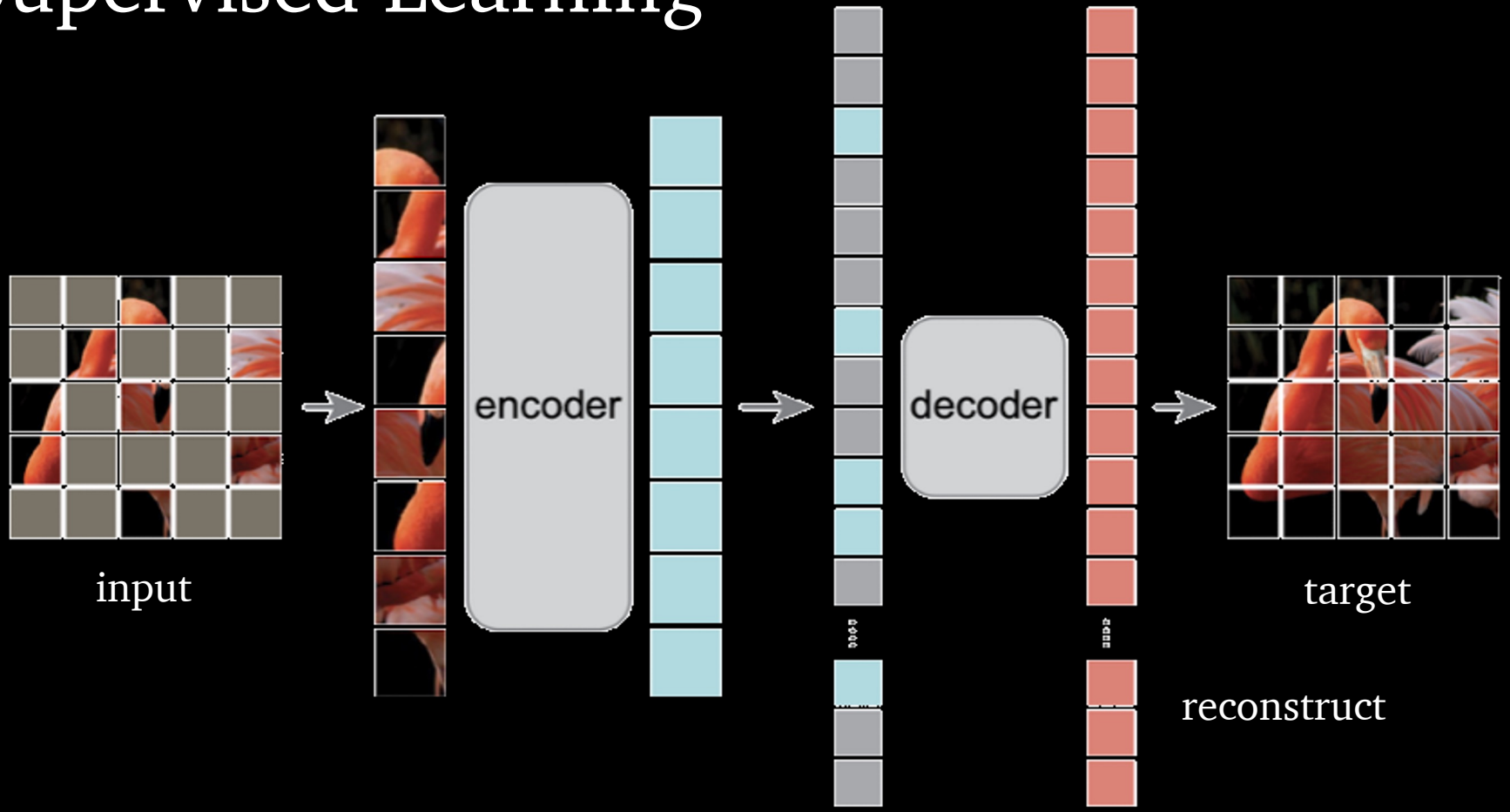
Ross Wightman @wightmanr · Jan 29

Rough summary:
* convnext_base_w @ 256x256 - 70.8 - 71.5 top-1 ZS
* convnext_base_w @ 320x320 - 71.0 - 71.7 top-1 ZS
* convnext_large_d @ 256x256 - 75.9

L/14 is 1.65x more GMAC, 1.45x more activations, 1.22x more params for 75.3 on same dataset and at 2x samples seen!

ConvNeXt is more efficient & more friendly to higher resolution input.

ConvNet vs Transformer:
Self-Supervised Learning



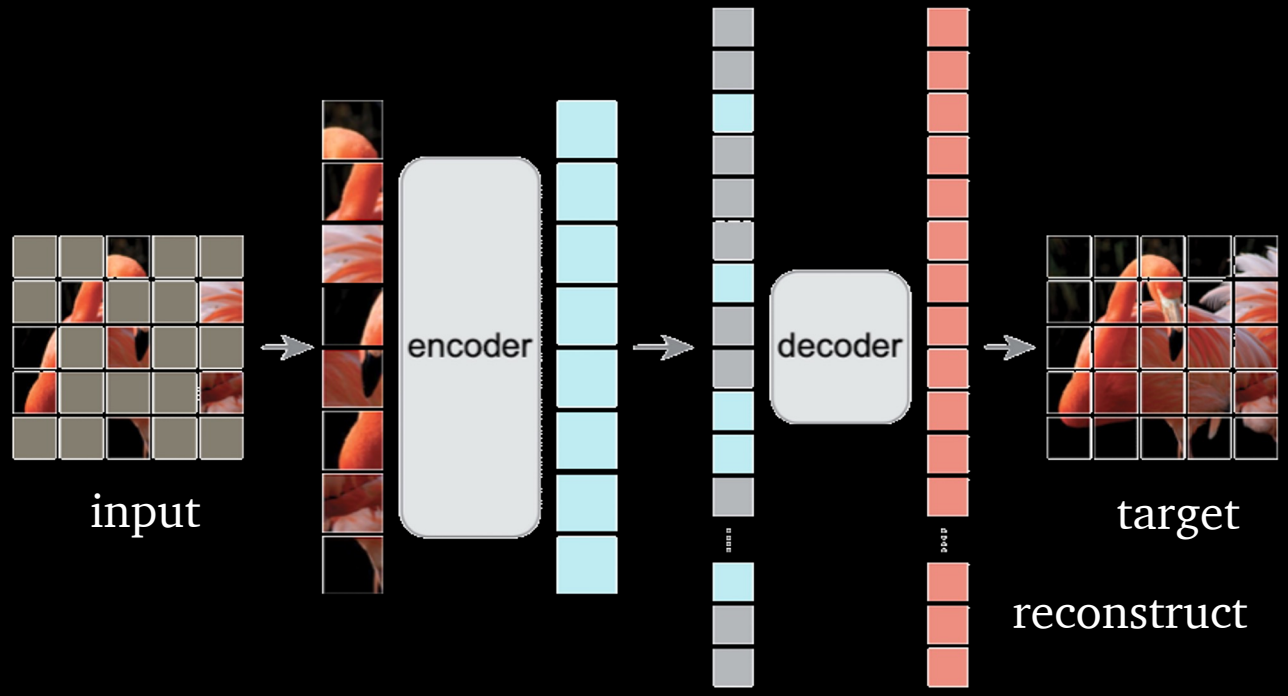
Masked Autoencoders (MAE)

[He et al., CVPR 2022]

Previously, on T4V ...

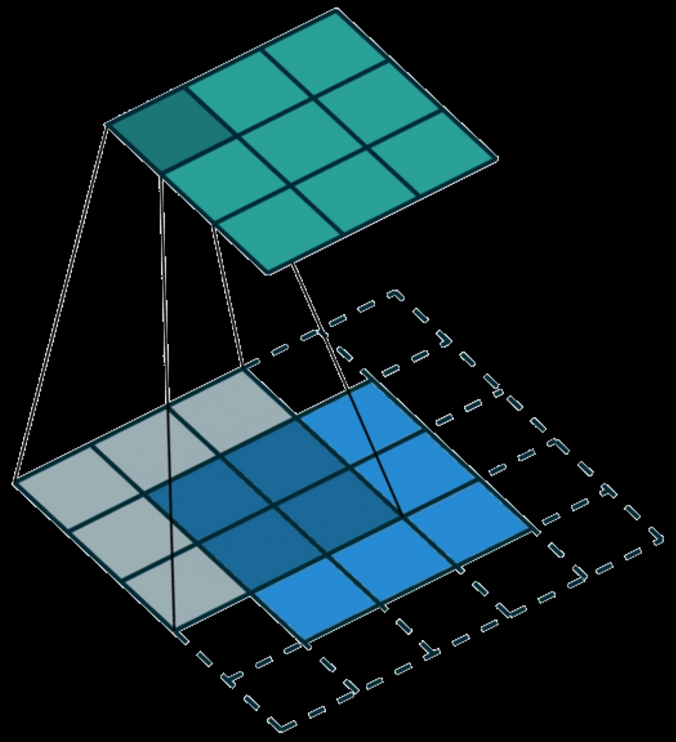
Q: Can we use ConvNeXt as a backbone network for self-supervised learning?

Masked Autoencoders (MAE)

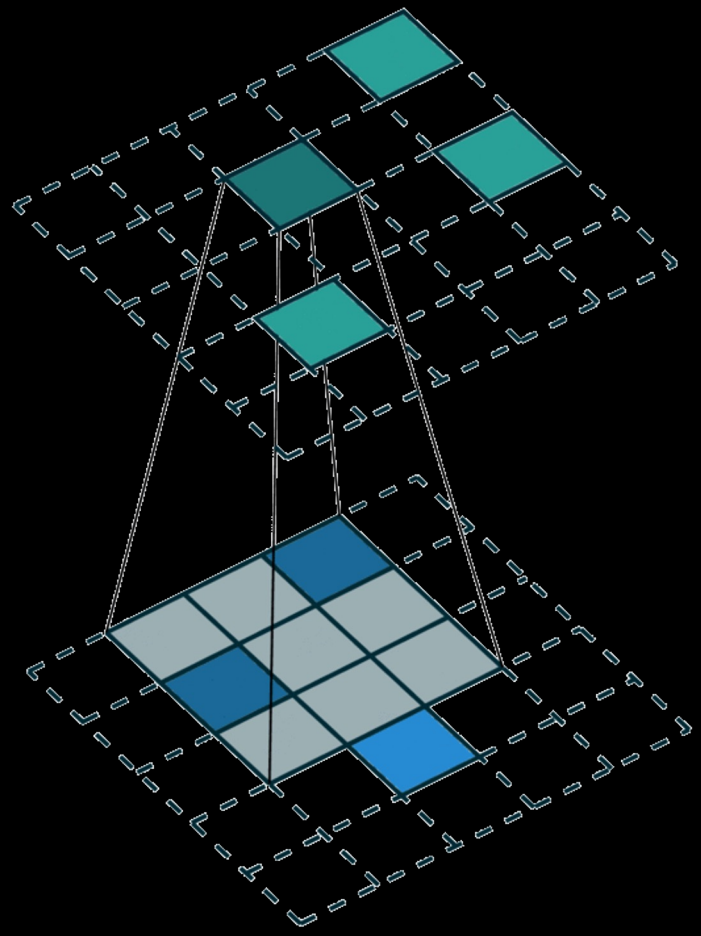


- Asymmetric encoder-decoder architecture.
- Only encoding visible patches.
- How can ConvNet also achieve this?

Dense ConvNets:
Image as 2D pixel grid

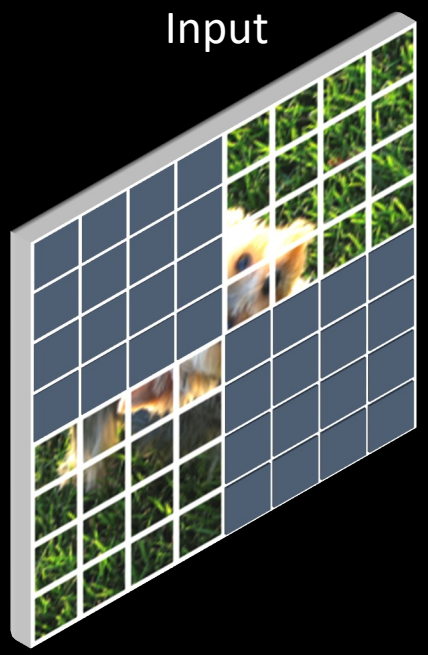


Sparse ConvNets:
Image as point cloud



[MinkowskiEngine, Choy et.al. 2019]

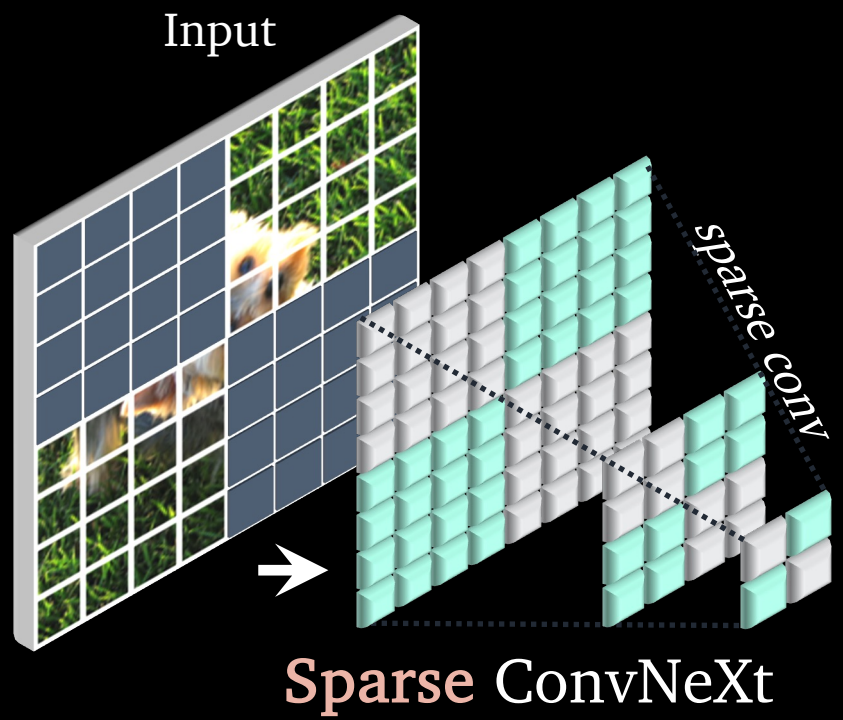
FCMAE: Fully Convolutional Masked Autoencoder



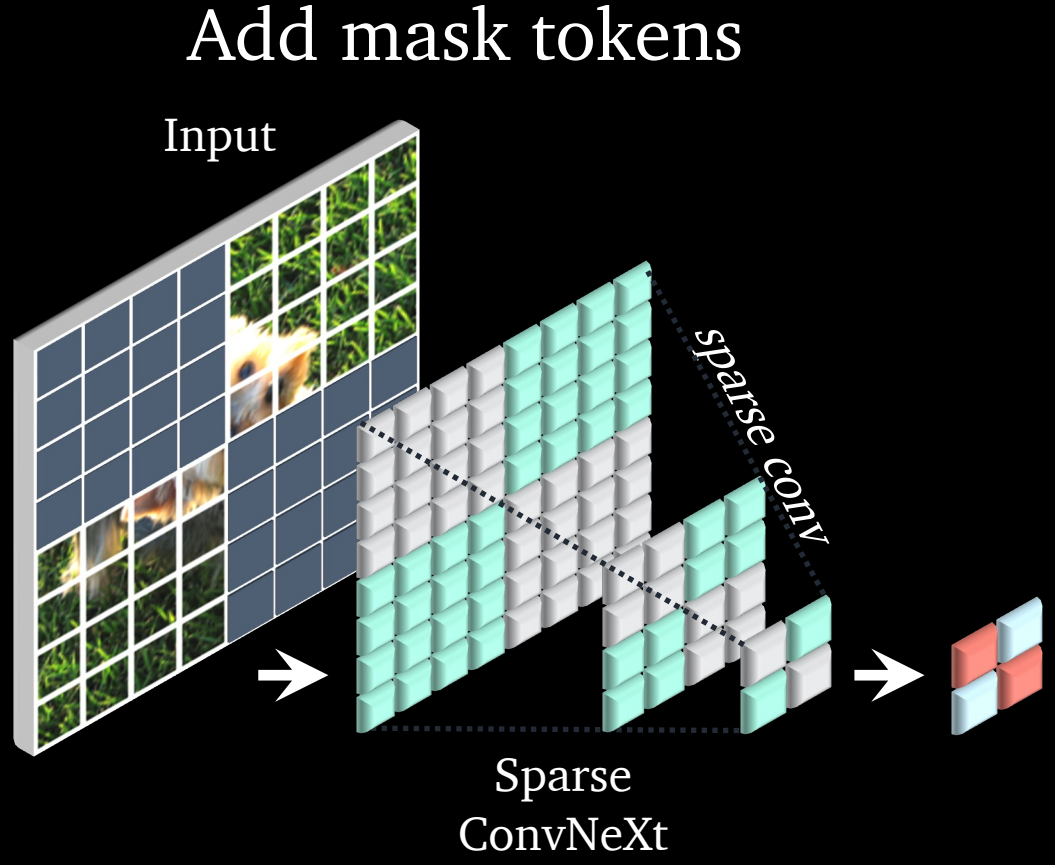
ConvNeXt V2: Co-designing and Scaling ConvNets with Masked Autoencoders, Woo et al., CVPR 2023.

FCMAE: Fully Convolutional Masked Autoencoder

Encode visible patches using **sparse convolutions**

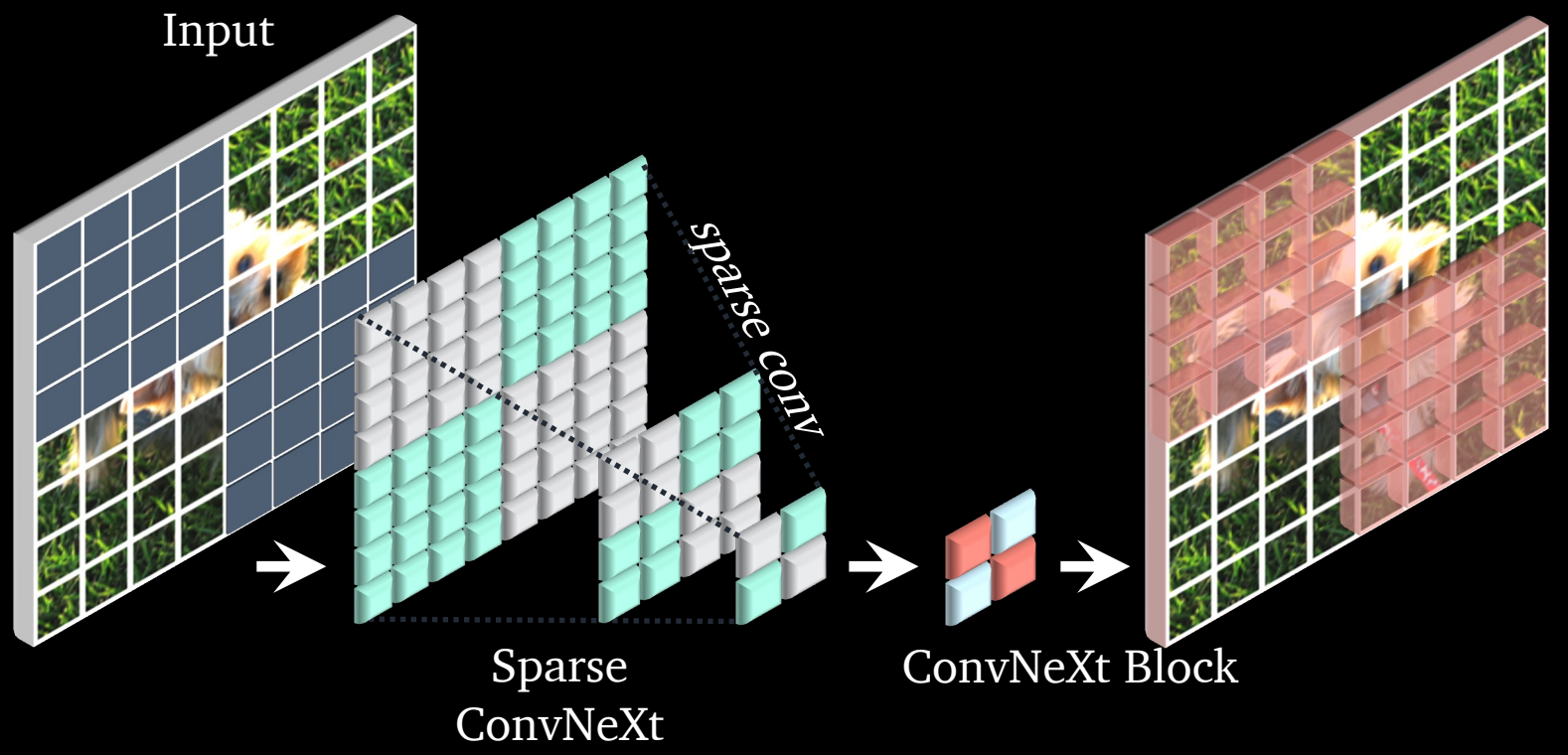


FCMAE: Fully Convolutional Masked Autoencoder



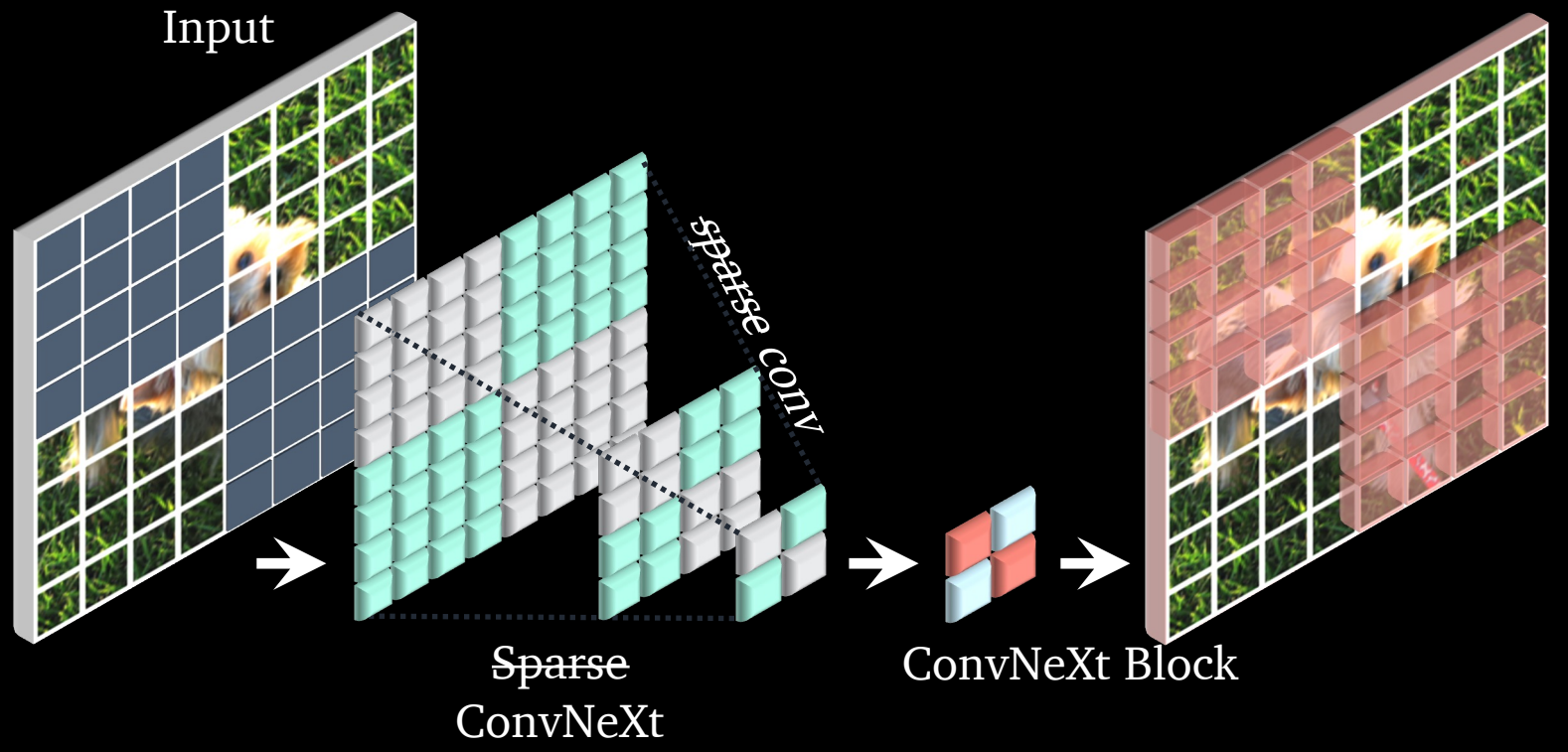
FCMAE: Fully Convolutional Masked Autoencoder

Reconstruct masked patches



FCMAE: Fully Convolutional Masked Autoencoder

After pre-training, the weights are converted back to standard layers without requiring special handling.



The impact of FCMAE

Train & Test Protocol

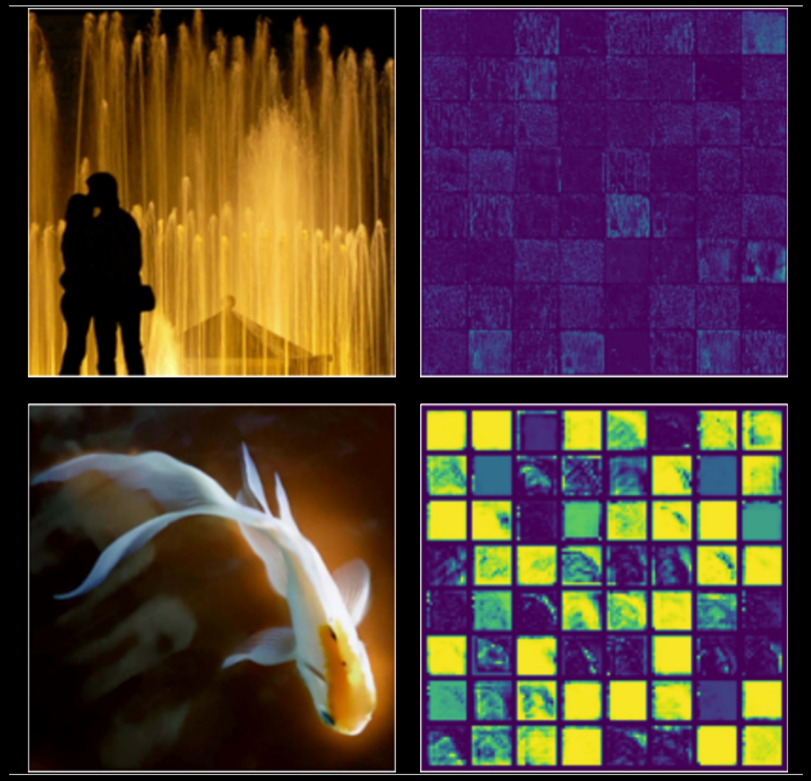
- 800ep Pre-training on ImageNet-1K *train*
- 100ep Fine-tuning on ImageNet-1K *train*
- Test on ImageNet-1K *val*

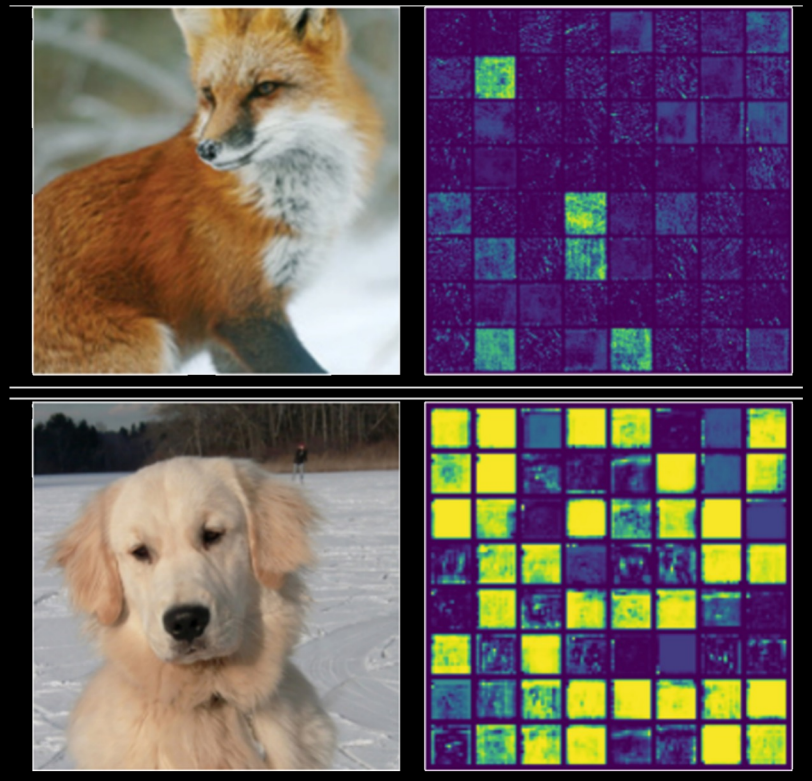
Sup, 100ep	Sup, 300ep. [33]	FCMAE
82.7	83.8	83.7

FCMAE pre-trained ConvNeXt surpasses random initialization but still lags the original 300ep supervised setup.

Feature Collapse

Feature collapse detected in the dimension-expansion MLP layers in FCMAE pre-trained ConvNeXtV1 blocks.

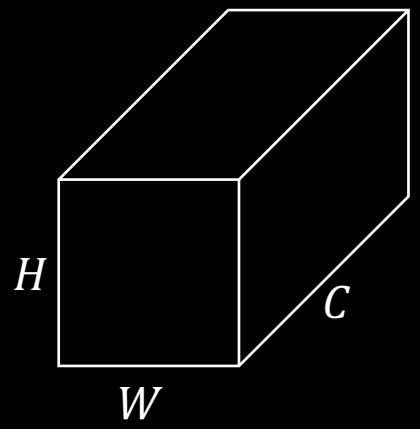




Dead Neurons

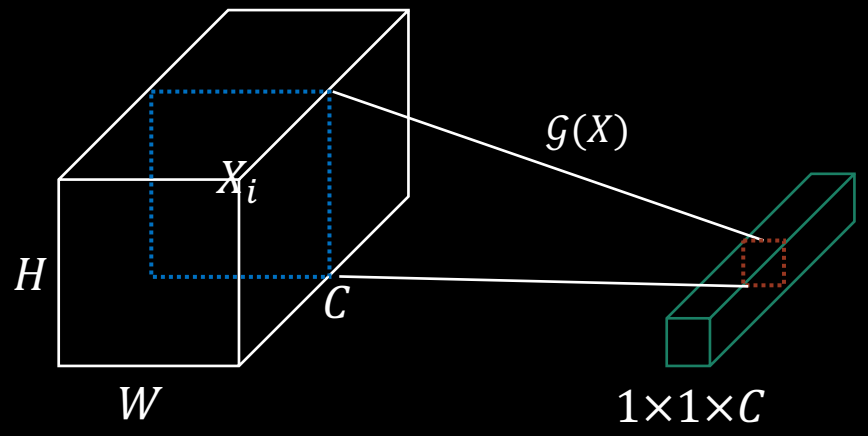
Saturated Neurons

GRN: Global Response Normalization



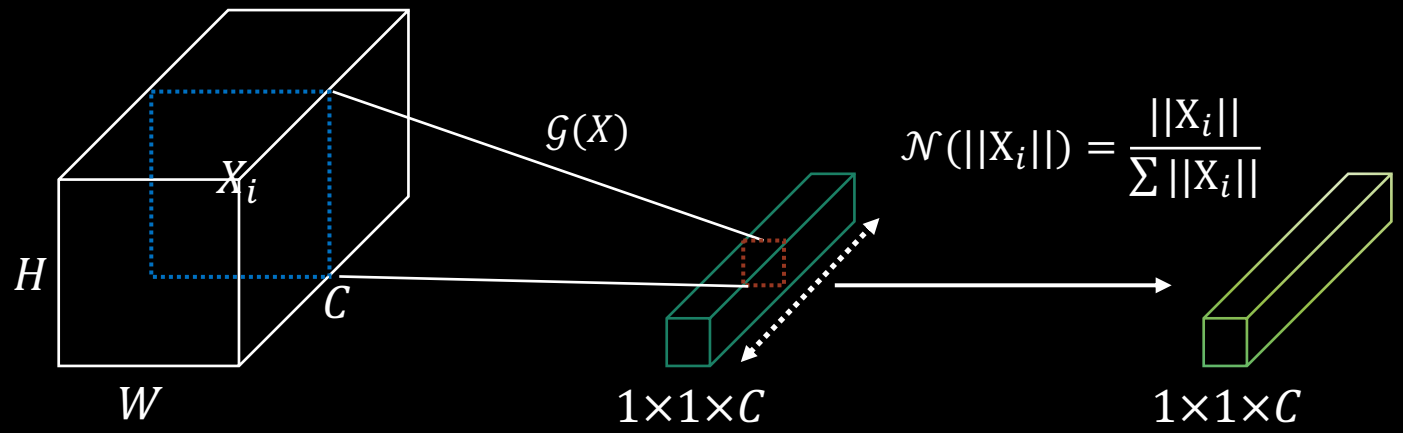
GRN: Global Response Normalization

1) Global feature aggregation



GRN: Global Response Normalization

1) Global feature aggregation
2) Feature normalization



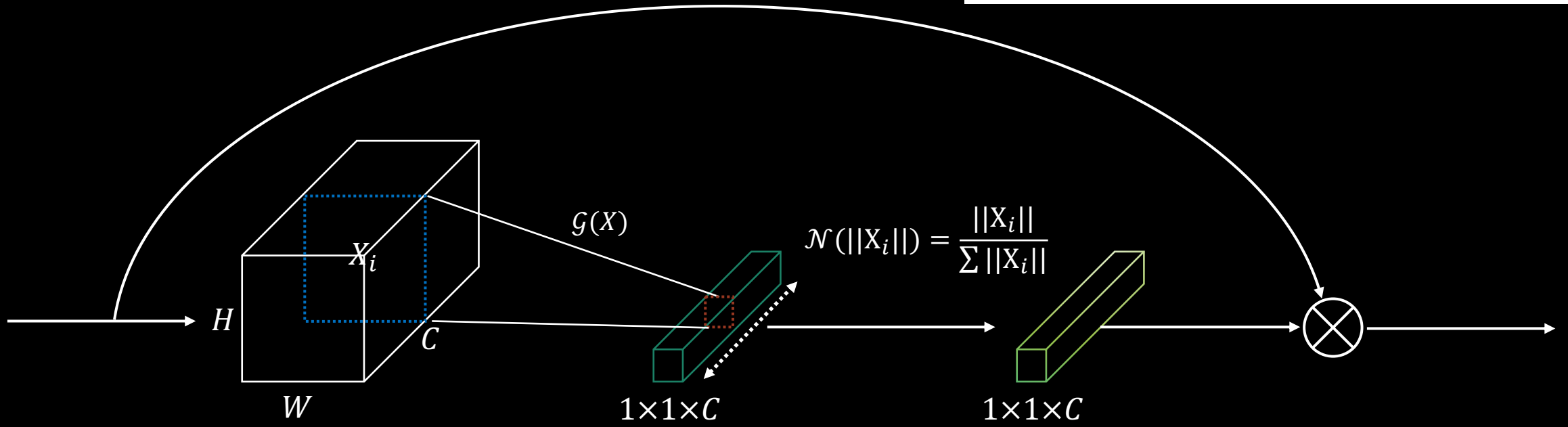
GRN: Global Response Normalization

1) Global feature aggregation
2) Feature normalization
3) Feature calibration

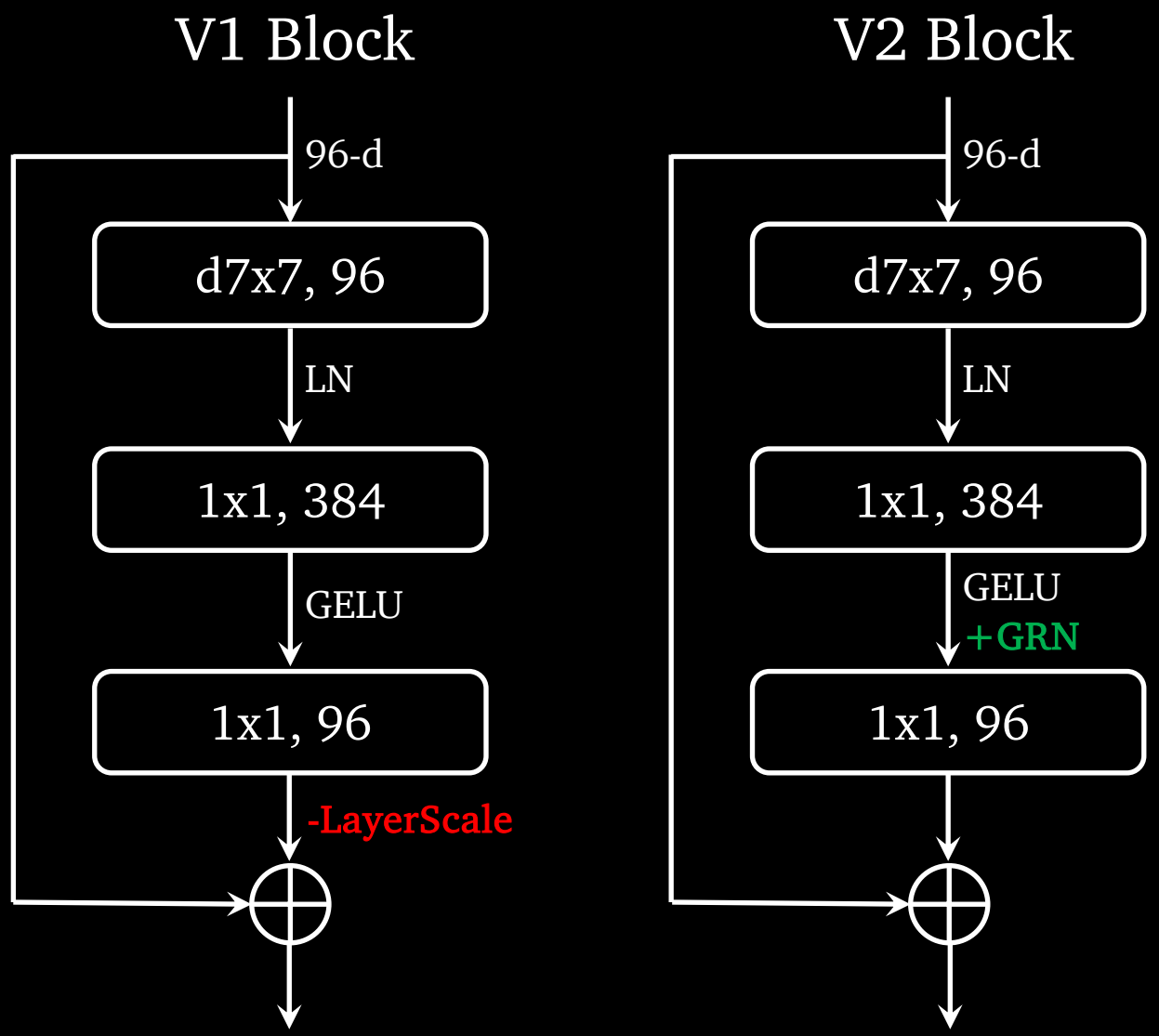
Three lines of code

Algorithm 1 Pseudocode of GRN in a PyTorch-like style.

