Inpainting Flu Dynamics Forecasts

Based on *The annotated diffusion model* from Niels Rogge and Kashif Rasul at Hugging Face. More information and annotations are found on `dev/HF-annotated-diffusion-ref.ipynb`

In [1]:
```
import math
from inspect import isfunction
from functools import partial

%matplotlib inline
import matplotlib.pyplot as plt
from tqdm.auto import tqdm
from einops import rearrange

import torch
from torch import nn, einsum
import torch.nn.functional as F

import numpy as np
import pandas as pd
import xarray as xr

import data_utils, data_classes
from torch.utils.data import DataLoader
from torchvision import transforms
import datetime

flusetup = data_utils.FluSetup.from_flusight(fluseason_startdate=pd.to_da
flusetup = data_utils.FluSetup.from_flusight(fluseason_startdate=pd.to_da

image_size = 64
channels = 1
batch_size = 128 * torch.cuda.device_count()
```

```
Spatial Setup with 53 locations.
Spatial Setup with 51 locations.
```

In [ ]:
```
```

Prepare ground-truth for inpainting

In [2]:
```
# if True, take a training data sample as gt:
if False:
    inpaintfrom_idx = 19
    gt_keep_mask = np.ones((channels,image_size,image_size))
    gt_keep_mask[:,inpaintfrom_idx:,:] = 0
    # mask is ones for the known pixels, and zero for the ones to be infe
    gt = data.getitem_nocast(4)
    print(gt.shape)
    show_tensor_image(gt, place=ipl)
    plt.show()
```

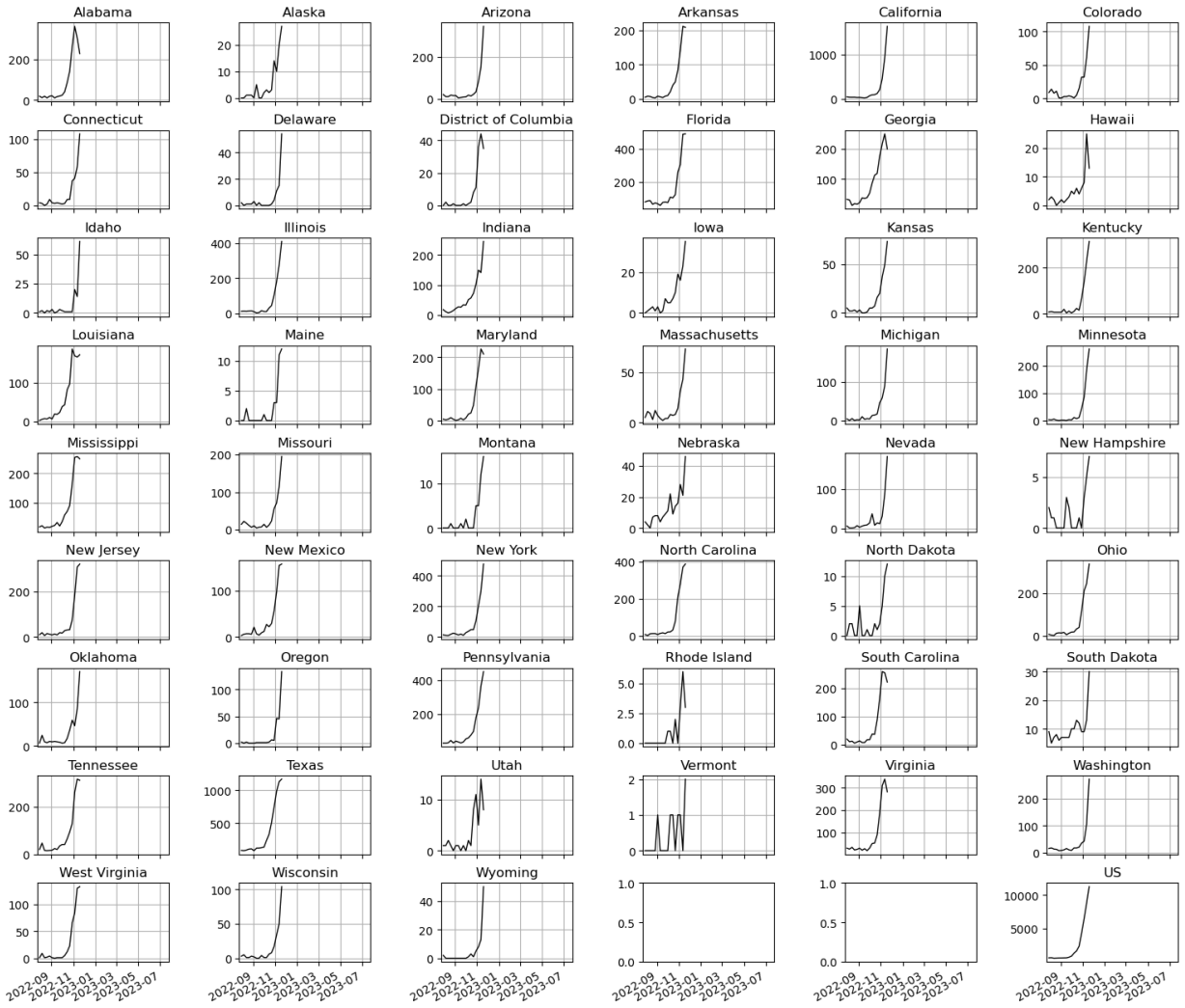
In [3]:
```
#flusetup = data_utils.FluSetup.from_flusight(fluseason_startdate=pd.to_d
flusight = data_utils.get_from_epidata(dataset="flusight", flusetup=fluse
```

⚠️ ⚠️ ⚠️ Make sure ./update_data.sh is ran AND that the fork is updated

In [4]:
```
gt_df = flusight.copy()
gt_df = gt_df[gt_df["fluseason"] == 2022]
gt_df = gt_df[gt_df["location_code"].isin(flusetup.locations)]
```

In [ ]:
```
```

In [5]:
```
fig, axes = plt.subplots(9, 6, sharex=True, figsize=(15,15))
gt_piv  = gt_df.pivot(index = "week_enddate", columns='location_code', va
for idx, pl in enumerate(gt_piv.columns):
    ax = axes.flat[idx]
    ax.plot(gt_piv[pl], lw=1, color='k')
    ax.set_title(flusetup.get_location_name(pl))
    ax.grid()
    ax.set_xlim(flusetup.fluseason_startdate, flusetup.fluseason_startdat
    #ax.set_xticks(flusetup.get_dates(52).resample("M"))
    #ax.plot(pd.date_range(flusetup.fluseason_startdate, flusetup.fluseas

ax = axes.flat[-1]
ax.plot(gt_piv[flusetup.locations].sum(axis=1), color="black", linewidth=
ax.set_title("US")
fig.tight_layout()
fig.autofmt_xdate()
```



In [6]:
```
gt_xarr = data_utils.dataframe_to_xarray(gt_df, flusetup=flusetup,
            xarray_name = "gt_flusight_incidHosp",
            xarrax_features = "incidHosp")
print(gt_xarr.shape)
```

```
(1, 64, 64)
```

In [7]:
```
inpaintfrom_idx = len(gt_df.week_enddate.unique())
print(f"{inpaintfrom_idx} weeks already in data, inpainting the next ones
```

```
17 weeks already in data, inpainting the next ones
```

In [8]:
```
gt_df.week_enddate.unique()
```

Out[8]:
```
array(['2022-07-30T00:00:00.000000000', '2022-08-06T00:00:00.000000000',
       '2022-08-13T00:00:00.000000000', '2022-08-20T00:00:00.000000000',
       '2022-08-27T00:00:00.000000000', '2022-09-03T00:00:00.000000000',
       '2022-09-10T00:00:00.000000000', '2022-09-17T00:00:00.000000000',
       '2022-09-24T00:00:00.000000000', '2022-10-01T00:00:00.000000000',
       '2022-10-08T00:00:00.000000000', '2022-10-15T00:00:00.000000000',
       '2022-10-22T00:00:00.000000000', '2022-10-29T00:00:00.000000000',
       '2022-11-05T00:00:00.000000000', '2022-11-12T00:00:00.000000000',
       '2022-11-19T00:00:00.000000000'], dtype='datetime64[ns]')
```

Define a PyTorch Dataset + DataLoader

In [9]:
```
data = data_classes.FluDataset.from_fluview(
                                            flusetup=flusetup,
                                            download=False,
                                            #transform=data_classes.transfor
                                            )

data = data_classes.FluDataset.from_csp_SMHR1('datasets/synthetic/CSP_Flu
data.test(6)
```

```
created dataset with max [19.4177], full dataset has shape (13, 1, 64, 64
)
created dataset with max [14603.53538403], full dataset has shape (1199,
1, 64, 64)
test passed: back and forth transformation are ok ✅
```

In [10]:
```
#data = data_classes.FluDataset.from_SMHR1_fluview(flusetup=flusetup, dow
```

In [11]:
```
data.flu_dyn.shape
```

Out[11]:
```
(1199, 1, 64, 64)
```

In [12]:
```
scaling_per_channel = np.sqrt(max(data.max_per_feature, gt_xarr.max(dim=[
scaling_per_channel
```

Out[12]:
```
array([120.84508837])
```

In [13]:
```
## define image transformations (e.g. using torchvision)
transform = transforms.Compose([
                    data_classes.transform_sqrt,
                    transforms.Lambda(lambda t: data_classes.transform_ch
                    transforms.Lambda(lambda t: data_classes.transform_ch
                    transforms.Lambda(lambda t: data_classes.transform_ra
                    transforms.Lambda(lambda t: data_classes.transform_ra

        ])

transform_inv = transforms.Compose([
                    data_classes.transform_sqrt_inv,
                    transforms.Lambda(lambda t: data_classes.transform_ch
                    transforms.Lambda(lambda t: data_classes.transform_ch
        ][::-1])      # important reverse the sequence
```

In [14]:
```
data.add_transform(transform=transform, transform_inv=transform_inv, bypa
```

In [15]:
```
sample = data[6]
print(f"There are {len(data)} samples in the dataset, and a single sample
```

```
There are 1199 samples in the dataset, and a single sample has shape torc
h.Size([1, 64, 64])
```

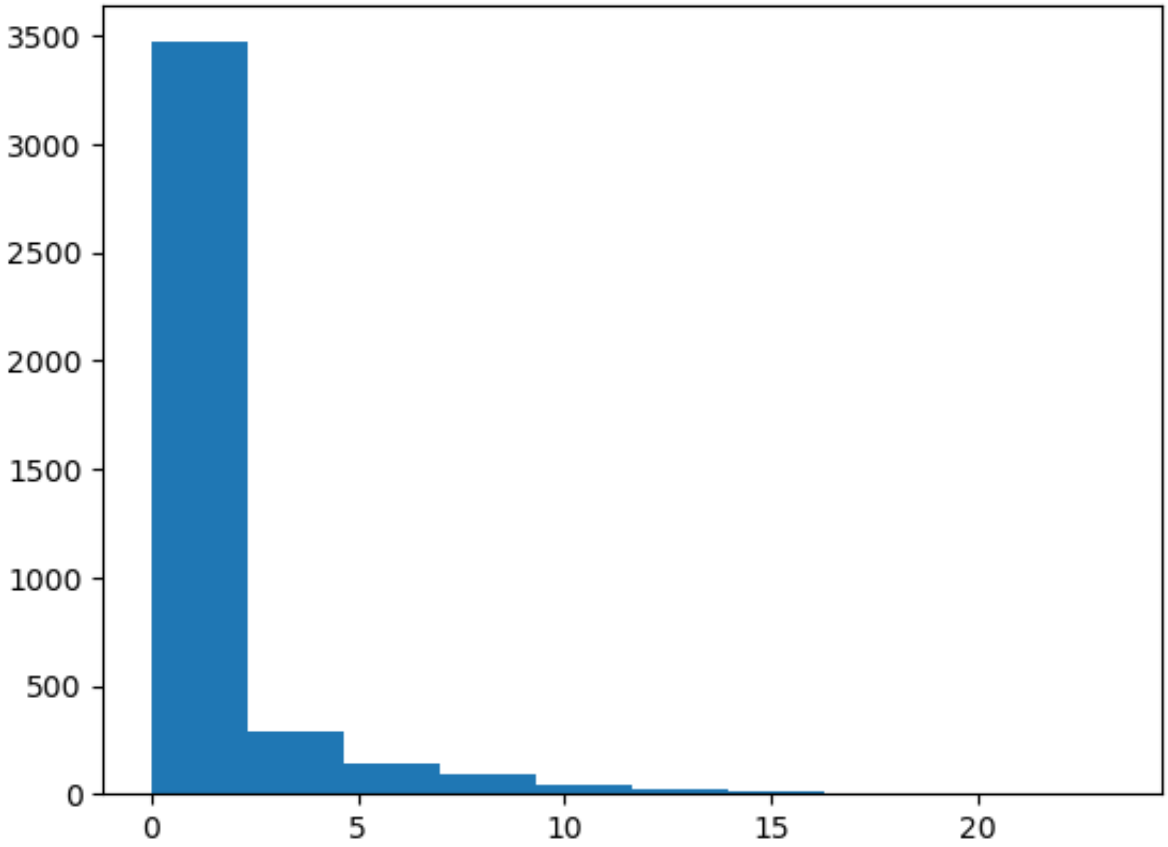
In [16]:
```
dataloader = DataLoader(data, batch_size=batch_size, shuffle=True, drop_l
batch = next(iter(dataloader))
print(f"batch shape {batch.shape}")
```

```
batch shape torch.Size([512, 1, 64, 64])
```

Next, we define a function which we'll apply on-the-fly on the entire dataset. We use the `with_transform` functionality for that. The function just applies some basic image preprocessing: random horizontal flips, rescaling and finally make them have values in the $[-1,1]$ range.

In [17]:
```
plt.hist(data.getitem_nocast(4).flatten())
```

Out[17]:
```
(array([3468.,  292.,  142.,   92.,   47.,   21.,   15.,    7.,    7.,
           5.]),
 array([ 0.        ,  2.33011891,  4.66023782,  6.99035673,  9.32047564,
        11.65059456, 13.98071347, 16.31083238, 18.64095129, 20.9710702 ,
        23.30118911]),
 <BarContainer object of 10 artists>)
```



In [ ]:
```
```

The neural network

Network helpers

First, we define some helper functions and classes which will be used when implementing the neural network. Importantly, we define a `Residual` module, which simply adds the input to the output of a particular function (in other words, adds a residual connection to a particular function).

We also define aliases for the up- and downsampling operations.

In [18]:
```
def exists(x):
    return x is not None

def default(val, d):
    if exists(val):
        return val
    return d() if isfunction(d) else d

class Residual(nn.Module):
    def __init__(self, fn):
        super().__init__()
        self.fn = fn

    def forward(self, x, *args, **kwargs):
        return self.fn(x, *args, **kwargs) + x

def Upsample(dim):
    return nn.ConvTranspose2d(dim, dim, 4, 2, 1)

def Downsample(dim):
    return nn.Conv2d(dim, dim, 4, 2, 1)
```

Position embeddings

The `SinusoidalPositionEmbeddings` module takes a tensor of shape `(batch_size, 1)` as input (i.e. the noise levels of several noisy images in a batch), and turns this into a tensor of shape `(batch_size, dim)`, with `dim` being the dimensionality of the position embeddings. This is then added to each residual block, as we will see further.

In [19]:
```
class SinusoidalPositionEmbeddings(nn.Module):
    def __init__(self, dim):
        super().__init__()
        self.dim = dim

    def forward(self, time):
        device = time.device
        half_dim = self.dim // 2
        embeddings = math.log(10000) / (half_dim - 1)
        embeddings = torch.exp(torch.arange(half_dim, device=device) * -e
        embeddings = time[:, None] * embeddings[None, :]
        embeddings = torch.cat((embeddings.sin(), embeddings.cos()), dim=
        return embeddings
```

ResNet/ConvNeXT block

Next, we define the core building block of the U-Net model. The DDPM authors employed a Wide ResNet block (Zagoruyko et al., 2016), but Phil Wang decided to also add support for a ConvNeXT block (Liu et al., 2022), as the latter has achieved great success in the image domain. One can choose one or another in the final U-Net architecture.

In [20]:

```python
class Block(nn.Module):
    def __init__(self, dim, dim_out, groups = 8):
        super().__init__()
        self.proj = nn.Conv2d(dim, dim_out, 3, padding = 1)
        self.norm = nn.GroupNorm(groups, dim_out)
        self.act = nn.SiLU()

    def forward(self, x, scale_shift = None):
        x = self.proj(x)
        x = self.norm(x)

        if exists(scale_shift):
            scale, shift = scale_shift
            x = x * (scale + 1) + shift

        x = self.act(x)
        return x

class ResnetBlock(nn.Module):
    """https://arxiv.org/abs/1512.03385"""

    def __init__(self, dim, dim_out, *, time_emb_dim=None, groups=8):
        super().__init__()
        self.mlp = (
            nn.Sequential(nn.SiLU(), nn.Linear(time_emb_dim, dim_out))
            if exists(time_emb_dim)
            else None
        )

        self.block1 = Block(dim, dim_out, groups=groups)
        self.block2 = Block(dim_out, dim_out, groups=groups)
        self.res_conv = nn.Conv2d(dim, dim_out, 1) if dim != dim_out else

    def forward(self, x, time_emb=None):
        h = self.block1(x)

        if exists(self.mlp) and exists(time_emb):
            time_emb = self.mlp(time_emb)
            h = rearrange(time_emb, "b c -> b c 1 1") + h

        h = self.block2(h)
        return h + self.res_conv(x)

class ConvNextBlock(nn.Module):
    """https://arxiv.org/abs/2201.03545"""

    def __init__(self, dim, dim_out, *, time_emb_dim=None, mult=2, norm=T
        super().__init__()
        self.mlp = (
            nn.Sequential(nn.GELU(), nn.Linear(time_emb_dim, dim))
            if exists(time_emb_dim)
            else None
        )

        self.ds_conv = nn.Conv2d(dim, dim, 7, padding=3, groups=dim)

        self.net = nn.Sequential(
```

```
            nn.GroupNorm(1, dim) if norm else nn.Identity(),
            nn.Conv2d(dim, dim_out * mult, 3, padding=1),
            nn.GELU(),
            nn.GroupNorm(1, dim_out * mult),
            nn.Conv2d(dim_out * mult, dim_out, 3, padding=1),
        )

        self.res_conv = nn.Conv2d(dim, dim_out, 1) if dim != dim_out else

    def forward(self, x, time_emb=None):
        h = self.ds_conv(x)

        if exists(self.mlp) and exists(time_emb):
            assert exists(time_emb), "time embedding must be passed in"
            condition = self.mlp(time_emb)
            h = h + rearrange(condition, "b c -> b c 1 1")

        h = self.net(h)
        return h + self.res_conv(x)
```

Attention module

Next, we define the attention module, which the DDPM authors added in between the convolutional blocks. Attention is the building block of the famous Transformer architecture (Vaswani et al., 2017), which has shown great success in various domains of AI, from NLP and vision to protein folding. Phil Wang employs 2 variants of attention: one is regular multi-head self-attention (as used in the Transformer), the other one is a linear attention variant (Shen et al., 2018), whose time- and memory requirements scale linear in the sequence length, as opposed to quadratic for regular attention.

For an extensive explanation of the attention mechanism, we refer the reader to Jay Allamar's wonderful blog post.