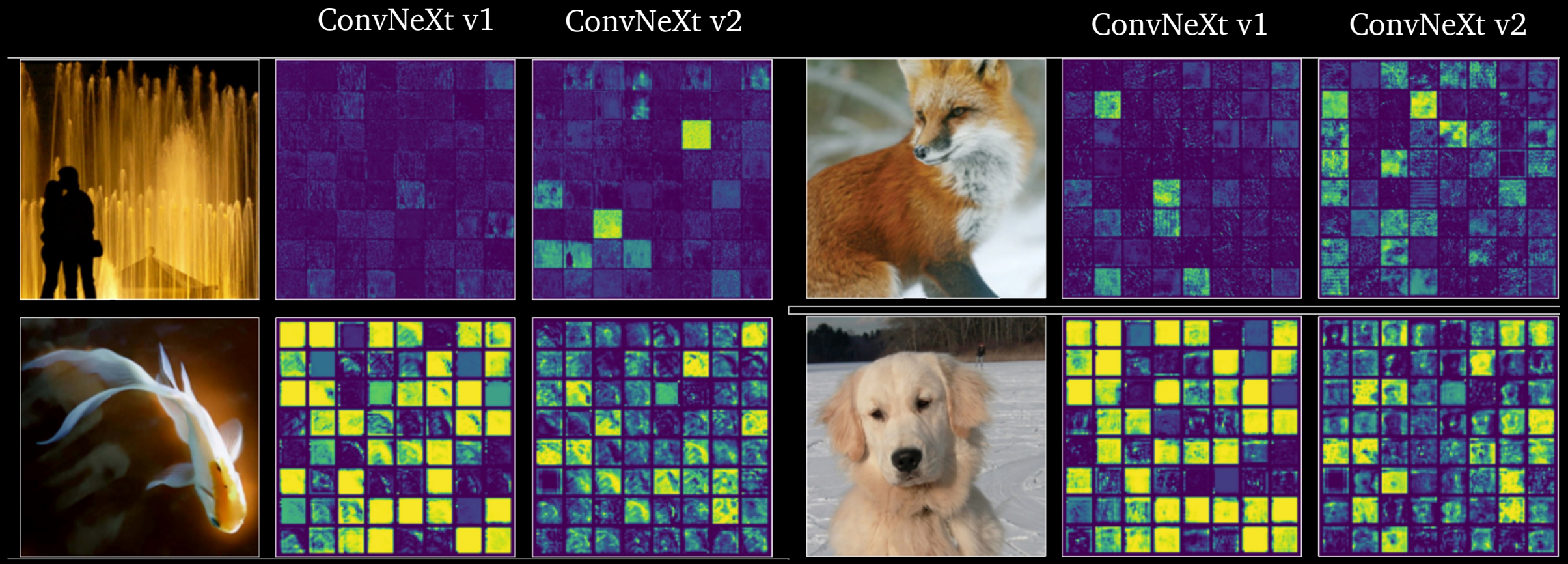
```
# gamma, beta: learnable affine transform parameters
# X: input of shape (N,H,W,C)

gx = torch.norm(X, p=2, dim=(1,2), keepdim=True)
nx = gx / (gx.mean(dim=-1, keepdim=True)+1e-6)
return gamma * (X * nx) + beta + X
```



ConvNeXt V2

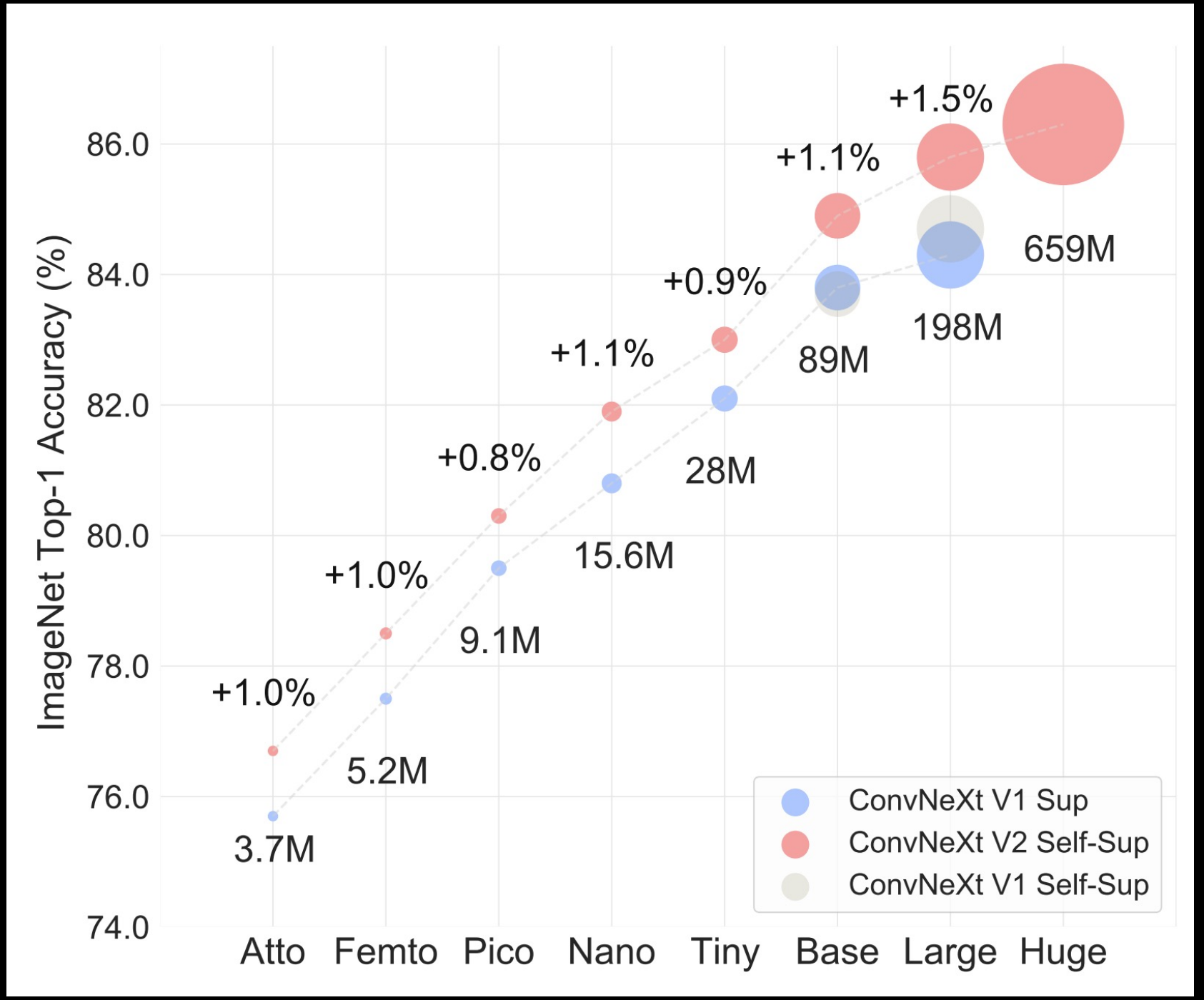

V1 Block
96-d
d7x7, 96
LN
1x1, 384
GELU
1x1, 96
-LayerScale
V2 Block
96-d
d7x7, 96
LN
1x1, 384
GELU
+GRN
1x1, 96

Impact of GRN

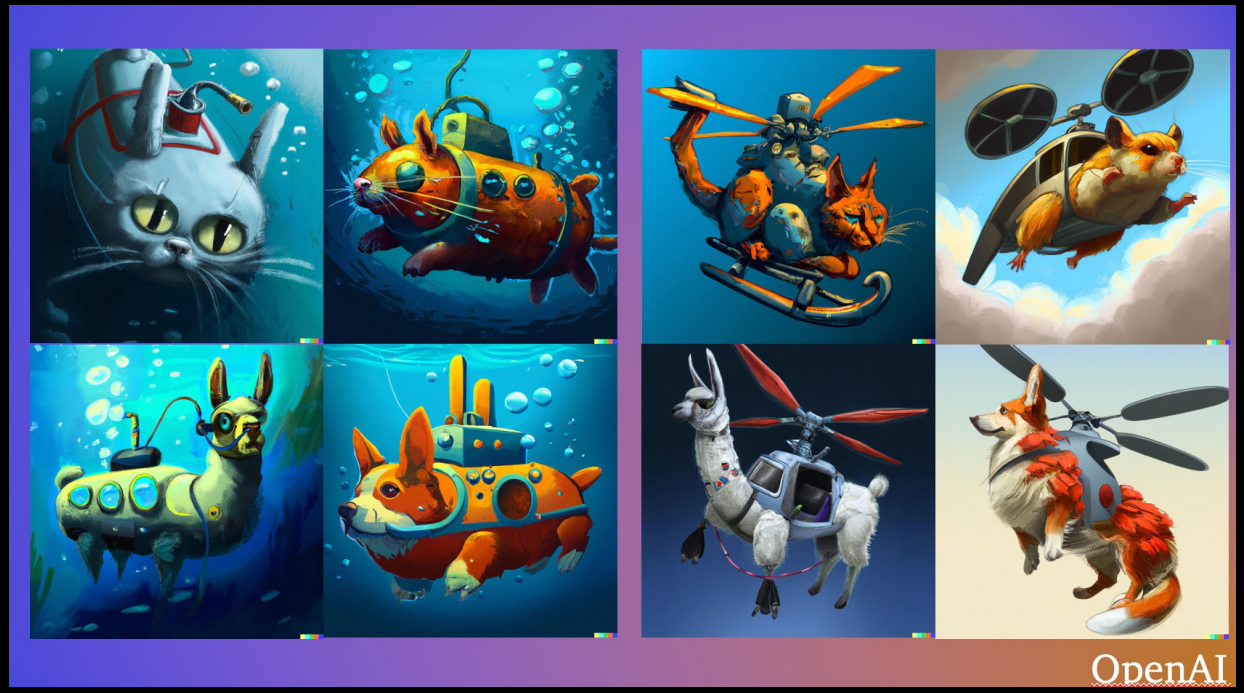


Results

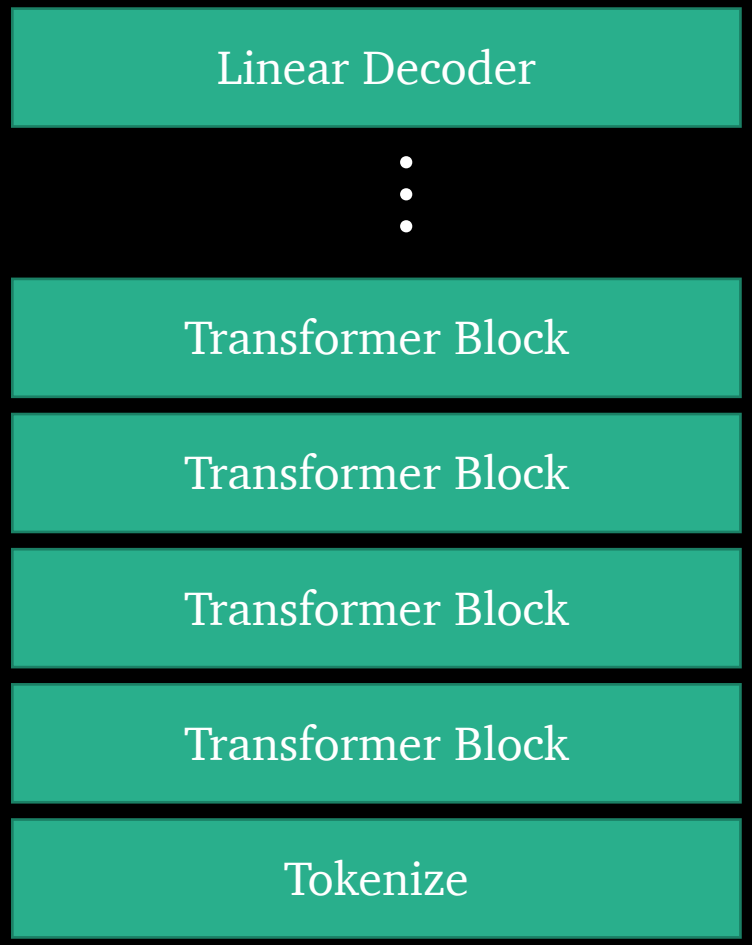
Works for tiny tiny models as well!



ConvNet vs Transformer:
Diffusion Models

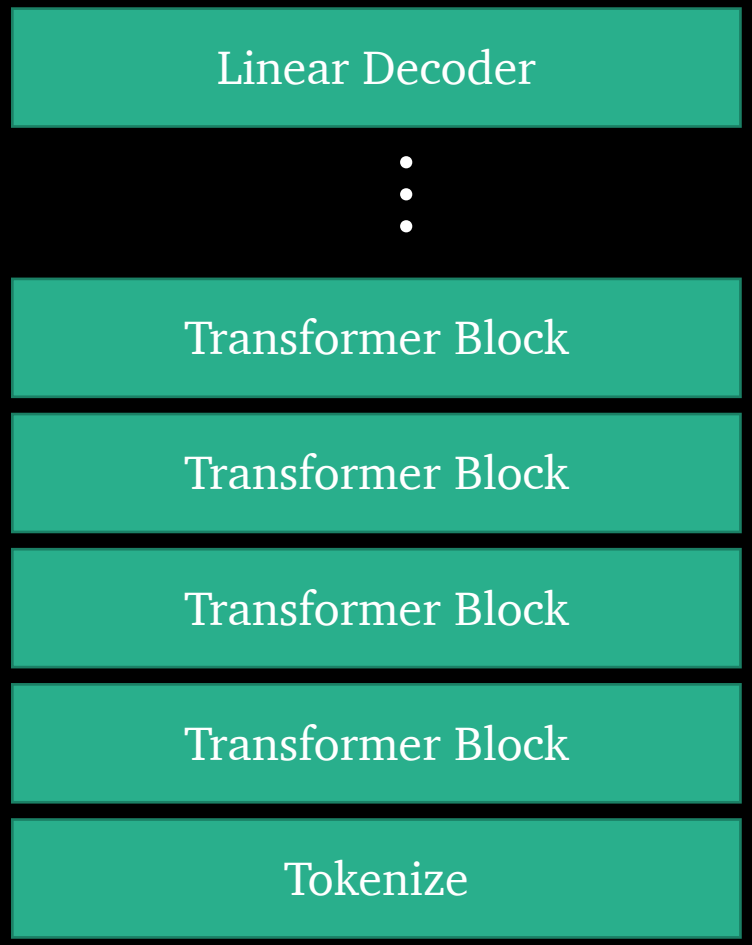


Diffusion models use highly-nonstandard architectures



GPT-3 / Chinchilla / …

Diffusion models use highly-nonstandard architectures



GPT-3 / Chinchilla / …

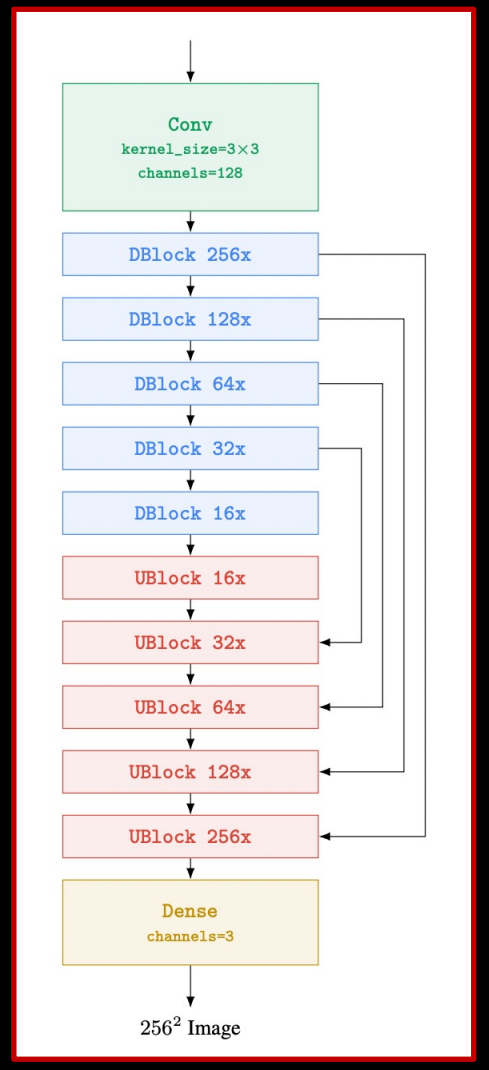


Imagen U-Nets

Diffusion models use highly-nonstandard architectures

Linear Decoder

⋮

Transformer Block

Transformer Block

Transformer Block

Transformer Block

Tokenize

"A dog walked over to the barn…"

GPT-3 / Chinchilla / …

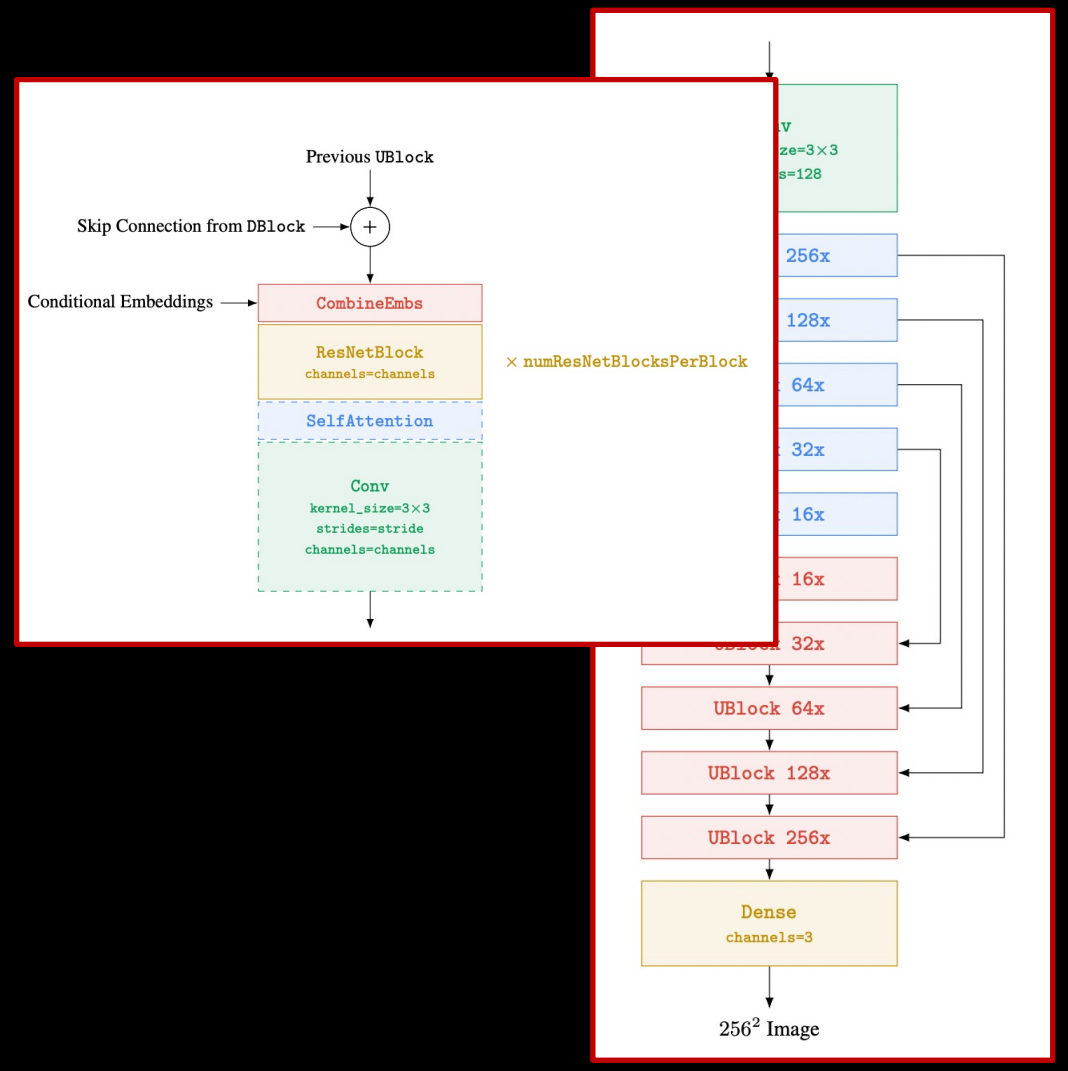


Imagen U-Nets

Diffusion models use highly-nonstandard architectures

Linear Decoder

⋮

Transformer Block

Transformer Block

Transformer Block

Transformer Block

Tokenize

"A dog walked over to the barn…"

GPT-3 / Chinchilla / …

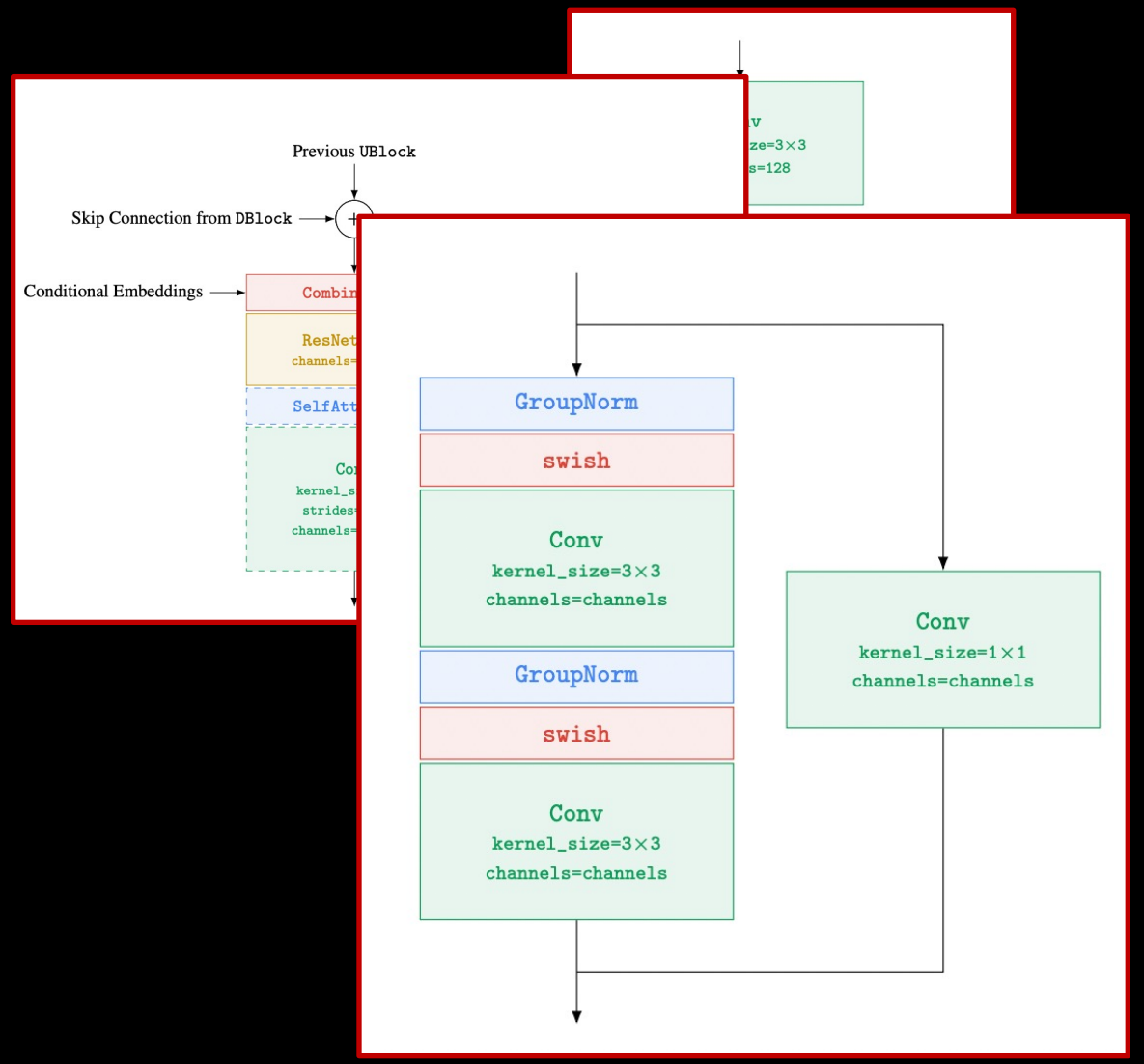


Imagen U-Nets

Diffusion models use highly-nonstandard architectures

Linear Decoder

⋮

Transformer Block

Transformer Block

Transformer Block

Transformer Block

Tokenize

"A dog walked over to the barn…"

GPT-3 / Chinchilla / …

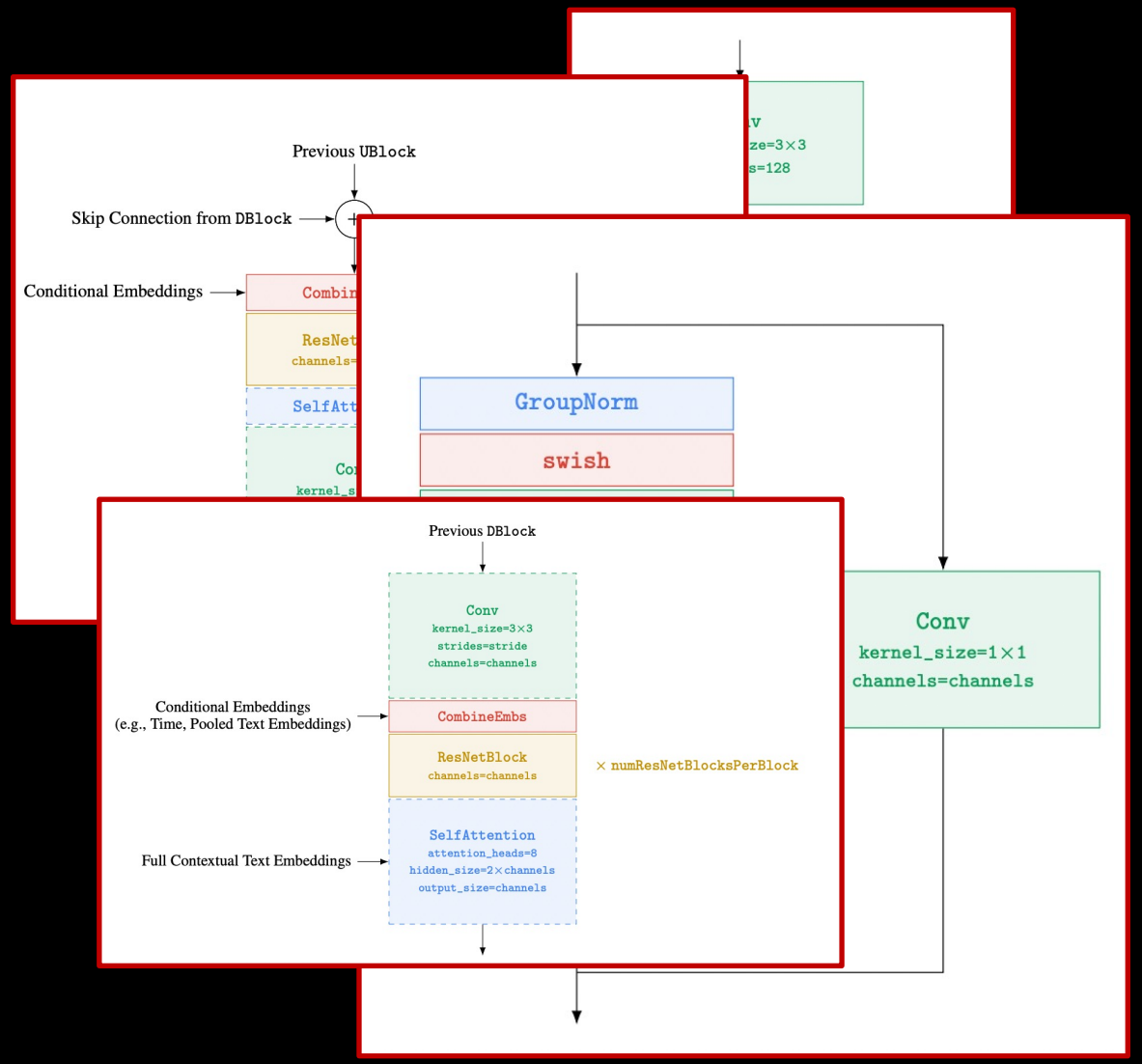
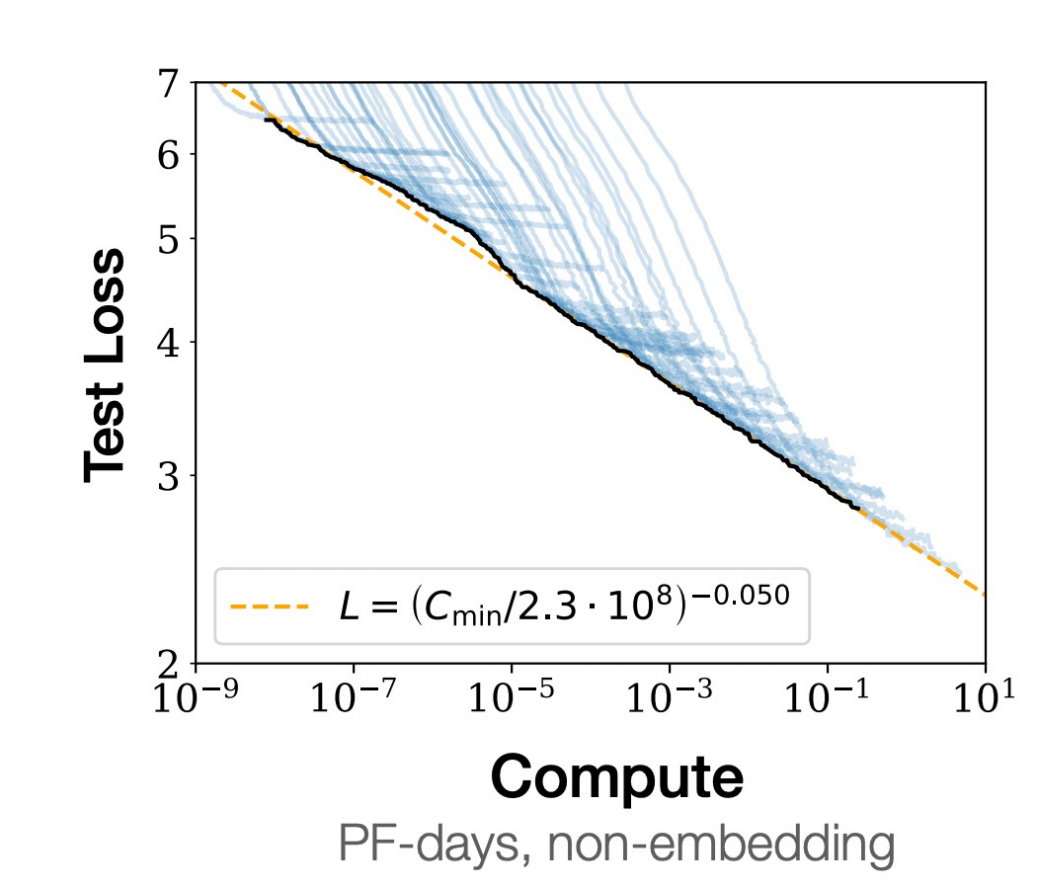
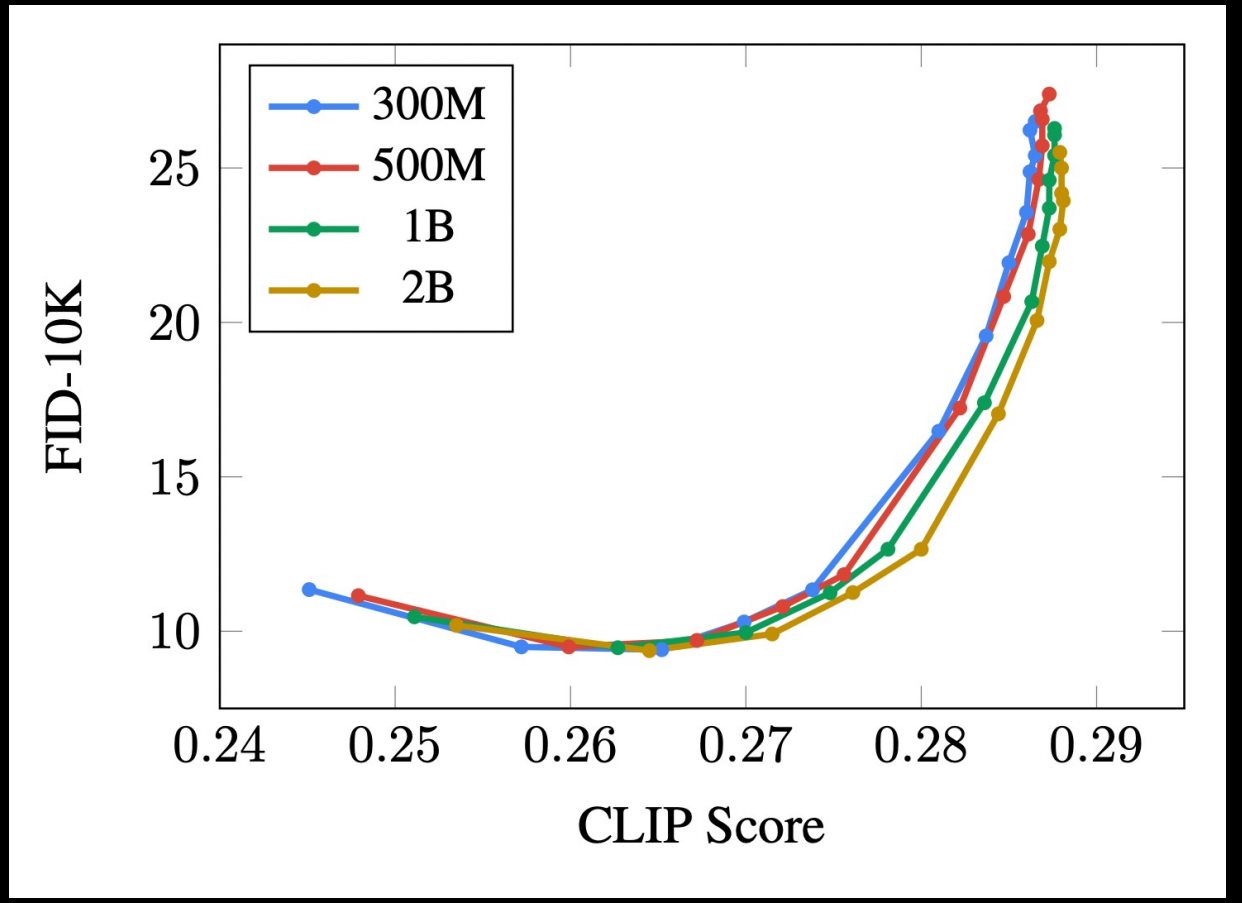


Imagen U-Nets

Language models are having all the fun



Scaling of the GPT Family for NLG



Scaling of the U-Net Family for Diffusion

Imagen. Google 2022.

Scaling Laws for Neural Language Models. OpenAI 2020.

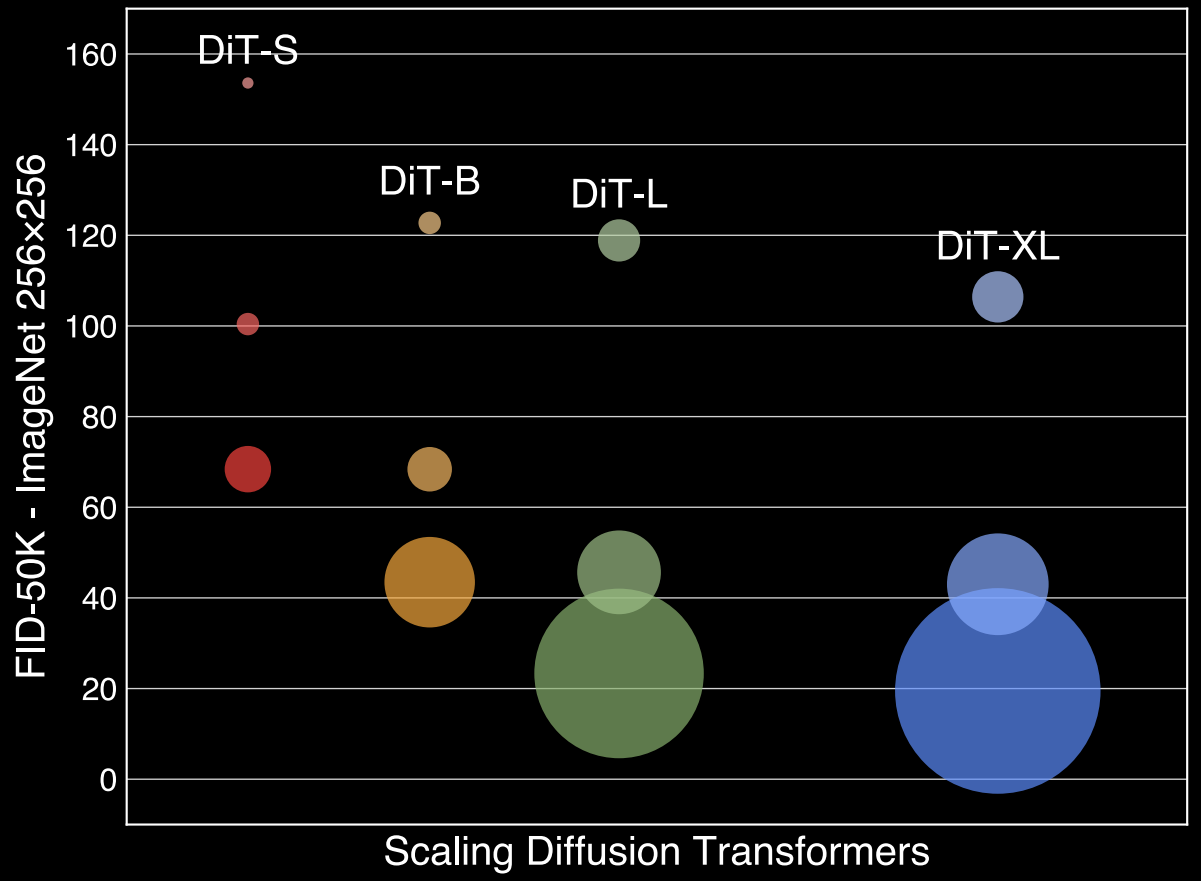
“Diffusion models are about denoising images and so a convolutional inductive bias that encodes locality is critical!! Furthermore, the U-Net architecture contains many components that are fundamentally important like long-range skip connections and an encoder-decoder structure!”

Can diffusion models benefit from simple, scalable architectures?

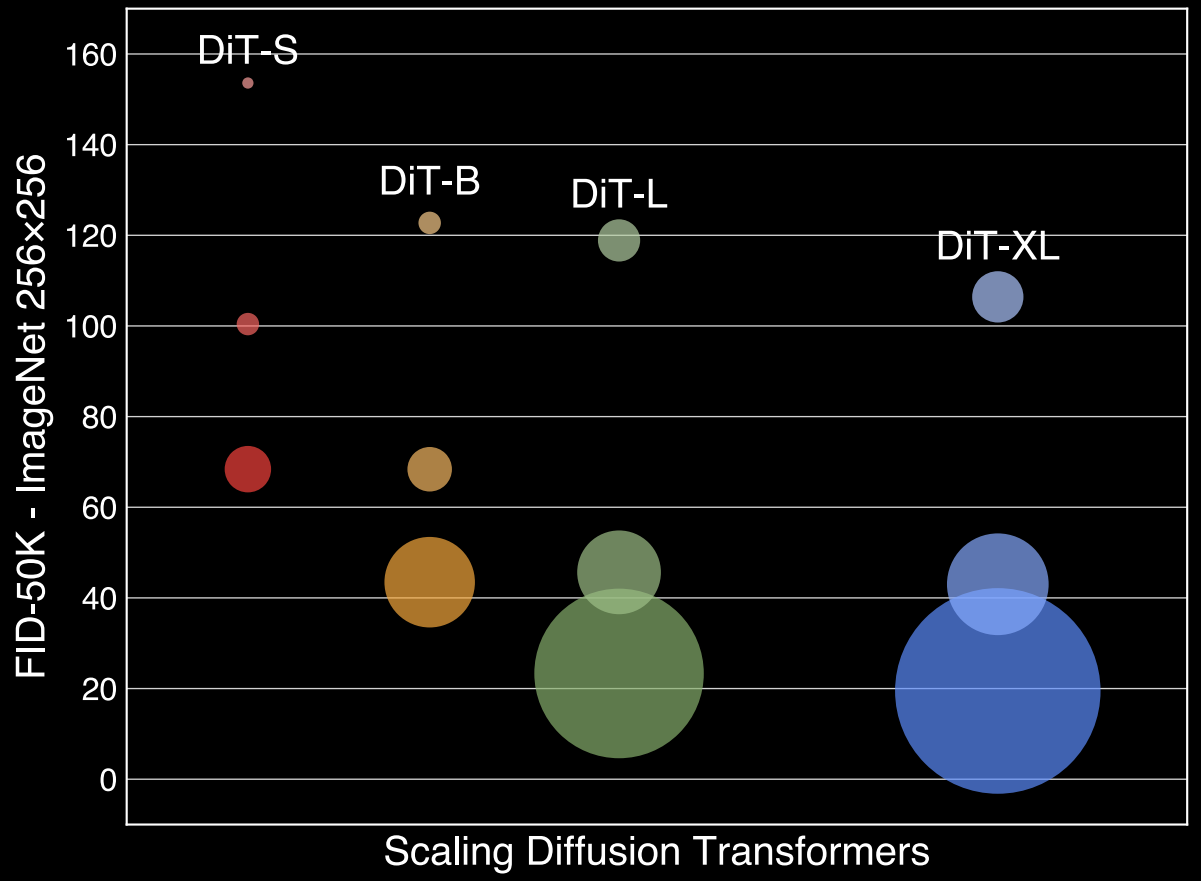
Scalable Diffusion Models with Transformers, Peebles and Xie, arXiv 2023.

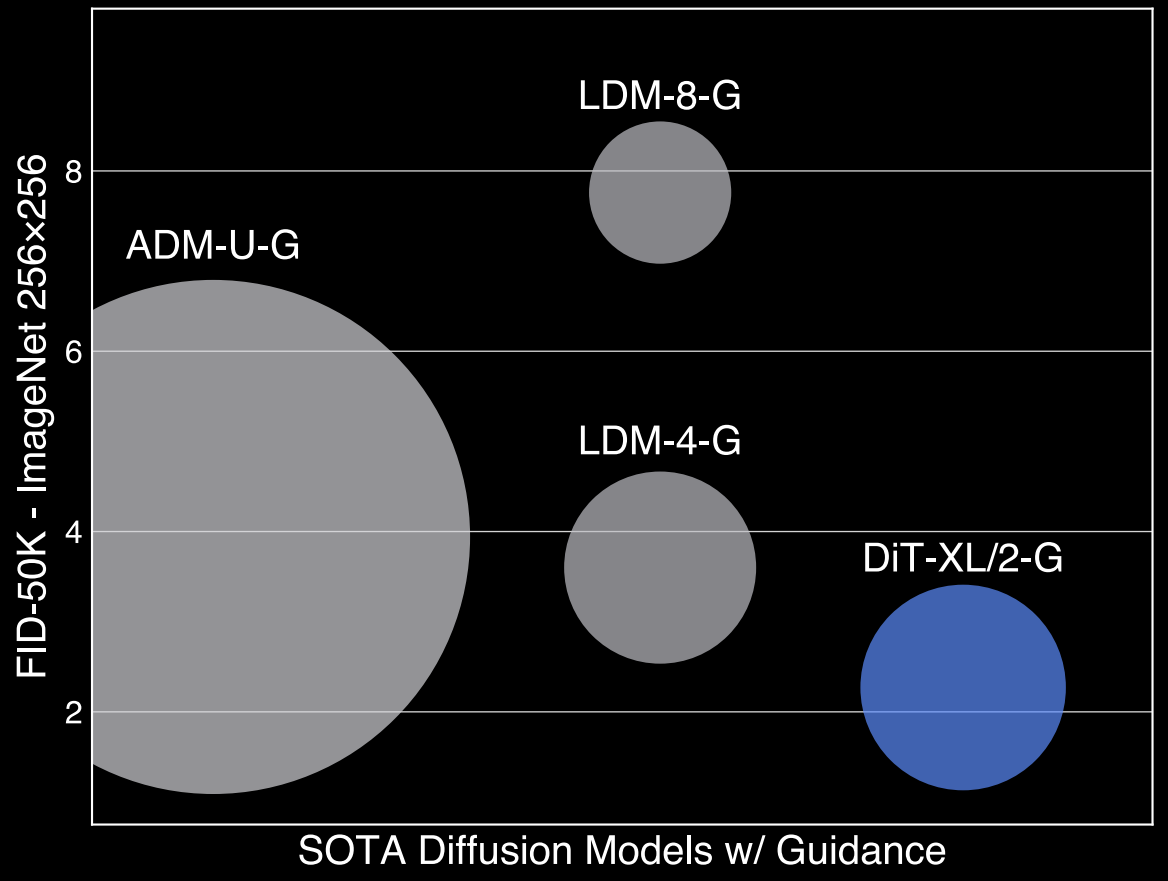
DiT: A New Class of Diffusion Models

DiT: A New Class of Diffusion Models

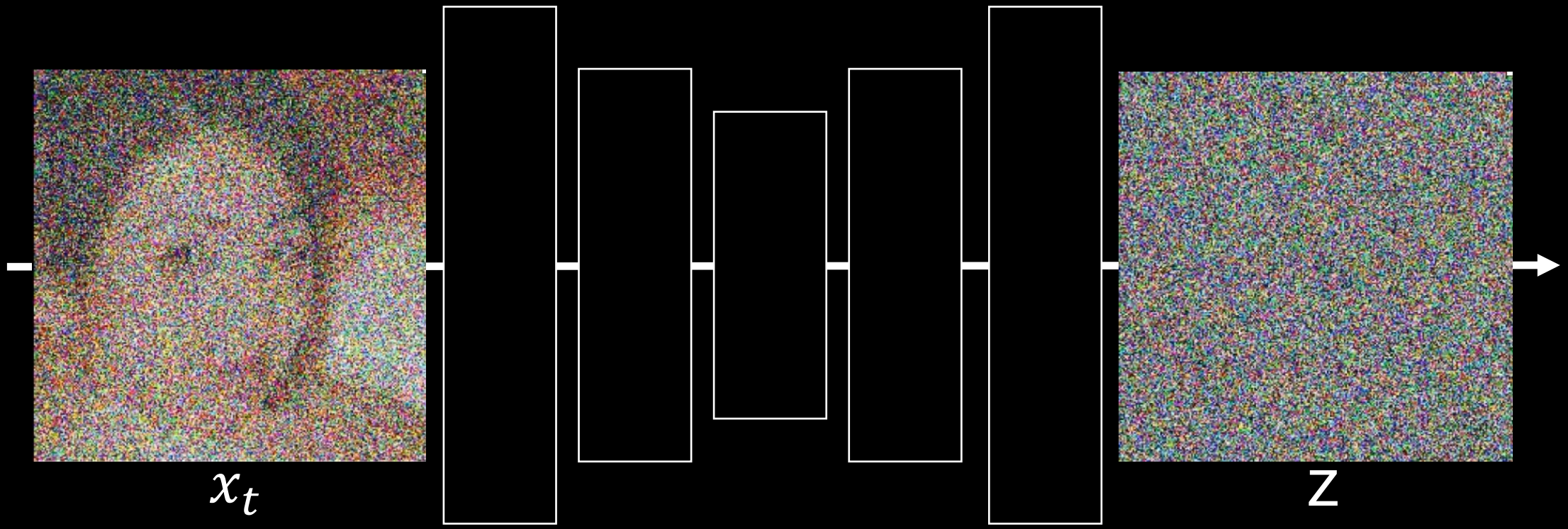


DiT: A New Class of Diffusion Models



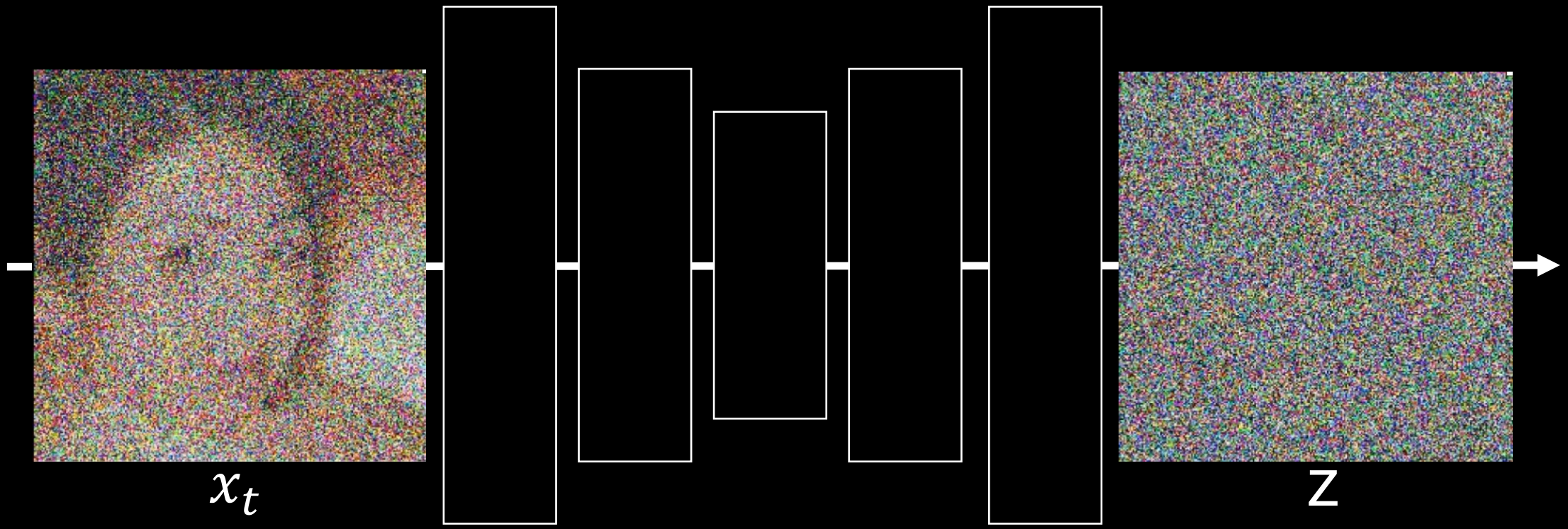


Diffusion Basics



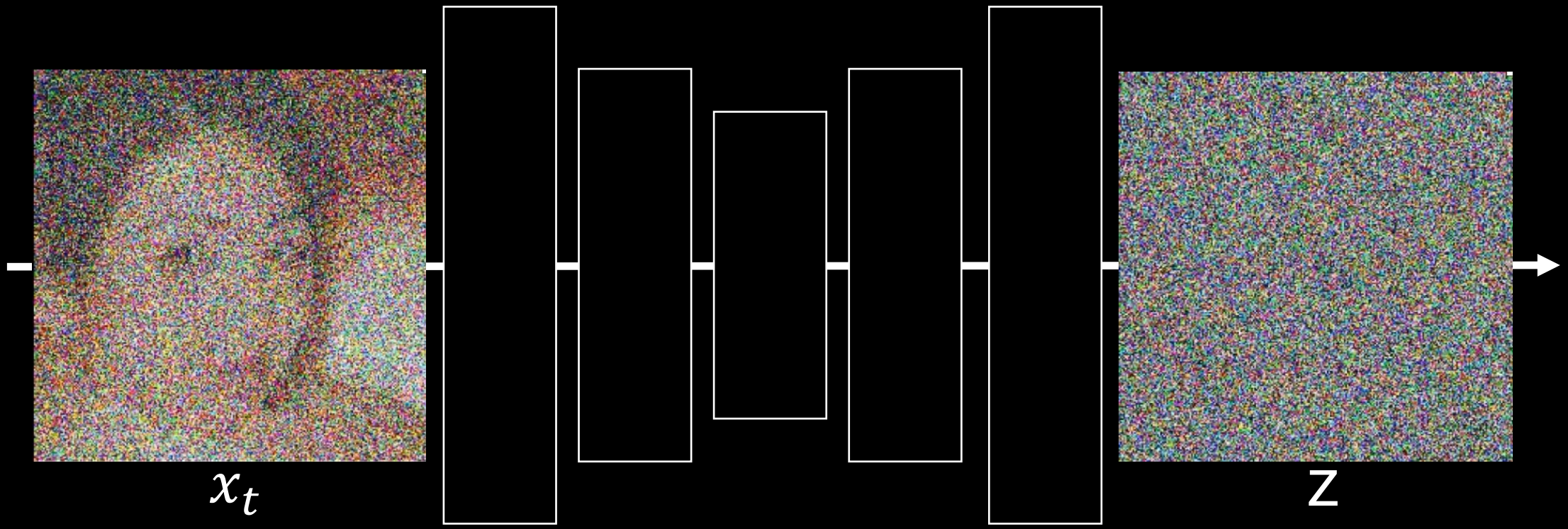
- Sample a real image x_0

Diffusion Basics



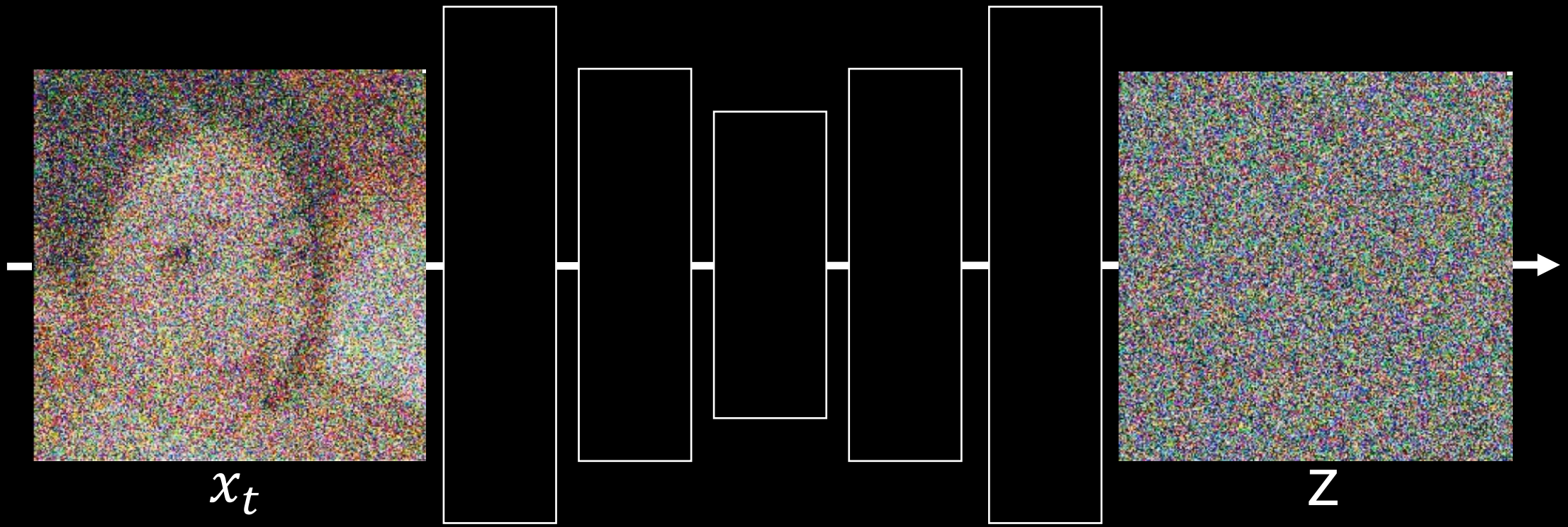
- Sample a real image x_0
- Sample noise z ~ N(0,I)

Diffusion Basics



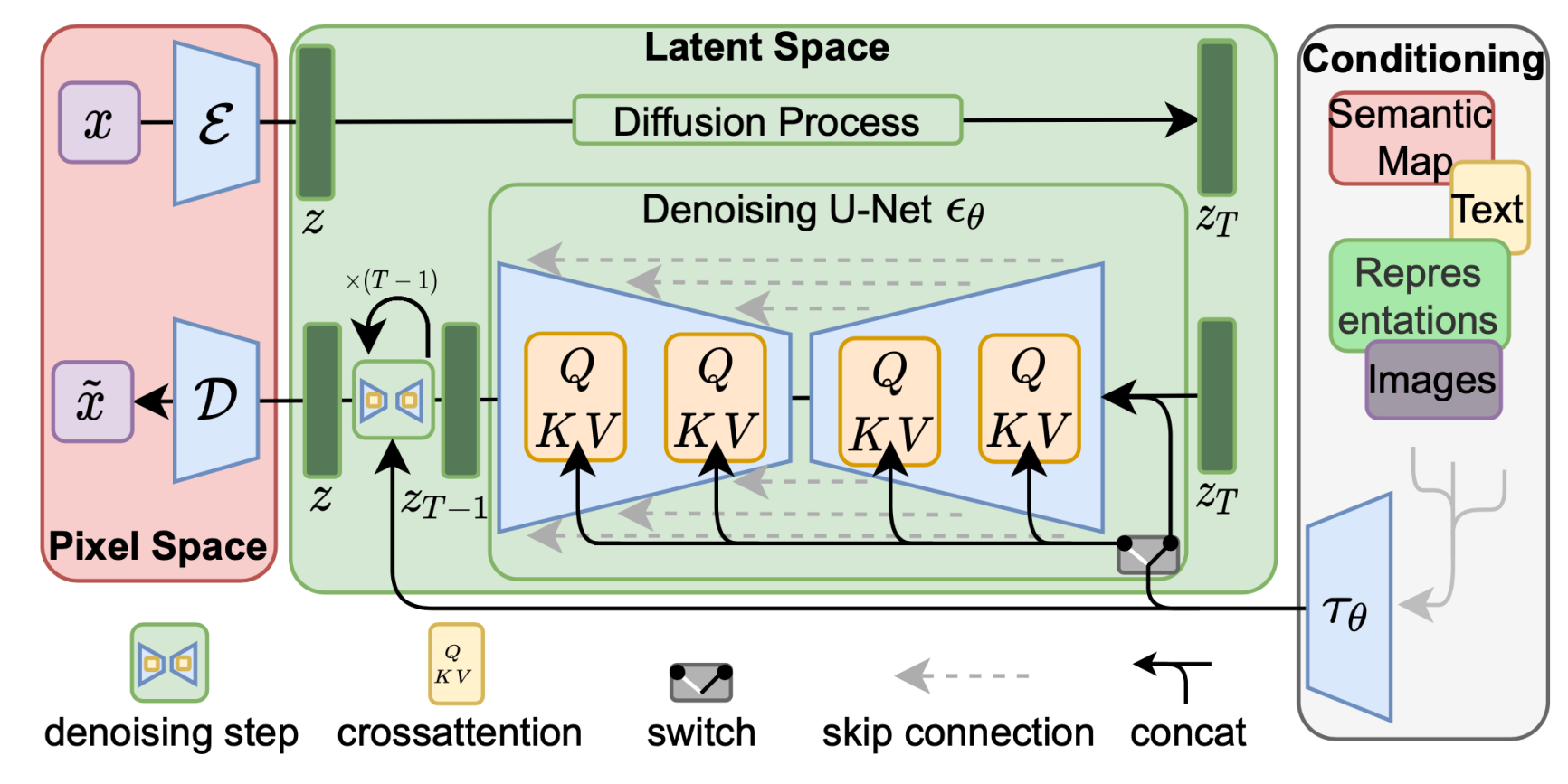
- Sample a real image x_0
- Sample noise z ~ N(0,I)
- Corrupt x_0 with z: call it x_t

Diffusion Basics



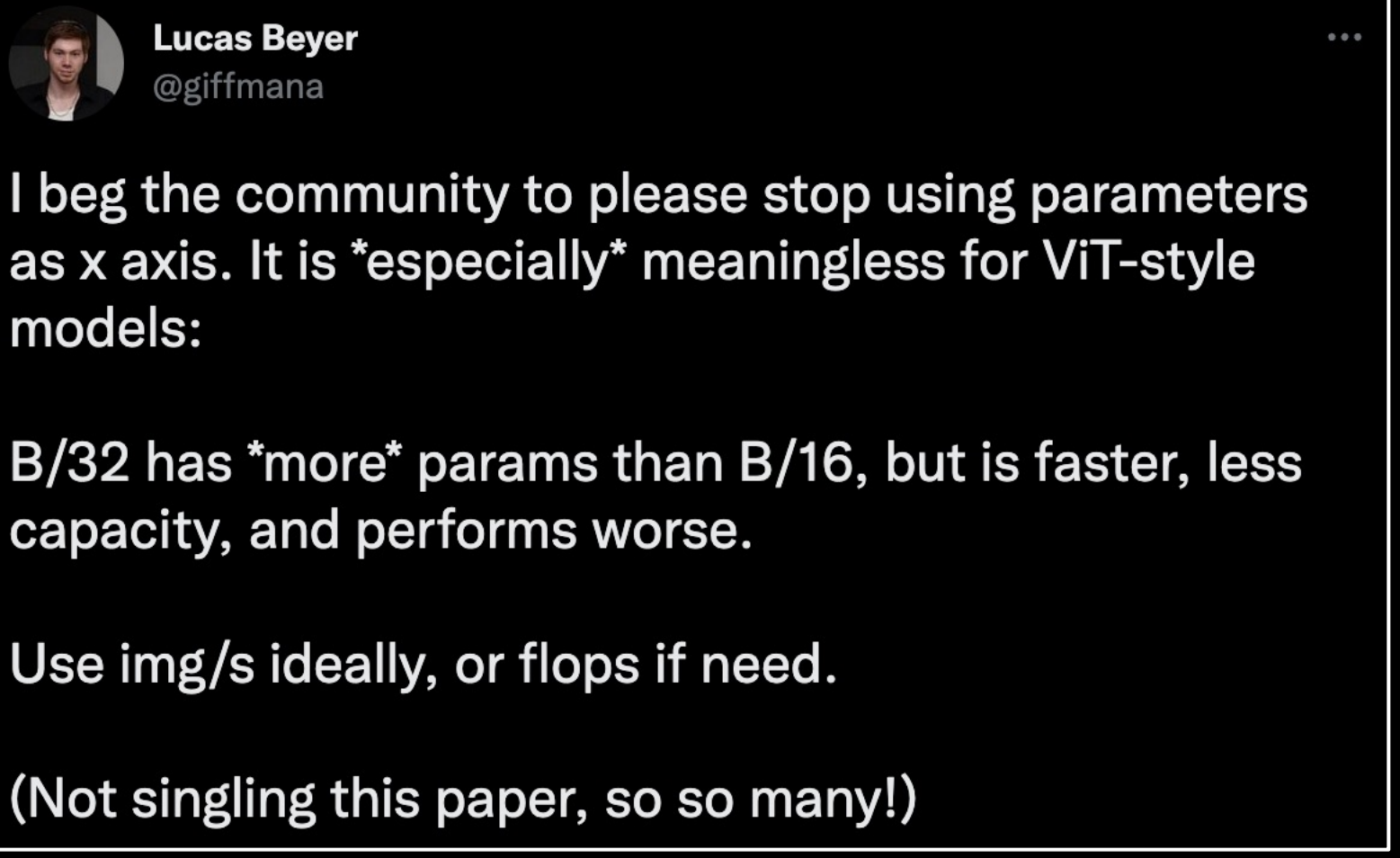
- Sample a real image x_0
- Sample noise z ~ N(0,I)
- Corrupt x_0 with z: call it x_t
- Train a neural net to predict z from x_t

Latent Diffusion Models



High-Resolution Image Synthesis with Latent Diffusion Models. Rombach et al., 202

Important to Scaling: Measuring Model Complexity



Measuring Model Complexity

Parameter counts = bad

FLOPs = much better (*but not perfect)

DiT Design

Latent DiTs: Training

Input image

t

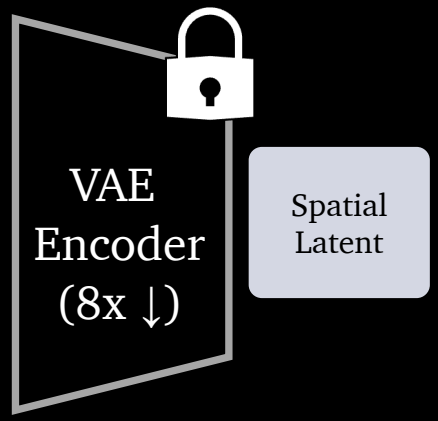
Forward process timestep

c

Conditional info (label, text, etc.)