In [21]:

```python
class Attention(nn.Module):
    def __init__(self, dim, heads=4, dim_head=32):
        super().__init__()
        self.scale = dim_head**-0.5
        self.heads = heads
        hidden_dim = dim_head * heads
        self.to_qkv = nn.Conv2d(dim, hidden_dim * 3, 1, bias=False)
        self.to_out = nn.Conv2d(hidden_dim, dim, 1)

    def forward(self, x):
        b, c, h, w = x.shape
        qkv = self.to_qkv(x).chunk(3, dim=1)
        q, k, v = map(
            lambda t: rearrange(t, "b (h c) x y -> b h c (x y)", h=self.h
        )
        q = q * self.scale

        sim = einsum("b h d i, b h d j -> b h i j", q, k)
        sim = sim - sim.amax(dim=-1, keepdim=True).detach()
        attn = sim.softmax(dim=-1)

        out = einsum("b h i j, b h d j -> b h i d", attn, v)
        out = rearrange(out, "b h (x y) d -> b (h d) x y", x=h, y=w)
        return self.to_out(out)

class LinearAttention(nn.Module):
    def __init__(self, dim, heads=4, dim_head=32):
        super().__init__()
        self.scale = dim_head**-0.5
        self.heads = heads
        hidden_dim = dim_head * heads
        self.to_qkv = nn.Conv2d(dim, hidden_dim * 3, 1, bias=False)

        self.to_out = nn.Sequential(nn.Conv2d(hidden_dim, dim, 1),
                                    nn.GroupNorm(1, dim))

    def forward(self, x):
        b, c, h, w = x.shape
        qkv = self.to_qkv(x).chunk(3, dim=1)
        q, k, v = map(
            lambda t: rearrange(t, "b (h c) x y -> b h c (x y)", h=self.h
        )

        q = q.softmax(dim=-2)
        k = k.softmax(dim=-1)

        q = q * self.scale
        context = torch.einsum("b h d n, b h e n -> b h d e", k, v)

        out = torch.einsum("b h d e, b h d n -> b h e n", context, q)
        out = rearrange(out, "b h c (x y) -> b (h c) x y", h=self.heads,
        return self.to_out(out)
```

Group normalization

The DDPM authors interleave the convolutional/attention layers of the U-Net with group normalization (Wu et al., 2018). Below, we define a `PreNorm` class, which will be used to apply groupnorm before the attention layer, as we'll see further. Note that there's been a debate about whether to apply normalization before or after attention in Transformers.

In [22]:
```
class PreNorm(nn.Module):
    def __init__(self, dim, fn):
        super().__init__()
        self.fn = fn
        self.norm = nn.GroupNorm(1, dim)

    def forward(self, x):
        x = self.norm(x)
        return self.fn(x)
```

Conditional U-Net

Now that we've defined all building blocks (position embeddings, ResNet/ConvNeXT blocks, attention and group normalization), it's time to define the entire neural network. Recall that the job of the network $\mathbf{\epsilon}_\theta(\mathbf{x}_t, t)$ is to take in a batch of noisy images + noise levels, and output the noise added to the input. More formally:

- the network takes a batch of noisy images of shape `(batch_size, num_channels, height, width)` and a batch of noise levels of shape `(batch_size, 1)` as input, and returns a tensor of shape `(batch_size, num_channels, height, width)`

The network is built up as follows:

- first, a convolutional layer is applied on the batch of noisy images, and position embeddings are computed for the noise levels
- next, a sequence of downsampling stages are applied. Each downsampling stage consists of 2 ResNet/ConvNeXT blocks + groupnorm + attention + residual connection + a downsample operation
- at the middle of the network, again ResNet or ConvNeXT blocks are applied, interleaved with attention
- next, a sequence of upsampling stages are applied. Each upsampling stage consists of 2 ResNet/ConvNeXT blocks + groupnorm + attention + residual connection + an upsample operation
- finally, a ResNet/ConvNeXT block followed by a convolutional layer is applied.

Ultimately, neural networks stack up layers as if they were lego blocks (but it's important to understand how they work).

In [23]:

```
class Unet(nn.Module):
    def __init__(
        self,
        dim,
        init_dim=None,
        out_dim=None,
        dim_mults=(1, 2, 4, 8),
        channels=3,
        with_time_emb=True,
        resnet_block_groups=8,
        use_convnext=True,
        convnext_mult=2,
    ):
        super().__init__()

        # determine dimensions
        self.channels = channels

        init_dim = default(init_dim, dim // 3 * 2)
        self.init_conv = nn.Conv2d(channels, init_dim, 7, padding=3)

        dims = [init_dim, *map(lambda m: dim * m, dim_mults)]
        in_out = list(zip(dims[:-1], dims[1:]))

        if use_convnext:
            block_klass = partial(ConvNextBlock, mult=convnext_mult)
        else:
            block_klass = partial(ResnetBlock, groups=resnet_block_groups

        # time embeddings
        if with_time_emb:
            time_dim = dim * 4
            self.time_mlp = nn.Sequential(
                SinusoidalPositionEmbeddings(dim),
                nn.Linear(dim, time_dim),
                nn.GELU(),
                nn.Linear(time_dim, time_dim),
            )
        else:
            time_dim = None
            self.time_mlp = None

        # layers
        self.downs = nn.ModuleList([])
        self.ups = nn.ModuleList([])
        num_resolutions = len(in_out)

        for ind, (dim_in, dim_out) in enumerate(in_out):
            is_last = ind >= (num_resolutions - 1)

            self.downs.append(
                nn.ModuleList(
                    [
                        block_klass(dim_in, dim_out, time_emb_dim=time_di
                        block_klass(dim_out, dim_out, time_emb_dim=time_d
                        Residual(PreNorm(dim_out, LinearAttention(dim_out
```

```
                    Downsample(dim_out) if not is_last else nn.Identi
                ]
            )
        )

        mid_dim = dims[-1]
        self.mid_block1 = block_klass(mid_dim, mid_dim, time_emb_dim=time
        self.mid_attn = Residual(PreNorm(mid_dim, Attention(mid_dim)))
        self.mid_block2 = block_klass(mid_dim, mid_dim, time_emb_dim=time

        for ind, (dim_in, dim_out) in enumerate(reversed(in_out[1:])):
            is_last = ind >= (num_resolutions - 1)

            self.ups.append(
                nn.ModuleList(
                    [
                        block_klass(dim_out * 2, dim_in, time_emb_dim=tim
                        block_klass(dim_in, dim_in, time_emb_dim=time_dim
                        Residual(PreNorm(dim_in, LinearAttention(dim_in))
                        Upsample(dim_in) if not is_last else nn.Identity(
                    ]
                )
            )

        out_dim = default(out_dim, channels)
        self.final_conv = nn.Sequential(
            block_klass(dim, dim), nn.Conv2d(dim, out_dim, 1)
        )

    def forward(self, x, time):
        x = self.init_conv(x)

        t = self.time_mlp(time) if exists(self.time_mlp) else None

        h = []

        # downsample
        for block1, block2, attn, downsample in self.downs:
            x = block1(x, t)
            x = block2(x, t)
            x = attn(x)
            h.append(x)
            x = downsample(x)

        # bottleneck
        x = self.mid_block1(x, t)
        x = self.mid_attn(x)
        x = self.mid_block2(x, t)

        # upsample
        for block1, block2, attn, upsample in self.ups:
            x = torch.cat((x, h.pop()), dim=1)
            x = block1(x, t)
            x = block2(x, t)
            x = attn(x)
            x = upsample(x)
```

```
        return self.final_conv(x)
```

Defining the forward diffusion process

The forward diffusion process gradually adds noise to an image from the real distribution, in a number of time steps T. This happens according to a **variance schedule**. The original DDPM authors employed a linear schedule:

> We set the forward process variances to constants

increasing linearly from $\beta_1 = 10^{−4}$ to $\beta_T = 0.02$.

However, it was shown in (Nichol et al., 2021) that better results can be achieved when employing a cosine schedule.

Below, we define various schedules for the T timesteps, as well as corresponding variables which we'll need, such as cumulative variances.

In [24]:
```
def cosine_beta_schedule(timesteps, s=0.008):
    """
    cosine schedule as proposed in https://arxiv.org/abs/2102.09672
    """
    steps = timesteps + 1
    x = torch.linspace(0, timesteps, steps)
    alphas_cumprod = torch.cos(((x / timesteps) + s) / (1 + s) * torch.pi
    alphas_cumprod = alphas_cumprod / alphas_cumprod[0]
    betas = 1 - (alphas_cumprod[1:] / alphas_cumprod[:-1])
    return torch.clip(betas, 0.0001, 0.9999)

def linear_beta_schedule(timesteps):
    beta_start = 0.0001
    beta_end = 0.02
    return torch.linspace(beta_start, beta_end, timesteps)

def quadratic_beta_schedule(timesteps):
    beta_start = 0.0001
    beta_end = 0.02
    return torch.linspace(beta_start**0.5, beta_end**0.5, timesteps) ** 2

def sigmoid_beta_schedule(timesteps):
    beta_start = 0.0001
    beta_end = 0.02
    betas = torch.linspace(-6, 6, timesteps)
    return torch.sigmoid(betas) * (beta_end - beta_start) + beta_start
```

To start with, let's use the linear schedule for $T=200$ time steps and define the various variables from the β_t which we will need, such as the cumulative product of the variances $\bar{\alpha}_t$. Each of the variables below are just 1-dimensional tensors, storing values from t to T. Importantly, we also define an `extract` function, which will allow us to extract the appropriate t index for a batch of indices.

In [25]:
```
timesteps = 200

# define beta schedule
betas = linear_beta_schedule(timesteps=timesteps)

# define alphas
alphas = 1. - betas
alphas_cumprod = torch.cumprod(alphas, axis=0)
alphas_cumprod_prev = F.pad(alphas_cumprod[:-1], (1, 0), value=1.0)
sqrt_recip_alphas = torch.sqrt(1.0 / alphas)

# calculations for diffusion q(x_t | x_{t-1}) and others
sqrt_alphas_cumprod = torch.sqrt(alphas_cumprod)
sqrt_one_minus_alphas_cumprod = torch.sqrt(1. - alphas_cumprod)

# calculations for posterior q(x_{t-1} | x_t, x_0)
posterior_variance = betas * (1. - alphas_cumprod_prev) / (1. - alphas_cu

def extract(a, t, x_shape):
    batch_size = t.shape[0]
    out = a.gather(-1, t.cpu())
    return out.reshape(batch_size, *((1,) * (len(x_shape) - 1))).to(t.dev
```

Forward diffusion sample

In [26]:
```
x_start = next(iter(dataloader))
print(f"batch shape {batch.shape}")
```

```
batch shape torch.Size([512, 1, 64, 64])
```

In [27]:
```
# forward diffusion
def q_sample(x_start, t, noise=None):
    if noise is None:
        noise = torch.randn_like(x_start)

    sqrt_alphas_cumprod_t = extract(sqrt_alphas_cumprod, t, x_start.shape
    sqrt_one_minus_alphas_cumprod_t = extract(
        sqrt_one_minus_alphas_cumprod, t, x_start.shape
    )

    return sqrt_alphas_cumprod_t * x_start + sqrt_one_minus_alphas_cumpro
```

In [28]:
```
def get_noisy_image(x_start, t):
  # add noise
  x_noisy = q_sample(x_start, t=t)

  # turn back into PIL image
  noisy_image = reverse_transform(x_noisy.squeeze())

  return noisy_image
```

In [29]:
```
# use seed for reproducability
torch.manual_seed(0)
def plot(imgs, with_orig=False, row_title=None, **imshow_kwargs):
    if not isinstance(imgs[0], list):
        # Make a 2d grid even if there's just 1 row
        imgs = [imgs]

    num_rows = len(imgs)
    num_cols = len(imgs[0]) + with_orig
    fig, axs = plt.subplots(figsize=(200,200), nrows=num_rows, ncols=num_
    for row_idx, row in enumerate(imgs):
        row = [image] + row if with_orig else row
        for col_idx, img in enumerate(row):
            ax = axs[row_idx, col_idx]
            ax.imshow(np.asarray(img), **imshow_kwargs)
            ax.set(xticklabels=[], yticklabels=[], xticks=[], yticks=[])

    if with_orig:
        axs[0, 0].set(title='Original image')
        axs[0, 0].title.set_size(8)
    if row_title is not None:
        for row_idx in range(num_rows):
            axs[row_idx, 0].set(ylabel=row_title[row_idx])

    plt.tight_layout()
```

In [30]:
```
# use seed for reproducability
torch.manual_seed(0)
def plot(imgs, with_orig=False, row_title=None, **imshow_kwargs):
    if not isinstance(imgs[0], list):
        # Make a 2d grid even if there's just 1 row
        imgs = [imgs]

    num_rows = len(imgs)
    num_cols = len(imgs[0]) + with_orig
    fig, axs = plt.subplots(figsize=(200,200), nrows=num_rows, ncols=num_
    for row_idx, row in enumerate(imgs):
        row = [image] + row if with_orig else row
        for col_idx, img in enumerate(row):
            ax = axs[row_idx, col_idx]
            ax.imshow(np.asarray(img), **imshow_kwargs)
            ax.set(xticklabels=[], yticklabels=[], xticks=[], yticks=[])

    if with_orig:
        axs[0, 0].set(title='Original image')
        axs[0, 0].title.set_size(8)
    if row_title is not None:
        for row_idx in range(num_rows):
            axs[row_idx, 0].set(ylabel=row_title[row_idx])

    plt.tight_layout()
```

In [31]:
```
#plot([get_noisy_image(x_start, torch.tensor([t])) for t in [0, 50, 100,
```

This means that we can now define the loss function given the model as follows:

In [32]:
```
def p_losses(denoise_model, x_start, t, noise=None, loss_type="l1"):
    if noise is None:
        noise = torch.randn_like(x_start)

    x_noisy = q_sample(x_start=x_start, t=t, noise=noise)
    predicted_noise = denoise_model(x_noisy, t)

    if loss_type == 'l1':
        loss = F.l1_loss(noise, predicted_noise)
    elif loss_type == 'l2':
        loss = F.mse_loss(noise, predicted_noise)
    elif loss_type == "huber":
        loss = F.smooth_l1_loss(noise, predicted_noise)
    else:
        raise NotImplementedError()

    return loss
```

In [33]:
```
def cuda_mem_info():
    # print(torch.cuda.memory_summary(device=None, abbreviated=False)) is
    convert_to_gb = 1024**3
    return f"{torch.cuda.get_device_name(0)} -- Allocated: {torch.cuda.me
print(cuda_mem_info())
```

```
Tesla V100-SXM2-16GB -- Allocated: 0.0GB, Cached: 0.0GB -- 15.1/15.8 (fre
e/total)
```

In [ ]:
```

```

The `denoise_model` will be our U-Net defined above. We'll employ the Huber loss between the true and the predicted noise.

Sampling

In [34]:
```
@torch.no_grad()
def p_sample(model, x, t, t_index):
    betas_t = extract(betas, t, x.shape)
    sqrt_one_minus_alphas_cumprod_t = extract(
        sqrt_one_minus_alphas_cumprod, t, x.shape
    )
    sqrt_recip_alphas_t = extract(sqrt_recip_alphas, t, x.shape)

    # Equation 11 in the paper
    # Use our model (noise predictor) to predict the mean
    model_mean = sqrt_recip_alphas_t * (
        x - betas_t * model(x, t) / sqrt_one_minus_alphas_cumprod_t
    )

    if t_index == 0:
        return model_mean
    else:
        posterior_variance_t = extract(posterior_variance, t, x.shape)
        noise = torch.randn_like(x)
        # Algorithm 2 line 4:
        return model_mean + torch.sqrt(posterior_variance_t) * noise

# Algorithm 2 but save all images:
@torch.no_grad()
def p_sample_loop(model, shape):
    device = next(model.parameters()).device

    b = shape[0]
    # start from pure noise (for each example in the batch)
    img = torch.randn(shape, device=device)
    imgs = []

    for i in tqdm(reversed(range(0, timesteps)), desc='sampling loop time
        img = p_sample(model, img, torch.full((b,), i, device=device, dty
        imgs.append(img.cpu().numpy())
    return imgs

@torch.no_grad()
def sample(model, image_size, batch_size=16, channels=3):
    return p_sample_loop(model, shape=(batch_size, channels, image_size,
```

Train the model

In [35]:

```python
from pathlib import Path

def num_to_groups(num, divisor):
    groups = num // divisor
    remainder = num % divisor
    arr = [divisor] * groups
    if remainder > 0:
        arr.append(remainder)
    return arr

results_folder = Path("./results")
results_folder.mkdir(exist_ok = True)
save_and_sample_every = 1000
```