Latent DiTs: Training

Input image



Spatial
Latent

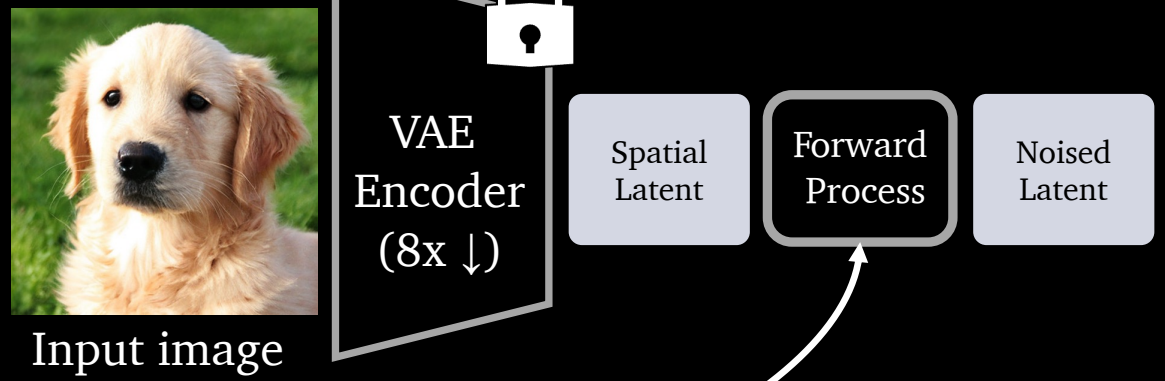
t

Forward process
timestep

c

Conditional info
(label, text, etc.)

Latent DiTs: Training



Latent DiTs: Training

Input image

VAE Encoder (8x ↓)

Spatial Latent

Forward Process

Noised Latent

DiT

t

Forward process timestep

c

Conditional info (label, text, etc.)

Latent DiTs: Training

Input image

VAE Encoder (8x ↓)

Spatial Latent

Forward Process

Noised Latent

DiT

Noise Prediction

Covariance

t

Forward process timestep

c

Conditional info (label, text, etc.)

Latent DiTs: Training

Input image

VAE Encoder (8x ↓)

Spatial Latent

Forward Process

Noised Latent

DiT

Noise Prediction

Covariance

Training losses

t

Forward process timestep

c

Conditional info (label, text, etc.)

Latent DiTs: Training

Input image

VAE Encoder (8x ↓)

Spatial Latent

Forward Process

Noised Latent

DiT

Noise Prediction

Covariance

Training losses

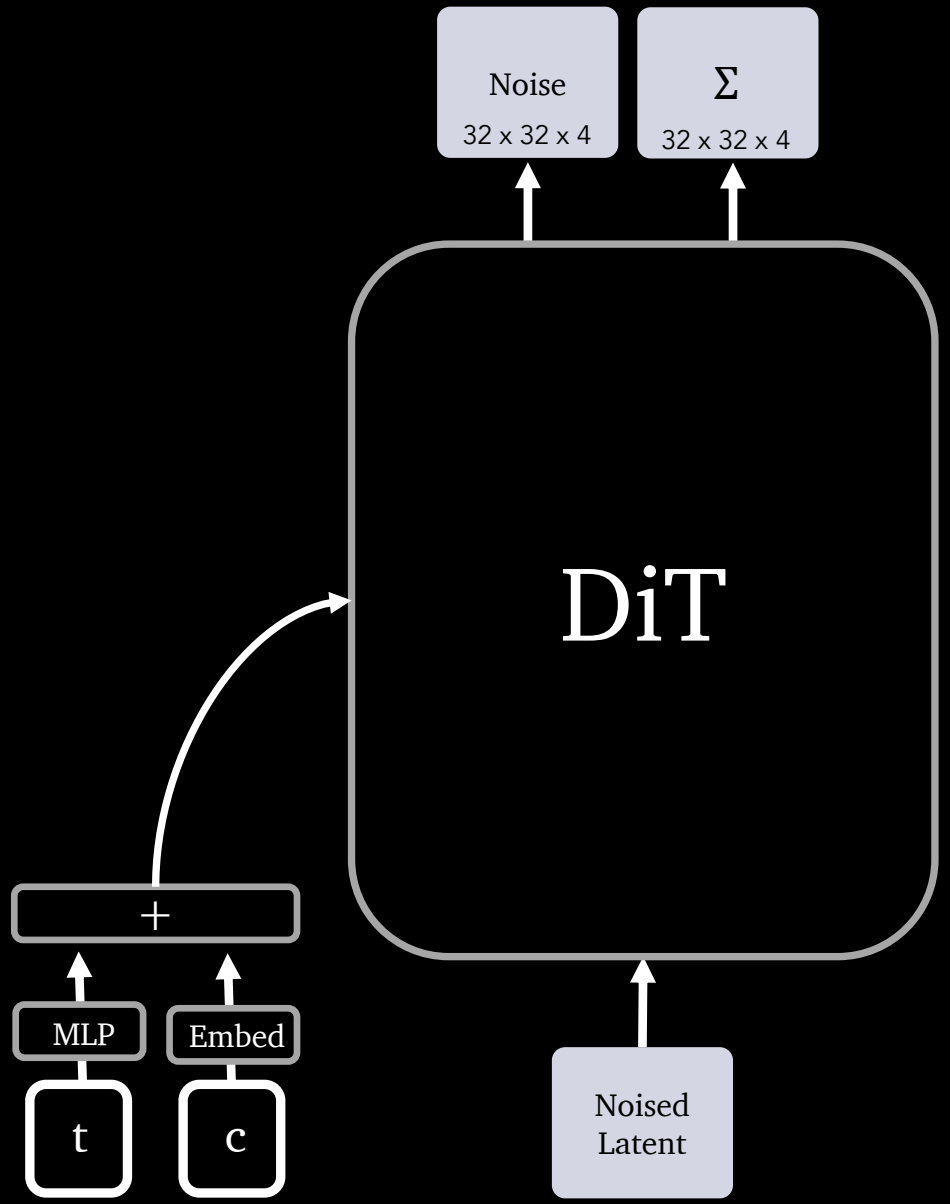
t

Forward process timestep

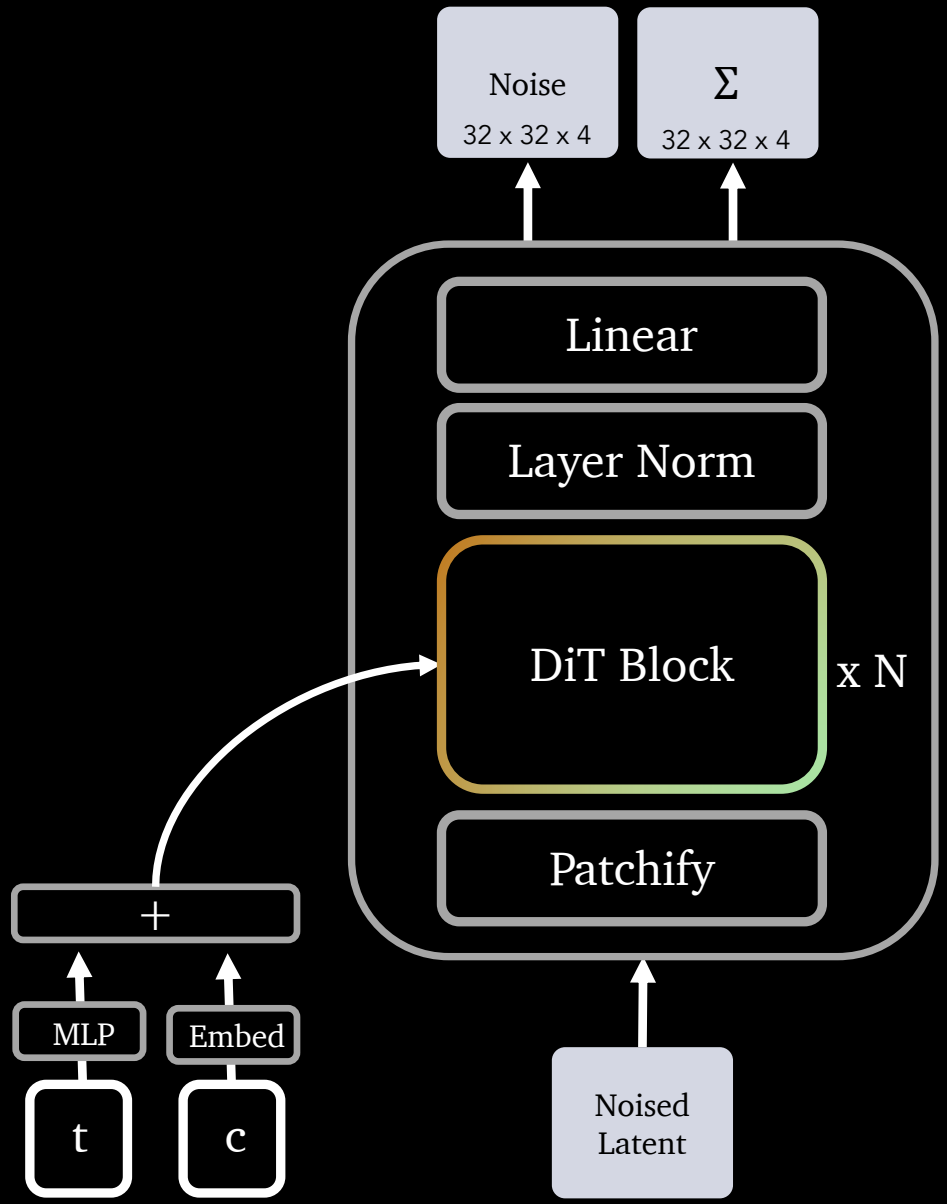
c

Conditional info (label, text, etc.)