In [36]:
```
from torchvision.utils import save_image

from torch.optim import Adam

device = "cuda" if torch.cuda.is_available() else "cpu"
print(device)
model = Unet(
    dim=image_size,
    channels=channels,
    dim_mults=(1, 2, 4,),
    use_convnext=False
)
if torch.cuda.device_count() > 1:
    model = nn.DataParallel(model)

model.to(device)

optimizer = Adam(model.parameters(), lr=1e-3)
print(cuda_mem_info())

# scheduler1 = torch.optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.

losses = []
epochs = 400

for epoch in range(epochs):
    for step, batch in enumerate(dataloader):
      optimizer.zero_grad()

      #batch_size = batch["pixel_values"].shape[0]
      #batch = batch["pixel_values"].to(device)
      batch_size = batch.shape[0]
      batch = batch.to(device)


      # Algorithm 1 line 3: sample t uniformally for every example in the
      t = torch.randint(0, timesteps, (batch_size,), device=device).long(

      loss = p_losses(model, batch, t,  loss_type="huber") #loss_type="l2

      if step % 100 == 0:
        print(f"Epoch: {epoch:<4} -- Loss: {loss.item()} \n   -- {cuda_me
      loss.backward()
      optimizer.step()

      # save generated images
      if step != 0 and step % save_and_sample_every == 0:
        milestone = step // save_and_sample_every
        batches = num_to_groups(4, batch_size)
        all_images_list = list(map(lambda n: sample(model, batch_size=n,
        all_images = torch.cat(all_images_list, dim=0)
        all_images = (all_images + 1) * 0.5
        save_image(all_images, str(results_folder / f'sample-{milestone}.

    #scheduler1.step()
```

```
cuda
```

```
Tesla V100-SXM2-16GB -- Allocated: 0.0GB, Cached: 0.0GB -- 15.0/15.8 (fre
e/total)
Epoch: 0    -- Loss: 0.4580073058605194
   -- Tesla V100-SXM2-16GB -- Allocated: 7.9GB, Cached: 8.2GB -- 6.5/15.8
(free/total)
Epoch: 1    -- Loss: 0.6302535533905029
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 2    -- Loss: 0.45431795716285706
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 3    -- Loss: 0.4898484945297241
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 4    -- Loss: 0.45206916332244873
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 5    -- Loss: 0.44079262018203735
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 6    -- Loss: 0.4377913475036621
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 7    -- Loss: 0.4265657365322113
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 8    -- Loss: 0.42891937494277954
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 9    -- Loss: 0.42240971326828003
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 10   -- Loss: 0.4254440665245056
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 11   -- Loss: 0.42258989810943604
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 12   -- Loss: 0.4211808443069458
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 13   -- Loss: 0.42049771547317505
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 14   -- Loss: 0.4186062216758728
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 15   -- Loss: 0.4176069498062134
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 16   -- Loss: 0.41622406244277954
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 17   -- Loss: 0.41303175687789917
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 18   -- Loss: 0.4093429148197174
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 19   -- Loss: 0.4052460193634033
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 20   -- Loss: 0.3998216390609741
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 21   -- Loss: 0.3914836645126343
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 22   -- Loss: 0.3797597885131836
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 23   -- Loss: 0.35805022716522217
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 24   -- Loss: 0.326522558927536
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 25   -- Loss: 0.2838994860649109
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 26   -- Loss: 0.2376454770565033
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 27   -- Loss: 0.19799309968948364
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 28   -- Loss: 0.19333237409591675
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 29   -- Loss: 0.15822792053222656
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 30   -- Loss: 0.14959236979484558
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 31   -- Loss: 0.12348610162734985
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 32   -- Loss: 0.10032502561807632
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 33   -- Loss: 0.09365551173686981
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 34   -- Loss: 0.07987258583307266
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 35   -- Loss: 0.07448810338973999
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 36   -- Loss: 0.07877905666828156
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 37   -- Loss: 0.06948050856590271
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 38   -- Loss: 0.0667458027601242
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 39   -- Loss: 0.059970173984766006
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 40   -- Loss: 0.059776850044727325
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 41   -- Loss: 0.05519559979438782
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 42   -- Loss: 0.05440610274672508
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 43   -- Loss: 0.05041583999991417
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 44   -- Loss: 0.05373396724462509
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 45   -- Loss: 0.04646095633506775
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 46   -- Loss: 0.05136077478528023
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 47   -- Loss: 0.04562029242515564
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 48   -- Loss: 0.047573719173669815
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 49   -- Loss: 0.046157896518707275
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 50   -- Loss: 0.04576282203197479
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 51   -- Loss: 0.04335344582796097
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 52   -- Loss: 0.04533493518829346
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 53   -- Loss: 0.044938359409570694
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 54   -- Loss: 0.04454401880502701
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 55   -- Loss: 0.0392475351691246
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 56   -- Loss: 0.040443550795316696
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 57   -- Loss: 0.04114872217178345
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 58   -- Loss: 0.03964657336473465
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 59   -- Loss: 0.037039484828710556
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 60   -- Loss: 0.04146336764097214
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 61   -- Loss: 0.04075465351343155
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 62   -- Loss: 0.04311235249042511
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 63   -- Loss: 0.039735663682222366
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 64   -- Loss: 0.04099077731370926
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 65   -- Loss: 0.03739394247531891
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 66   -- Loss: 0.03913722187280655
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 67   -- Loss: 0.0365014523267746
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 68   -- Loss: 0.03639626130461693
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 69   -- Loss: 0.03572770953178406
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 70   -- Loss: 0.035368598997592926
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 71   -- Loss: 0.033935900777578354
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 72   -- Loss: 0.03463044762611389
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 73   -- Loss: 0.03292768448591232
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 74   -- Loss: 0.03386254608631134
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 75   -- Loss: 0.0343470573425293
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 76   -- Loss: 0.03325042128562927
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 77   -- Loss: 0.03374457731842995
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 78   -- Loss: 0.030946623533964157
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 79   -- Loss: 0.030999306589365005
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 80   -- Loss: 0.0292702354490757
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 81   -- Loss: 0.03105035051703453
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 82   -- Loss: 0.030659982934594154
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 83   -- Loss: 0.028424054384231567
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 84   -- Loss: 0.03173311799764633
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 85   -- Loss: 0.030731387436389923
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 86   -- Loss: 0.03032691776752472
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 87   -- Loss: 0.030907243490219116
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 88   -- Loss: 0.028923995792865753
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 89   -- Loss: 0.030225088819861412
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 90   -- Loss: 0.030674457550048828
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 91   -- Loss: 0.028041351586580276
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 92   -- Loss: 0.028167594224214554
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 93   -- Loss: 0.026031343266367912
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 94   -- Loss: 0.02665204368531704
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 95   -- Loss: 0.027873555198311806
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 96   -- Loss: 0.02488628402352333
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 97   -- Loss: 0.02622980996966362
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 98   -- Loss: 0.025913497433066368
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 99   -- Loss: 0.026839355006814003
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 100  -- Loss: 0.025580549612641335
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 101  -- Loss: 0.027133461087942123
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 102  -- Loss: 0.026526879519224167
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 103  -- Loss: 0.023376433178782463
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 104  -- Loss: 0.027242062613368034
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 105  -- Loss: 0.02541699819266796
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 106  -- Loss: 0.024108706042170525
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 107  -- Loss: 0.026353009045124054
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 108  -- Loss: 0.02529235929250717
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 109  -- Loss: 0.025438886135816574
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 110  -- Loss: 0.023297734558582306
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 111  -- Loss: 0.023211780935525894
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 112  -- Loss: 0.023691948503255844
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 113  -- Loss: 0.022641129791736603
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 114  -- Loss: 0.024040697142481804
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 115  -- Loss: 0.022997144609689713
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 116  -- Loss: 0.024784591048955917
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 117  -- Loss: 0.022927597165107727
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 118  -- Loss: 0.02257109060883522
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 119  -- Loss: 0.024422897025942802
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 120  -- Loss: 0.022230617702007294
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 121  -- Loss: 0.022554270923137665
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 122  -- Loss: 0.023080527782440186
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 123  -- Loss: 0.0229201503098011
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 124  -- Loss: 0.021681655198335648
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 125  -- Loss: 0.022983700037002563
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 126  -- Loss: 0.022246789187192917
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 127  -- Loss: 0.022718239575624466
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 128  -- Loss: 0.02175363525748253
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 129  -- Loss: 0.023356756195425987
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 130  -- Loss: 0.023259365931153297
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 131  -- Loss: 0.023407042026519775
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 132  -- Loss: 0.02292807027697563
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 133  -- Loss: 0.023179702460765 84
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 134  -- Loss: 0.023158146068453 79
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 135  -- Loss: 0.024107616394758224
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 136  -- Loss: 0.022713022306561 47
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 137  -- Loss: 0.022250887006521225
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 138  -- Loss: 0.022825304418802 26
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 139  -- Loss: 0.023066103458404 54
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 140  -- Loss: 0.023670151829719543
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 141  -- Loss: 0.021388236433267593
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 142  -- Loss: 0.022415794432163 24
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 143  -- Loss: 0.022134982049465 18
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 144  -- Loss: 0.022302901372313 5
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 145  -- Loss: 0.021916314959526062
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 146  -- Loss: 0.020856019109487534
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 147  -- Loss: 0.020955946296453476
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 148  -- Loss: 0.021775523200631 14
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 149  -- Loss: 0.021093513816595078
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 150  -- Loss: 0.021495046094059944
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 151  -- Loss: 0.021067002788186073
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 152  -- Loss: 0.020507387816905975
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 153  -- Loss: 0.021835055202245712
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 154  -- Loss: 0.01943107508122921
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 155  -- Loss: 0.021336201578378677
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 156  -- Loss: 0.021767619997262955
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 157  -- Loss: 0.021342258900403976
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 158  -- Loss: 0.02530112862586975
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 159  -- Loss: 0.02649611048400402
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 160  -- Loss: 0.02497127093374729
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 161  -- Loss: 0.024256132543087006
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 162  -- Loss: 0.026161737740039825
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 163  -- Loss: 0.023314015939831734
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 164  -- Loss: 0.02191575989127159
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 165  -- Loss: 0.023814285174012184
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 166  -- Loss: 0.021586494520306587
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 167  -- Loss: 0.021060140803456306
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 168  -- Loss: 0.021147586405277252
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 169  -- Loss: 0.02167738974094391
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 170  -- Loss: 0.01977887749671936
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 171  -- Loss: 0.020721014589071274
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 172  -- Loss: 0.021991955116391182
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 173  -- Loss: 0.0198800191283226
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 174  -- Loss: 0.02134307660162449
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 175  -- Loss: 0.019568927586078644
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 176  -- Loss: 0.020481249317526817
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 177  -- Loss: 0.020184073597192764
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 178  -- Loss: 0.020029118284583092
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 179  -- Loss: 0.020070523023605347
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 180  -- Loss: 0.01911059021949768
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 181  -- Loss: 0.01938669942319393
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 182  -- Loss: 0.02050524763762951
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 183  -- Loss: 0.019592490047216415
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 184  -- Loss: 0.021464338526129723
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 185  -- Loss: 0.020612191408872604
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 186  -- Loss: 0.020802315324544907
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 187  -- Loss: 0.018876943737268448
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 188  -- Loss: 0.01999945193529129
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 189  -- Loss: 0.019577812403440475
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 190  -- Loss: 0.018161315470933914
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 191  -- Loss: 0.020086532458662987
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 192  -- Loss: 0.019432833418250084
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 193  -- Loss: 0.02016693726181984
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 194  -- Loss: 0.020781736820936203
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 195  -- Loss: 0.01950923353433609
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 196  -- Loss: 0.020748555660247803
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 197  -- Loss: 0.019783616065979004
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 198  -- Loss: 0.02120226062834263
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 199  -- Loss: 0.018437452614307404
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 200  -- Loss: 0.02009819820523262
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 201  -- Loss: 0.019448811188340187
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 202  -- Loss: 0.019395487383008003
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 203  -- Loss: 0.0192004032433033
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 204  -- Loss: 0.018741698935627937
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 205  -- Loss: 0.02078065276145935
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 206  -- Loss: 0.01949591562151909
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 207  -- Loss: 0.01963171735405922
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 208  -- Loss: 0.01804923266172409
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 209  -- Loss: 0.019002821296453476
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 210  -- Loss: 0.019860967993736267
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 211  -- Loss: 0.016894366592168808
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 212  -- Loss: 0.019159404560923576
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 213  -- Loss: 0.018582411110401154
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 214  -- Loss: 0.01767216995358467
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 215  -- Loss: 0.019302699714899063
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 216  -- Loss: 0.02181411348283291
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 217  -- Loss: 0.020891264081001282
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 218  -- Loss: 0.021097080782055855
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 219  -- Loss: 0.02084214985370636
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 220  -- Loss: 0.01940816268324852
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 221  -- Loss: 0.01941993460059166
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 222  -- Loss: 0.018170800060033798
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 223  -- Loss: 0.017799802124500275
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 224  -- Loss: 0.019081667065620422
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 225  -- Loss: 0.020758111029863358
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 226  -- Loss: 0.017745135352015495
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 227  -- Loss: 0.01882927492260933
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 228  -- Loss: 0.01873837783932686
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 229  -- Loss: 0.019057810306549072
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 230  -- Loss: 0.018987290561199188
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 231  -- Loss: 0.017604166641831398
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 232  -- Loss: 0.017398256808519363
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 233  -- Loss: 0.017773672938346863
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 234  -- Loss: 0.019015992060303688
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 235  -- Loss: 0.017410967499017715
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 236  -- Loss: 0.018975643441081047
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 237  -- Loss: 0.017606204375624657
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 238  -- Loss: 0.018350698053836823
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 239  -- Loss: 0.0165035929530859
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 240  -- Loss: 0.01793873868882656
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 241  -- Loss: 0.017363112419843674
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 242  -- Loss: 0.018648598343133926
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 243  -- Loss: 0.017409125342965126
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 244  -- Loss: 0.017909634858369827
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 245  -- Loss: 0.016306813806295395
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 246  -- Loss: 0.019065022468566895
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 247  -- Loss: 0.018424328416585922
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 248  -- Loss: 0.017111770808696747
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 249  -- Loss: 0.017050612717866898
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 250  -- Loss: 0.015906179323792458
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 251  -- Loss: 0.01542685180902481
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 252  -- Loss: 0.017094019800424576
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 253  -- Loss: 0.017523078247904778
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 254  -- Loss: 0.017757417634129524
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 255  -- Loss: 0.018655456602573395
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 256  -- Loss: 0.018605105578899384
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 257  -- Loss: 0.01610185205936432
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 258  -- Loss: 0.018400005996227264
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 259  -- Loss: 0.017471812665462494
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 260  -- Loss: 0.017122432589530945
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 261  -- Loss: 0.015580595470964909
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 262  -- Loss: 0.01535271666944027
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 263  -- Loss: 0.016252148896455765
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 264  -- Loss: 0.017773998901247978
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 265  -- Loss: 0.015086425468325615
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 266  -- Loss: 0.01745046302676201
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 267  -- Loss: 0.01873023808002472
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 268  -- Loss: 0.016033027321100235
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 269  -- Loss: 0.017302192747592926
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 270  -- Loss: 0.017563536763191223
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 271  -- Loss: 0.015504922717809677
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 272  -- Loss: 0.016125038266181946
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 273  -- Loss: 0.017008446156978607
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 274  -- Loss: 0.015903852880001068
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 275  -- Loss: 0.015541717410087585
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 276  -- Loss: 0.016899269074201584
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 277  -- Loss: 0.016943112015724182
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 278  -- Loss: 0.015291983261704445
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 279  -- Loss: 0.01710248738527298
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 280  -- Loss: 0.016910165548324585
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 281  -- Loss: 0.01657579466700554
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 282  -- Loss: 0.015283269807696342
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 283  -- Loss: 0.016087505966424942
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 284  -- Loss: 0.016696780920028687
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 285  -- Loss: 0.01621990278363228
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 286  -- Loss: 0.016443312168121338
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 287  -- Loss: 0.016970597207546234
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 288  -- Loss: 0.015859369188547134
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 289  -- Loss: 0.017462369054555893
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 290  -- Loss: 0.01713755540549755
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 291  -- Loss: 0.016497259959578514
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 292  -- Loss: 0.01685589924454689
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 293  -- Loss: 0.016785208135843277
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 294  -- Loss: 0.01654369756579399
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 295  -- Loss: 0.01683064177632332
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 296  -- Loss: 0.017153669148683548
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 297  -- Loss: 0.01640503481030464
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 298  -- Loss: 0.015525835566222668
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 299  -- Loss: 0.01544623076915741
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 300  -- Loss: 0.015173124149441719
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 301  -- Loss: 0.01716592162847519
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 302  -- Loss: 0.015630407258868217
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 303  -- Loss: 0.014462875202298164
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 304  -- Loss: 0.014767042361199856
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 305  -- Loss: 0.014972886070609093
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 306  -- Loss: 0.015549000352621078
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 307  -- Loss: 0.016008790582418442
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 308  -- Loss: 0.016323374584317207
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 309  -- Loss: 0.016493186354637146
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 310  -- Loss: 0.016192644834518433
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 311  -- Loss: 0.014624551869928837
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 312  -- Loss: 0.016627896577119827
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 313  -- Loss: 0.01721101626753807
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 314  -- Loss: 0.015668829903006554
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 315  -- Loss: 0.015523623675107956
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 316  -- Loss: 0.015058085322380066
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 317  -- Loss: 0.015377855859696865
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 318  -- Loss: 0.01613077148795128
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 319  -- Loss: 0.01399985607713461
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 320  -- Loss: 0.014761088415980339
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 321  -- Loss: 0.014234902337193489
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 322  -- Loss: 0.015169385820627213
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 323  -- Loss: 0.015424340963363647
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 324  -- Loss: 0.014500810764729977
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 325  -- Loss: 0.01568678766489029
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 326  -- Loss: 0.01586056500673294
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 327  -- Loss: 0.014184242114424706
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 328  -- Loss: 0.015424497425556183
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 329  -- Loss: 0.014507077634334564
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 330  -- Loss: 0.015291675925254822
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 331  -- Loss: 0.015026042237877846
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 332  -- Loss: 0.013510104268789291
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 333  -- Loss: 0.014836112037301064
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 334  -- Loss: 0.016243822872638702
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 335  -- Loss: 0.013332544825971127
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 336  -- Loss: 0.013876481913030148
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 337  -- Loss: 0.01403418555855751
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 338  -- Loss: 0.012876121327280998
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 339  -- Loss: 0.015090661123394966
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 340  -- Loss: 0.014102248474955559
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 341  -- Loss: 0.015335962176322937
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 342  -- Loss: 0.015108062885701656
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 343  -- Loss: 0.014902572147548199
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 344  -- Loss: 0.015320423990488052
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 345  -- Loss: 0.015780357643961906
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 346  -- Loss: 0.014863520860671997
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 347  -- Loss: 0.015300968661904335
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 348  -- Loss: 0.016075965017080307
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 349  -- Loss: 0.014568896032869816
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 350  -- Loss: 0.014079980552196503
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 351  -- Loss: 0.015068641863763332
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 352  -- Loss: 0.013569341972470284
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 353  -- Loss: 0.013453126884996891
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 354  -- Loss: 0.014333013445138931
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 355  -- Loss: 0.0142368758097291
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 356  -- Loss: 0.014213385060429573
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 357  -- Loss: 0.014734323136508465
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 358  -- Loss: 0.01522257924079895
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 359  -- Loss: 0.014654753729701042
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 360  -- Loss: 0.014030186459422112
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 361  -- Loss: 0.014387596398591995
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 362  -- Loss: 0.015005961991846561
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 363  -- Loss: 0.014701277948915958
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 364  -- Loss: 0.014141218736767769
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 365  -- Loss: 0.013989346101880074
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 366  -- Loss: 0.013635514304041862
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 367  -- Loss: 0.012648608535528183
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 368  -- Loss: 0.013886088505387306
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 369  -- Loss: 0.015784472227096558
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 370  -- Loss: 0.014397361315786839
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 371  -- Loss: 0.014634120278060436
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 372  -- Loss: 0.015436630696058273
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 373  -- Loss: 0.013274936005473137
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 374  -- Loss: 0.014837154187262058
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 375  -- Loss: 0.015011928044259548
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 376  -- Loss: 0.015354504808783531
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 377  -- Loss: 0.013270804658532143
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 378  -- Loss: 0.014359831809997559
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 379  -- Loss: 0.01389152929186821
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 380  -- Loss: 0.014659332111477852
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 381  -- Loss: 0.015506040304899216
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 382  -- Loss: 0.013322320766746998
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 383  -- Loss: 0.013535590842366219
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 384  -- Loss: 0.01371462270617485
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 385  -- Loss: 0.014220558106899261
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 386  -- Loss: 0.012893016450107098
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 387  -- Loss: 0.013440227136015892
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 388  -- Loss: 0.013592474162578583
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 389  -- Loss: 0.013752693310379982
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 390  -- Loss: 0.013517523184418678
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 391  -- Loss: 0.014470061287283897
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 392  -- Loss: 0.013422982767224312
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 393  -- Loss: 0.01243470050394535
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 394  -- Loss: 0.015768101438879967
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 395  -- Loss: 0.014583313837647438
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 396  -- Loss: 0.015387683175504208
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 397  -- Loss: 0.014762740582227707
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 398  -- Loss: 0.01585860177874565
```

```
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
Epoch: 399  -- Loss: 0.014934438280761242
   -- Tesla V100-SXM2-16GB -- Allocated: 8.0GB, Cached: 10.6GB -- 3.7/15.
8 (free/total)
```

```
save_path = f"{results_folder}/checkpoint-{epoch}.pth")
torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'loss': loss,
            }, save_path)
```

```
model = Unet(
    dim=image_size,
    channels=channels,
    dim_mults=(1, 2, 4,),
    use_convnext=False
)
optimizer = Adam(model.parameters(), lr=1e-3)

checkpoint = torch.load(save_path)
model.load_state_dict(checkpoint['model_state_dict'])
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
epoch = checkpoint['epoch']
loss = checkpoint['loss']

model.eval()

model.train()
```

Sampling (inference)

To sample from the model, we can just use our sample function defined above:

In [37]:
```
print(cuda_mem_info())
torch.cuda.empty_cache() # make sure we don't keep old stuff
print(cuda_mem_info())
```

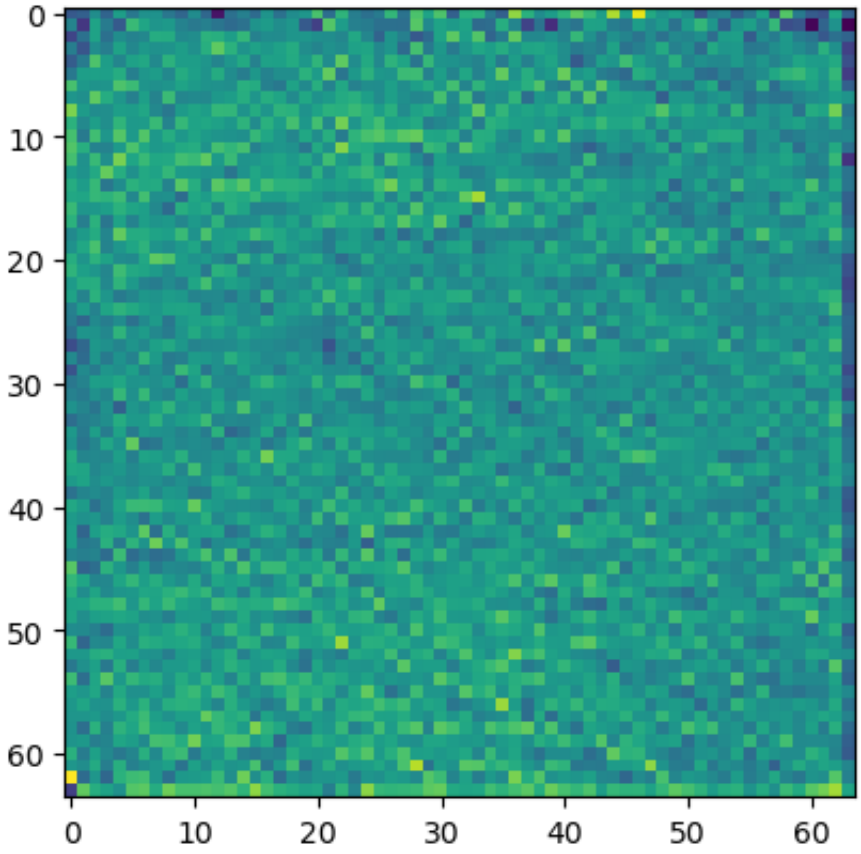
```
Tesla V100-SXM2-16GB -- Allocated: 0.1GB, Cached: 10.6GB -- 3.7/15.8 (fre
e/total)
Tesla V100-SXM2-16GB -- Allocated: 0.1GB, Cached: 0.4GB -- 13.9/15.8 (fre
e/total)
```

In [38]:
```
# sample 64 images
samples = sample(model, image_size=image_size, batch_size=batch_size, cha
```

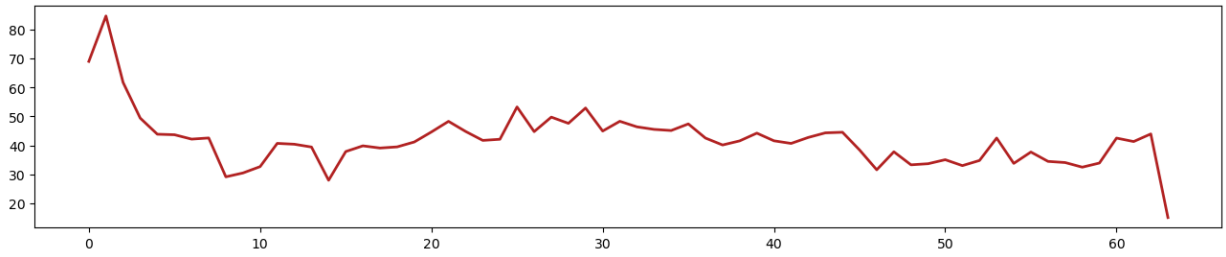
```
sampling loop time step:   0%|          | 0/200 [00:00<?, ?it/s]
```

In [39]:
```
# show a random one
random_index = 3
plt.imshow(samples[-1][random_index].reshape(image_size, image_size, chan
```

Out[39]: <matplotlib.image.AxesImage at 0x7f68387c2950>



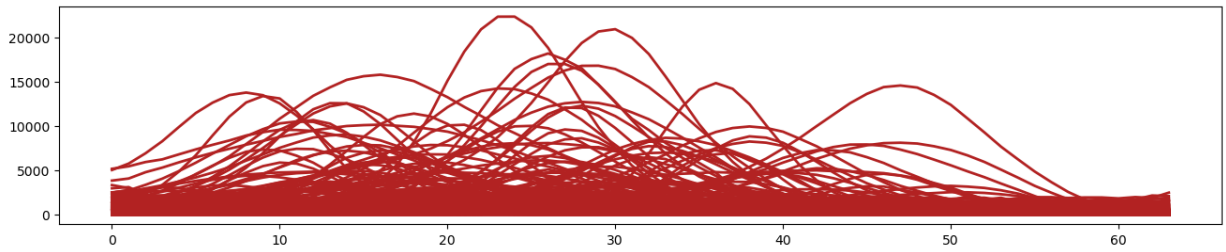
In [40]:
```
def plot_to_ax(array, ax=None, place=None, multi=False):
    c = ['firebrick', 'slateblue', 'orange']
    # (3, 48, 51)
    if len(array.shape) == 4: # it's a batch
        array = array[0, :, :, :]
    if ax is None:
        ax = plt.gca() # get current ax

    if place is None:
        array_to_plot = array.sum(axis=2)
    else:
        array_to_plot = array[:, :, place]
    for k in range(channels):
        if multi: # several lines will be plotted:
            ax.plot(array_to_plot[k], c = c[k], lw = .5, alpha=.5)
        else:
            ax.plot(array_to_plot[k], c = c[k], lw = 2, alpha=1)

def show_tensor_image(image, ax=None, place=None, multi=False):
    plot_to_ax(data.apply_transform_inv(image), ax=ax, place=place, multi

fig, axes = plt.subplots(1, 1, figsize=(16,3), dpi=100)
ax = axes # es.flat[i]
show_tensor_image(samples[-1][random_index], ax = ax)
plt.show()
```



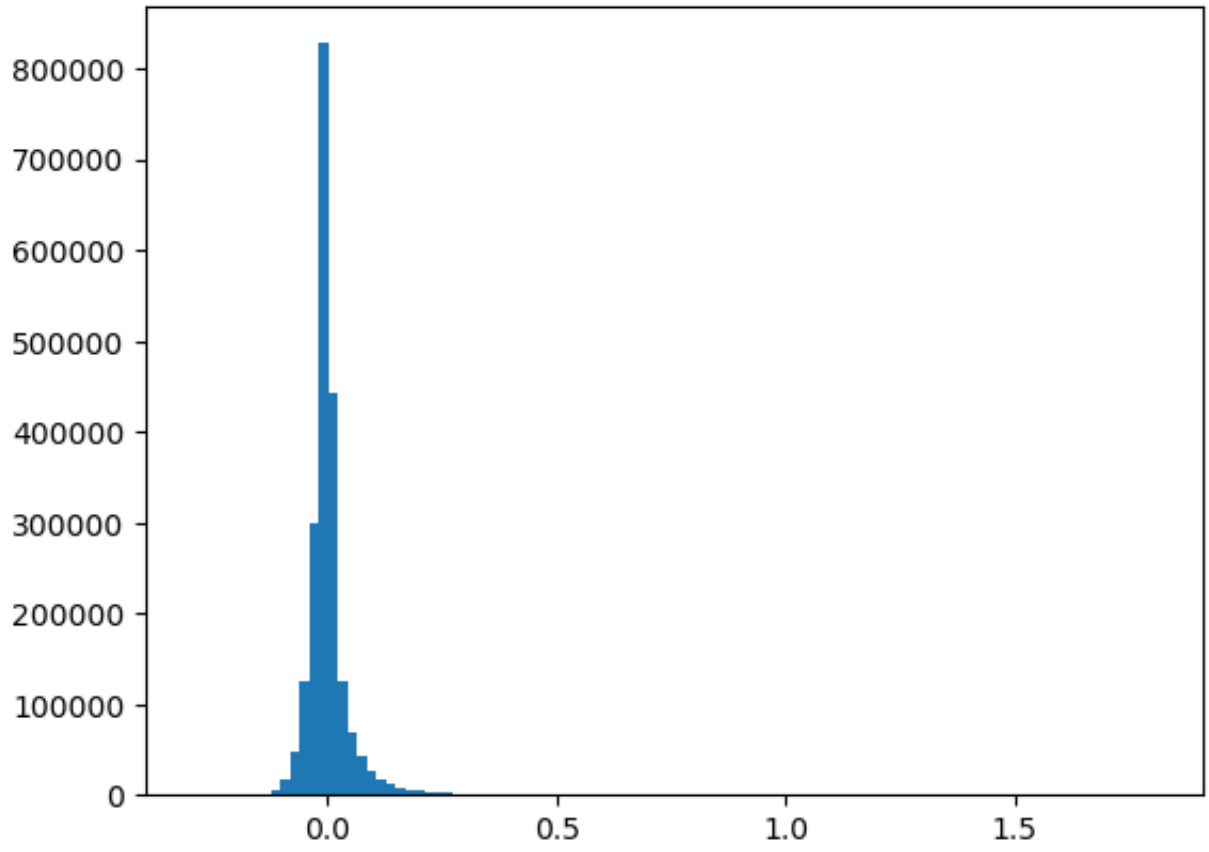
In [41]:
```
#till
```

In [ ]:
```
```

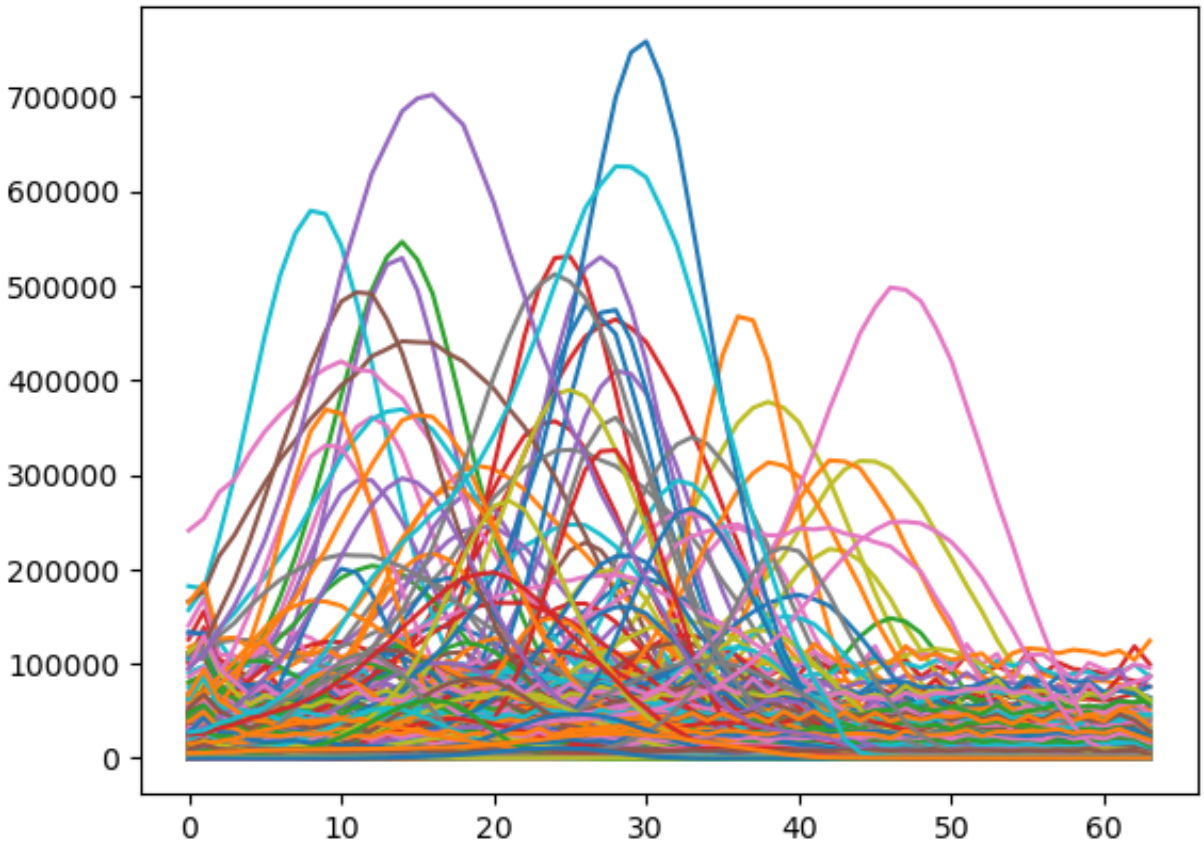
In [42]:
```
fig, axes = plt.subplots(1, 1, figsize=(16,3), dpi=100)
ax = axes # es.flat[i]
for i in range(batch_size):
    show_tensor_image(samples[-1][i], ax = ax)

#ax.set_ylim(0,10000)
```



In [43]:
```
plt.hist(samples[-1].flatten(), bins = 100);
```

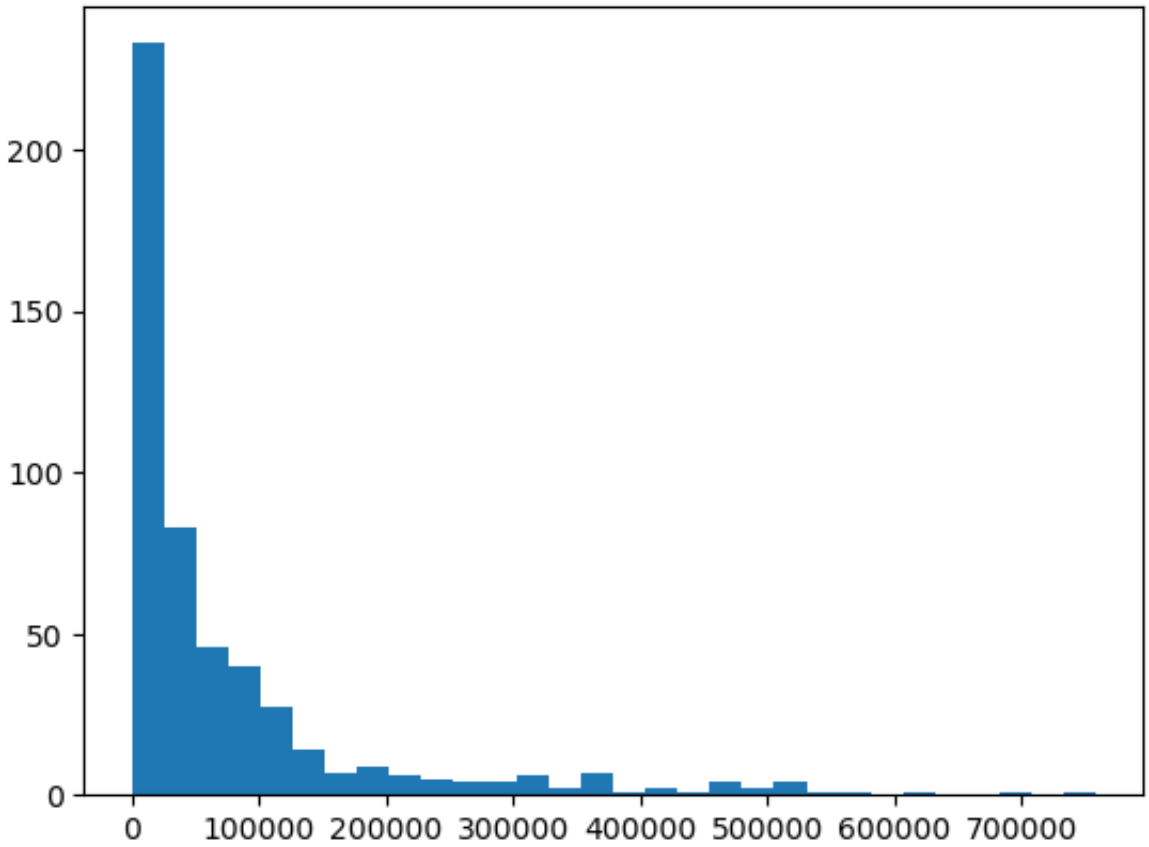


In [44]:
```
plt.plot(transform_inv(samples[-1][:,0,:,:].sum(axis=2)).T);
```



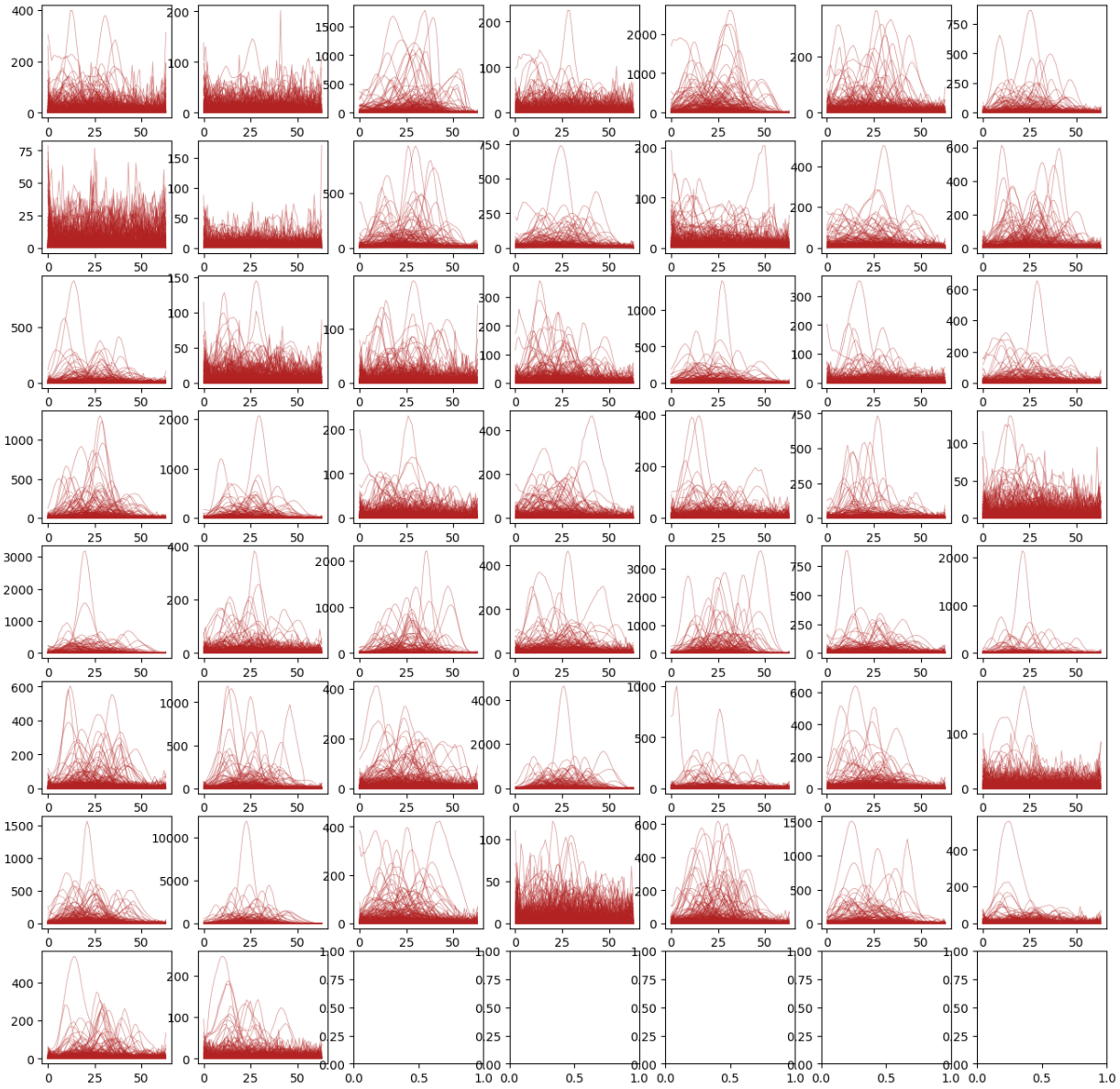
In [45]:
```
# histogram of peaks. In the US historically it's from 13k to 34k
plt.hist(transform_inv(samples[-1][:,0,:,:].sum(axis=2)).max(axis=1), bin
```

Out[45]:
```
(array([233.,  83.,  46.,  40.,  27.,  14.,   7.,   9.,   6.,   5.,   4.,
          4.,   6.,   2.,   7.,   1.,   2.,   1.,   4.,   2.,   4.,   1.,
          1.,   0.,   1.,   0.,   0.,   1.,   0.,   1.]),
 array([3.45451418e+02, 2.56076352e+04, 5.08698190e+04, 7.61320029e+04,
        1.01394187e+05, 1.26656370e+05, 1.51918554e+05, 1.77180738e+05,
        2.02442922e+05, 2.27705106e+05, 2.52967290e+05, 2.78229473e+05,
        3.03491657e+05, 3.28753841e+05, 3.54016025e+05, 3.79278209e+05,
        4.04540392e+05, 4.29802576e+05, 4.55064760e+05, 4.80326944e+05,
        5.05589128e+05, 5.30851311e+05, 5.56113495e+05, 5.81375679e+05,
        6.06637863e+05, 6.31900047e+05, 6.57162231e+05, 6.82424414e+05,
        7.07686598e+05, 7.32948782e+05, 7.58210966e+05]),
 <BarContainer object of 30 artists>)
```