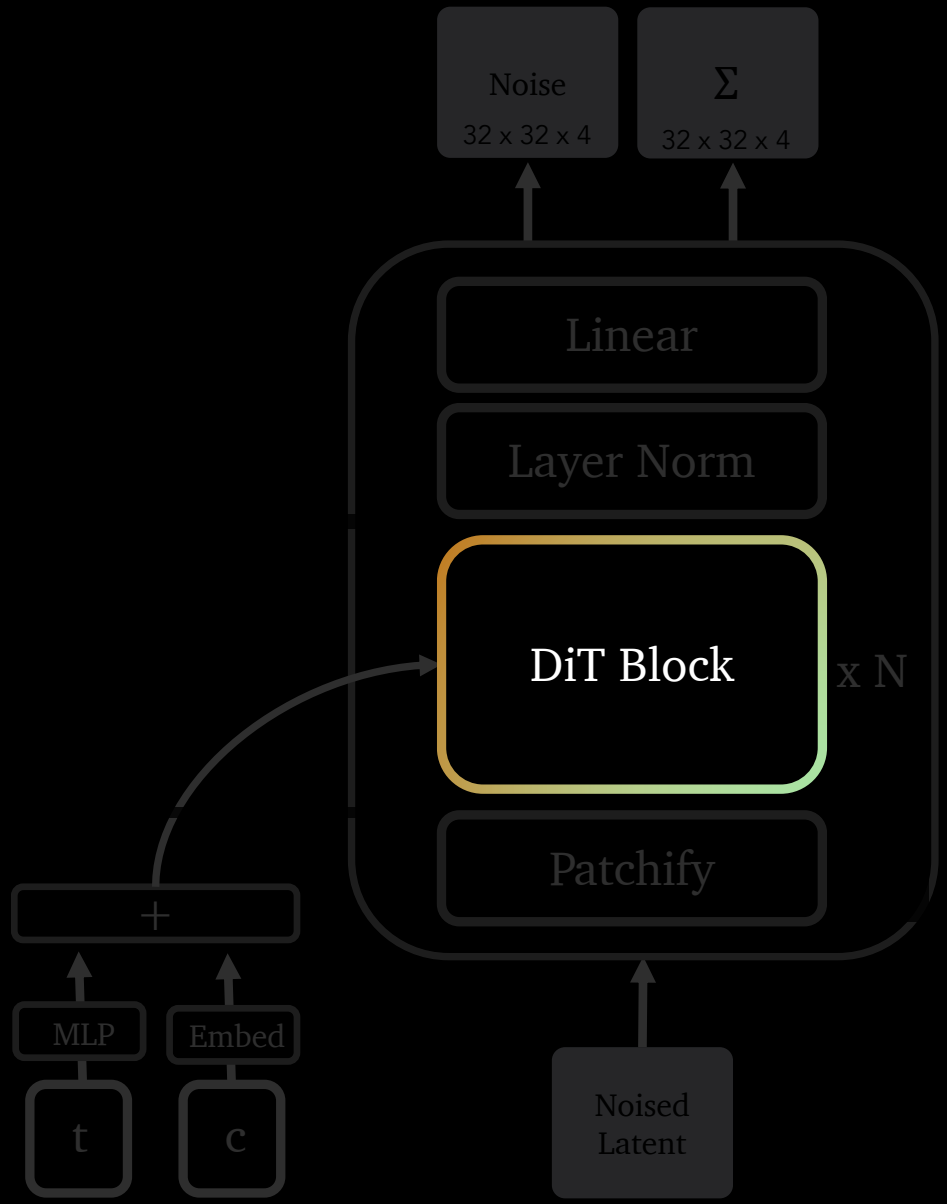
DiT Design Space: Transformer Block



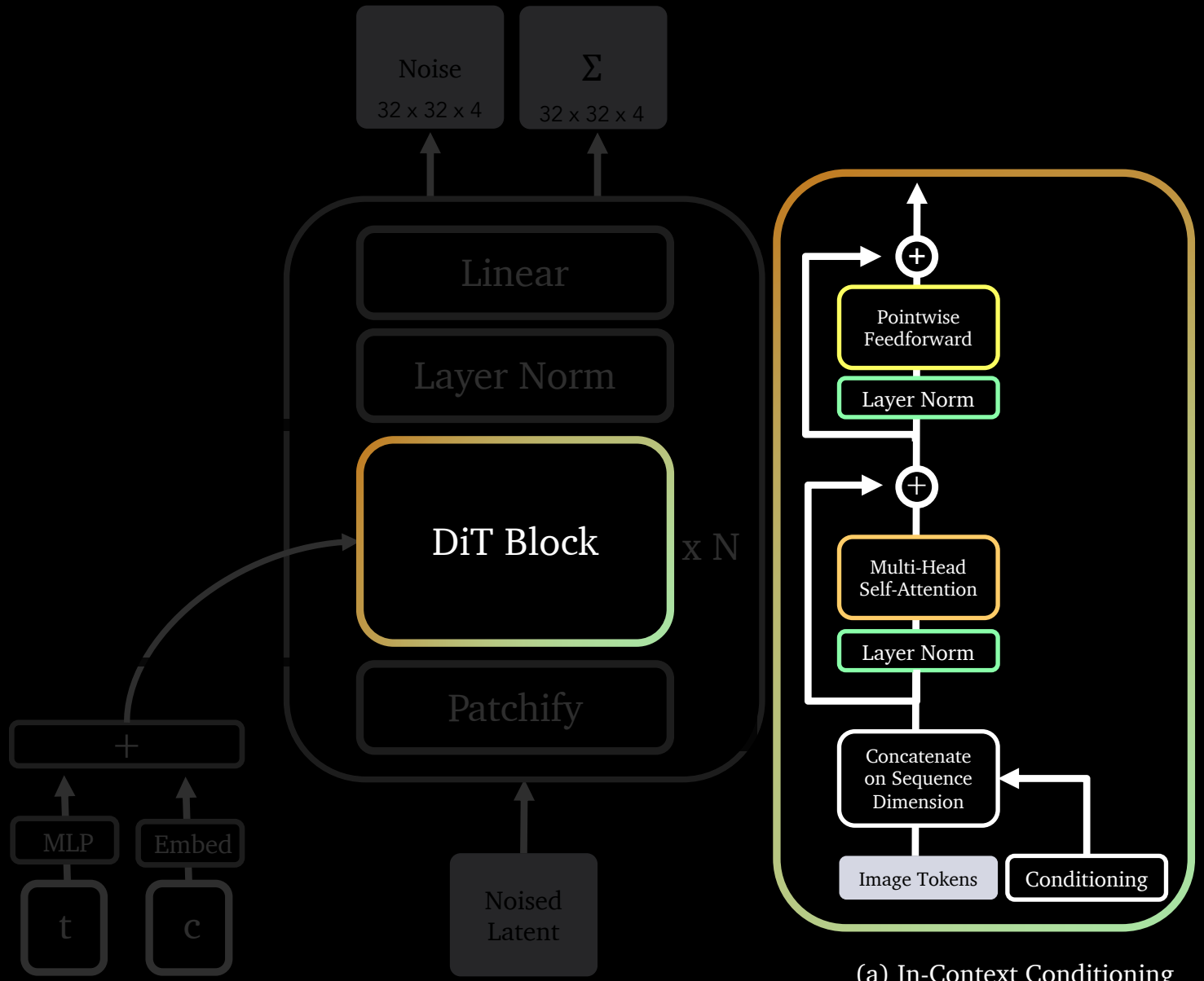
DiT Design Space: Transformer Block



DiT Design Space: Transformer Block

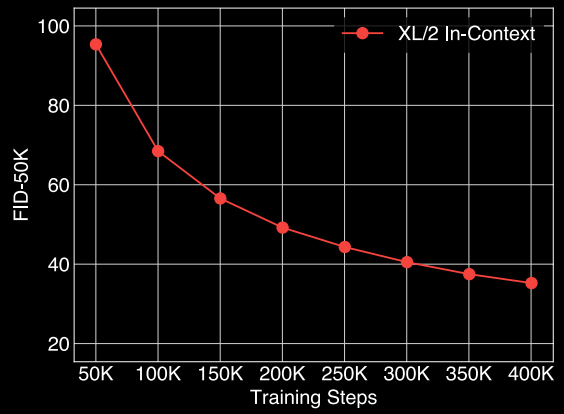


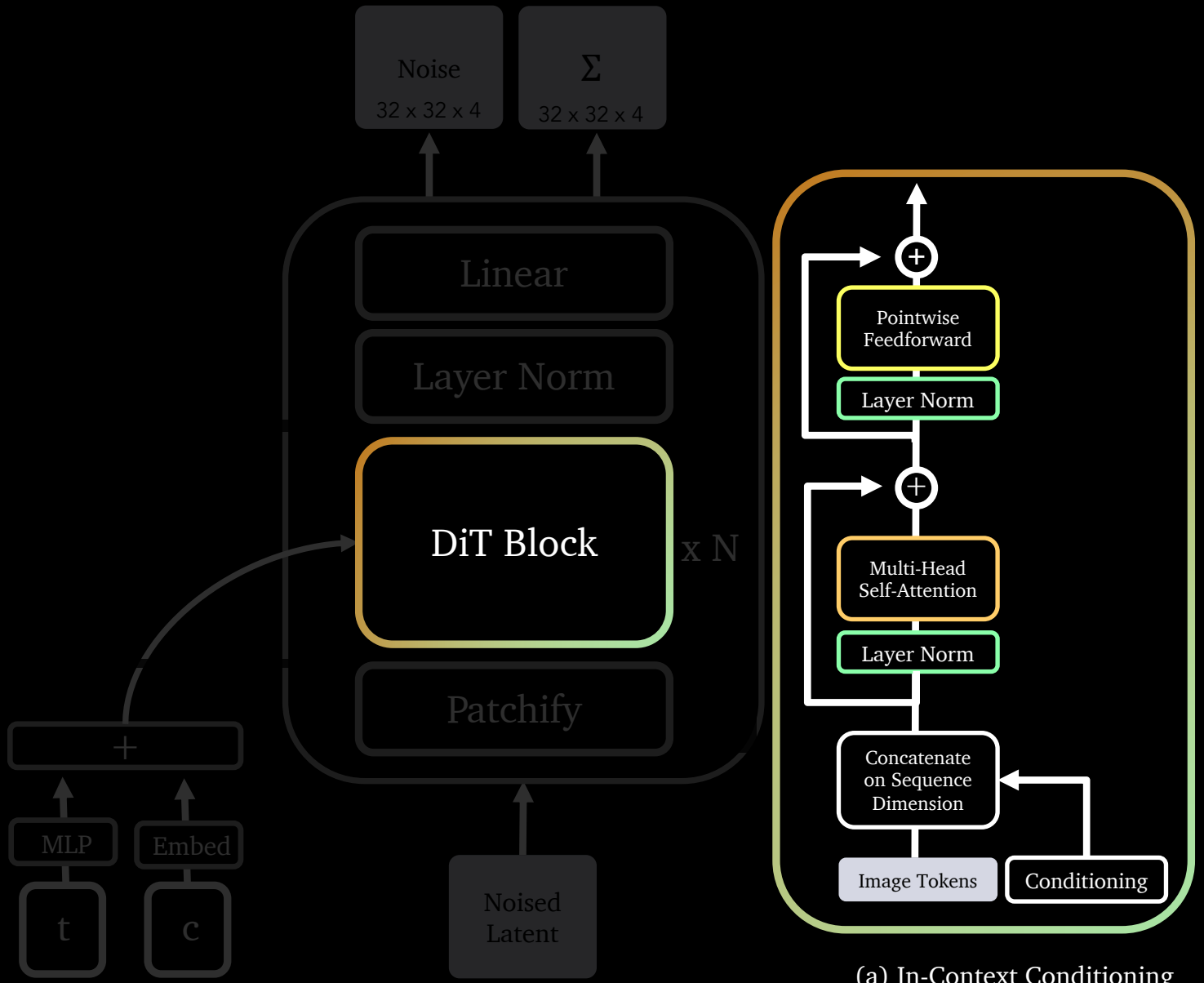
DiT Design Space: Transformer Block



(a) In-Context Conditioning

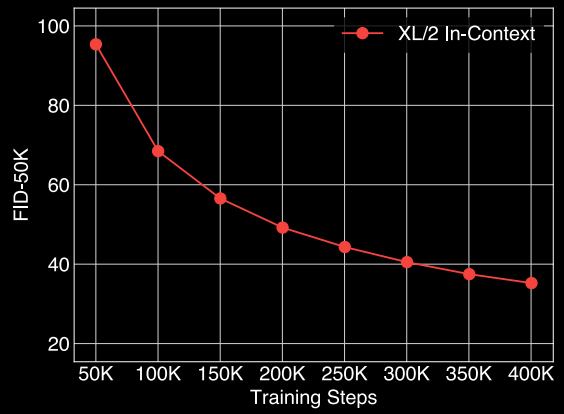
DiT Design Space: Transformer Block

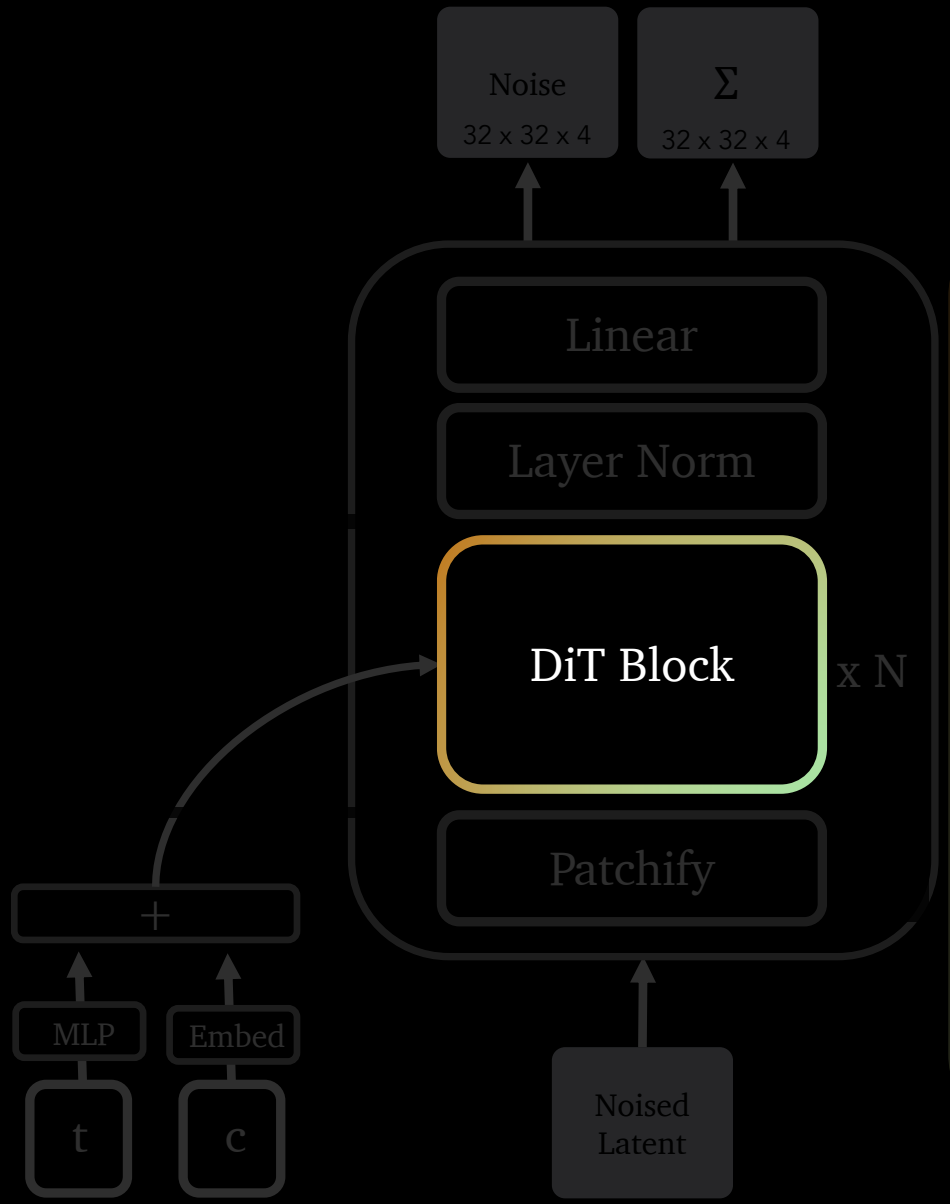


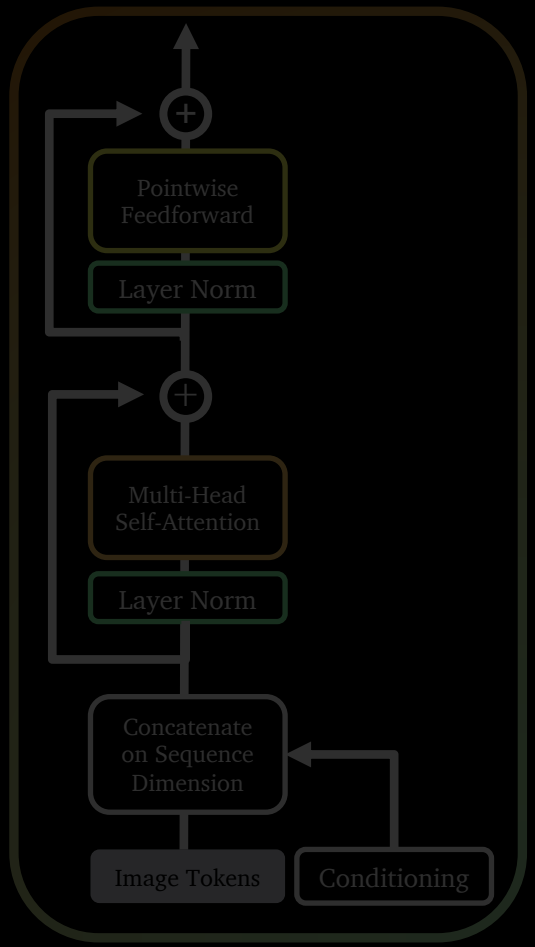


(a) In-Context Conditioning

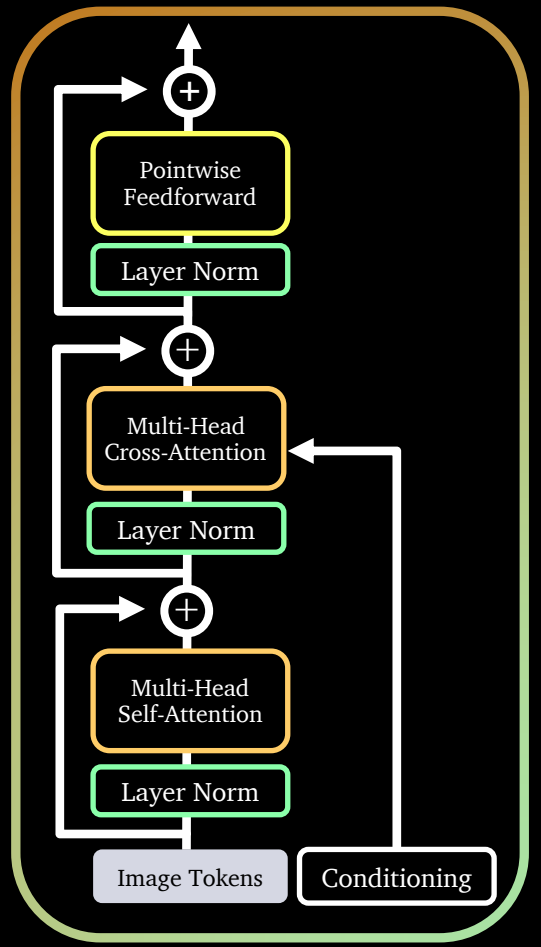
DiT Design Space: Transformer Block





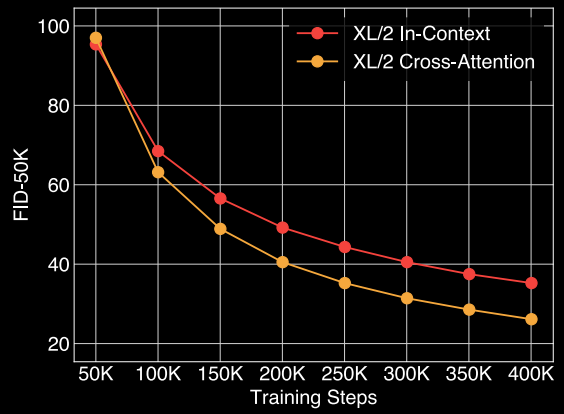


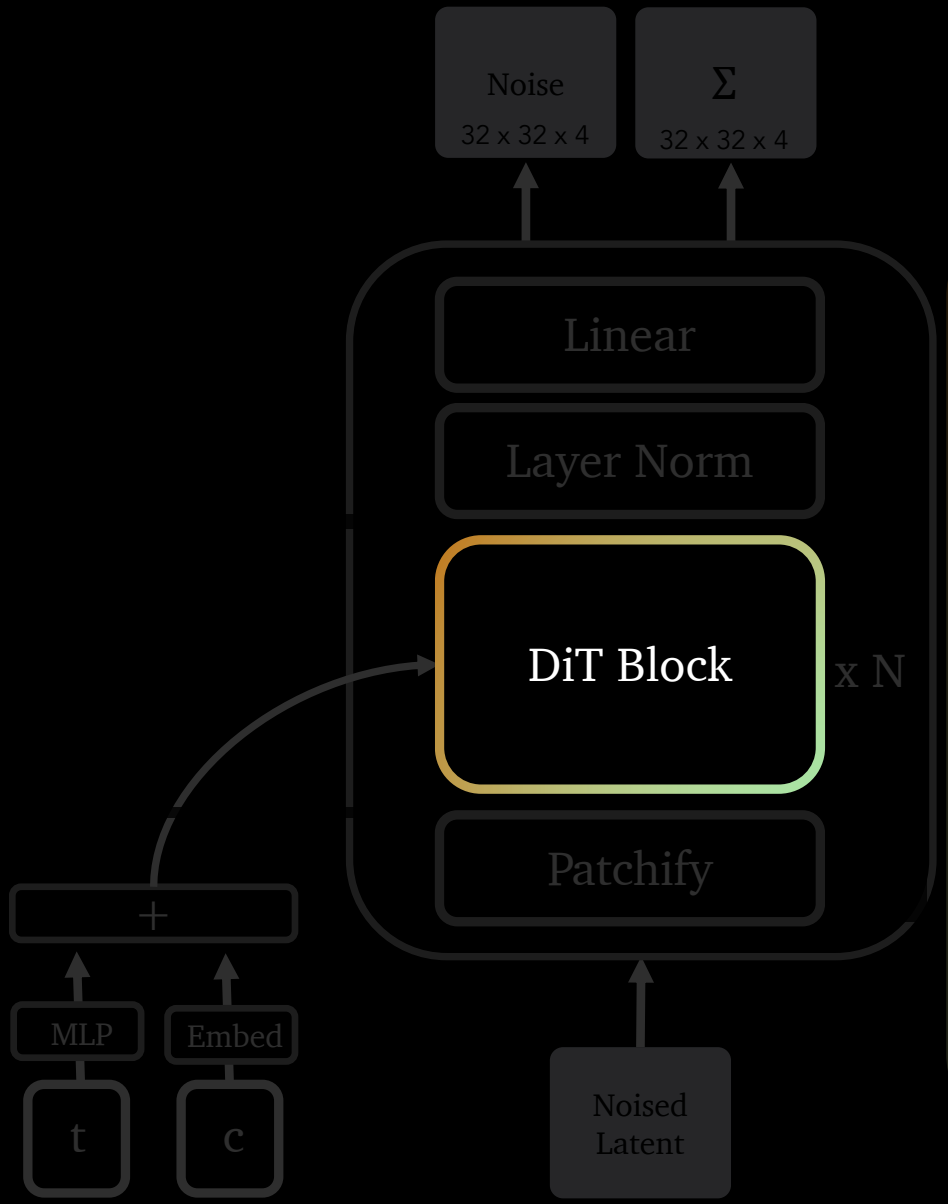
(a) In-Context Conditioning

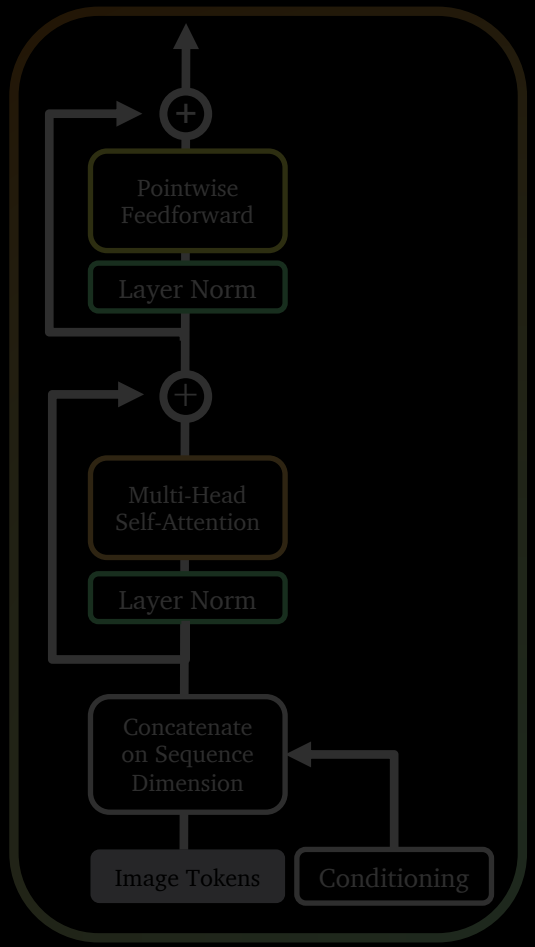


(b) Cross-Attention

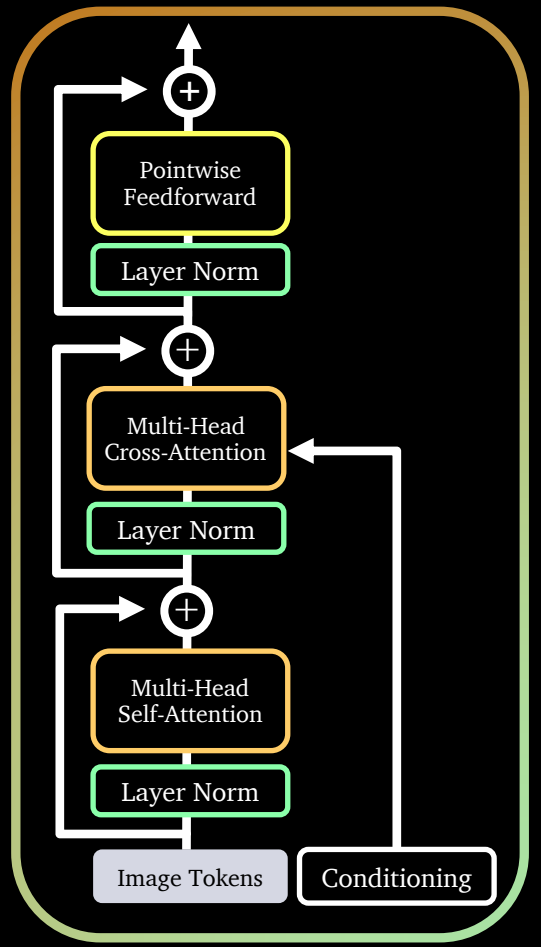
DiT Design Space: Transformer Block





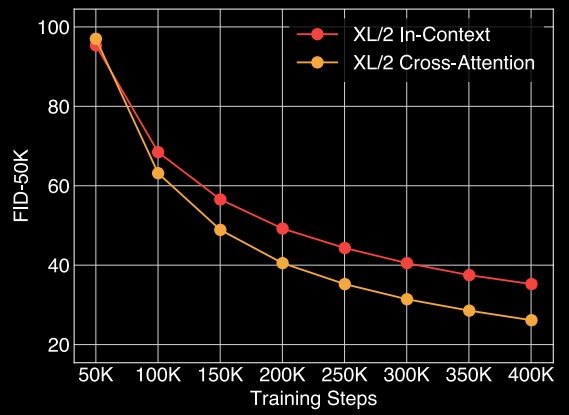


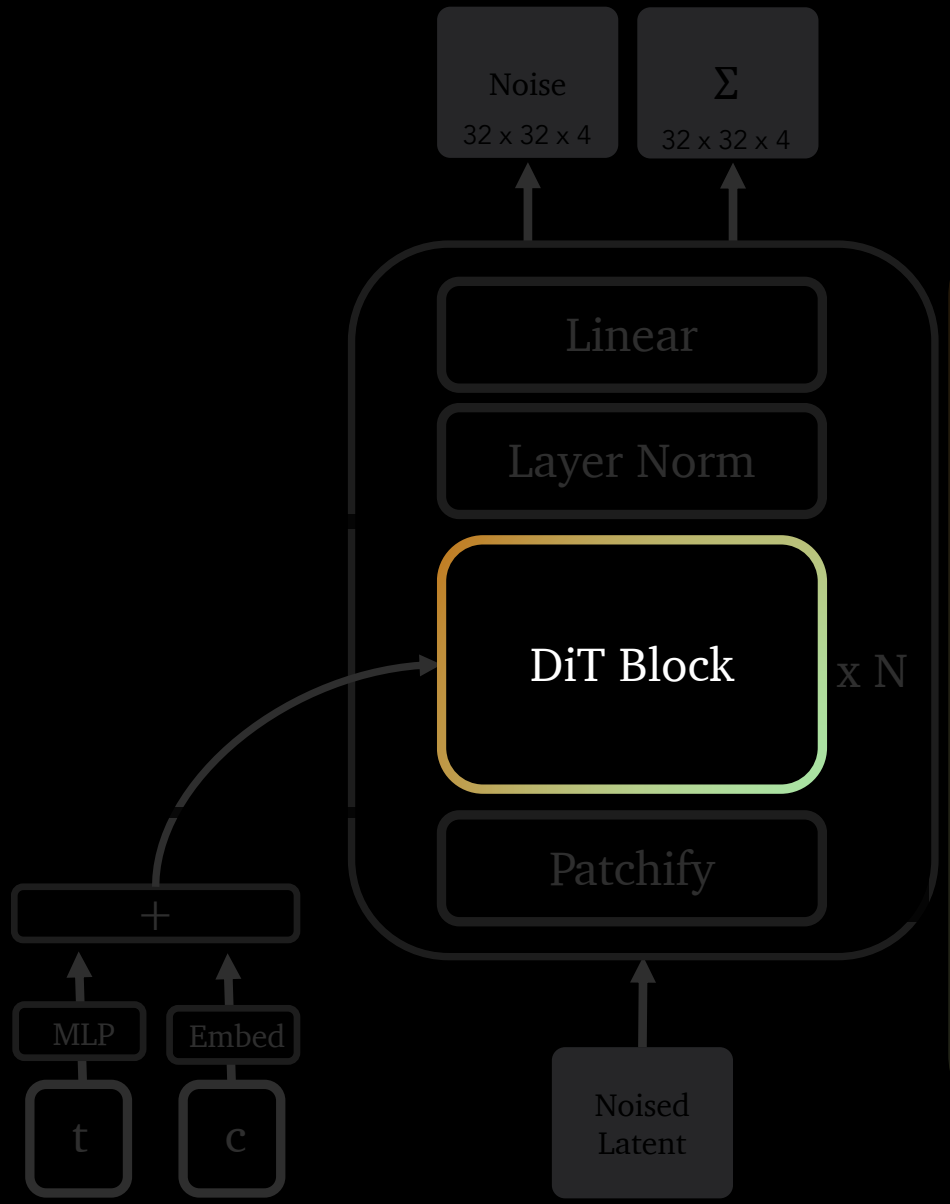
(a) In-Context Conditioning

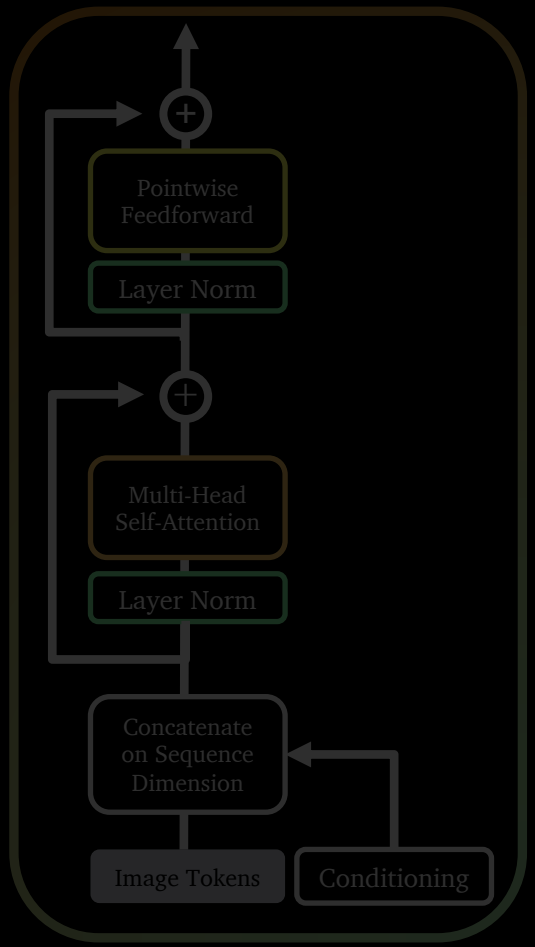


(b) Cross-Attention

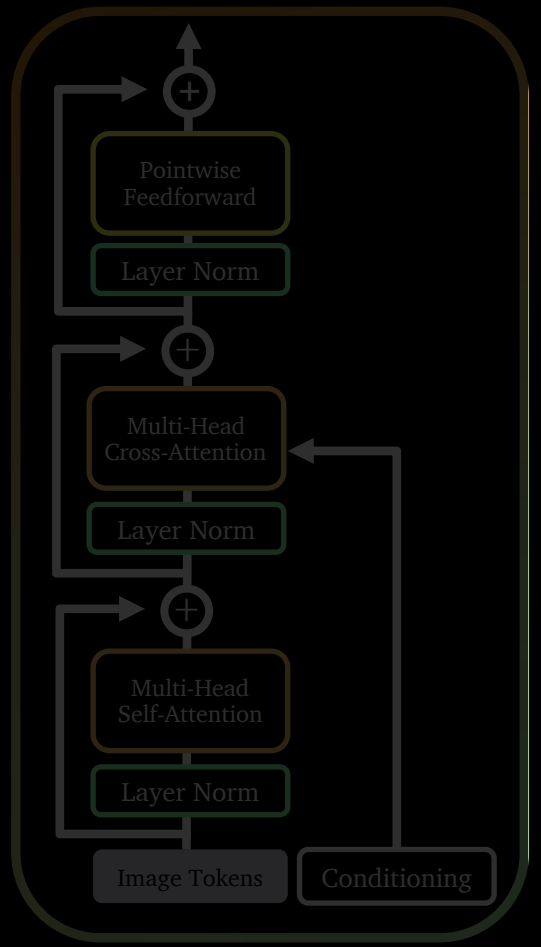
DiT Design Space: Transformer Block



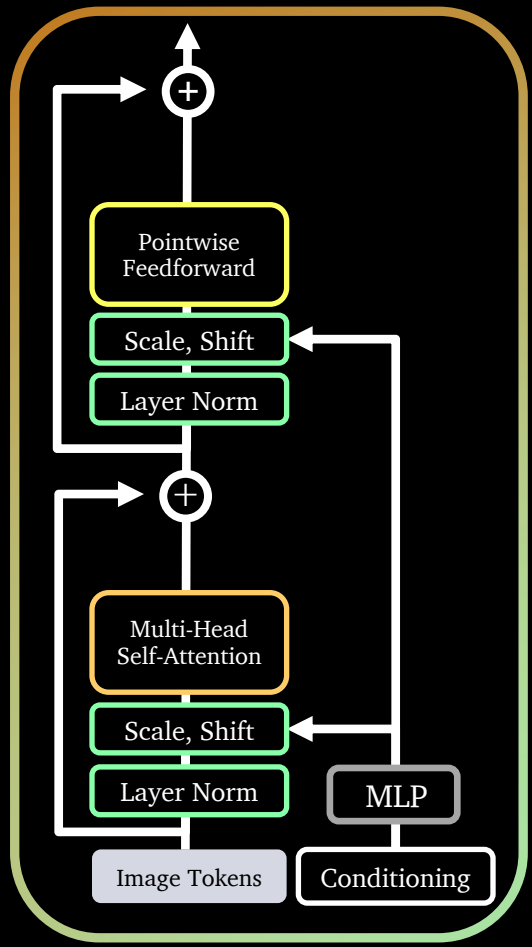




(a) In-Context Conditioning

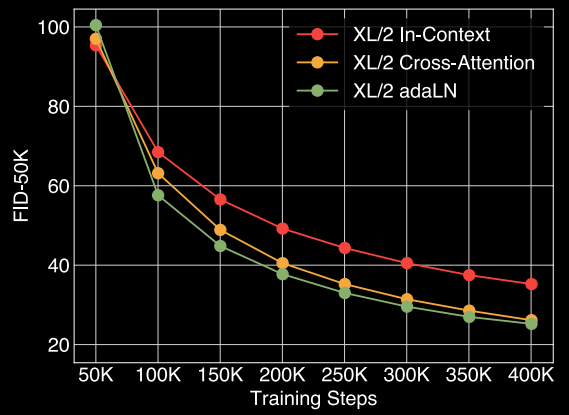


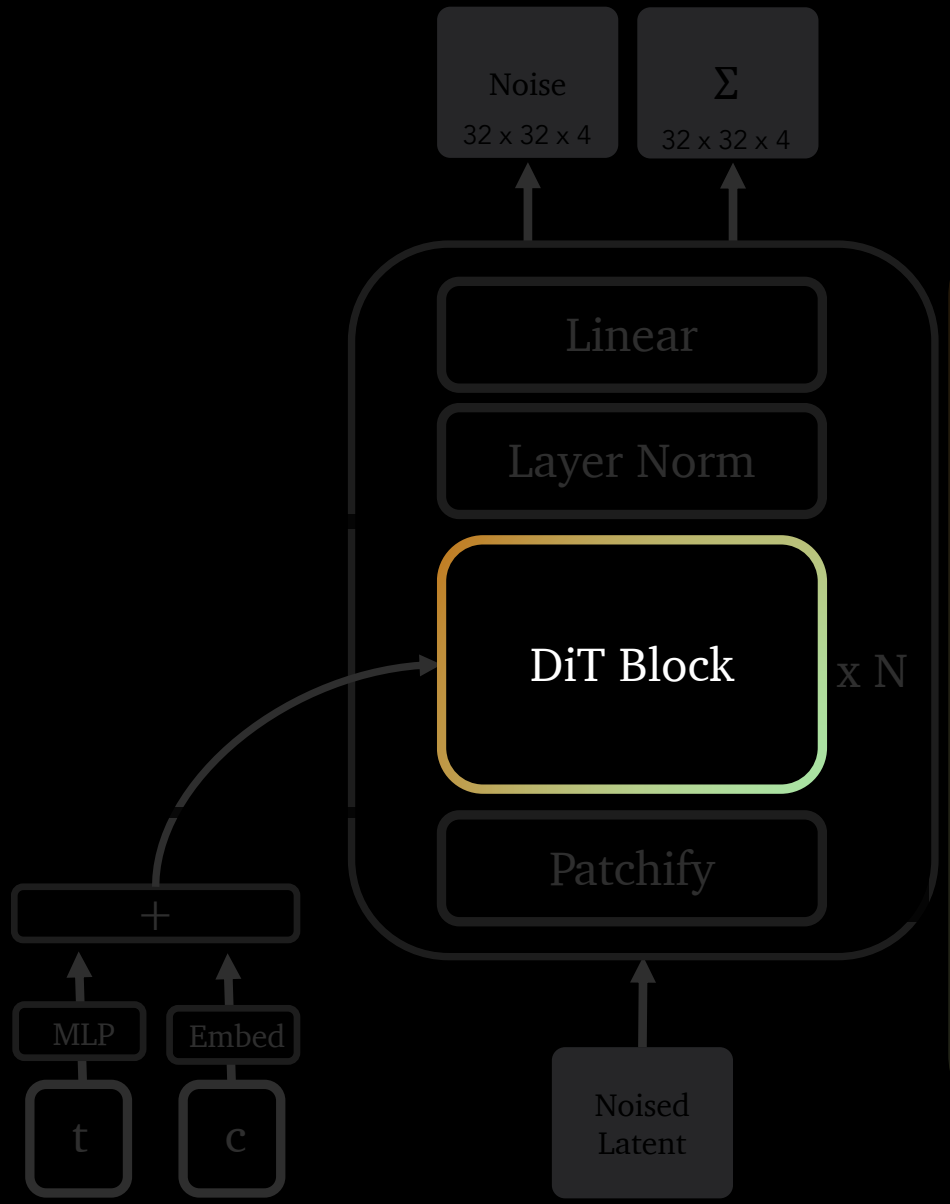
(b) Cross-Attention

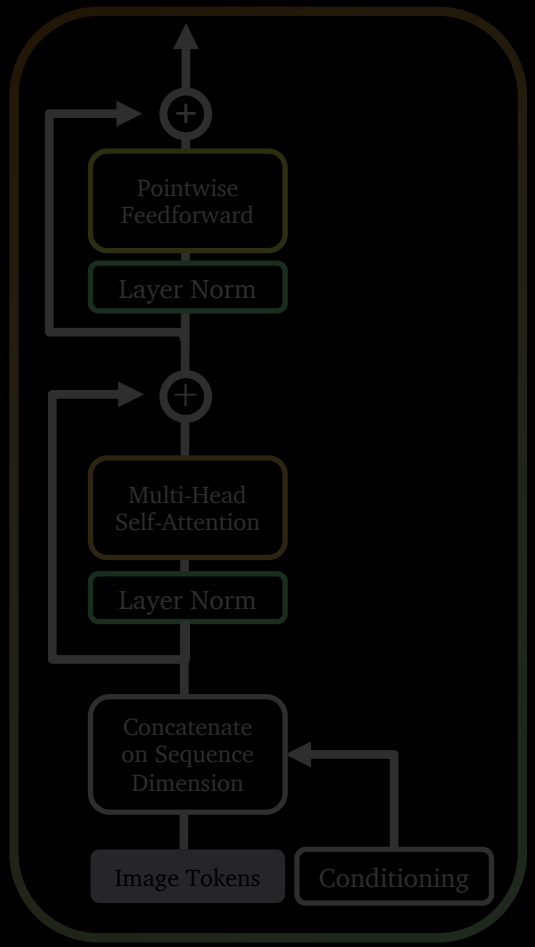


(c) Adaptive Layer Norm

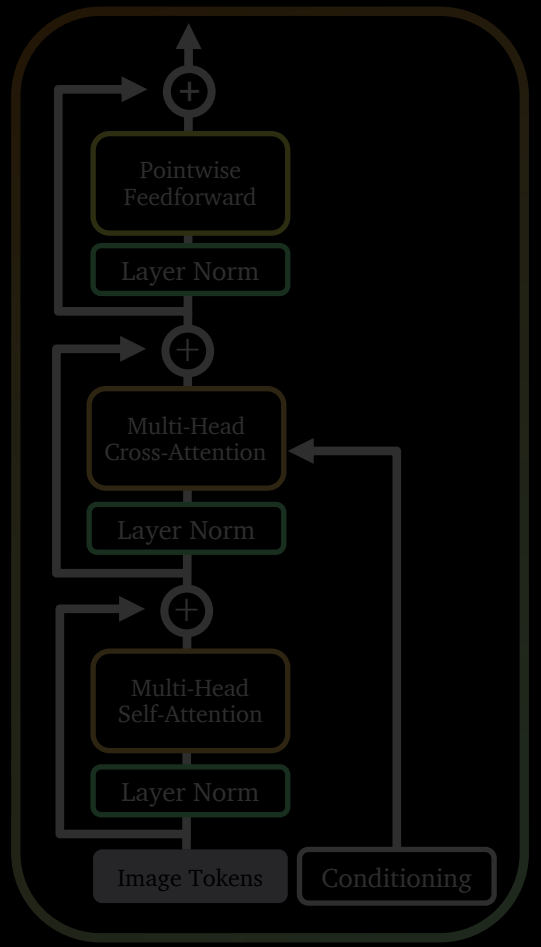
DiT Design Space: Transformer Block



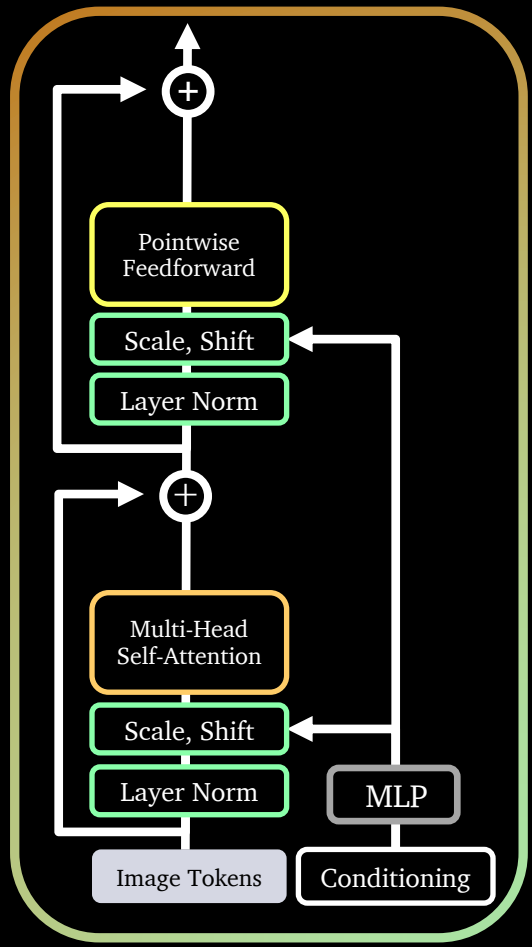




(a) In-Context Conditioning



(b) Cross-Attention



(c) Adaptive Layer Norm

Aside: Identity Initialization in ResNets

kn	η	γ-init	top-1 error (%)
256	0.1	1.0	23.84 $\pm$0.18
256	0.1	0.0	23.60 $\pm$0.12
8k	3.2	1.0	24.11 $\pm$0.07
8k	3.2	0.0	23.74 $\pm$0.09

(b) **Batch normalization γ initialization.** Initializing $\gamma = 0$ in the *last* BN layer of each residual block improves results for both small and large minibatches. This initialization leads to better optimization behavior which has a larger positive impact when training with large minibatches.

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. Goyal et. al 2018.

Aside: Identity Initialization in ResNets

```
def resnet_block(x, *, temb, name, out_ch=None, conv_shortcut=False, dropout):
  B, H, W, C = x.shape
  if out_ch is None:
    out_ch = C

  with tf.variable_scope(name):
    h = x

    h = nonlinearity(normalize(h, temb=temb, name='norm1'))
    h = nn.conv2d(h, name='conv1', num_units=out_ch)

    # add in timestep embedding
    h += nn.dense(nonlinearity(temb), name='temb_proj', num_units=out_ch)[:, None, None, :]

    h = nonlinearity(normalize(h, temb=temb, name='norm2'))
    h = tf.nn.dropout(h, rate=dropout)
    h = nn.conv2d(h, name='conv2', num_units=out_ch, init_scale=0.)

    if C != out_ch:
      if conv_shortcut:
        x = nn.conv2d(x, name='conv_shortcut', num_units=out_ch)
      else:
        x = nn.nin(x, name='nin_shortcut', num_units=out_ch)

    assert x.shape == h.shape
    print('{}: x={} temb={}'.format(tf.get_default_graph().get_name_scope(), x.shape, temb.shape))
    return x + h
```

Original U-Net code from Ho et. al

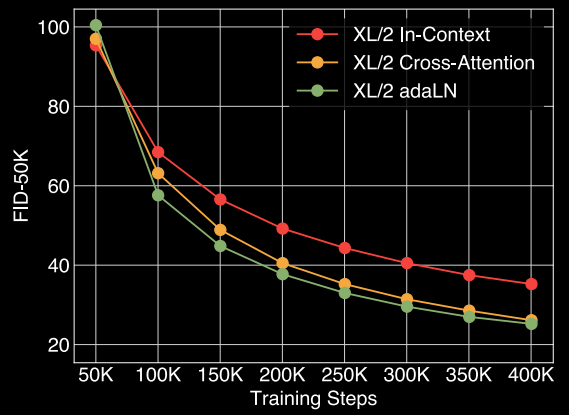
Aside: Identity Initialization in ResNets

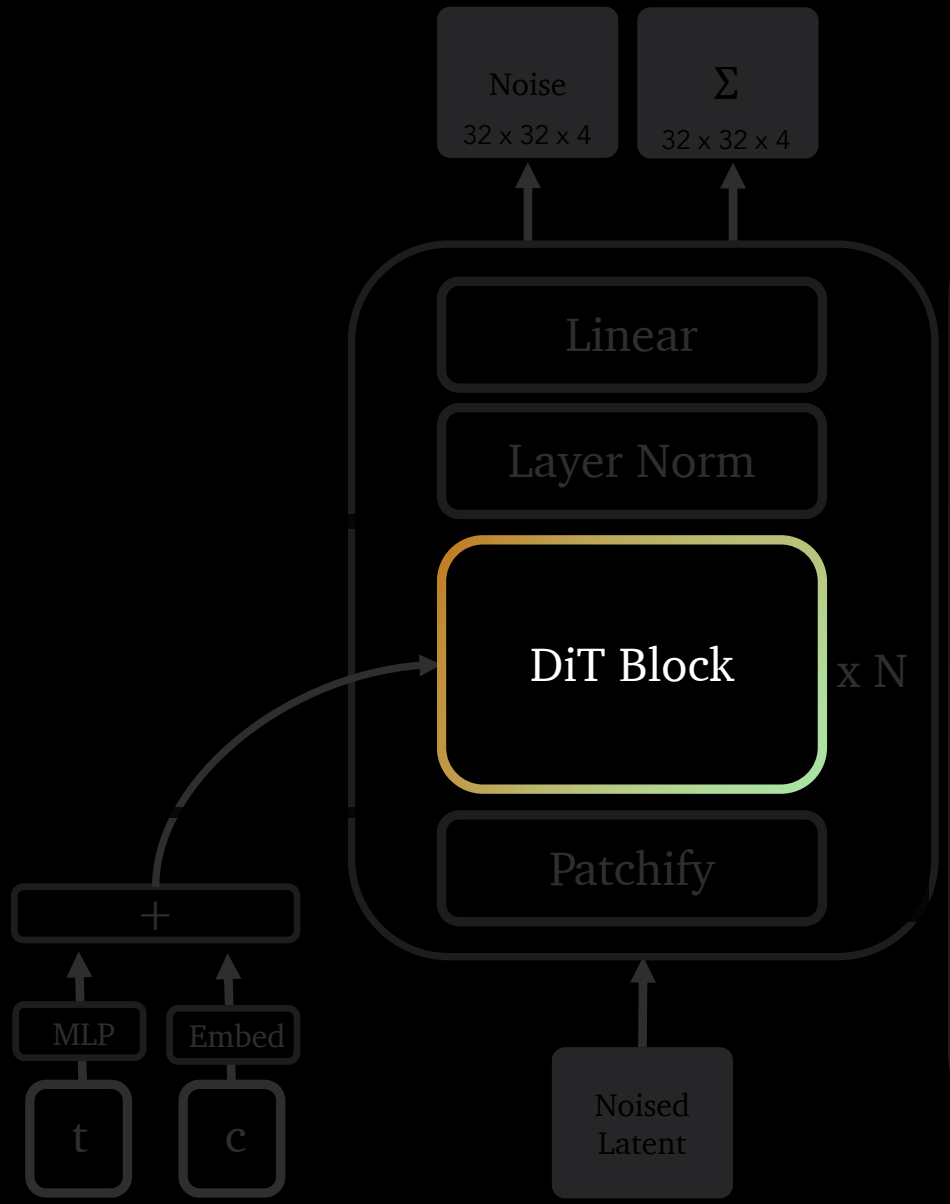
```python
def resnet_block(x, *, temb, name, out_ch=None, conv_shortcut=False, dropout):
  B, H, W, C = x.shape
  if out_ch is None:
    out_ch = C

  with tf.variable_scope(name):
    h = x

    h = nonlinearity(normalize(h, temb=temb, name='norm1'))
    h = nn.conv2d(h, name='conv1', num_units=out_ch)

    # add in timestep embedding
    h += nn.dense(nonlinearity(temb), name='temb_proj', num_units=out_ch)[:, None, None, :]

    h = nonlinearity(normalize(h, temb=temb, name='norm2'))
    h = tf.nn.dropout(h, rate=dropout)
    h = nn.conv2d(h, name='conv2', num_units=out_ch, init_scale=0.)

    if C != out_ch:
      if conv_shortcut:
        x = nn.conv2d(x, name='conv_shortcut', num_units=out_ch)
      else:
        x = nn.nin(x, name='nin_shortcut', num_units=out_ch)

    assert x.shape == h.shape
    print('{}: x={} temb={}'.format(tf.get_default_graph().get_name_scope(), x.shape, temb.shape))
    return x + h
```

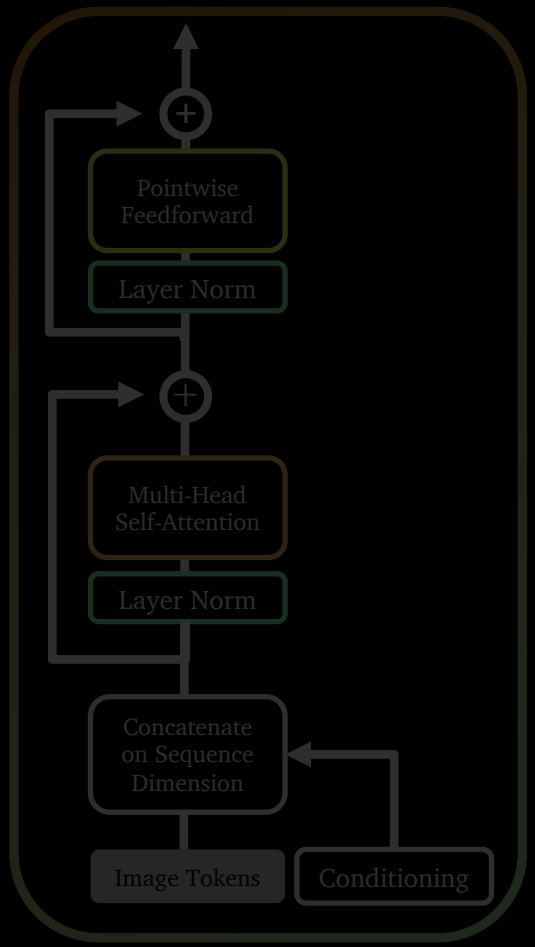
Original U-Net code from Ho et. al

Not mentioned in paper, but super important in practice!