In [ ]:

In [46]:
```
fig, axes = plt.subplots(8, 7, figsize=(16,16), dpi=100)

for ipl in range(51):
    ax = axes.flat[ipl]
    for i in range(batch_size):
        show_tensor_image(samples[-1][i], ax = ax, place=ipl, multi=True)
```



In [47]:

```
animate = False
if animate:
    import matplotlib.animation as animation

    random_index = 53
# TODO: the reshape shuffles the information
    fig = plt.figure()
    ims = []
    for i in range(timesteps):
        im = plt.imshow(samples[i][random_index].reshape(image_size, imag
        ims.append([im])

    animate = animation.ArtistAnimation(fig, ims, interval=50, blit=True,
    animate.save('diffusion.gif')
    plt.show()
```

In [48]:
```
if animate:
    plt.ioff()
    for ts in tqdm(range(0, timesteps, 5)):
        fig, axes = plt.subplots(8, 8, figsize=(10,10))
        for ipl in range(51):
            ax = axes.flat[ipl]
            for i in range(0,batch_size, 2):
                show_tensor_image(samples[ts][i], ax = ax, place=ipl)
        plt.savefig(f'results/{ts}.png')
        plt.close(fig)
```

Let's add repainting from the RePaint paper

Dimensions are chanel, time, place

In [49]:
```
transform_notdesc = transforms.Compose([
                    data_classes.transform_sqrt,
                    transforms.Lambda(lambda t: data_classes.transform_ch
                    transforms.Lambda(lambda t: data_classes.transform_ch
                    #transforms.Lambda(lambda t: data_classes.transform_r
        ])
```

In [50]:
```
gt = transform_notdesc(gt_xarr.data) # data.apply_transform
print(gt.shape)
```

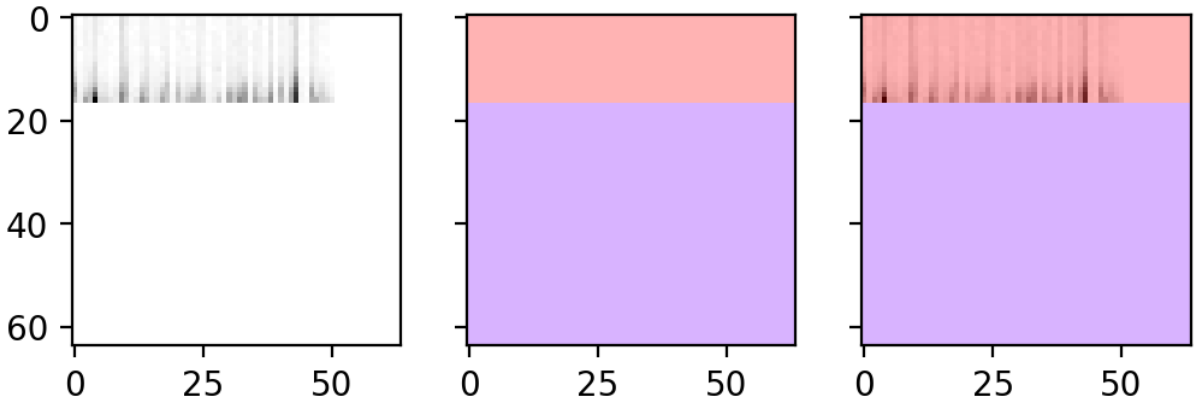
```
(1, 64, 64)
```

In [ ]:
```
```

In [51]:
```
gt_keep_mask = np.ones((channels,image_size,image_size))
gt_keep_mask[:,inpaintfrom_idx:,:] = 0
```

In [52]:
```
# check that it stitch
fig, axes = plt.subplots(1, 3, figsize=(6,6), dpi=200, sharex=True, share
axes[1].imshow(gt_keep_mask[0], alpha=.3, cmap = "rainbow")
axes[0].imshow(gt[0], cmap='Greys')


axes[2].imshow(gt[0], cmap='Greys')
axes[2].imshow(gt_keep_mask[0], alpha=.3, cmap = "rainbow")
```

Out[52]:
```
<matplotlib.image.AxesImage at 0x7f665dacb370>
```



In [53]:
```
device = next(model.parameters()).device
gt_keep_mask = torch.from_numpy(gt_keep_mask).type(torch.FloatTensor).to(
gt = torch.from_numpy(gt).type(torch.FloatTensor).to(device)
```

In [54]:
```
@torch.no_grad()
def p_sample_paint(model, x, t, t_index):

    # timestep parameters
    betas_t = extract(betas, t, x.shape)
    sqrt_one_minus_alphas_cumprod_t = extract(
        sqrt_one_minus_alphas_cumprod, t, x.shape
    )
    sqrt_recip_alphas_t = extract(sqrt_recip_alphas, t, x.shape)

    posterior_variance_t = extract(posterior_variance, t, x.shape)

    resampling_steps = 50

    for u in range(resampling_steps):
        # RePaint algorithm, line 4 and 6
        if t_index == 0:
            epsilon = 0
            z = 0
        else:
            epsilon = torch.randn_like(x)
            z = torch.randn_like(x)

        # RePaint algorithm, line 5
        x_tminus1_known = 1/ sqrt_recip_alphas_t * gt + sqrt_one_minus_al

        # RePaint algorithm, line 4 and 7
        x_tminus1_unknow = sqrt_recip_alphas_t * (
            x - betas_t * model(x, t) / sqrt_one_minus_alphas_cumprod_t
        ) + torch.sqrt(posterior_variance_t) * z

        x_tminus1 = x_tminus1_known * gt_keep_mask + x_tminus1_unknow * (

        if u < resampling_steps and (t > 1).all():
            # taken from q_sample:
            noise = torch.randn_like(x)
            sqrt_alphas_cumprod_t = extract(sqrt_alphas_cumprod, t-1, x.s
            sqrt_one_minus_alphas_cumprod_t = extract(sqrt_one_minus_alph
                                t-1, x.shape)
```

```
            x = sqrt_alphas_cumprod_t * x_tminus1 + sqrt_one_minus_alphas

    return x_tminus1

# Algorithm 2 but save all images:
@torch.no_grad()
def p_sample_loop_paint(model, shape):
    device = next(model.parameters()).device

    b = shape[0]
    # start from pure noise (for each example in the batch)
    img = torch.randn(shape, device=device)   # this is x_T
    imgs = []

    for i in tqdm(reversed(range(0, timesteps)), desc='sampling loop time
        img = p_sample_paint(model, img, torch.full((b,), i, device=devic
        imgs.append(img.cpu().numpy())
    return imgs

@torch.no_grad()
def sample_paint(model, image_size, batch_size=16, channels=3):
    return p_sample_loop_paint(model, shape=(batch_size, channels, image_
```

In [55]:
```
print(cuda_mem_info())
torch.cuda.empty_cache() # make sure we don't keep old stuff
print(cuda_mem_info())
```

```
Tesla V100-SXM2-16GB -- Allocated: 0.1GB, Cached: 2.3GB -- 12.0/15.8 (fre
e/total)
Tesla V100-SXM2-16GB -- Allocated: 0.1GB, Cached: 0.4GB -- 13.9/15.8 (fre
e/total)
```

In [ ]:
```
```

In [56]:
```
n_samples = 140
all_samples = []
for i in range(max(n_samples//batch_size,1)):
    samples = sample_paint(model, image_size=image_size, batch_size=batch
    all_samples.append(samples)
```

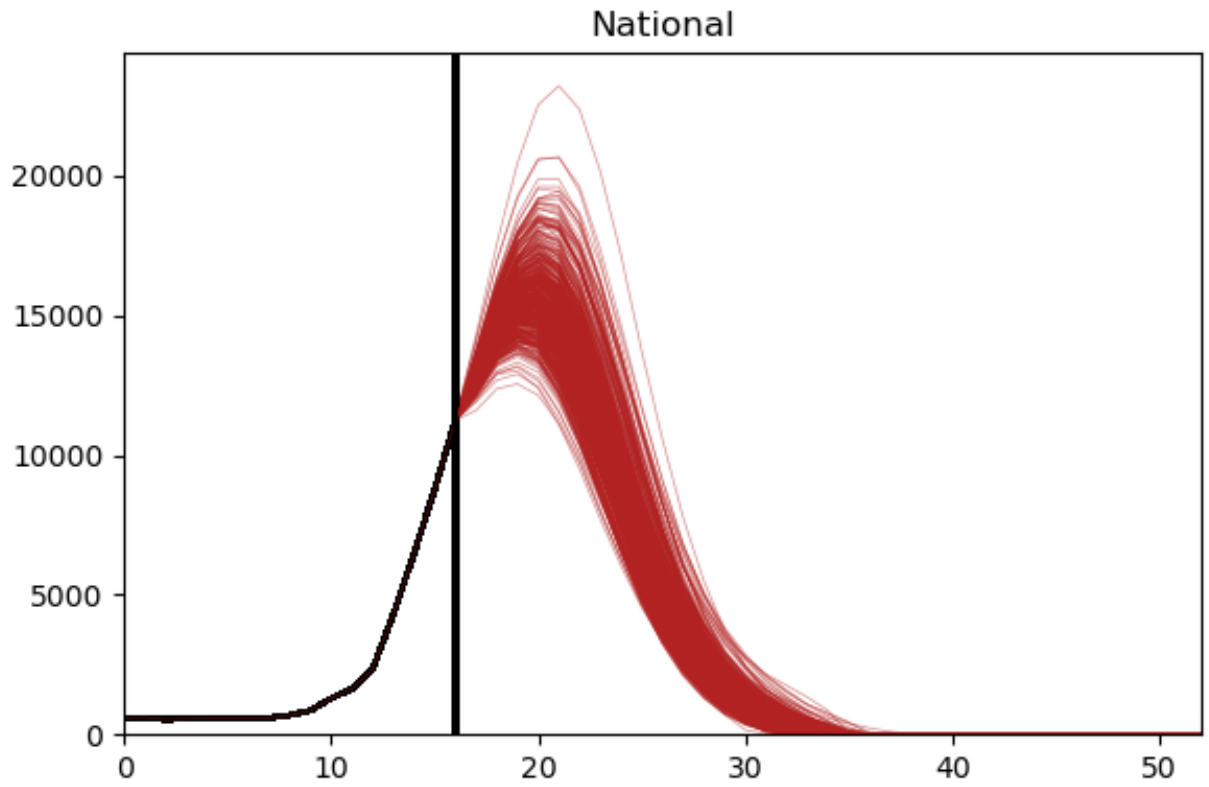
```
sampling loop time step:   0%|          | 0/200 [00:00<?, ?it/s]
```

In [57]:
```
samples[0].shape
```

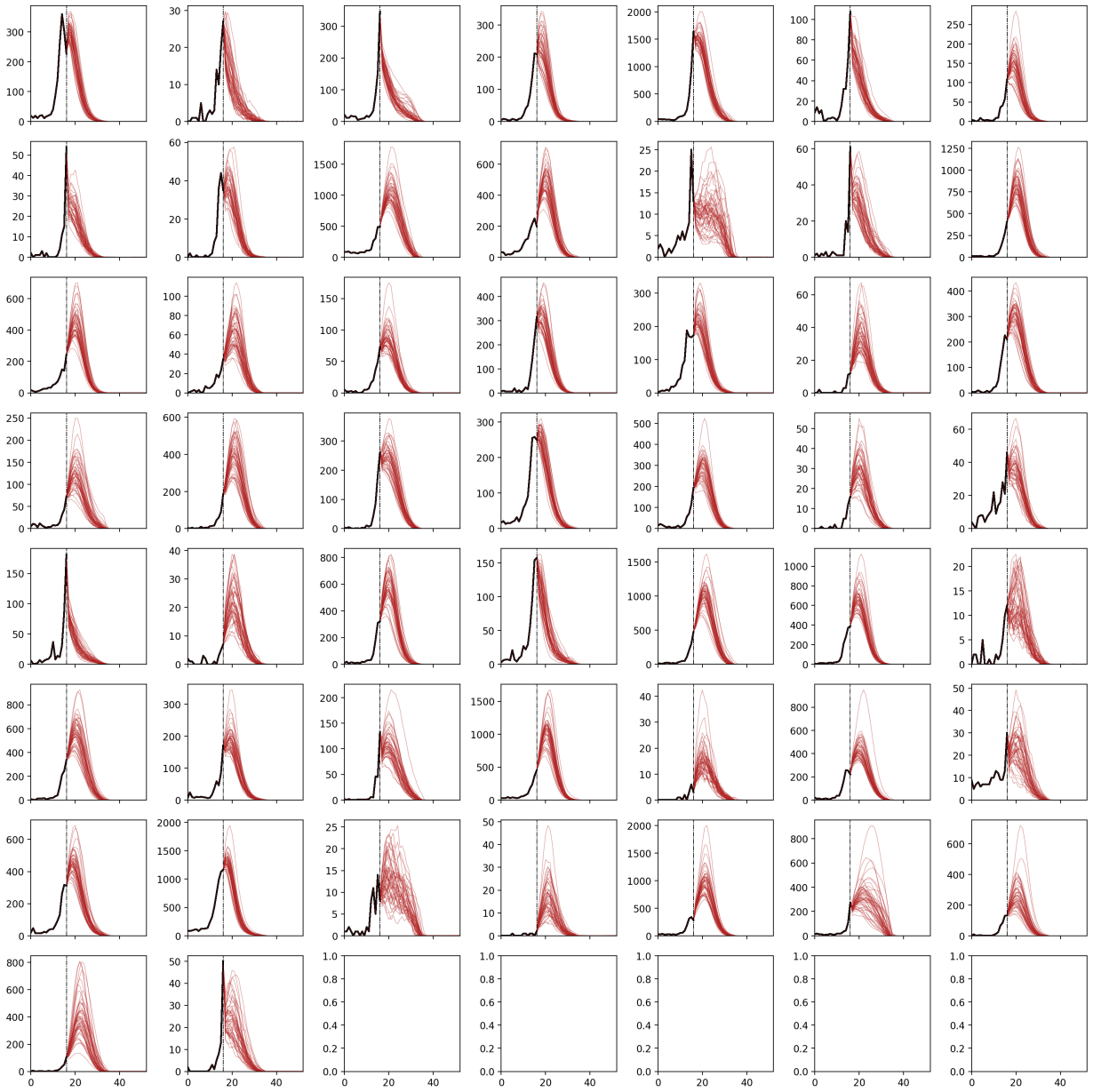
Out[57]:
```
(512, 1, 64, 64)
```

In [58]:
```
fluforecasts = -1*np.ones((batch_size*max(n_samples//batch_size,1), 1, 64
for i in range(max(n_samples//batch_size,1)):
    fluforecasts[i*batch_size:i*batch_size+batch_size] = all_samples[i][-
```

In [59]:
```
fig, axes = plt.subplots(1, 1, figsize=(6,4), dpi=100, sharex=True)

ax = axes
for i in range(batch_size):
    ax.plot(gt_xarr.data[0,:inpaintfrom_idx].sum(axis=1), color='k')
    show_tensor_image(samples[-1][i], ax = ax, multi=True)
    ax.axvline(inpaintfrom_idx-1, c='k')
    ax.set_xlim(0,52)
    ax.set_ylim(bottom=0, auto=True)
    #ax.grid(visible = True)
    ax.set_title("National")
fig.tight_layout()
plt.show()
```



In [60]:
```
fig, axes = plt.subplots(8, 7, figsize=(16,16), dpi=200, sharex=True)



for ipl in range(51):
    ax = axes.flat[ipl]
    for i in range(50):  # print max 50 sims
        ax.plot(gt_xarr.data[0,:inpaintfrom_idx, ipl], color='k')
        show_tensor_image(samples[-1][i], ax = ax, place=ipl, multi=True)
        ax.axvline(inpaintfrom_idx-1, c='k', lw=.7, ls='-.')
        ax.set_xlim(0,52)
        ax.set_ylim(bottom=0, auto=True)
        ax.grid()
        #ax.set_title(get_state_name(places[ipl]))
fig.tight_layout()
#plt.savefig("inpainting.pdf")
```



In [61]:
```
flusight_quantiles = np.append(np.append([0.01,0.025],np.arange(0.05,0.95
fluforecasts.shape
```

Out[61]:
```
(512, 1, 64, 64)
```

In [ ]:
```

```

In [62]:
```
flusight_quantile_pairs = np.array([flusight_quantiles[:11],flusight_quan
flusight_quantile_pairs
```

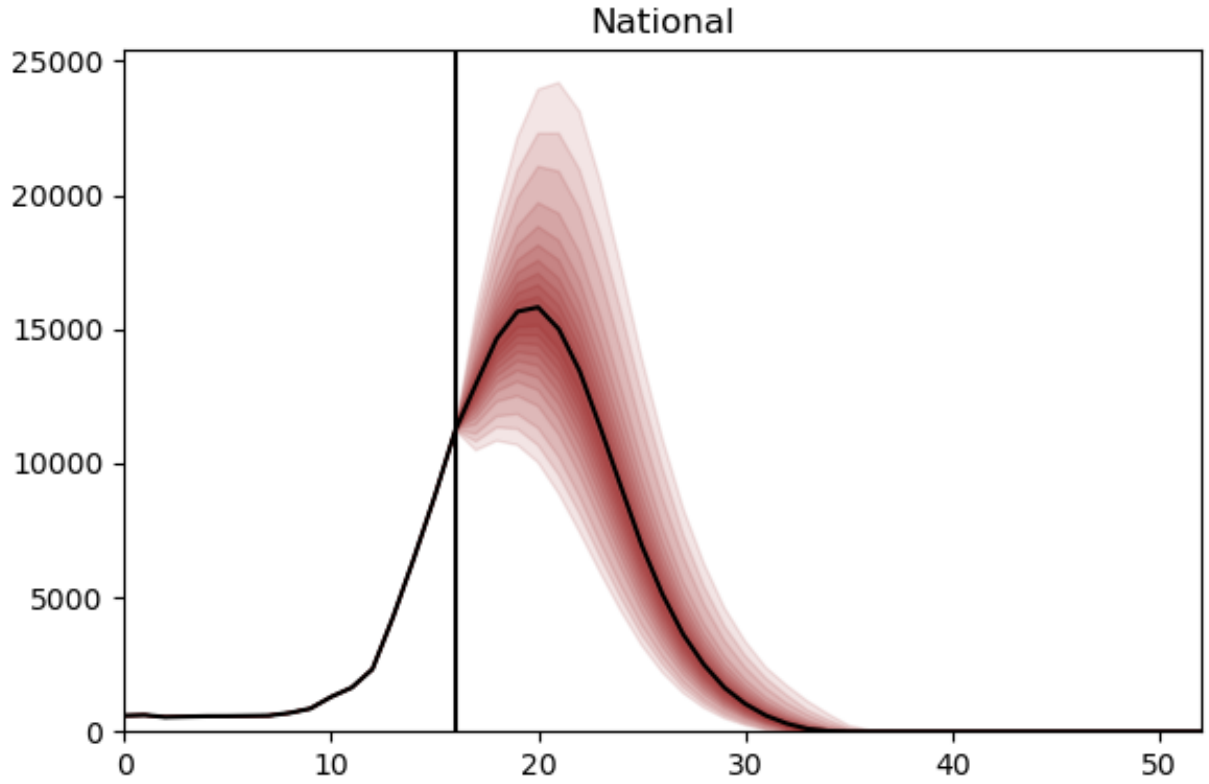
Out[62]:
```
array([[0.01 , 0.99 ],
       [0.025, 0.975],
       [0.05 , 0.95 ],
       [0.1  , 0.9  ],
       [0.15 , 0.85 ],
       [0.2  , 0.8  ],
       [0.25 , 0.75 ],
       [0.3  , 0.7  ],
       [0.35 , 0.65 ],
       [0.4  , 0.6  ],
       [0.45 , 0.55 ]])
```