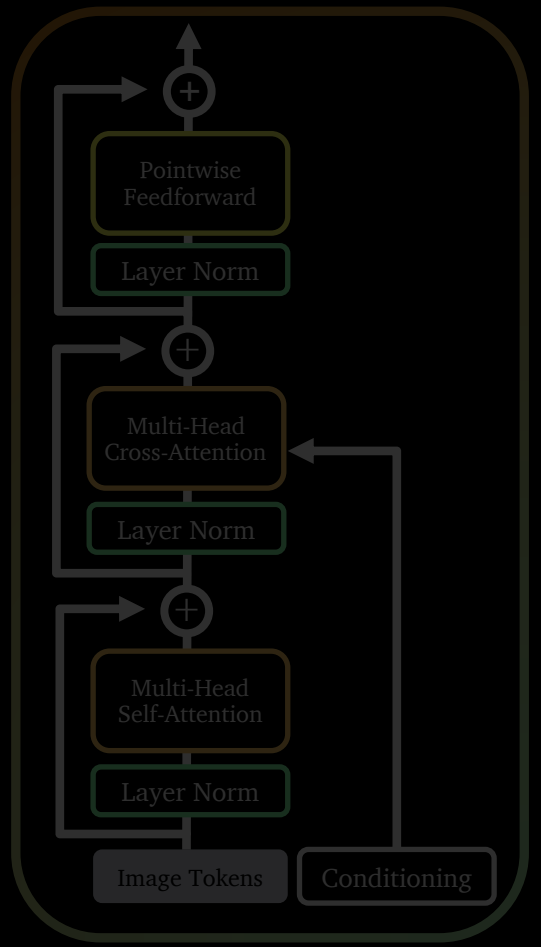
DiT Design Space: Transformer Block



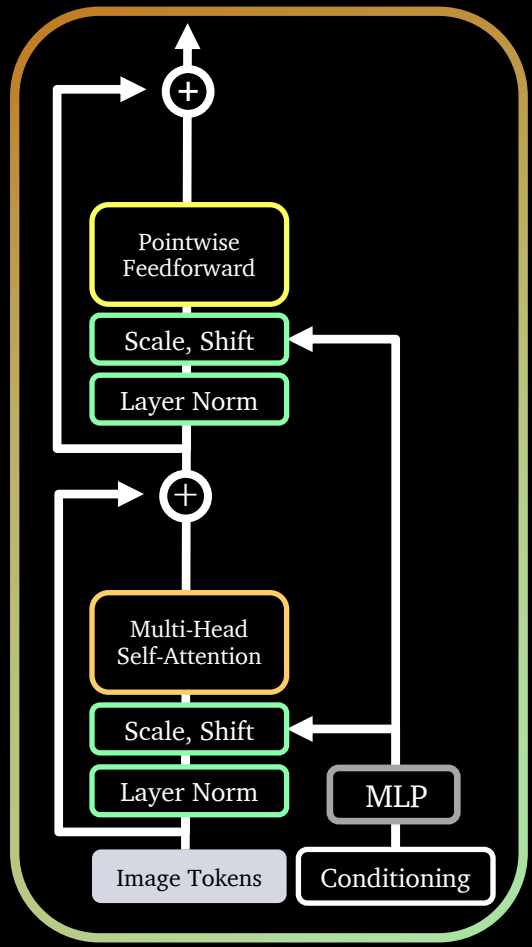




(a) In-Context Conditioning

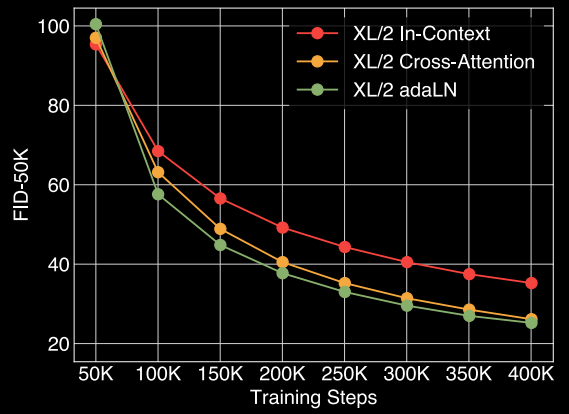


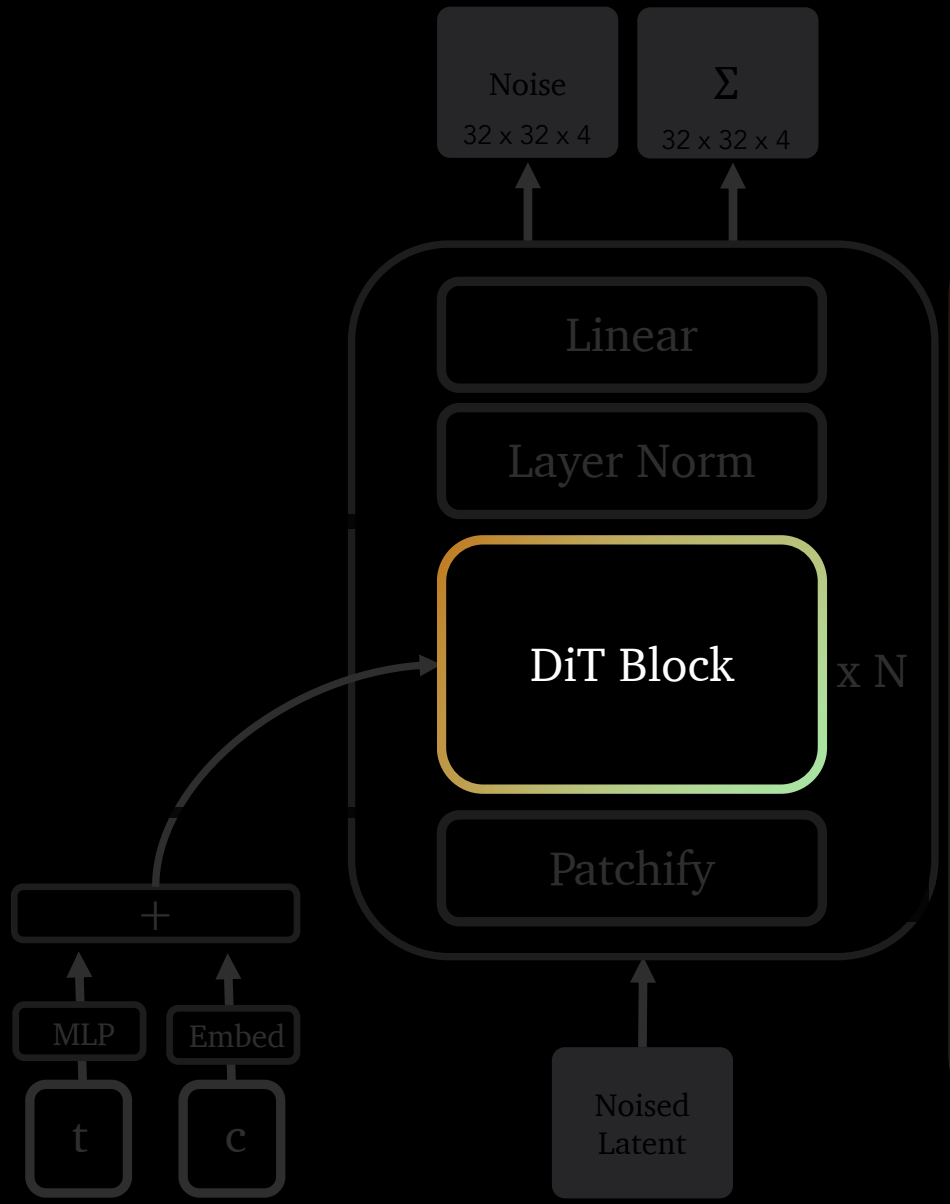
(b) Cross-Attention

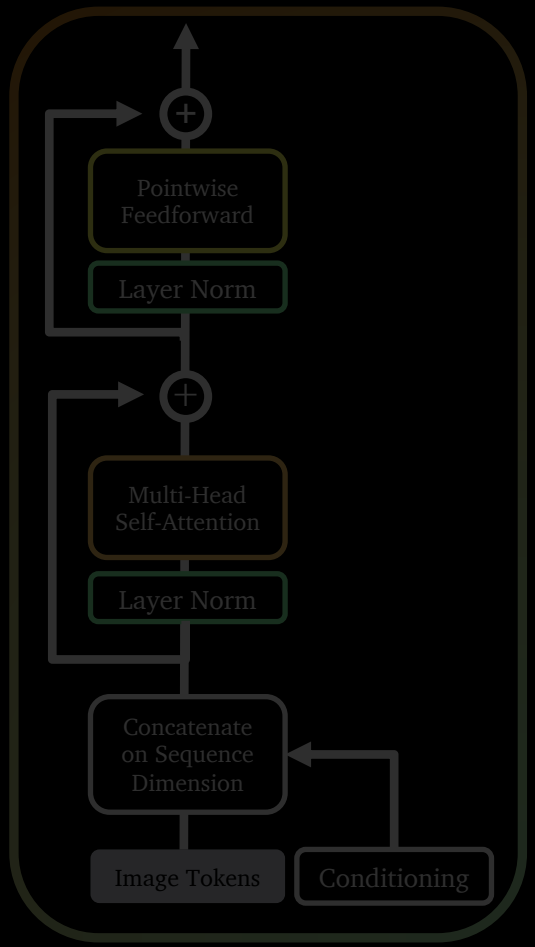


(c) Adaptive Layer Norm

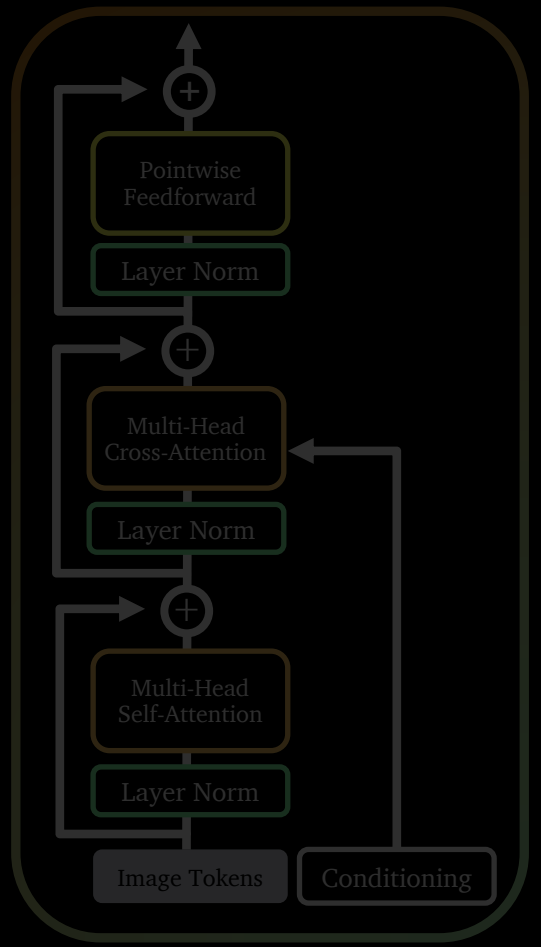
DiT Design Space: Transformer Block



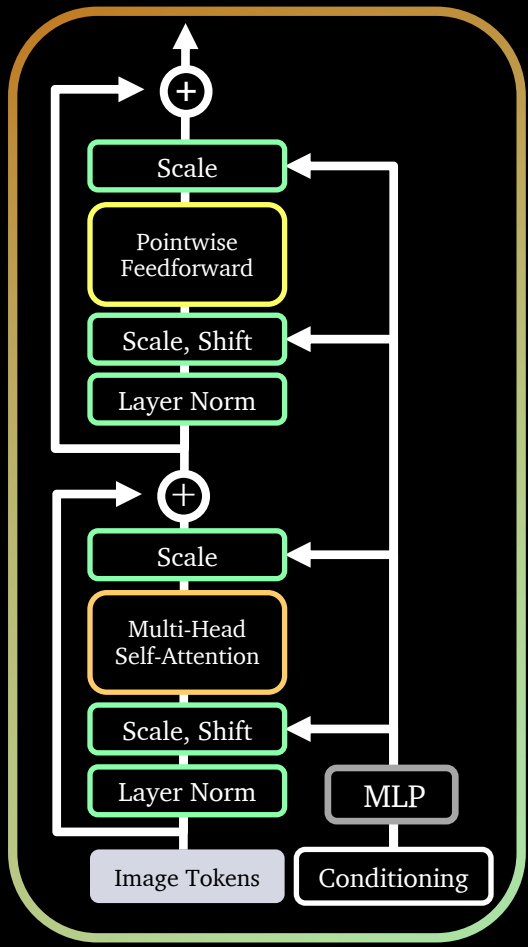




(a) In-Context Conditioning

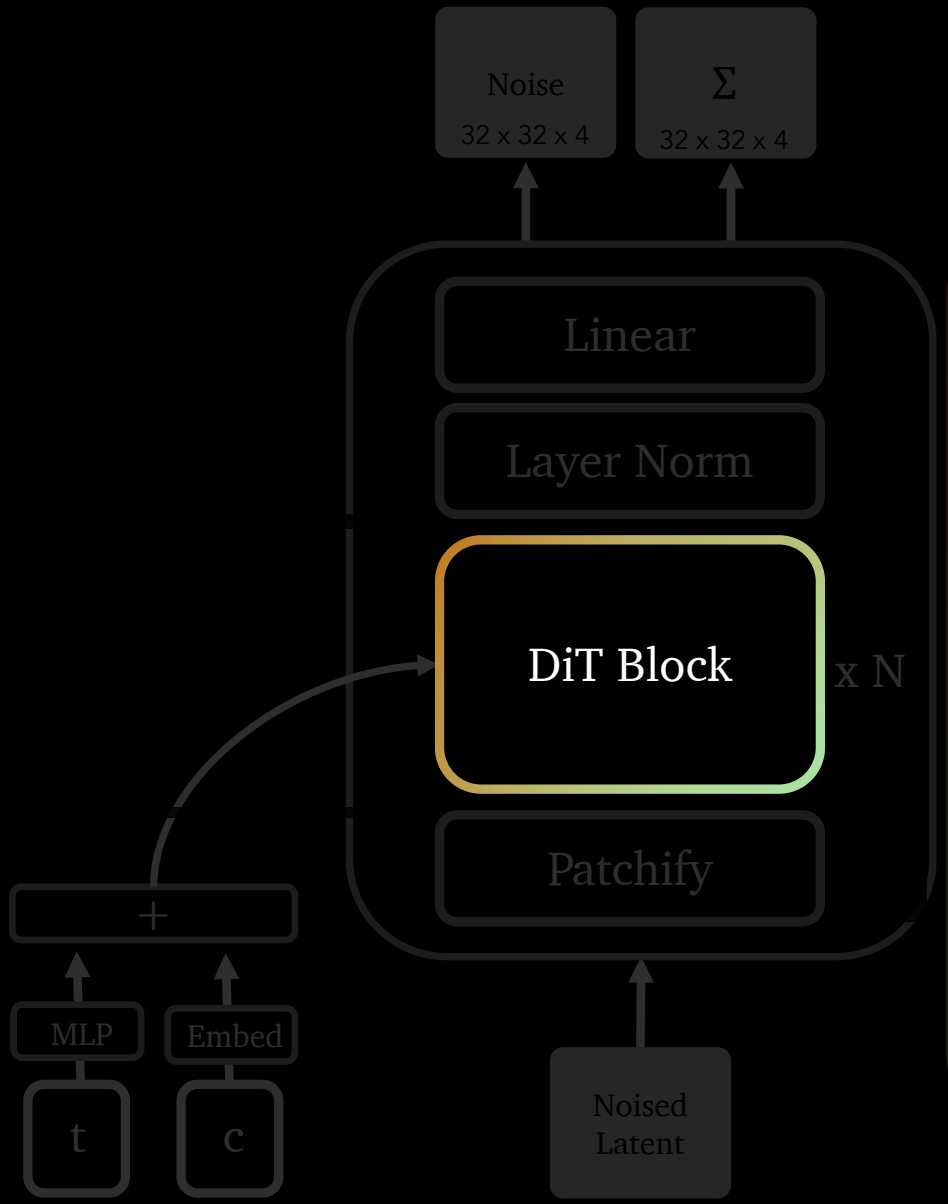


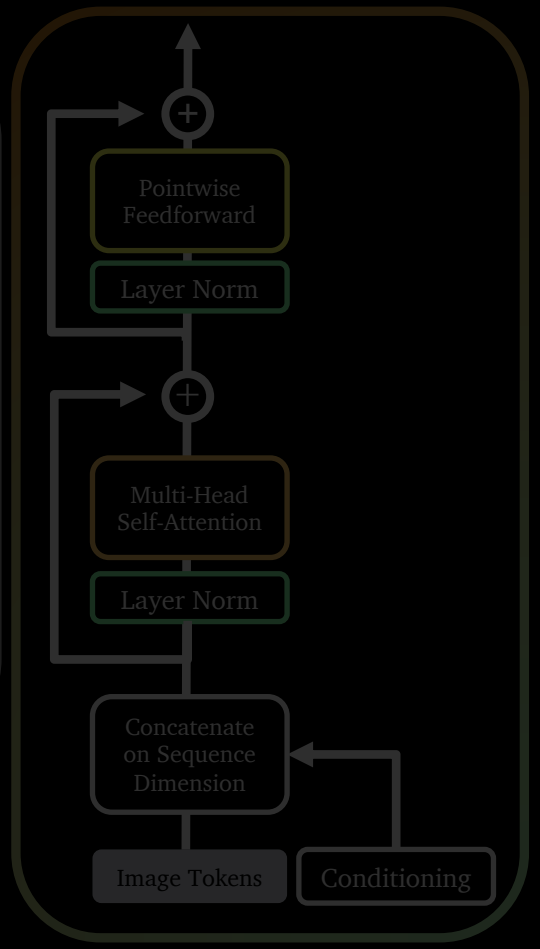
(b) Cross-Attention



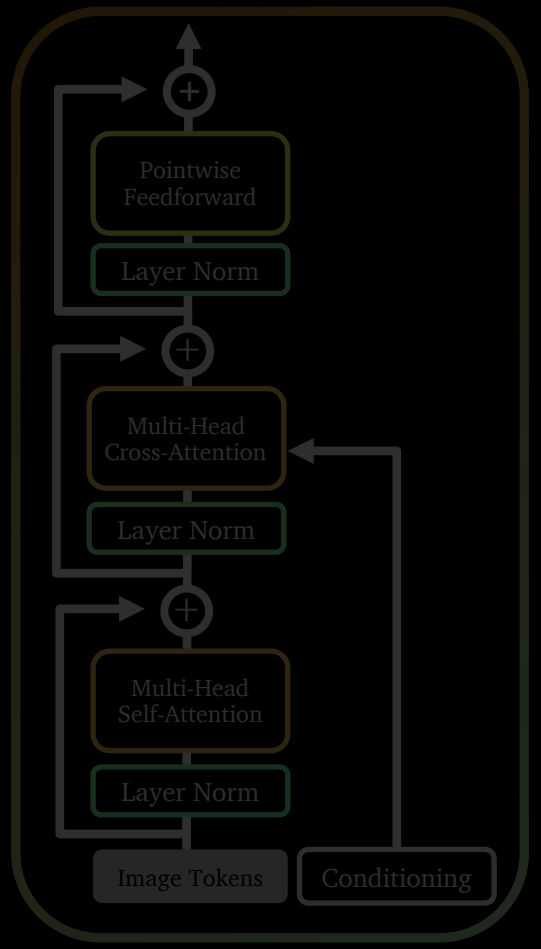
(d) Adaptive Layer Norm *Zero*

DiT Design Space: Transformer Block

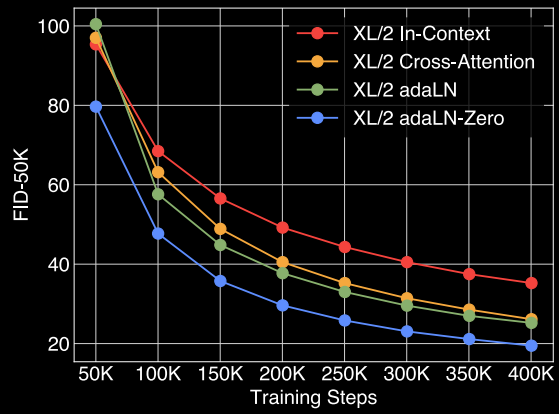


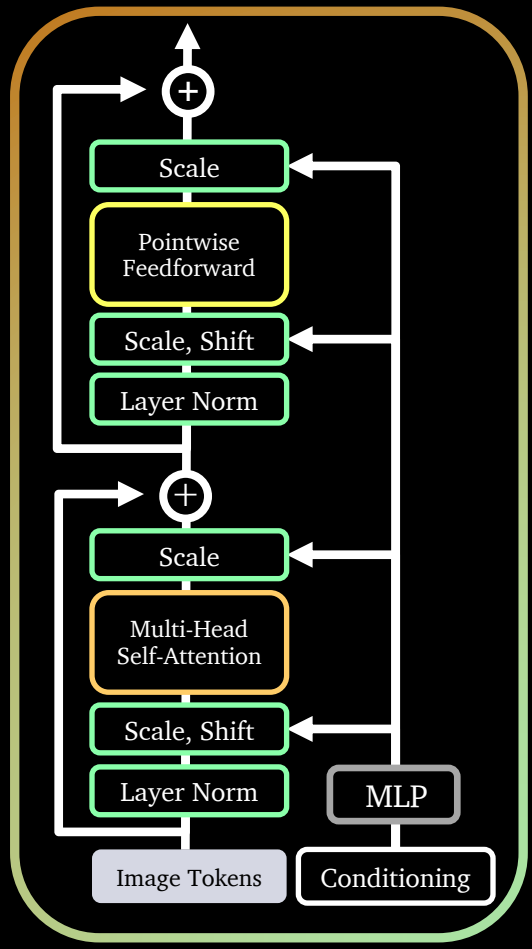


(a) In-Context Conditioning



(b) Cross-Attention

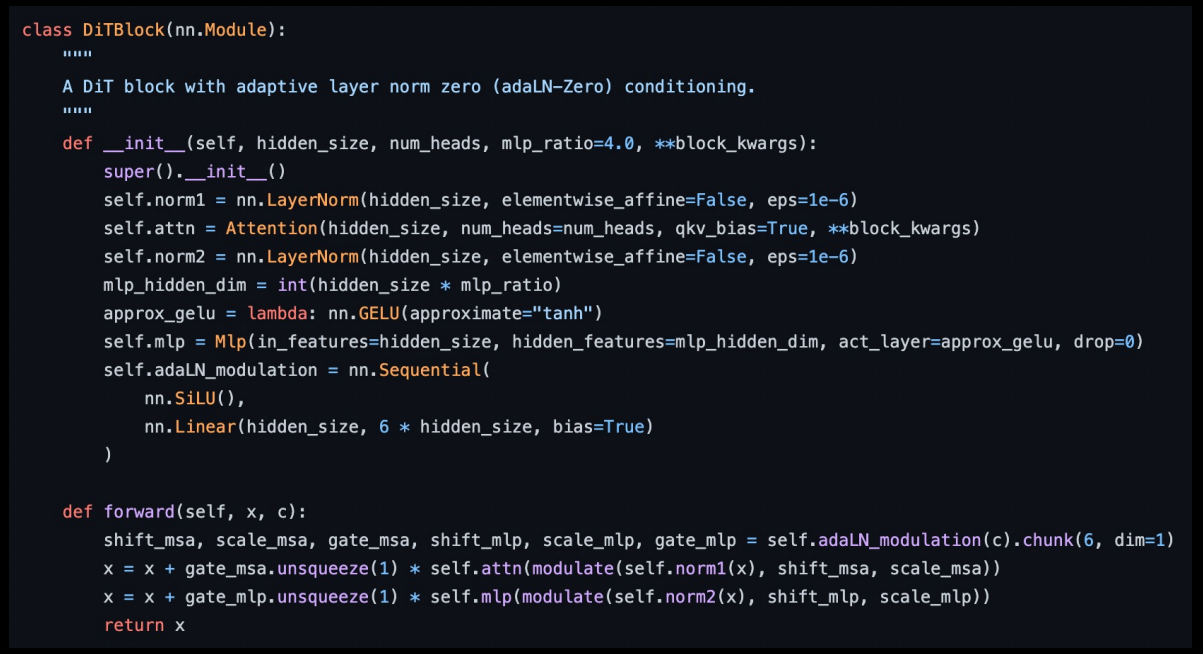




(d) Adaptive Layer Norm *Zero*

```python
class DiTBlock(nn.Module):
    """
    A DiT block with adaptive layer norm zero (adaLN-Zero) conditioning.
    """
    def __init__(self, hidden_size, num_heads, mlp_ratio=4.0, **block_kwargs):
        super().__init__()
        self.norm1 = nn.LayerNorm(hidden_size, elementwise_affine=False, eps=1e-6)
        self.attn = Attention(hidden_size, num_heads=num_heads, qkv_bias=True, **block_kwargs)
        self.norm2 = nn.LayerNorm(hidden_size, elementwise_affine=False, eps=1e-6)
        mlp_hidden_dim = int(hidden_size * mlp_ratio)
        approx_gelu = lambda: nn.GELU(approximate="tanh")
        self.mlp = Mlp(in_features=hidden_size, hidden_features=mlp_hidden_dim, act_layer=approx_gelu, drop=0)
        self.adaLN_modulation = nn.Sequential(
            nn.SiLU(),
            nn.Linear(hidden_size, 6 * hidden_size, bias=True)
        )

    def forward(self, x, c):
        shift_msa, scale_msa, gate_msa, shift_mlp, scale_mlp, gate_mlp = self.adaLN_modulation(c).chunk(6, dim=1)
        x = x + gate_msa.unsqueeze(1) * self.attn(modulate(self.norm1(x), shift_msa, scale_msa))
        x = x + gate_mlp.unsqueeze(1) * self.mlp(modulate(self.norm2(x), shift_mlp, scale_mlp))
        return x
```

DiT Block



```python
class DiTBlock(nn.Module):
    """
    A DiT block with adaptive layer norm zero (adaLN-Zero) conditioning.
    """
    def __init__(self, hidden_size, num_heads, mlp_ratio=4.0, **block_kwargs):
        super().__init__()
        self.norm1 = nn.LayerNorm(hidden_size, elementwise_affine=False, eps=1e-6)
        self.attn = Attention(hidden_size, num_heads=num_heads, qkv_bias=True, **block_kwargs)
        self.norm2 = nn.LayerNorm(hidden_size, elementwise_affine=False, eps=1e-6)
        mlp_hidden_dim = int(hidden_size * mlp_ratio)
        approx_gelu = lambda: nn.GELU(approximate="tanh")
        self.mlp = Mlp(in_features=hidden_size, hidden_features=mlp_hidden_dim, act_layer=approx_gelu, drop=0)
        self.adaLN_modulation = nn.Sequential(
            nn.SiLU(),
            nn.Linear(hidden_size, 6 * hidden_size, bias=True)
        )

    def forward(self, x, c):
        shift_msa, scale_msa, gate_msa, shift_mlp, scale_mlp, gate_mlp = self.adaLN_modulation(c).chunk(6, dim=1)
        x = x + gate_msa.unsqueeze(1) * self.attn(modulate(self.norm1(x), shift_msa, scale_msa))
        x = x + gate_mlp.unsqueeze(1) * self.mlp(modulate(self.norm2(x), shift_mlp, scale_mlp))
        return x
```

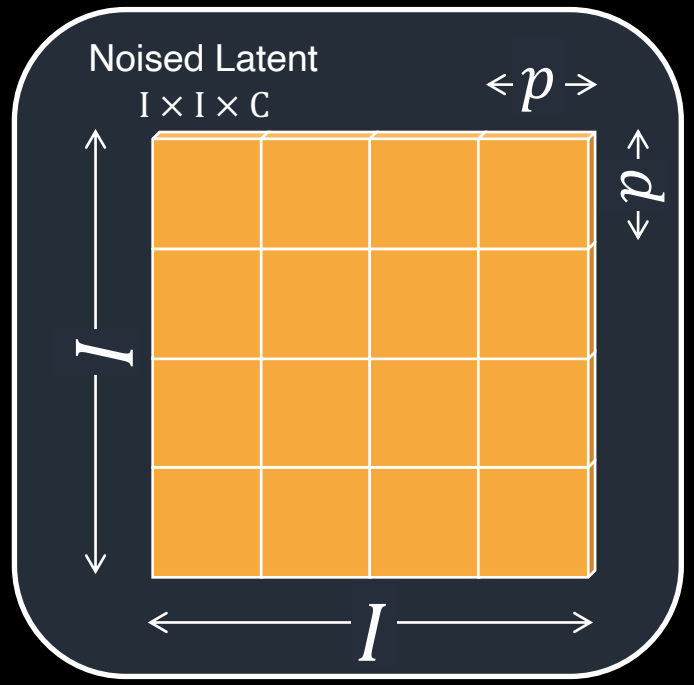
DiT Block

```python
class ResBlock(TimestepBlock):
    """
    A residual block that can optionally change the numt

    :param channels: the number of input channels.
    :param emb_channels: the number of timestep embeddir
    :param dropout: the rate of dropout.
    :param out_channels: if specified, the number of out
    :param use_conv: if True and out_channels is specifi
        convolution instead of a smaller 1x1 convolutior
        channels in the skip connection.
    :param dims: determines if the signal is 1D, 2D, or
    :param use_checkpoint: if True, use gradient checkpc
    :param up: if True, use this block for upsampling.
    :param down: if True, use this block for downsamplir
    """

    def __init__(
        self,
        channels,
        emb_channels,
        dropout,
        out_channels=None,
        use_conv=False,
        use_scale_shift_norm=False,
        dims=2,
        use_checkpoint=False,
        up=False,
        down=False,
    ):
        super().__init__()
        self.channels = channels
        self.emb_channels = emb_channels
        self.dropout = dropout
        self.out_channels = out_channels or channels
        self.use_conv = use_conv
        self.use_checkpoint = use_checkpoint
        self.use_scale_shift_norm = use_scale_shift_norm

        self.in_layers = nn.Sequential(
            normalization(channels),
            nn.SiLU(),
            conv_nd(dims, channels, self.out_channels, 3
        )

        self.updown = up or down

        if up:
            self.h_upd = Upsample(channels, False, dims)
            self.x_upd = Upsample(channels, False, dims)
        elif down:
            self.h_upd = Downsample(channels, False, dim
            self.x_upd = Downsample(channels, False, dim
        else:
            self.h_upd = self.x_upd = nn.Identity()

        self.emb_layers = nn.Sequential(
            nn.SiLU(),
            linear(
                emb_channels,
                2 * self.out_channels if use_scale_shift
            ),
        )
        self.out_layers = nn.Sequential(
            normalization(self.out_channels),
            nn.SiLU(),
            nn.Dropout(p=dropout),
            zero_module(
                conv_nd(dims, self.out_channels, self.ou
            ),
        )

        if self.out_channels == channels:
            self.skip_connection = nn.Identity()
        elif use_conv:
            self.skip_connection = conv_nd(
                dims, channels, self.out_channels, 3, pa
            )
        else:
            self.skip_connection = conv_nd(dims, channel

    def forward(self, x, emb):
        """
        Apply the block to a Tensor, conditioned on a timestep embedding.

        :param x: an [N x C x ...] Tensor of features.
        :param emb: an [N x emb_channels] Tensor of timestep embeddings.
        :return: an [N x C x ...] Tensor of outputs.
        """
        return checkpoint(
            self._forward, (x, emb), self.parameters(), self.use_checkpoint
        )

    def _forward(self, x, emb):
        if self.updown:
            in_rest, in_conv = self.in_layers[:
            h = in_rest(x)
            h = self.h_upd(h)
            x = self.x_upd(x)
            h = in_conv(h)
        else:
            h = self.in_layers(x)
        emb_out = self.emb_layers(emb).type(h.d
        while len(emb_out.shape) < len(h.shape)
            emb_out = emb_out[..., None]
        if self.use_scale_shift_norm:
            out_norm, out_rest = self.out_layer
            scale, shift = th.chunk(emb_out, 2,
            h = out_norm(h) * (1 + scale) + shi
            h = out_rest(h)
        else:
            h = h + emb_out
            h = self.out_layers(h)
        return self.skip_connection(x) + h


class AttentionBlock(nn.Module):
    """
    An attention block that allows spatial posi

    Originally ported from here, but adapted to
    https://github.com/hojonathanho/diffusion/b
    """

    def __init__(
        self,
        channels,
        num_heads=1,
        num_head_channels=-1,
        use_checkpoint=False,
        use_new_attention_order=False,
    ):
        super().__init__()
        self.channels = channels
        if num_head_channels == -1:
            self.num_heads = num_heads
        else:
            assert (
                channels % num_head_channels ==
            ), f"q,k,v channels {channels} is n
            self.num_heads = channels // num_he
        self.use_checkpoint = use_checkpoint
        self.norm = normalization(channels)
        self.qkv = conv_nd(1, channels, channel
        if use_new_attention_order:
            # split qkv before split heads
            self.attention = QKVAttention(self.
        else:
            # split heads before split qkv
            self.attention = QKVAttentionLegacy

        self.proj_out = zero_module(conv_nd(1,

    def forward(self, x):
        return checkpoint(self._forward, (x,),

    def _forward(self, x):
        b, c, *spatial = x.shape
        x = x.reshape(b, c, -1)
        qkv = self.qkv(self.norm(x))
        h = self.attention(qkv)
        h = self.proj_out(h)
        return (x + h).reshape(b, c, *spatial)


class Upsample(nn.Module):
    """
    An upsampling layer with an optional convolution.

    :param channels: channels in the inputs and outputs.
    :param use_conv: a bool determining if a convolution is applied.
    :param dims: determines if the signal is 1D, 2D, or 3D. If 3D, then
                 upsampling occurs in the inner-two dimensions.
    """

    def __init__(self, channels, use_conv, dims=2, out_channels=None):
        super().__init__()
        self.channels = channels
        self.out_channels = out_channels or channels
        self.use_conv = use_conv
        self.dims = dims
        if use_conv:
            self.conv = conv_nd(dims, self.channels, self.out_channels, 3, padding=1)

    def forward(self, x):
        assert x.shape[1] ==
        if self.dims == 3:
            x = F.interpolat
                x, (x.shape[
            )
        else:
            x = F.interpolat
        if self.use_conv:
            x = self.conv(x)
        return x


class Downsample(nn.Module):
    """
    A downsampling layer wit

    :param channels: channel
    :param use_conv: a bool
    :param dims: determines
                 downsamplir
    """

    def __init__(self, chanr
        super().__init__()
        self.channels = char
        self.out_channels =
        self.use_conv = use_
        self.dims = dims
        stride = 2 if dims !
        if use_conv:
            self.op = conv_f
                dims, self.c
            )
        else:
            assert self.char
            self.op = avg_pc

    def forward(self, x):
        assert x.shape[1] ==
        return self.op(x)


for level, mult in enumerate(channel_mult):
    for _ in range(num_res_blocks):
        layers = [
            ResBlock(
                ch,
                time_embed_dim,
                dropout,
                out_channels=int(mult * model_channels),
                dims=dims,
                use_checkpoint=use_checkpoint,
                use_scale_shift_norm=use_scale_shift_norm,
            )
        ]
        ch = int(mult * model_channels)
        if ds in attention_resolutions:
            layers.append(
                AttentionBlock(
                    ch,
                    use_checkpoint=use_checkpoint,
                    num_heads=num_heads,
                    num_head_channels=num_head_channels,
                    use_new_attention_order=use_new_attention_order,
                )
            )
        self.input_blocks.append(TimestepEmbedSequential(*layers))
        self._feature_size += ch
        input_block_chans.append(ch)
    if level != len(channel_mult) - 1:
        out_ch = ch
        self.input_blocks.append(
            TimestepEmbedSequential(
                ResBlock(
                    ch,
                    time_embed_dim,
                    dropout,
                    out_channels=out_ch,
                    dims=dims,
                    use_checkpoint=use_checkpoint,
                    use_scale_shift_norm=use_scale_shift_norm,
                    down=True,
                )
                if resblock_updown
                else Downsample(
                    ch, conv_resample, dims=dims, out_channels=out_ch
                )
            )
        )
        ch = out_ch
        input_block_chans.append(ch)
        ds *= 2
        self._feature_size += ch
```

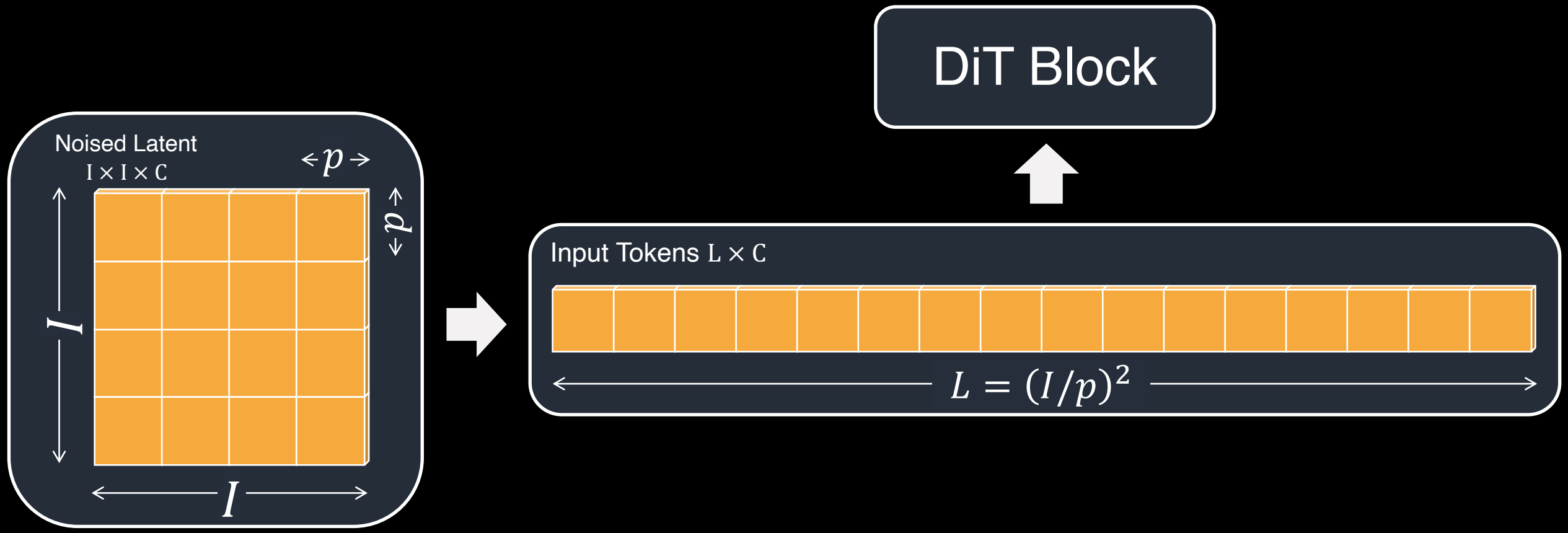
ADM/LDM U-Net Block

Scaling DiT

DiT Design Space: Patch Size

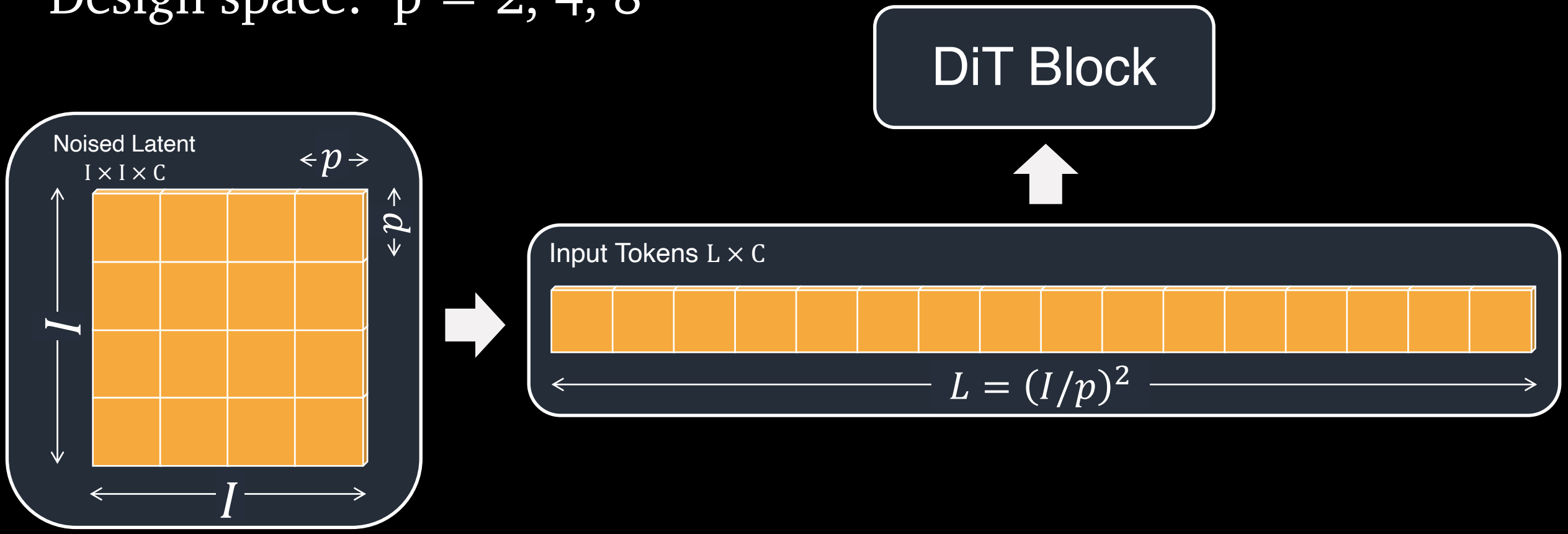


DiT Design Space: Patch Size



DiT Design Space: Patch Size

Design space: $p = 2, 4, 8$



DiT Design Space: Model Size

Model	Layers N	Hidden size d	Heads	Gflops (I=32, p=4)
DiT-S	12	384	6	1.4
DiT-B	12	768	12	5.6
DiT-L	24	1024	16	19.7
DiT-XL	28	1152	16	29.1

DiT Design Space: Model Size

Design space: S, B, L, XL model configs

Model	Layers N	Hidden size d	Heads	Gflops (I=32, p=4)
DiT-S	12	384	6	1.4
DiT-B	12	768	12	5.6
DiT-L	24	1024	16	19.7
DiT-XL	28	1152	16	29.1

DiT Design Space

Block design: adaLN, cross-attention, in-context blocks

Patch size: 2, 4, 8

Model size: S, B, L, XL configs

Experiments

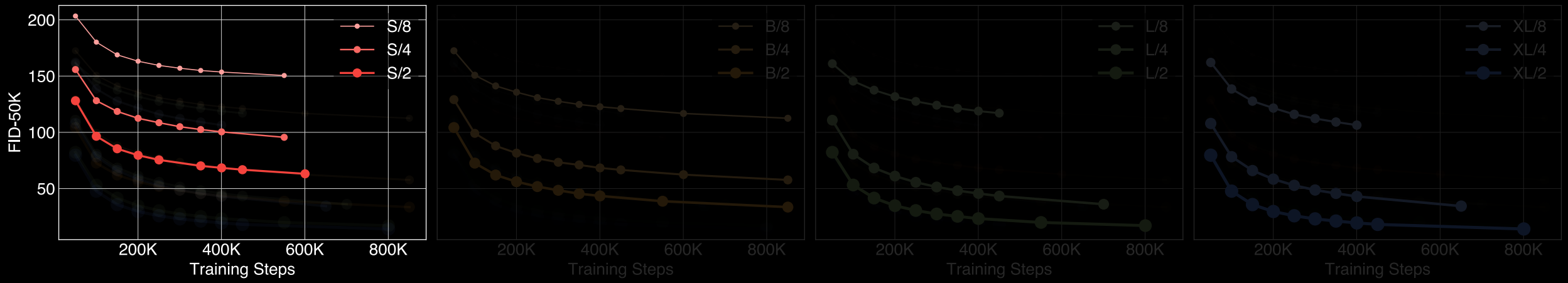
DiT Training Recipe

Training hyperparameters are almost entirely retained from ADM, and identical across all model sizes and patch sizes.

Did not tune learning rates, decay/warm-up schedules, Adam $\beta 1/\beta 2$ or weight decays.

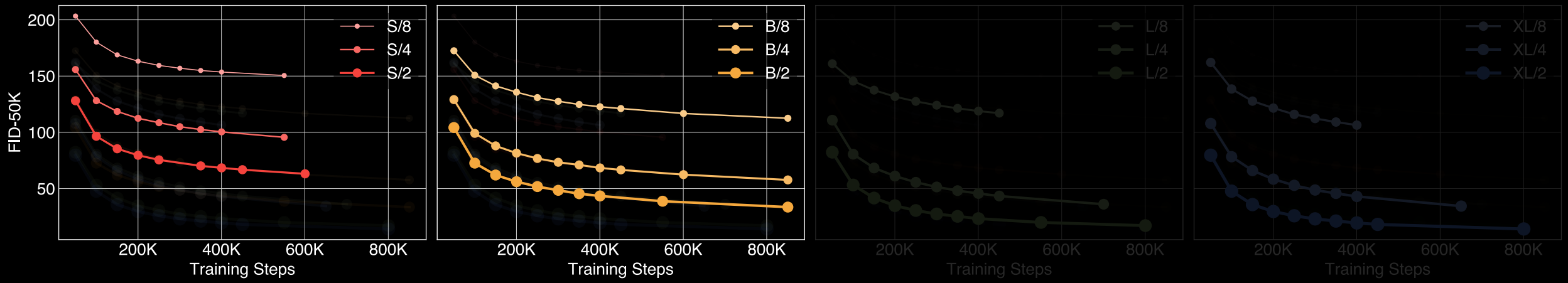
Stability issues with U-Net.