In [ ]:

In [63]:
```
fluforecasts_ti = data.apply_transform_inv(fluforecasts)
```

In [64]:
```
fig, axes = plt.subplots(1, 1, figsize=(6,4), dpi=100, sharex=True)

ax = axes
for iqt in range(11):
    print(flusight_quantile_pairs[iqt,0], flusight_quantile_pairs[iqt,1])
    ax.fill_between(np.arange(64),
                    np.quantile(fluforecasts_ti, flusight_quantile_pairs[
                    np.quantile(fluforecasts_ti, flusight_quantile_pairs[

ax.plot(np.arange(64),
        np.quantile(fluforecasts_ti, flusight_quantiles[12], axis=0)[0].s
ax.axvline(inpaintfrom_idx-1, c='k')
ax.set_xlim(0,52)
ax.set_ylim(bottom=0, auto=True)
#ax.grid(visible = True)
ax.set_title("National")
fig.tight_layout()
plt.show()
```

```
0.01 0.99
0.025 0.975
0.05 0.9500000000000001
0.1 0.9000000000000001
0.15000000000000002 0.8500000000000001
0.2 0.8
0.25 0.7500000000000001
0.3 0.7000000000000001
0.35000000000000003 0.6500000000000001
0.4 0.6000000000000001
0.45 0.55
```

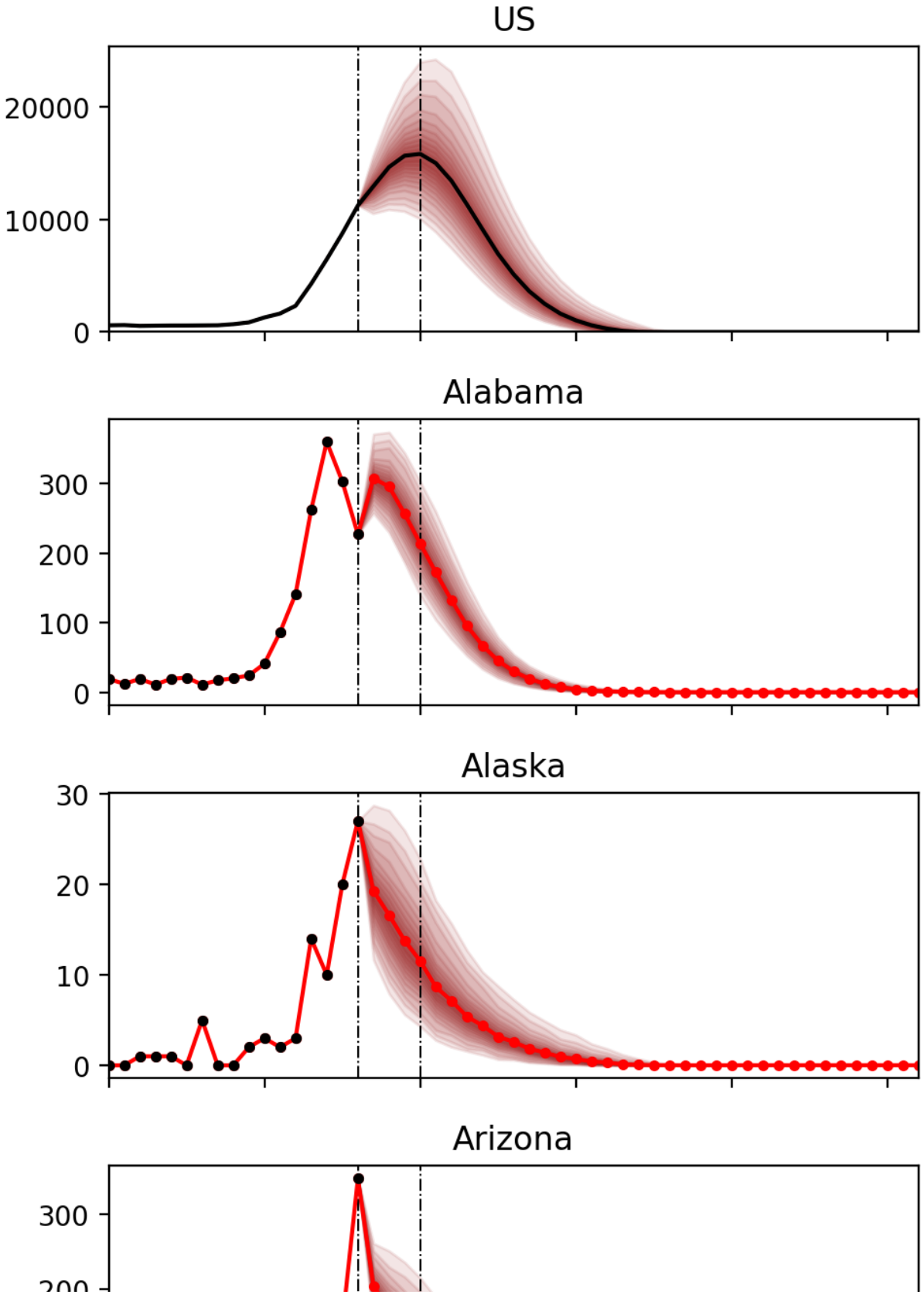


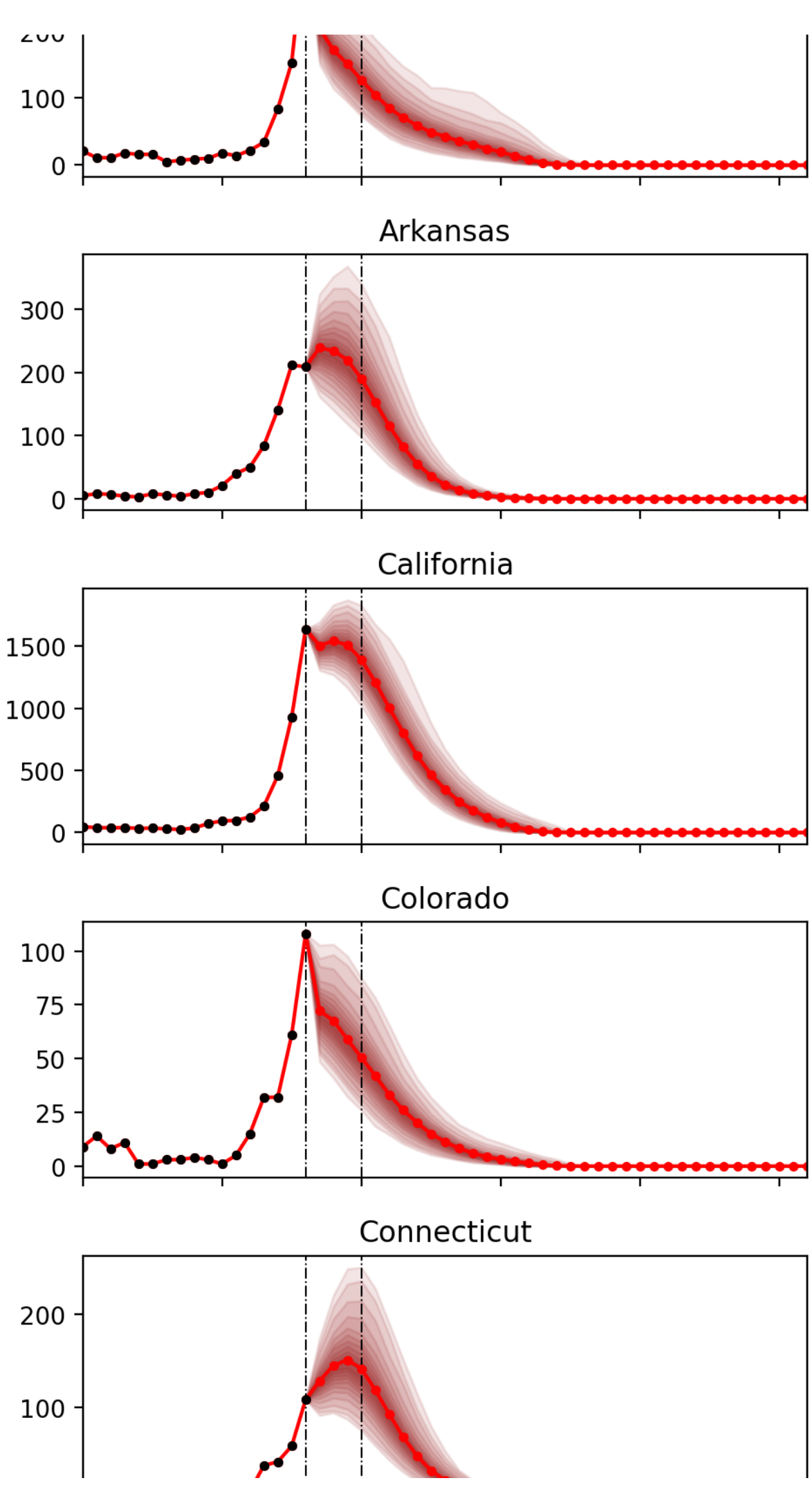
In [ ]:

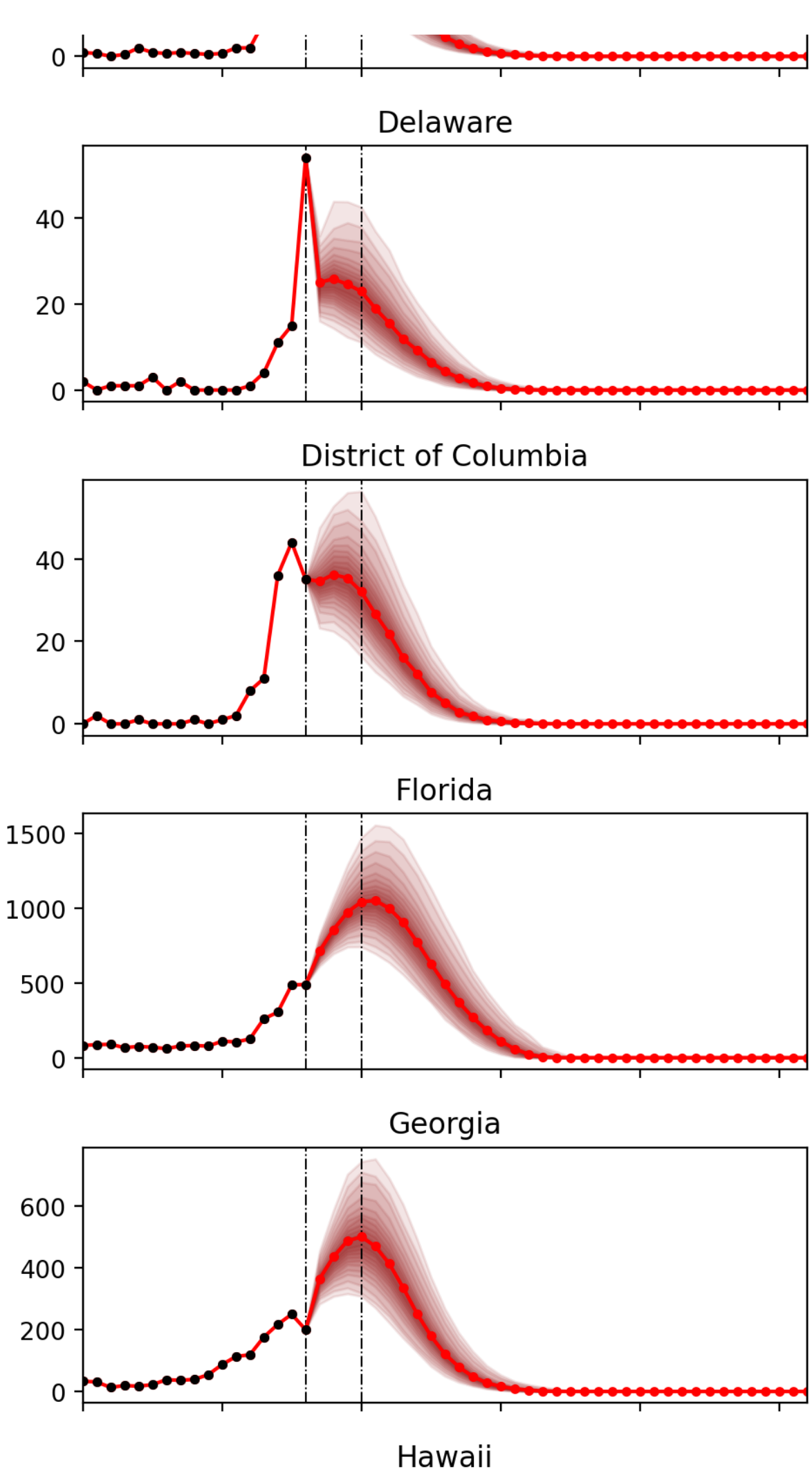
In [ ]:

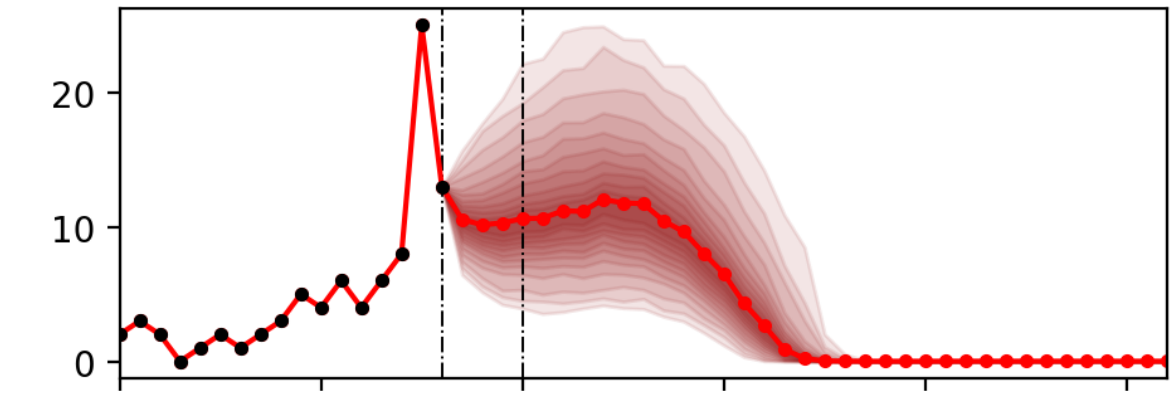
In [65]:
```
fig, axes = plt.subplots(52, 1, figsize=(5,100), dpi=200, sharex=True)

ax = axes[0]
for iqt in range(11):
    print(flusight_quantile_pairs[iqt,0], flusight_quantile_pairs[iqt,1])
    ax.fill_between(np.arange(64),
                    np.quantile(fluforecasts_ti, flusight_quantile_pairs[
                    np.quantile(fluforecasts_ti, flusight_quantile_pairs[

ax.plot(np.arange(64),
        np.quantile(fluforecasts_ti, flusight_quantiles[12], axis=0)[0].s
ax.set_xlim(0,52)
ax.set_ylim(bottom=0, auto=True)

ax.axvline(inpaintfrom_idx-1, c='k', lw=.7, ls='-.')
ax.axvline(inpaintfrom_idx-1+4, c='k', lw=.7, ls='-.')
ax.set_title("US")

for iqt in range(11):
    x = np.arange(64)
    yup = np.quantile(fluforecasts_ti, flusight_quantile_pairs[iqt,0], ax
    ylo = np.quantile(fluforecasts_ti, flusight_quantile_pairs[iqt,1], ax

    for ipl in range(51):
        ax = axes.flat[ipl+1]
        ax.fill_between(x[inpaintfrom_idx-1:],  yup[inpaintfrom_idx-1:,ip


for ipl in range(51):
    ax = axes.flat[ipl+1]
    ax.axvline(inpaintfrom_idx-1, c='k', lw=.7, ls='-.')
    ax.axvline(inpaintfrom_idx-1+4, c='k', lw=.7, ls='-.')

    ax.set_xlim(0,52)
    ax.set_ylim(bottom=0, auto=True)
    ax.plot(np.arange(64),
            np.quantile(fluforecasts_ti, flusight_quantiles[12], axis=0)[
    ax.plot(gt_xarr.data[0,:inpaintfrom_idx, ipl], color='k', ls = '', ma

    ax.set_title(flusetup.get_location_name(flusetup.locations[ipl]))
fig.tight_layout()
plt.savefig("inpainting.pdf")
```

```
0.01 0.99
0.025 0.975
0.05 0.9500000000000001
0.1 0.9000000000000001
0.15000000000000002 0.8500000000000001
0.2 0.8
0.25 0.7500000000000001
0.3 0.7000000000000001
0.35000000000000003 0.6500000000000001
0.4 0.6000000000000001
0.45 0.55
```

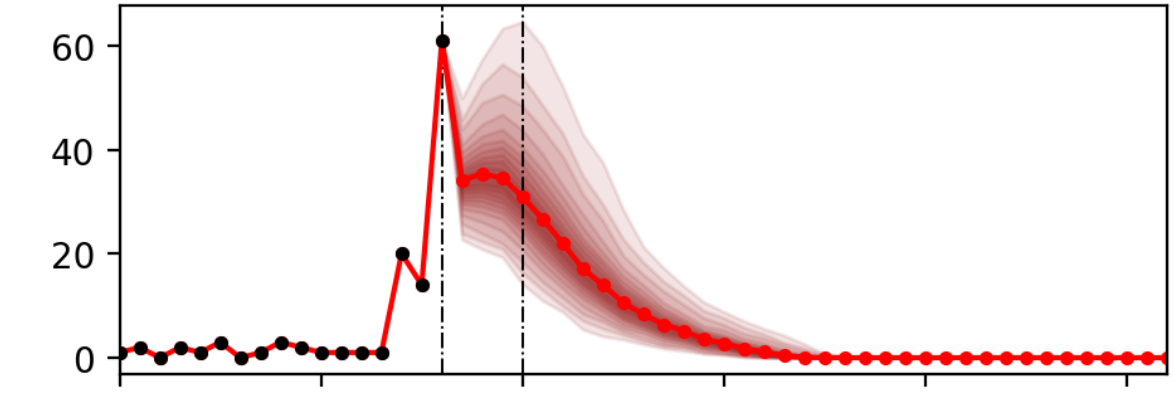



200
100
0
Arkansas
300
200
100
0
California
1500
1000
500
0
Colorado
100
75
50
25
0
Connecticut
200
100