More compute = better generative models



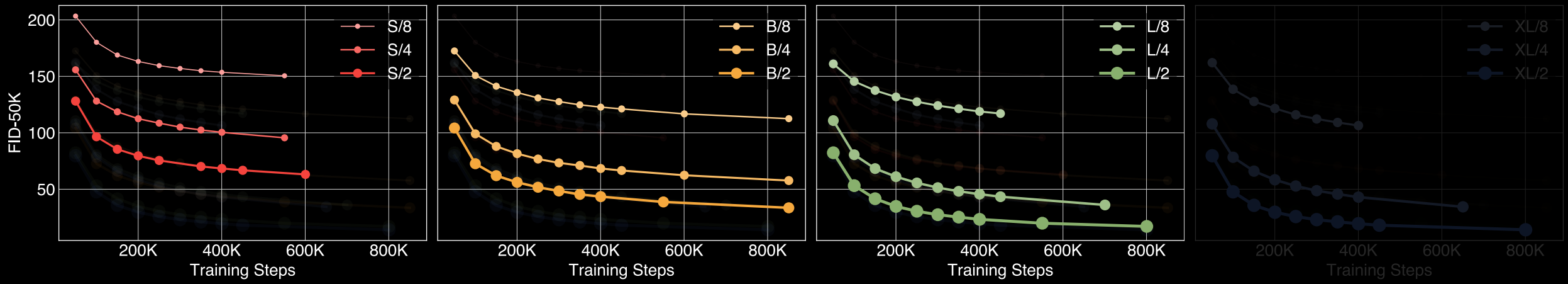
More tokens

More compute = better generative models



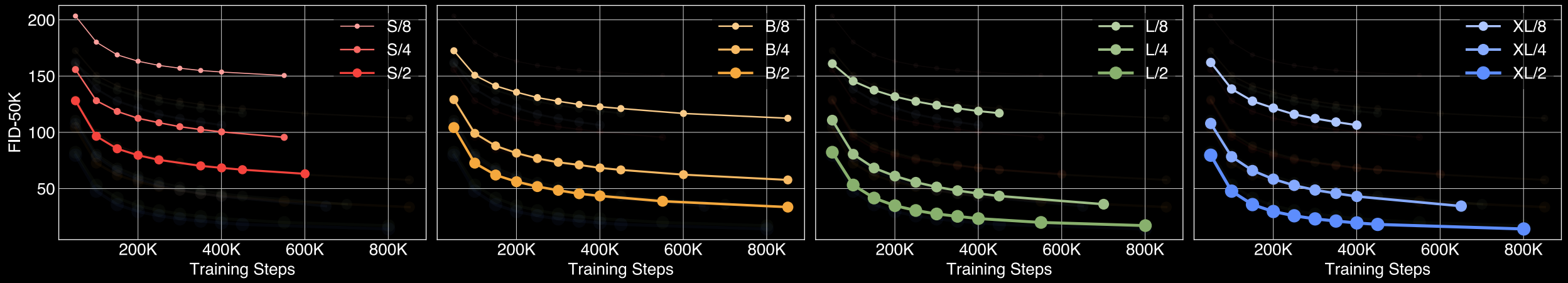
More tokens

More compute = better generative models



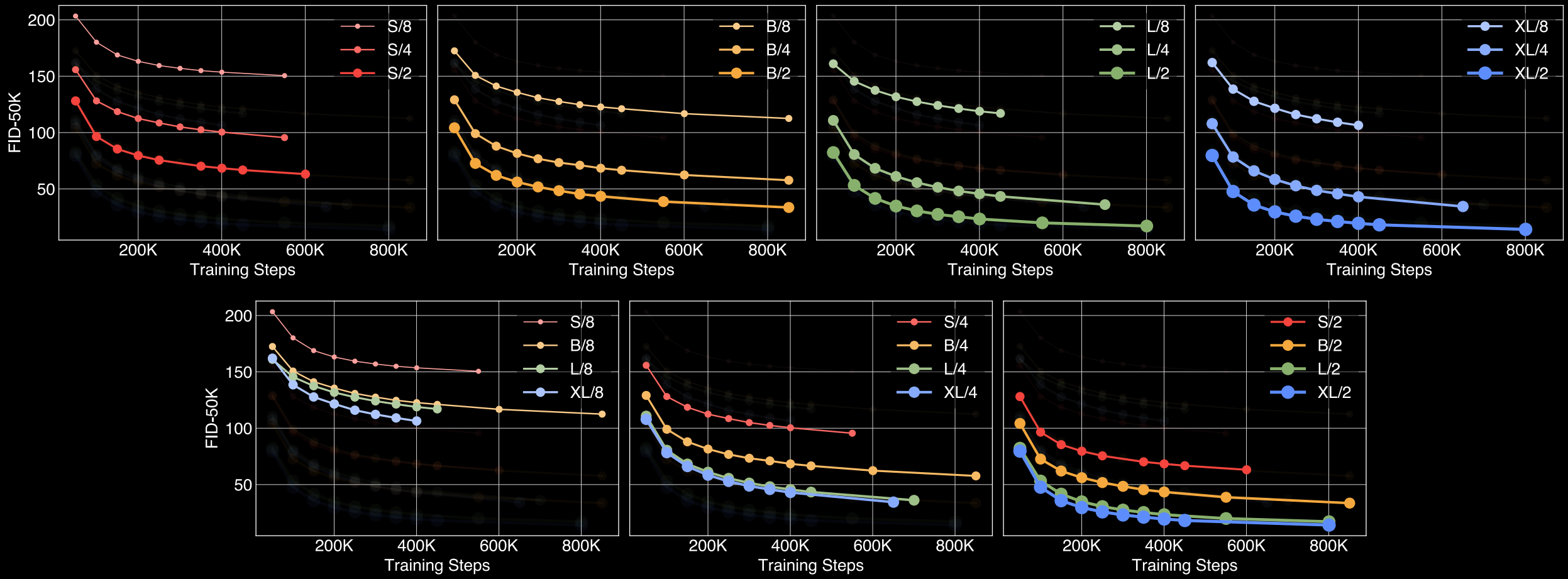
More tokens

More compute = better generative models



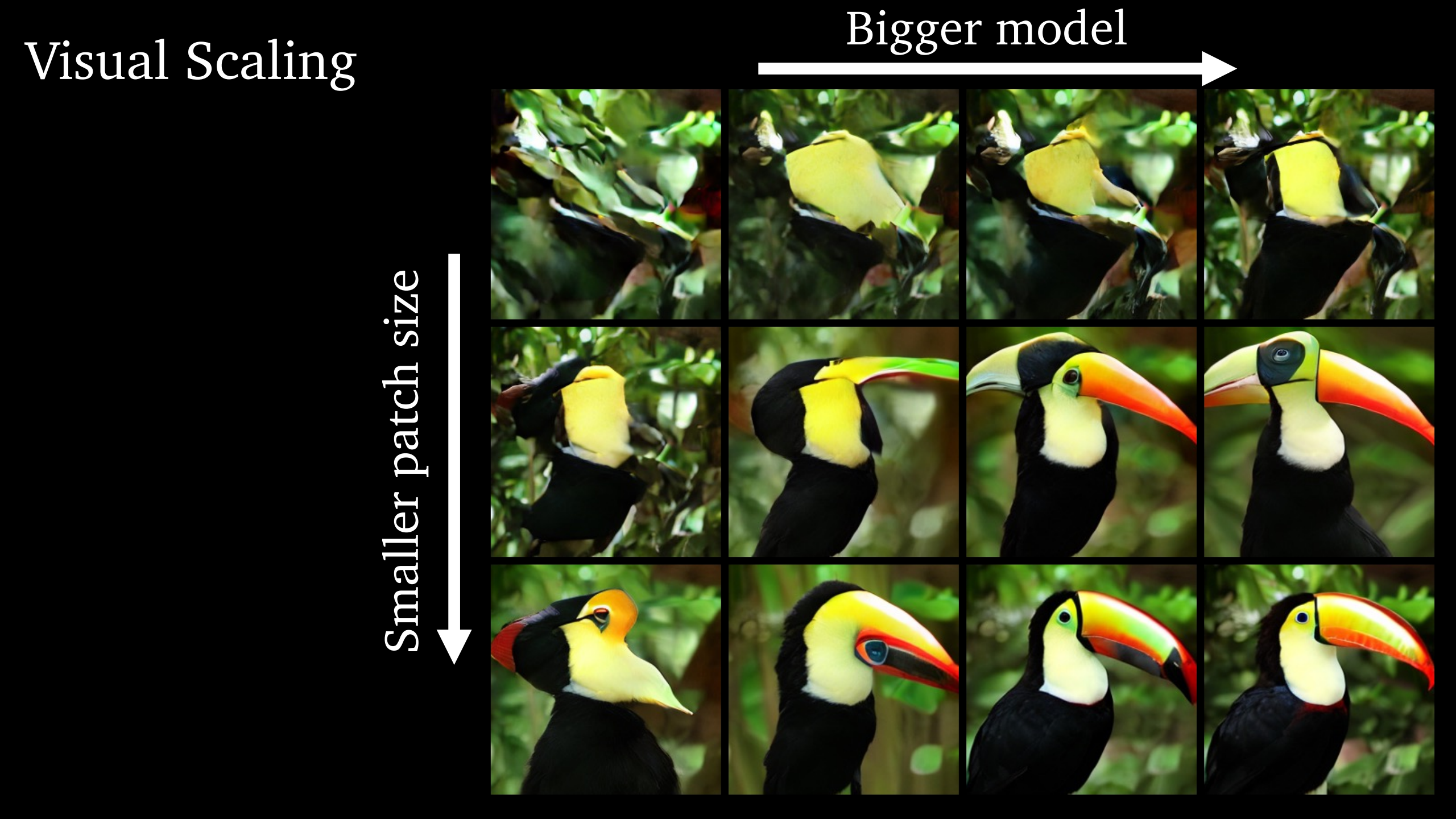
More tokens

More compute = better generative models

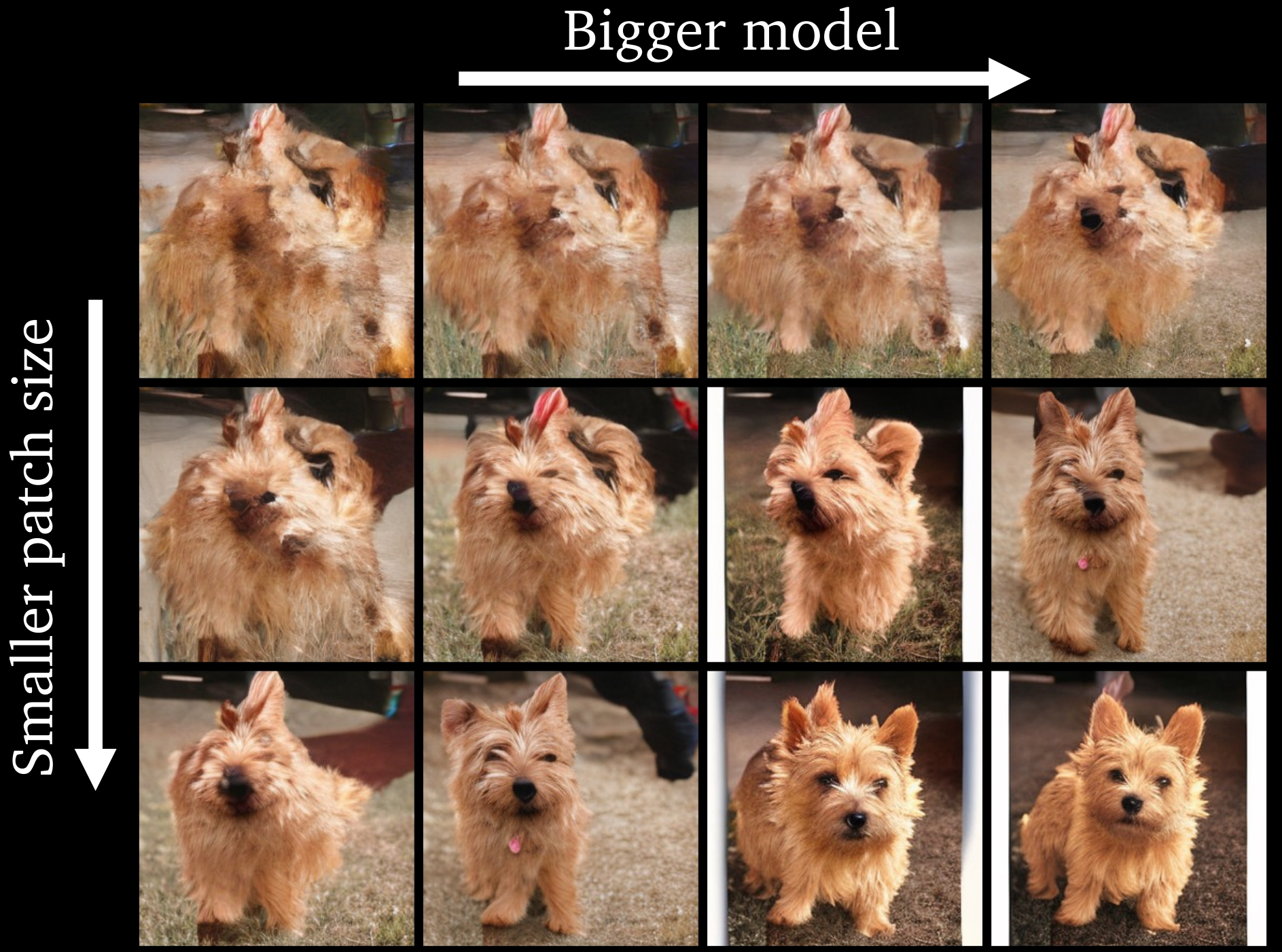


Bigger DiTs

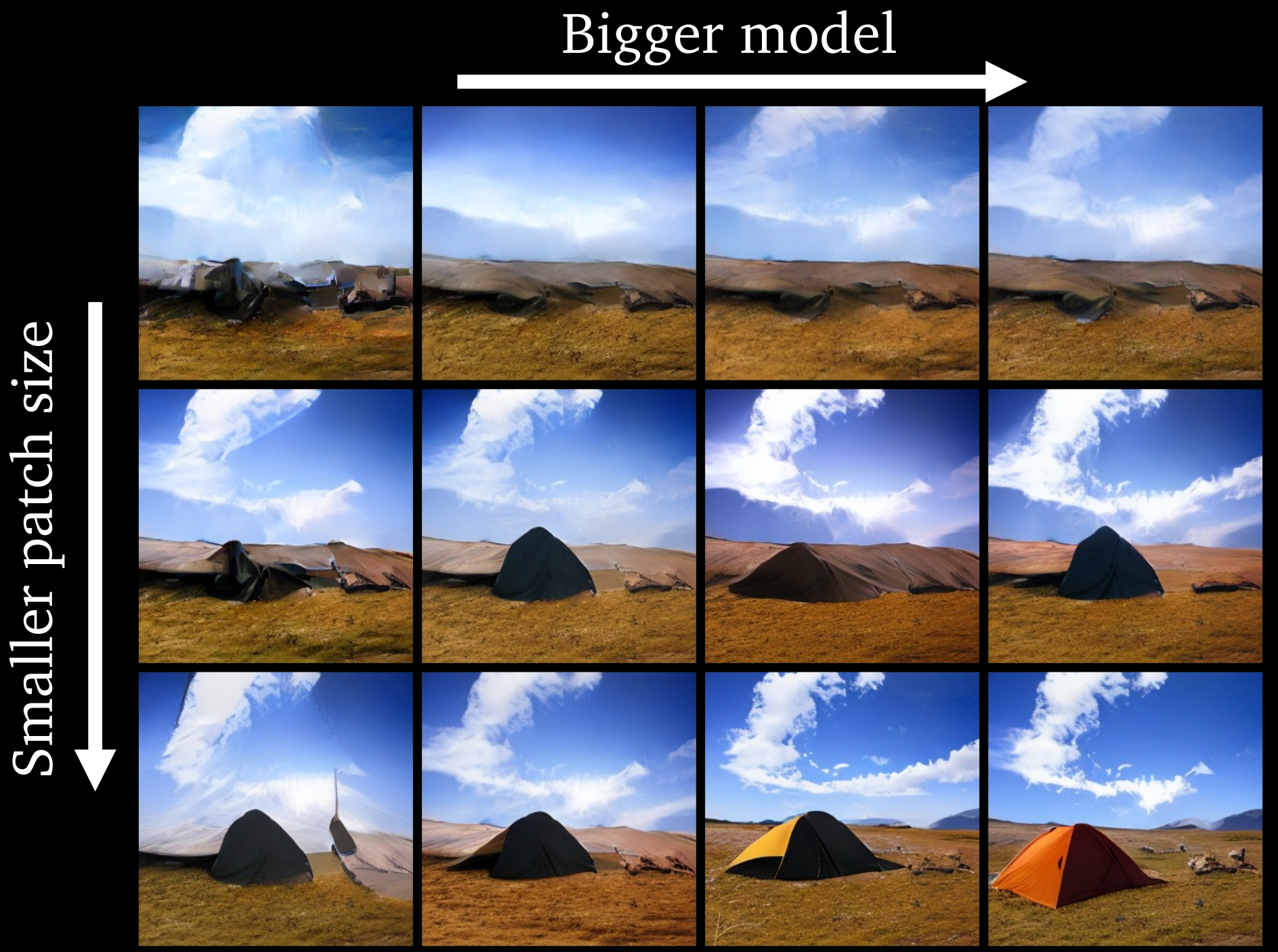
Visual Scaling



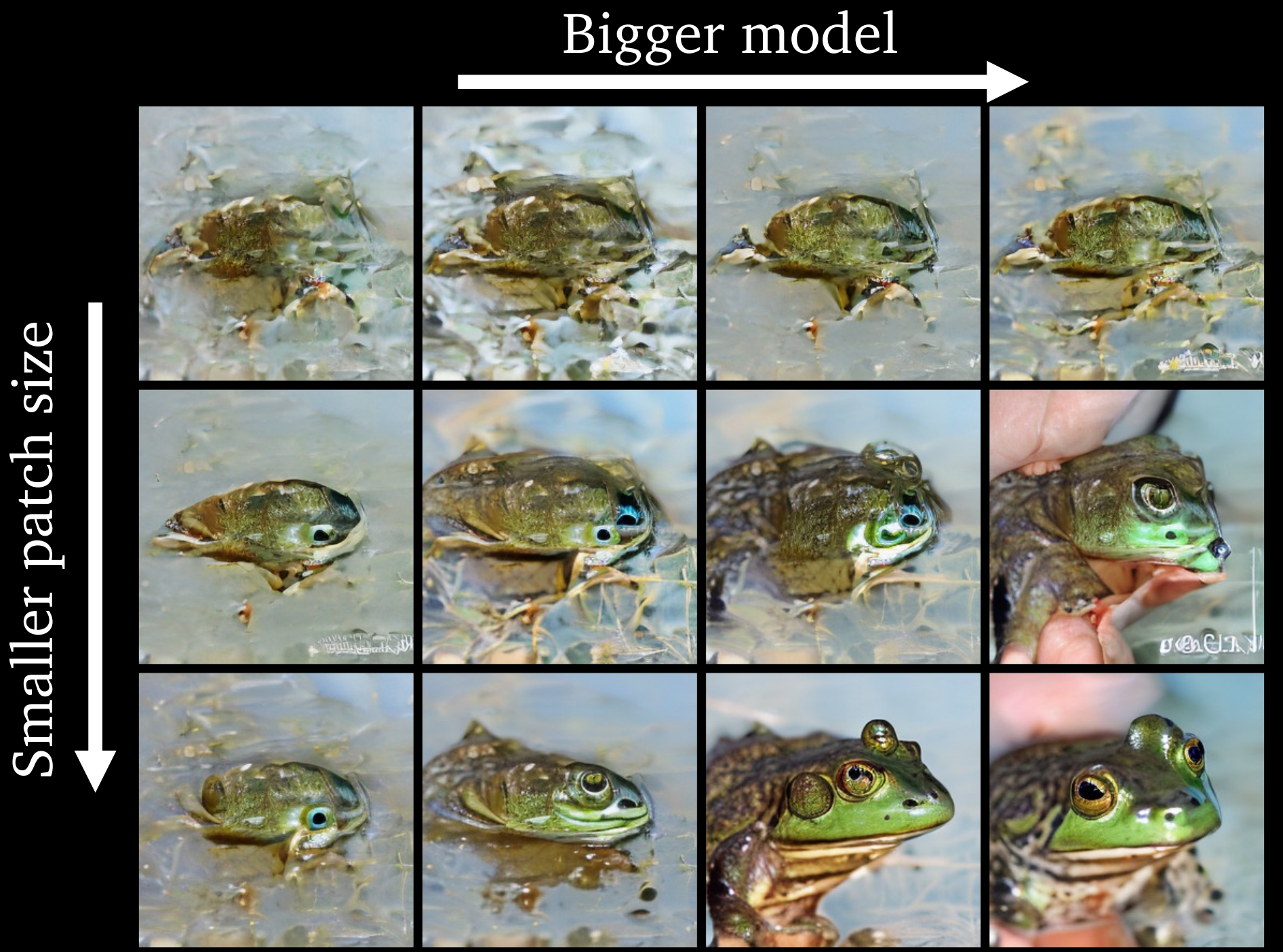
Visual Scaling



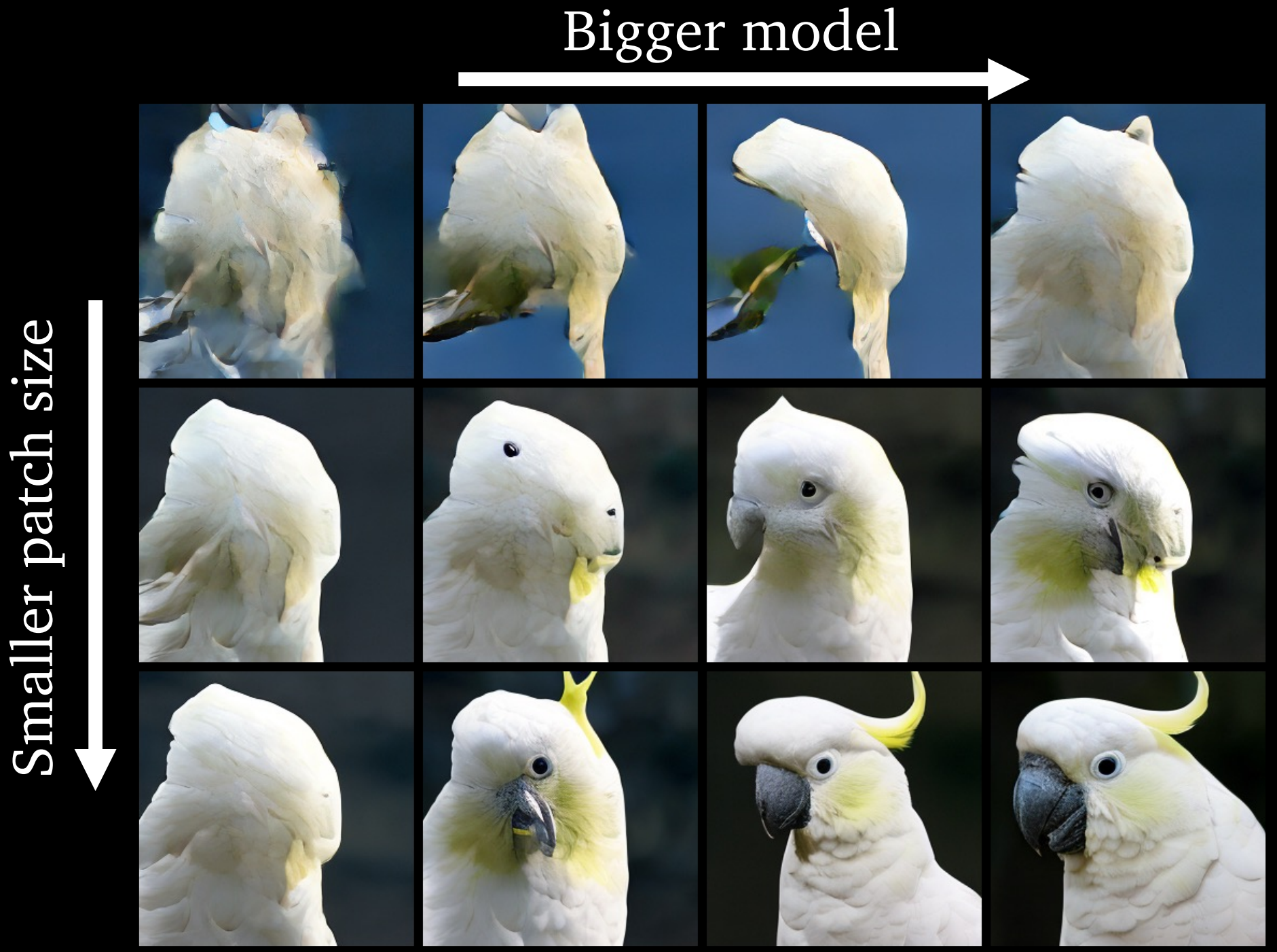
Visual Scaling



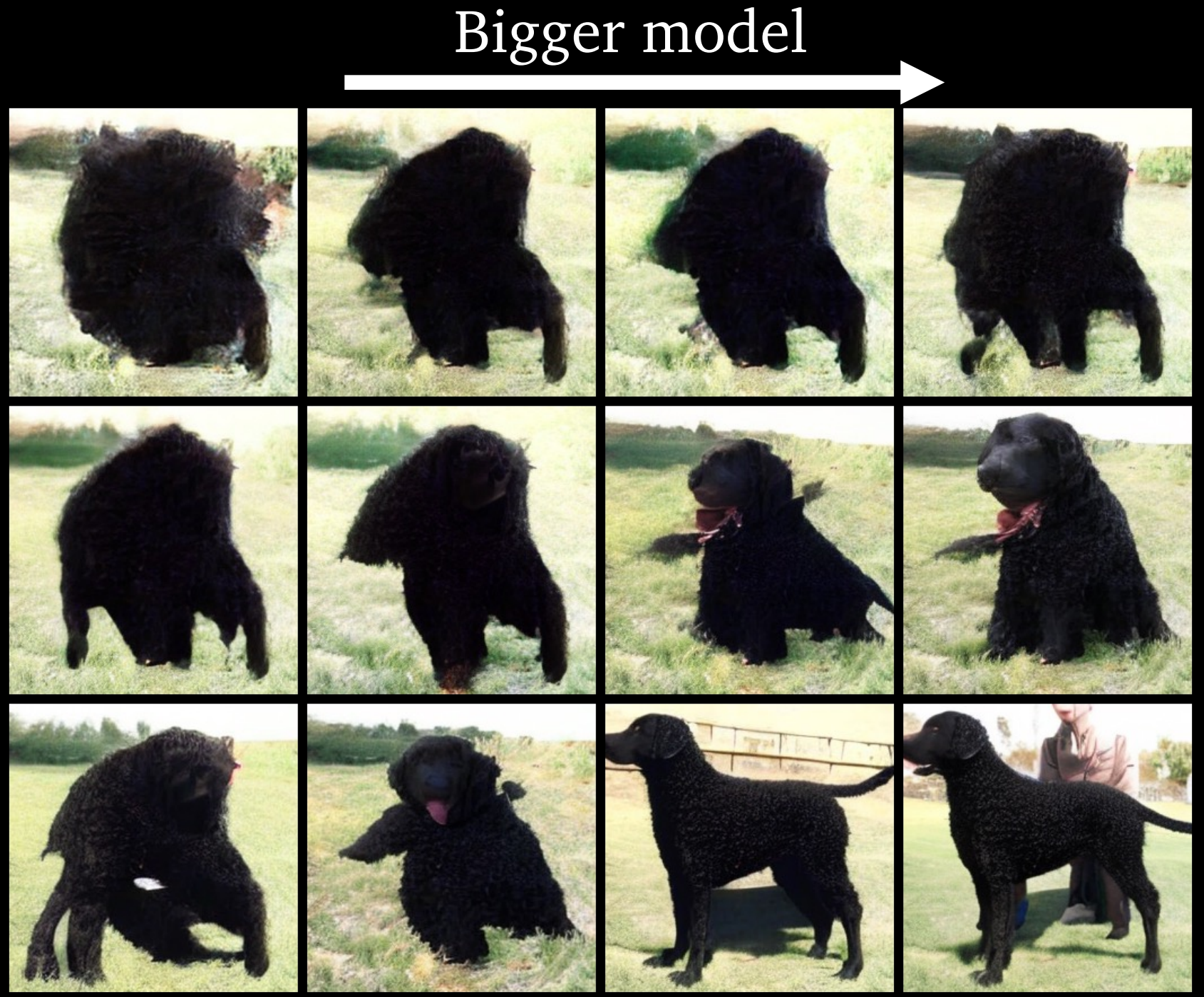
Visual Scaling



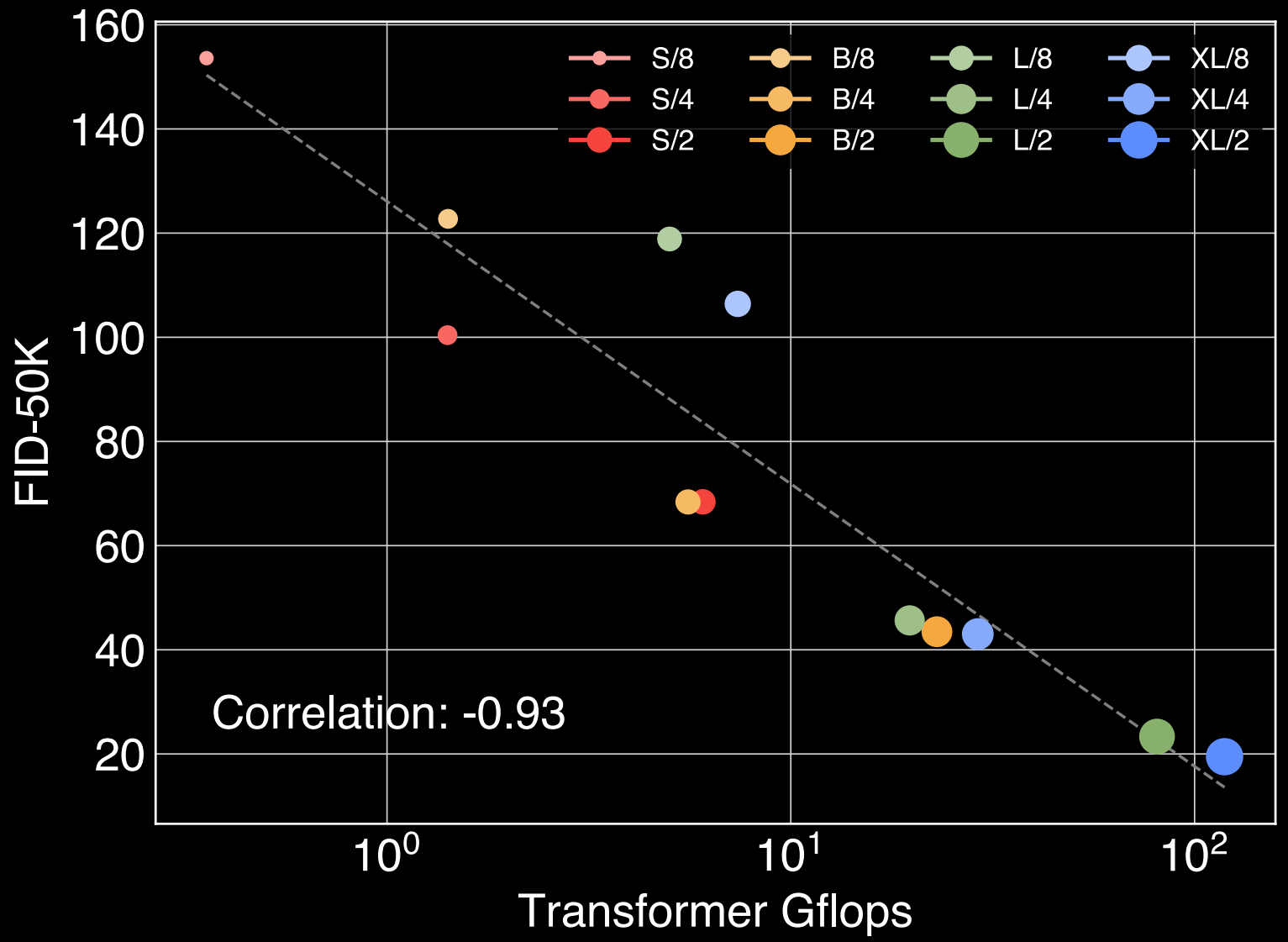
Visual Scaling



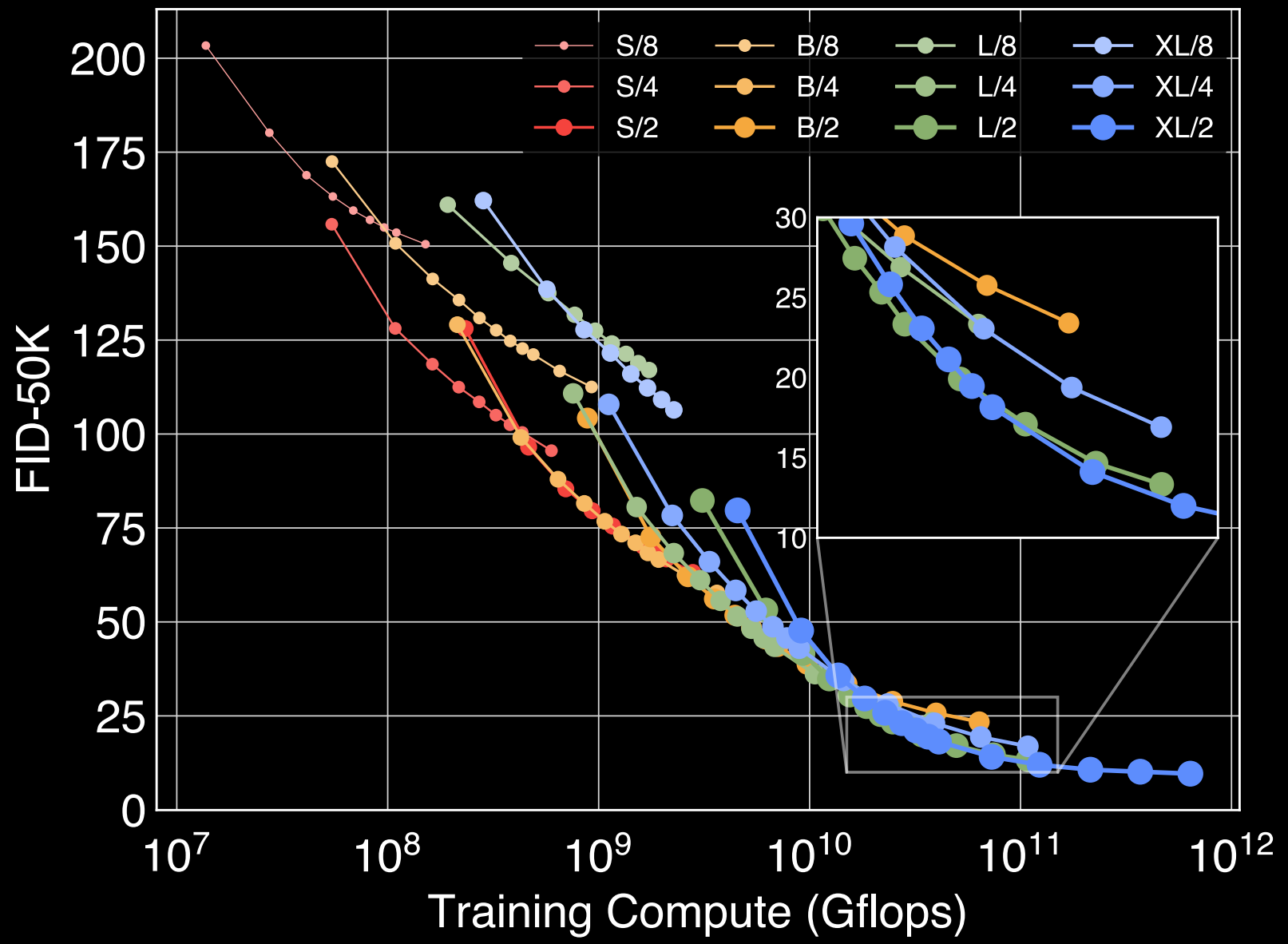
Visual Scaling



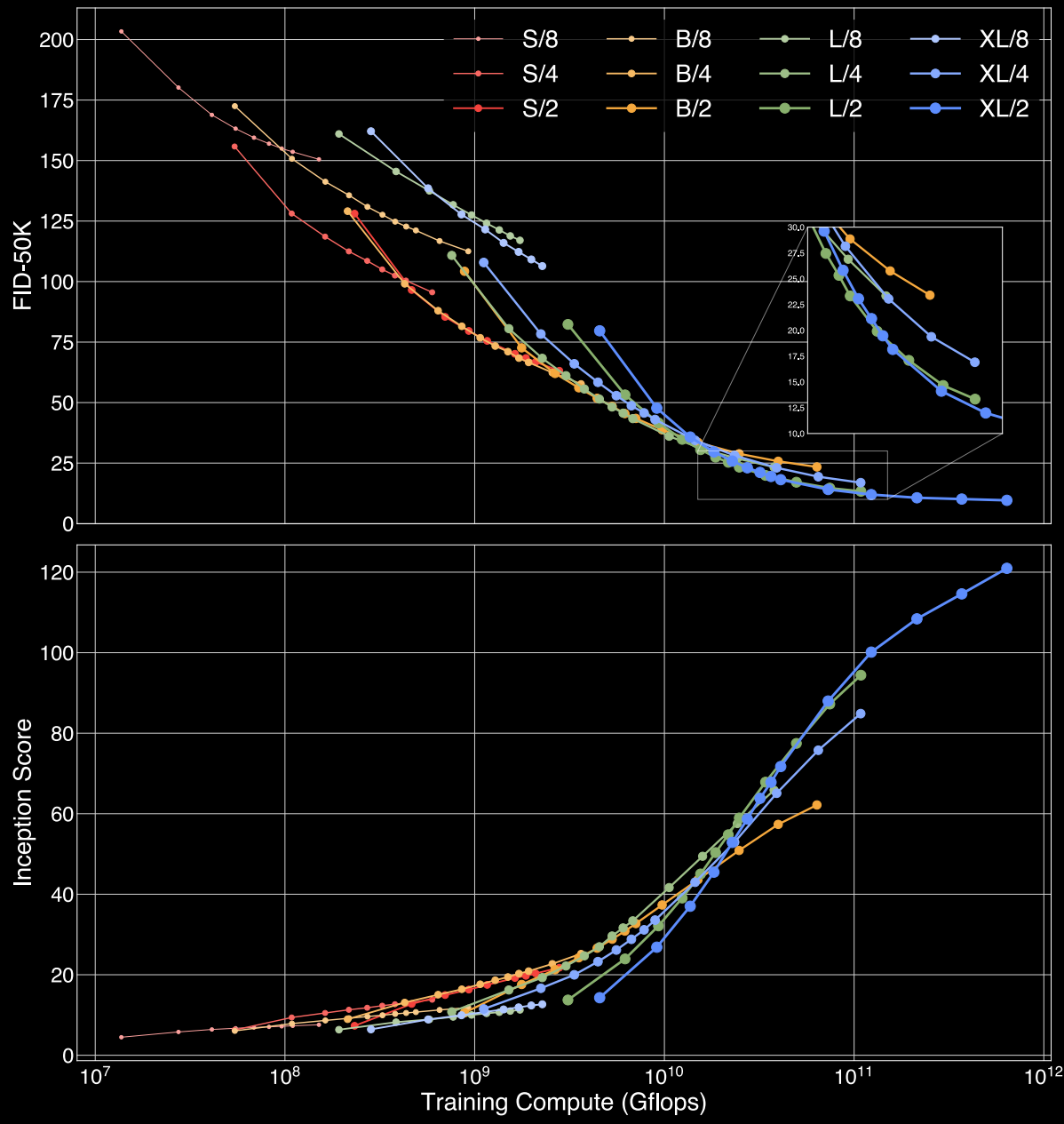
Compute is all that matters

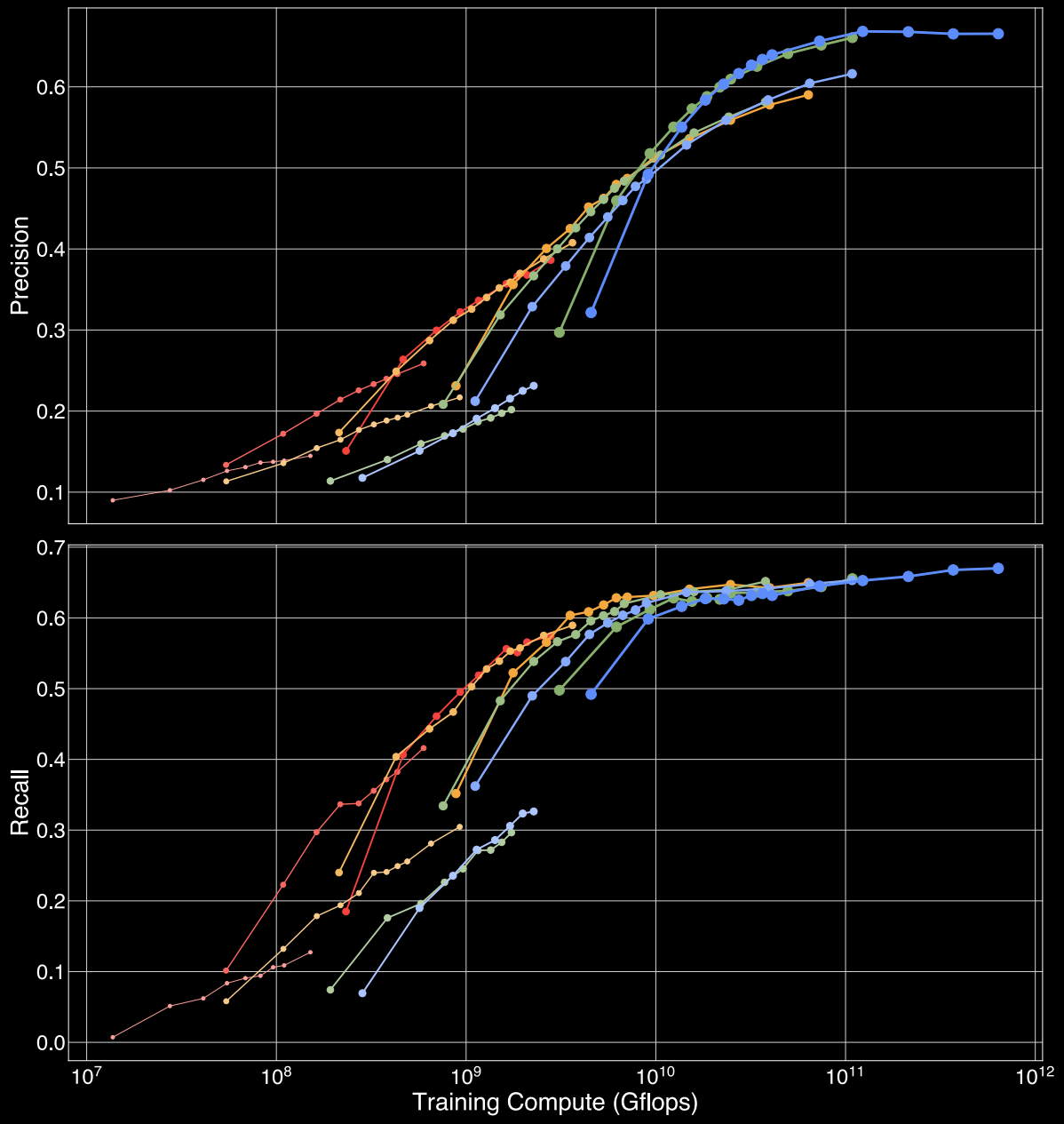


Compute is all that matters

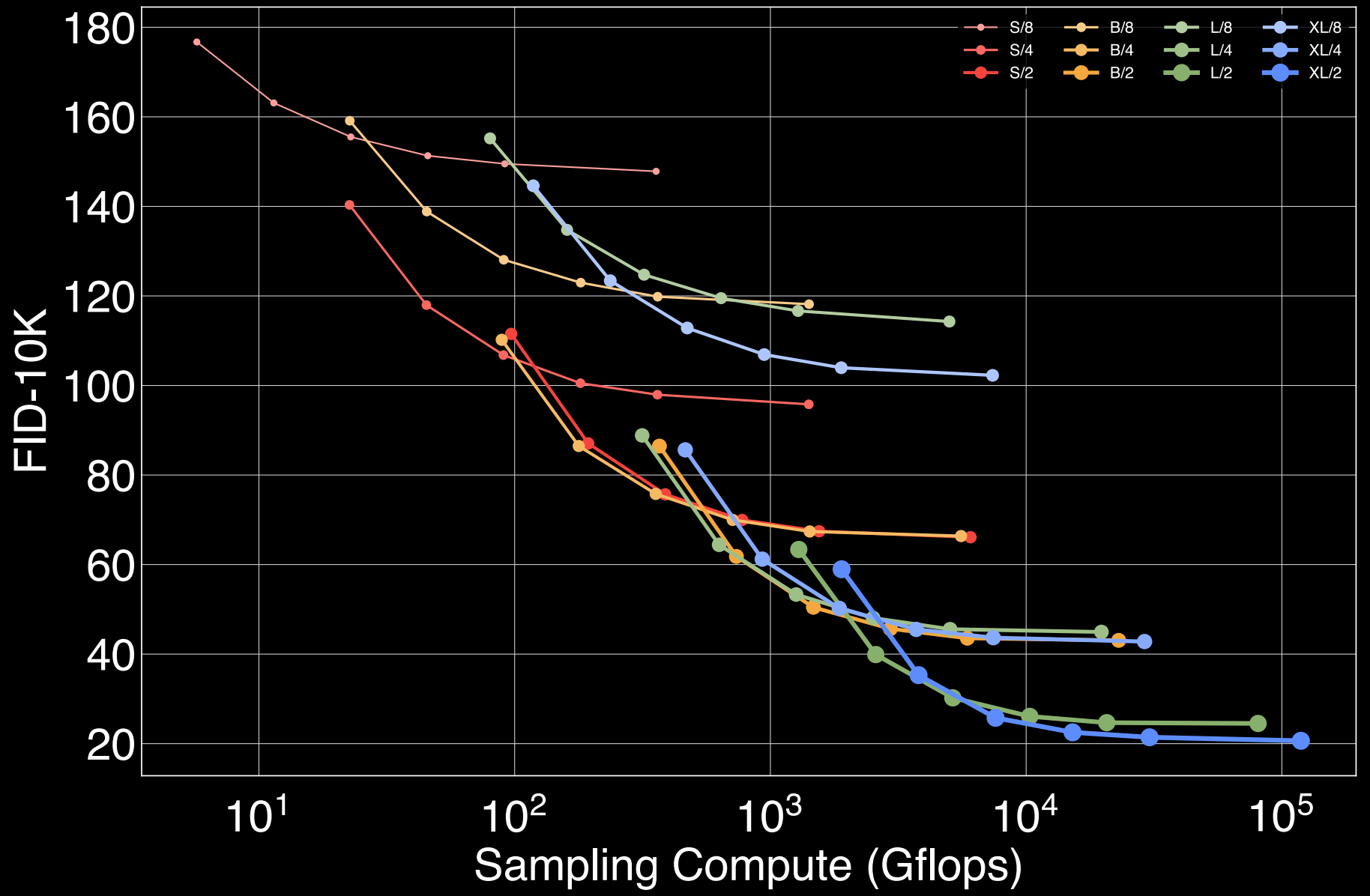


Compute is all that matters





Sampling compute **cannot** compensate for model compute



DiT-XL/2 512x512 Samples

DiT-XL/2 512x512: High Guidance






DiT-XL/2 512x512: High Guidance






DiT-XL/2 512x512: High Guidance






DiT-XL/2 512x512: High Guidance






DiT-XL/2 512x512: High Guidance






DiT-XL/2 256x256: Effects of Guidance Scale

s = 1.5

s = 2.0

s = 4.0

s = 6.0

DiT Latent Walk (512x512)

DiT Class Embedding Walk (512x512)

DiT Class Embedding Walk (512x512)

DiT Dogball

BigGAN Dogball

Comparison to SOTA generative models

Class-Conditional ImageNet 256×256					
Model	FID↓	sFID↓	IS↑	Precision↑	Recall↑
BigGAN-deep [2]	6.95	7.36	171.4	0.87	0.28
StyleGAN-XL [51]	2.30	4.02	265.12	0.78	0.53
ADM [9]	10.94	6.02	100.98	0.69	0.63
ADM-U	7.49	5.13	127.49	0.72	0.63
ADM-G	4.59	5.25	186.70	0.82	0.52
ADM-G, ADM-U	3.94	6.14	215.84	0.83	0.53
CDM [20]	4.88	-	158.71	-	-
LDM-8 [46]	15.51	-	79.03	0.65	0.63
LDM-8-G	7.76	-	209.52	0.84	0.35
LDM-4	10.56	-	103.49	0.71	0.62
LDM-4-G (cfg=1.25)	3.95	-	178.22	0.81	0.55
LDM-4-G (cfg=1.50)	3.60	-	247.67	**0.87**	0.48
DiT-XL/2	9.62	6.85	121.50	0.67	**0.67**
DiT-XL/2-G (cfg=1.25)	3.22	5.28	201.77	0.76	0.62
DiT-XL/2-G (cfg=1.50)	**2.27**	**4.60**	**278.24**	0.83	0.57

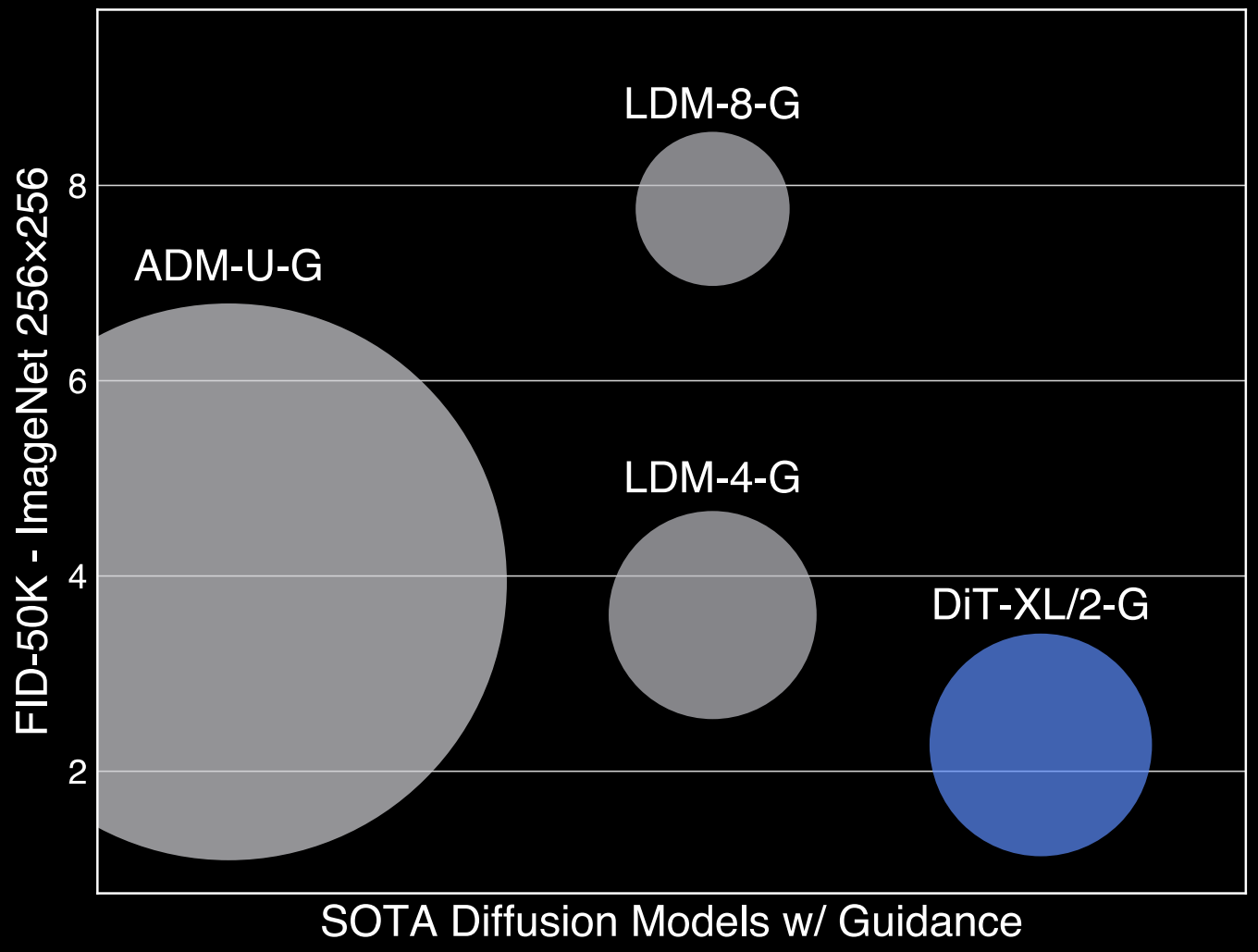
Table 2. **Benchmarking class-conditional image generation on ImageNet 256×256.** DiT-XL/2 achieves state-of-the-art FID.

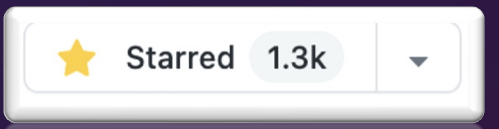
Comparison to SOTA generative models

Class-Conditional ImageNet 512×512					
Model	FID↓	sFID↓	IS↑	Precision↑	Recall↑
BigGAN-deep [2]	8.43	8.13	177.90	0.88	0.29
StyleGAN-XL [51]	2.41	4.06	267.75	0.77	0.52
ADM [9]	23.24	10.19	58.06	0.73	0.60
ADM-U	9.96	5.62	121.78	0.75	**0.64**
ADM-G	7.72	6.57	172.71	**0.87**	0.42
ADM-G, ADM-U	3.85	5.86	221.72	0.84	0.53
DiT-XL/2	12.03	7.12	105.25	0.75	**0.64**
DiT-XL/2-G (cfg=1.25)	4.64	5.77	174.77	0.81	0.57
DiT-XL/2-G (cfg=1.50)	**3.04**	**5.02**	**240.82**	0.84	0.54

Table 3. **Benchmarking class-conditional image generation on ImageNet 512×512.** Note that prior work [9] measures Precision and Recall using 1000 real samples for 512 × 512 resolution; for consistency, we do the same.

Comparison to SOTA generative models

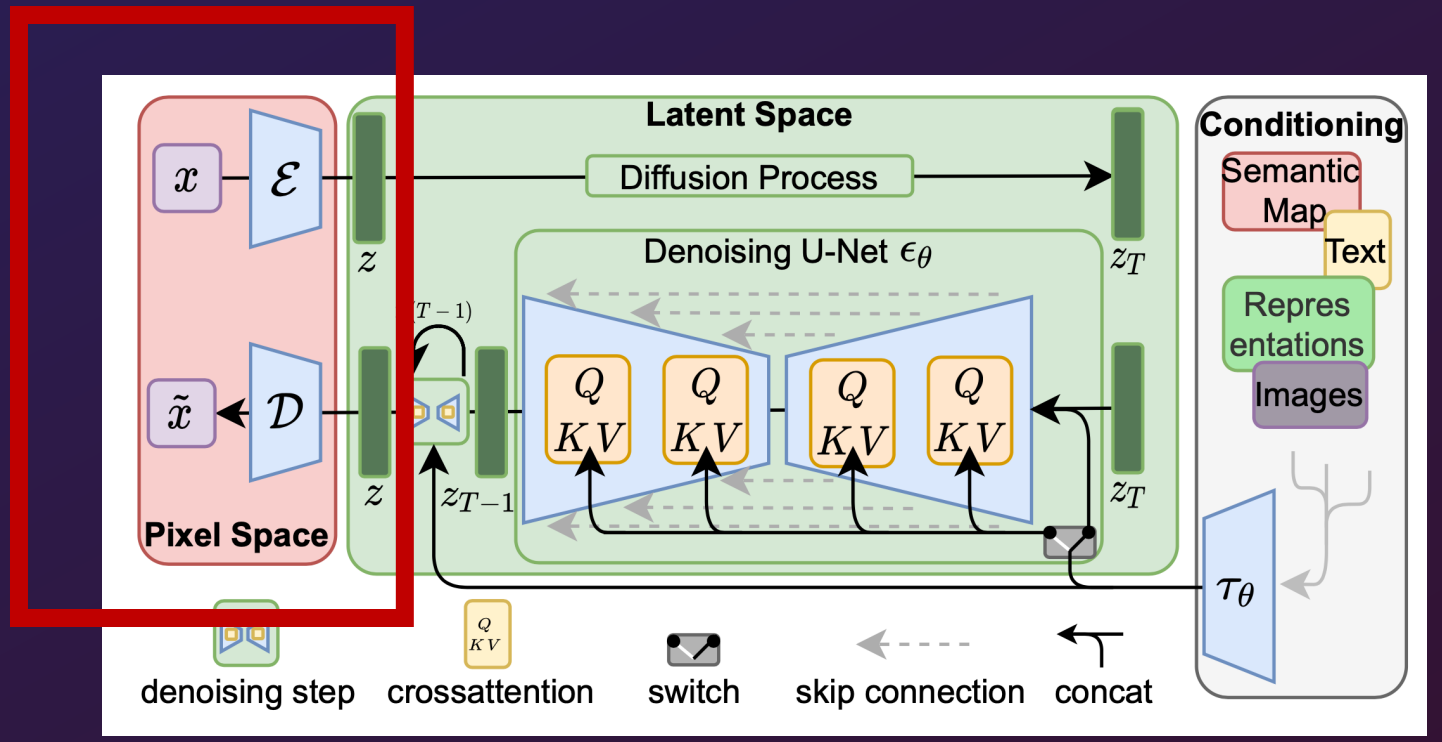




Code: github.com/facebookresearch/DiT

Also available in the Hugging Face `diffusers` library

Plot Twist:
DiT is also a hybrid architecture!



ConvNet vs. Transformer
ROUND 2 Winner:

Goes to the simple, standardized and scalable architecture, that dances perfectly to the rhythm of YOUR task!

Okay, But why I see people are still using UNet?

DiT with COCO captions

a giraffe standing in front of a grassy plain and blue sky.

a big black bear laying in the grass surrounded by trees.

two images of open suitecases full of toiletries.

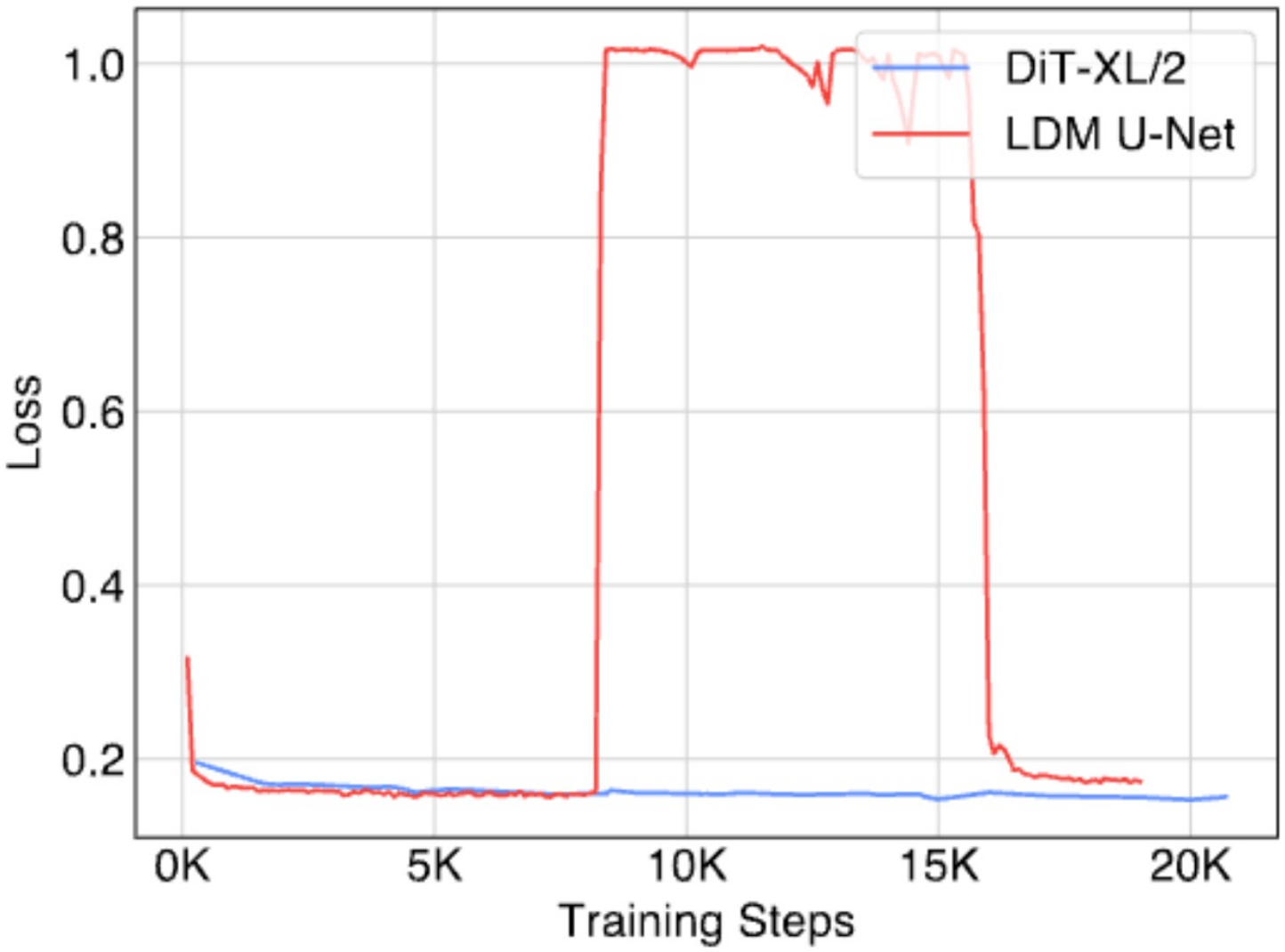


Figure 2. **DiT-XL/2 versus a scaled-up LDM U-Net.**