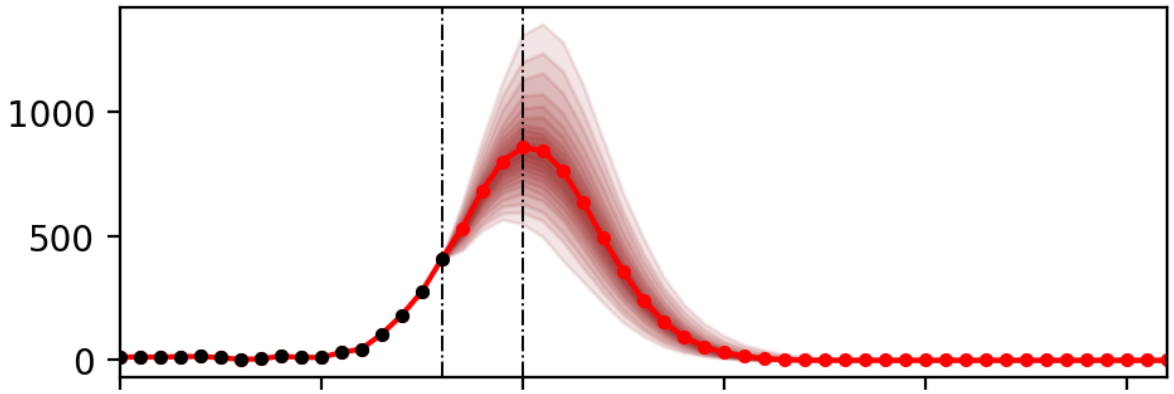

0
Delaware
40
20
0
District of Columbia
40
20
0
Florida
1500
1000
500
0
Georgia
600
400
200
0
Hawaii



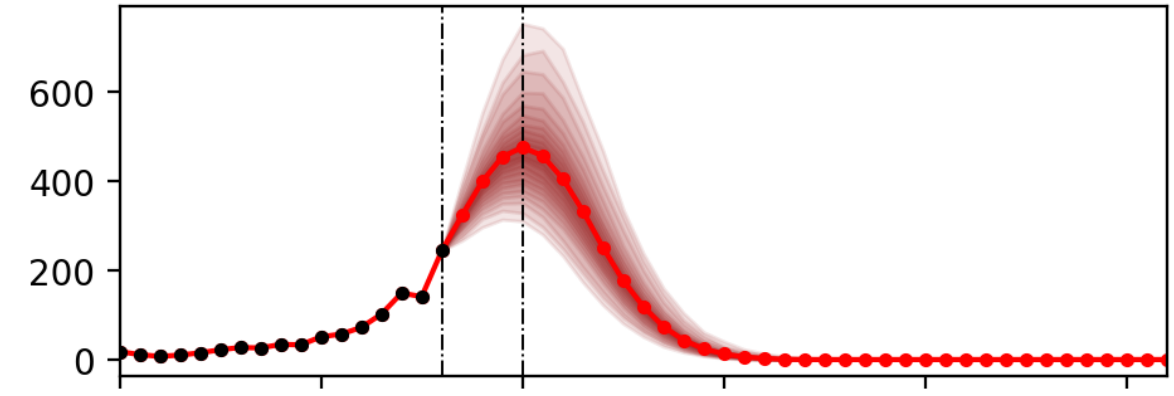
Idaho



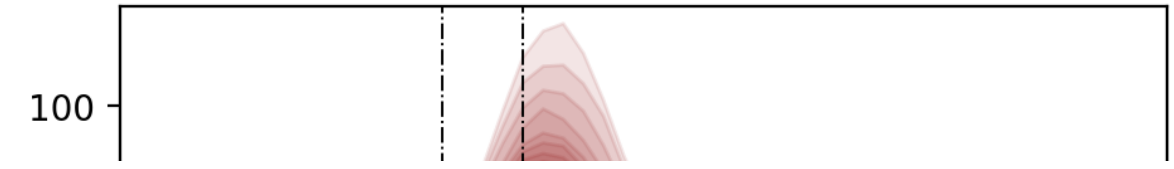
Illinois

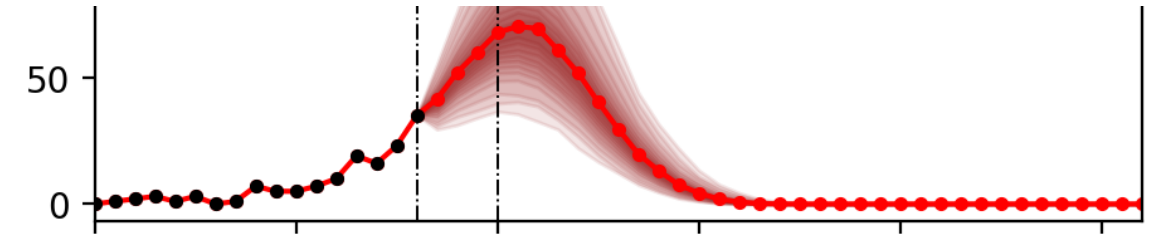


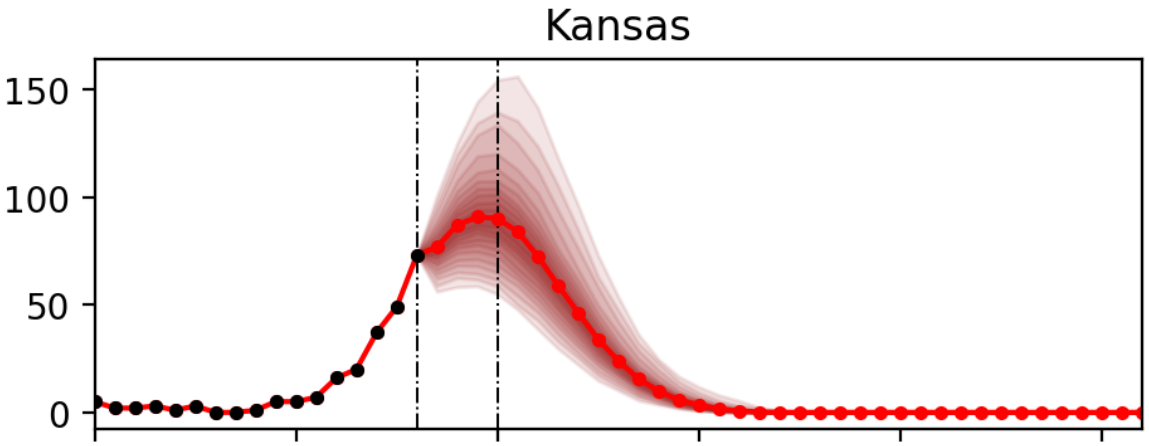
Indiana

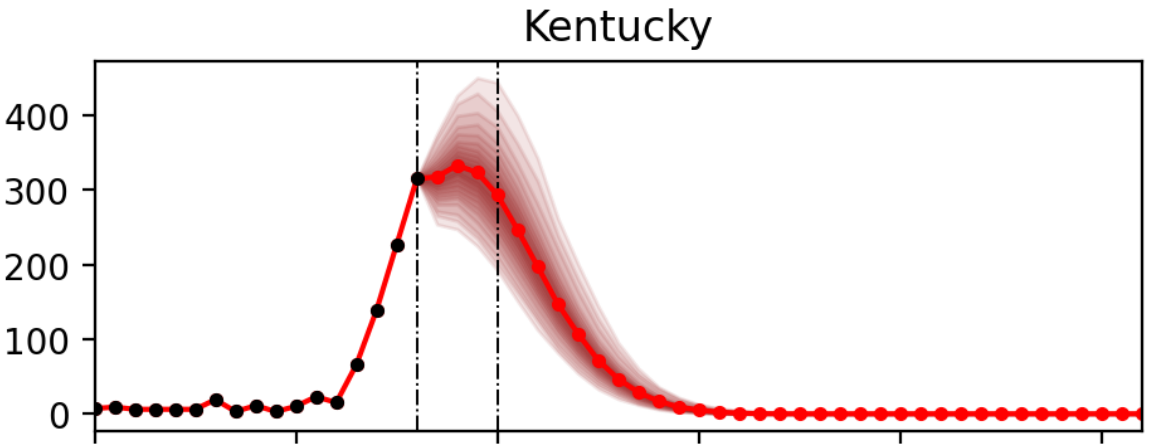


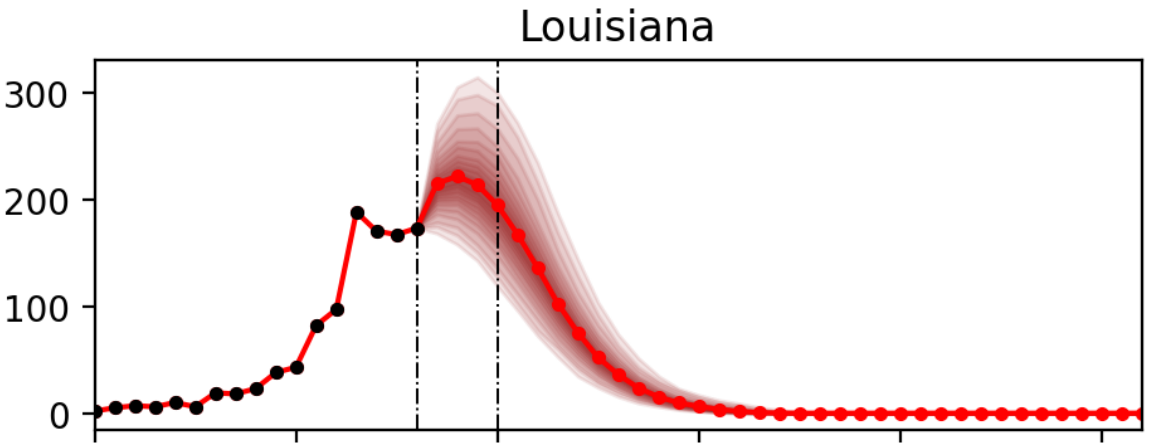
Iowa

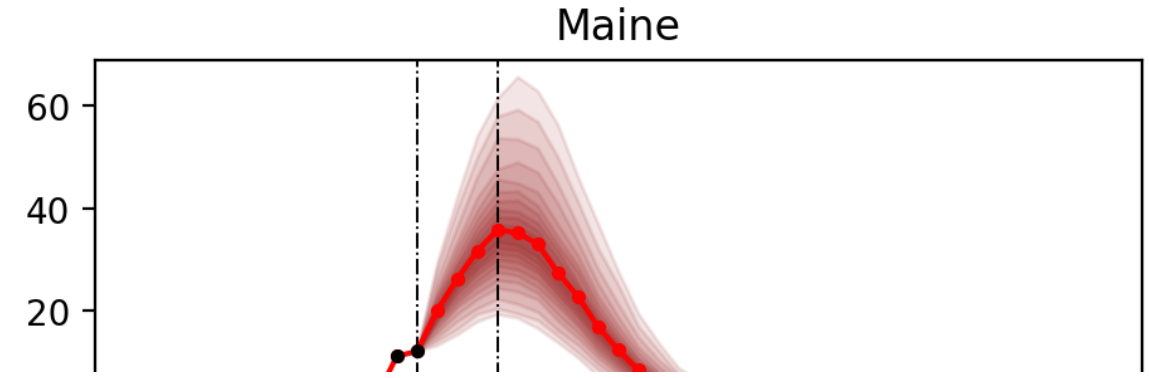


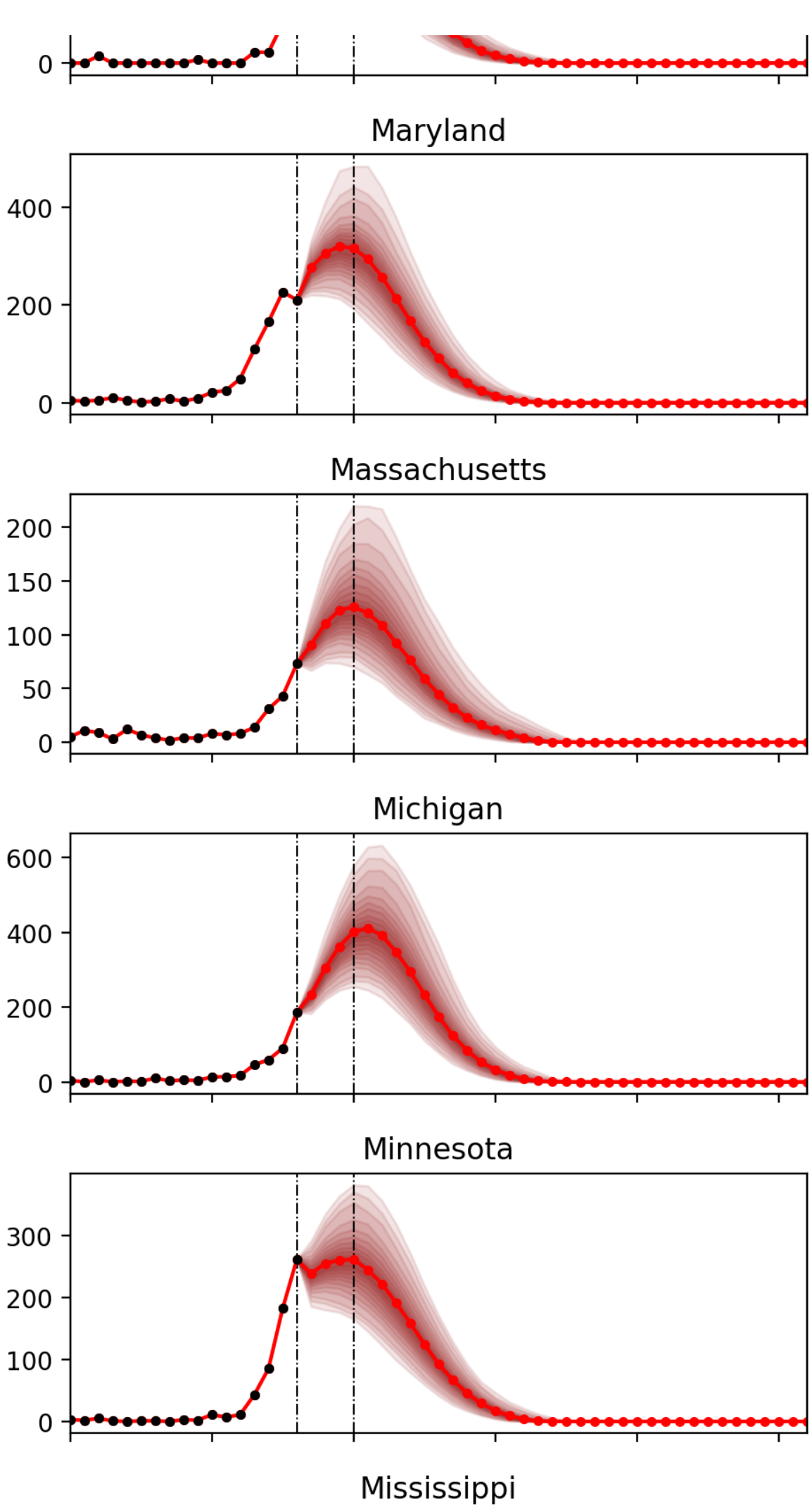

50
0

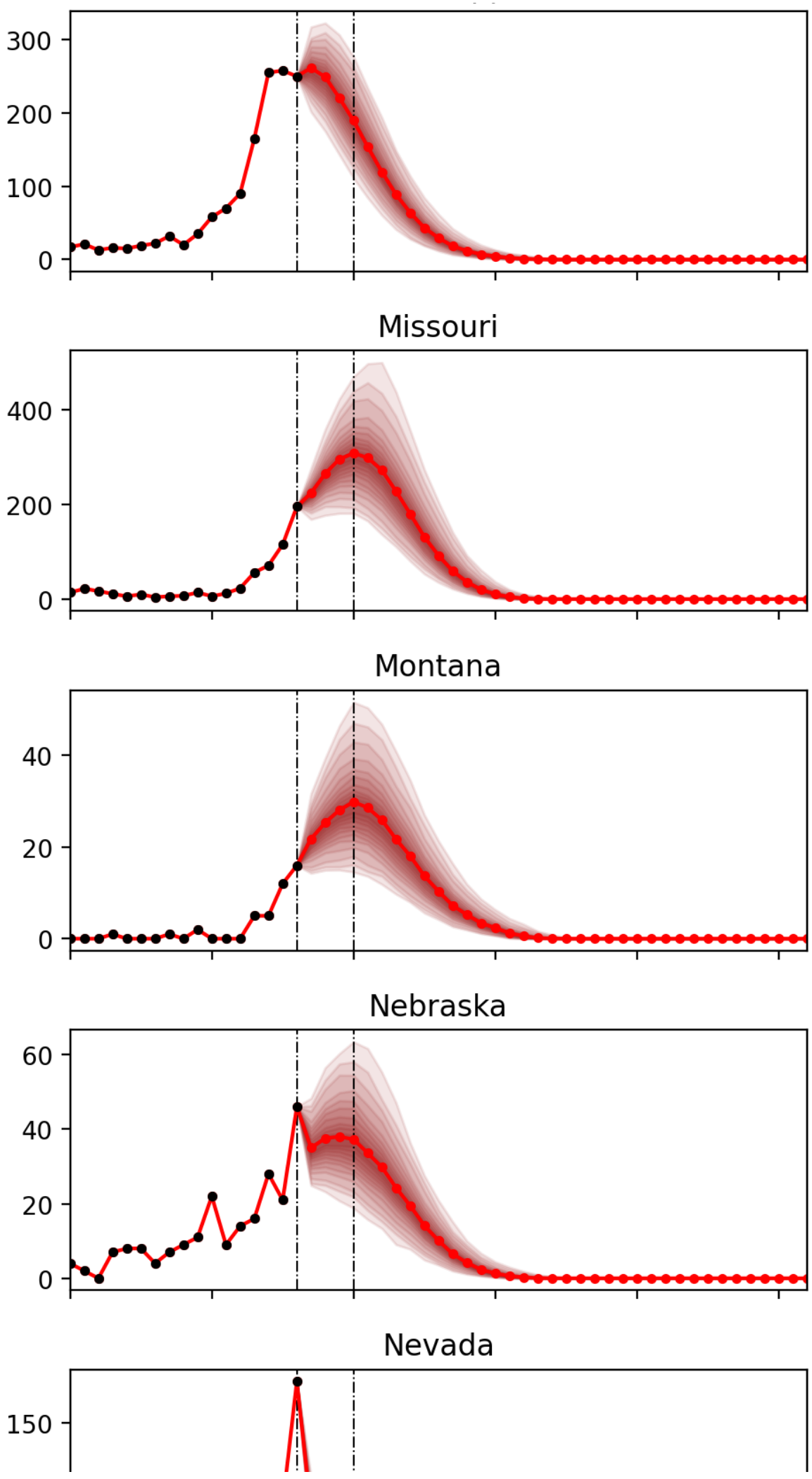

Kansas
150
100
50
0

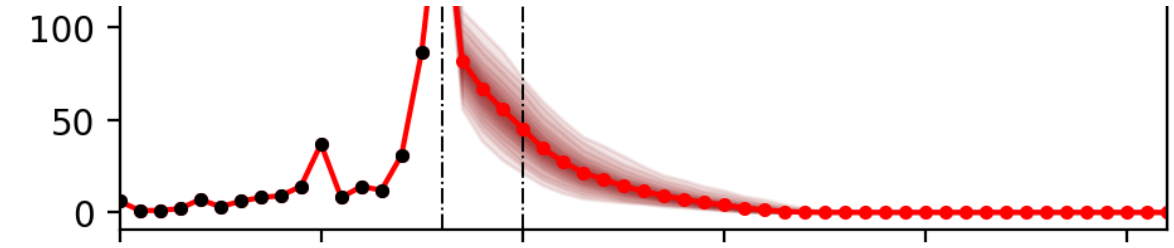

Kentucky
400
300
200
100
0

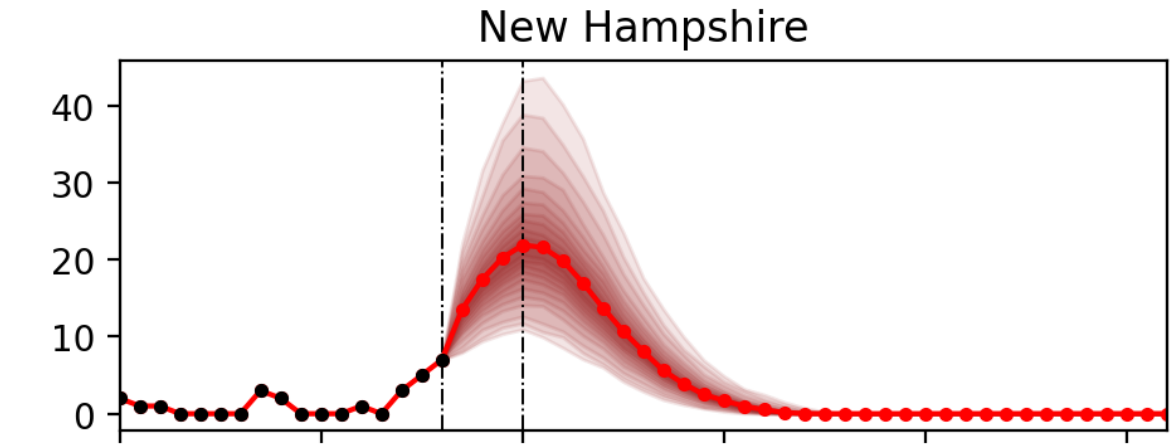

Louisiana
300
200
100
0

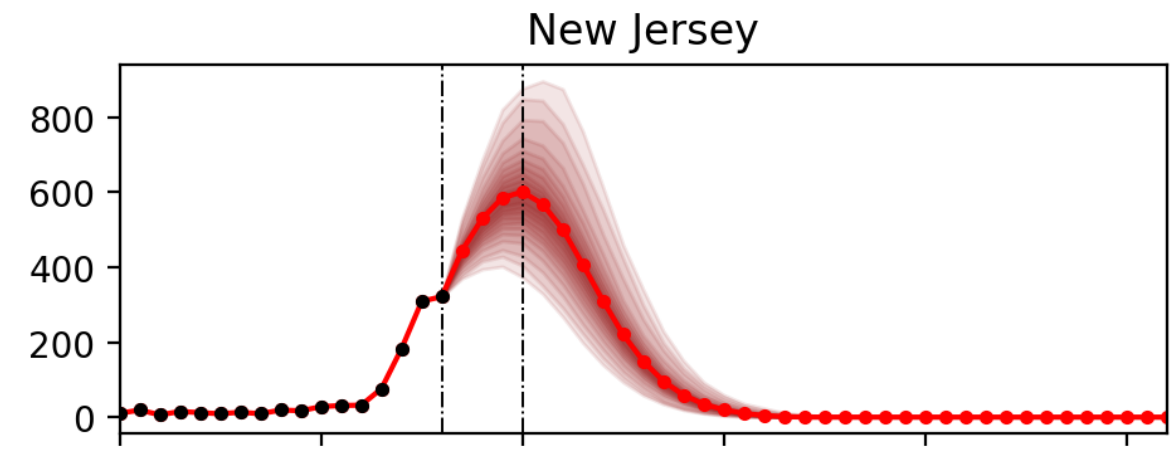

Maine
60
40
20

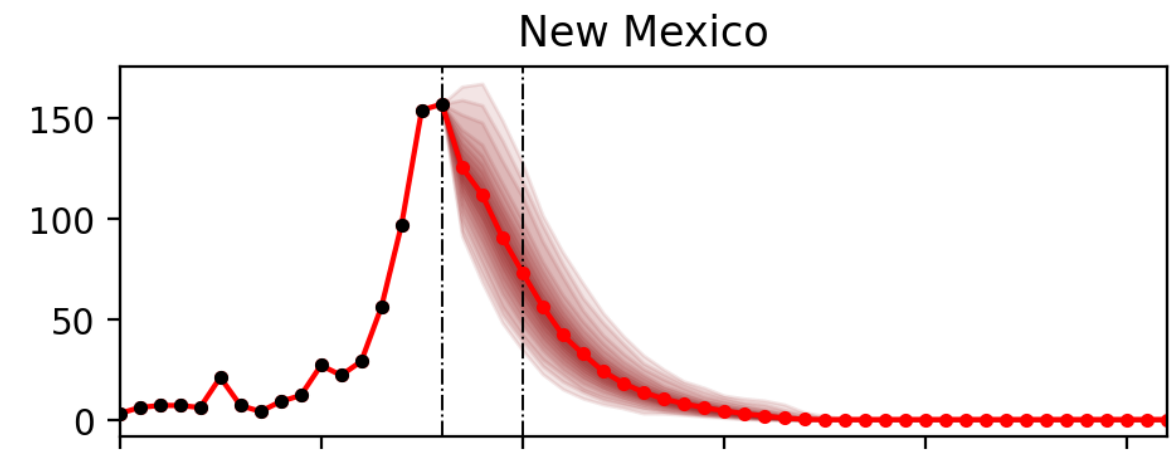

0
Maryland
400
200
0
Massachusetts
200
150
100
50
0
Michigan
600
400
200
0
Minnesota
300
200
100
0
Mississippi

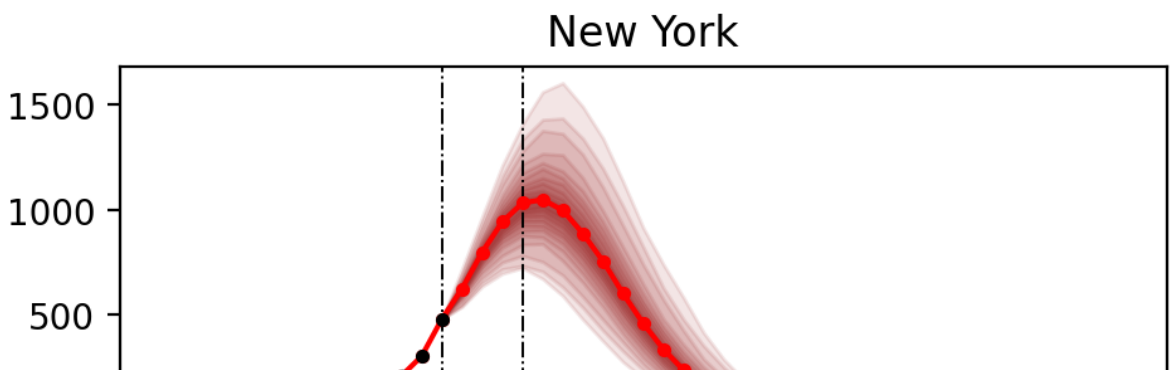

300
200
100
0
Missouri
400
200
0
Montana
40
20
0
Nebraska
60
40
20
0
Nevada
150

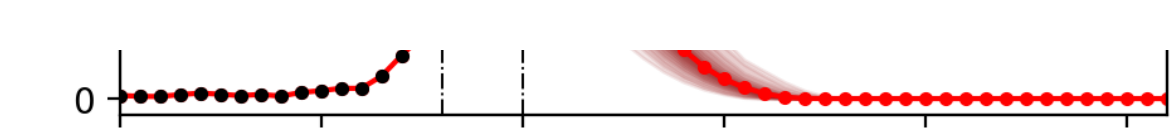

100
50
0


New Hampshire
40
30
20
10
0

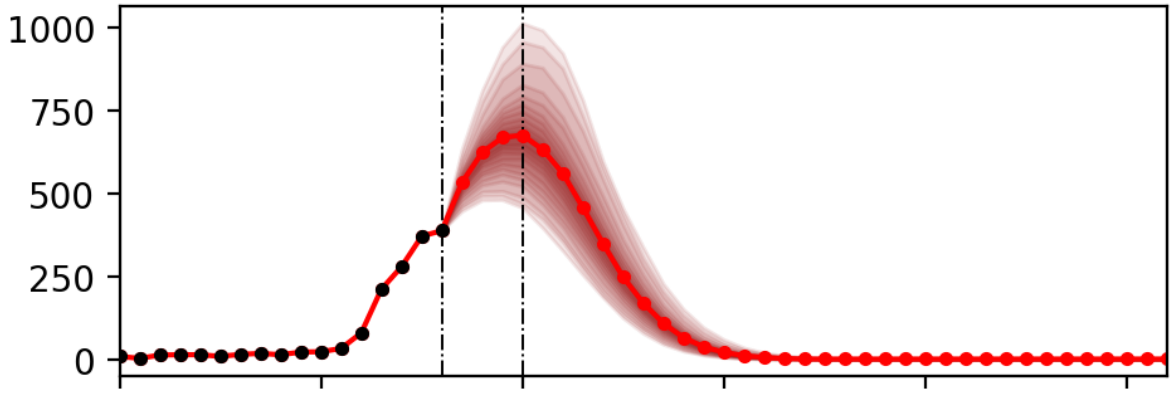

New Jersey
800
600
400
200
0


New Mexico
150
100
50
0

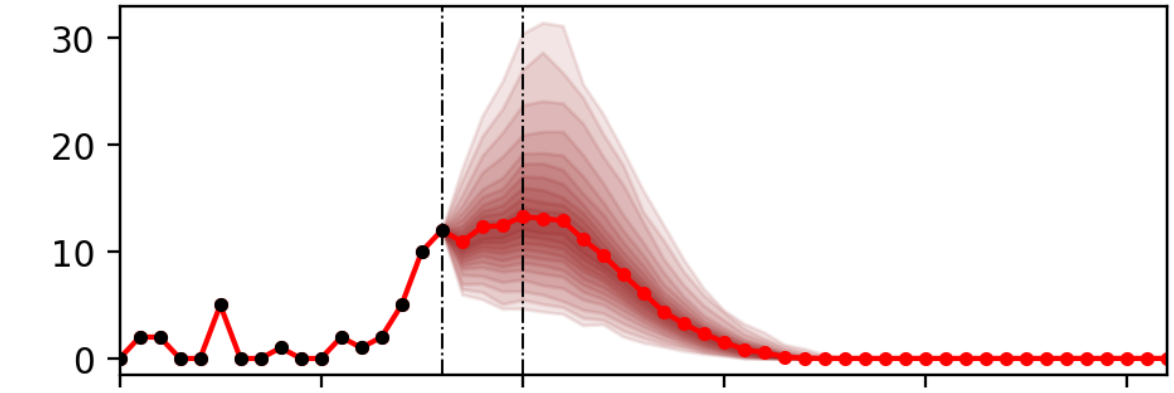

New York
1500
1000
500



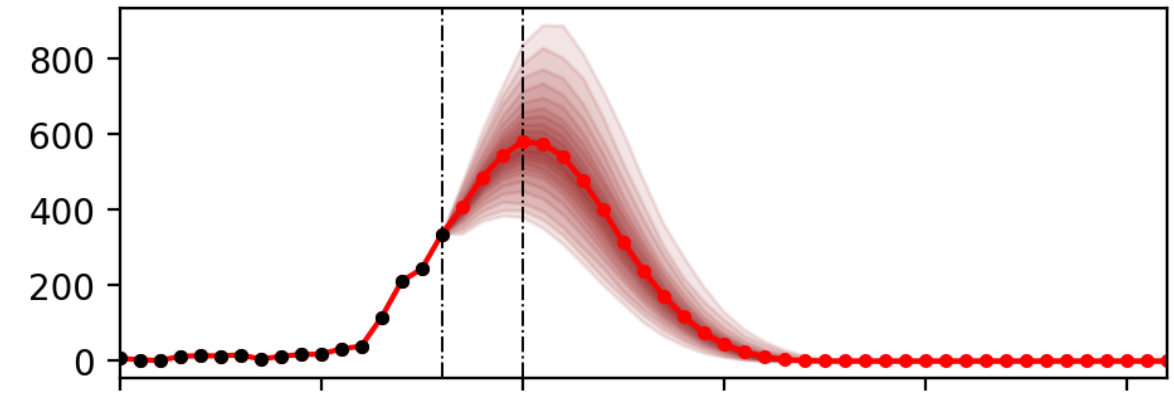
North Carolina



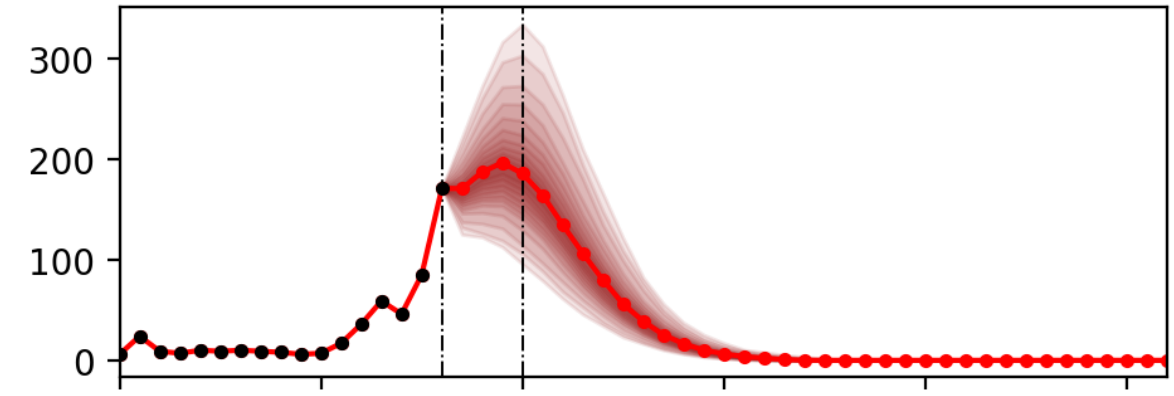
North Dakota



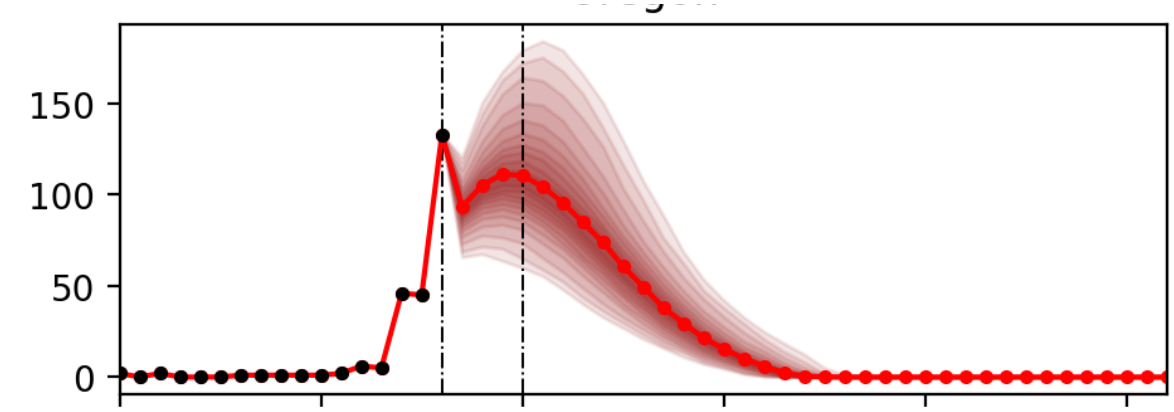
Ohio

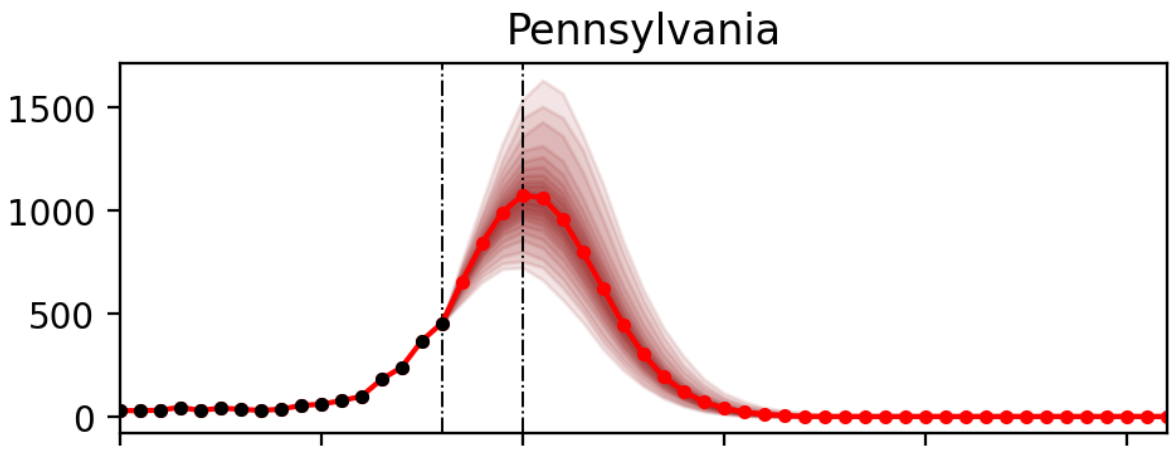


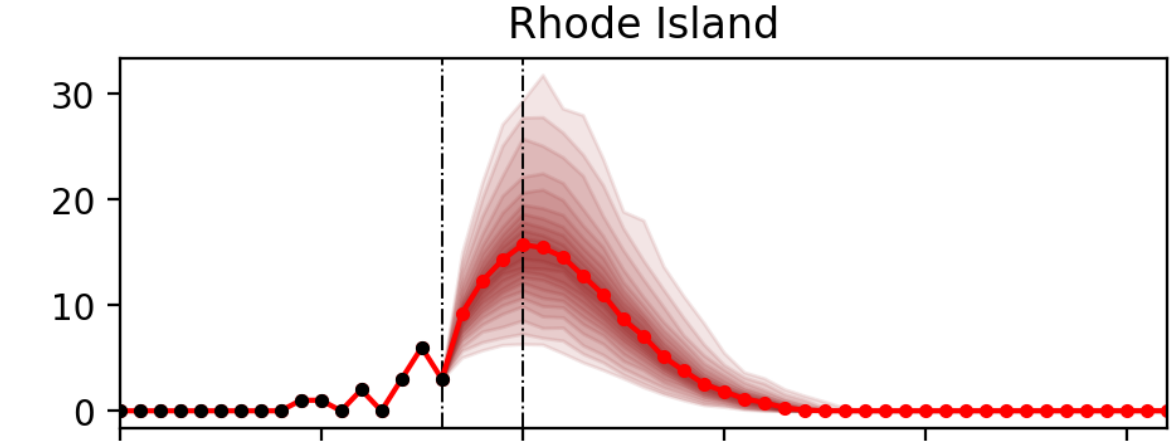
Oklahoma

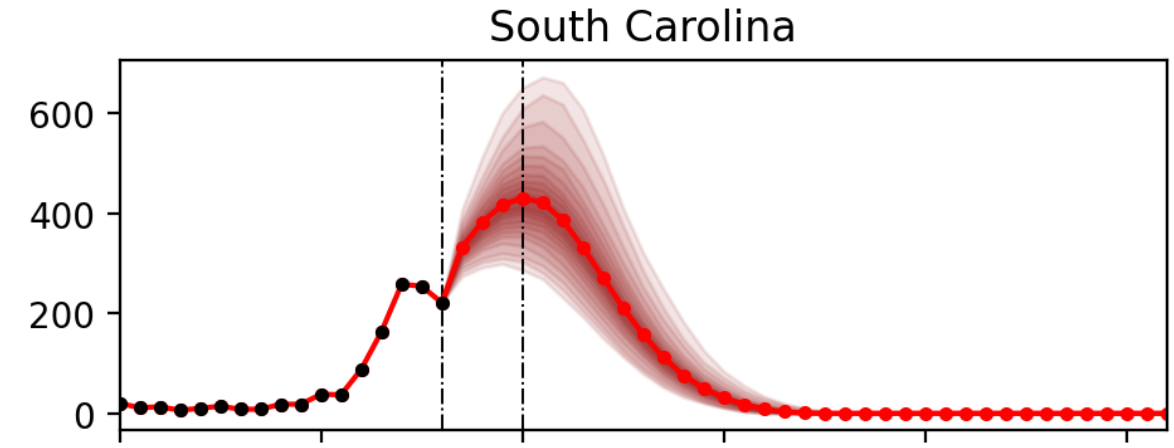


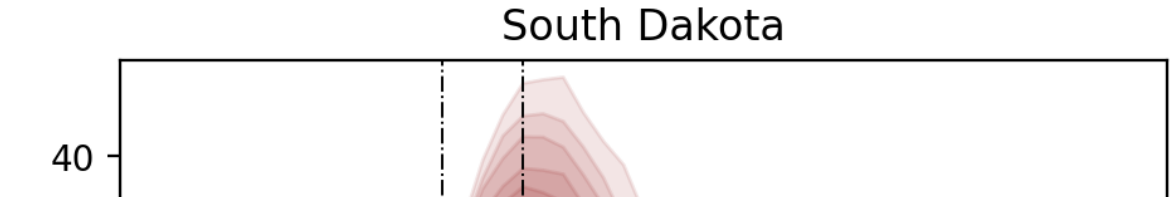
Oregon

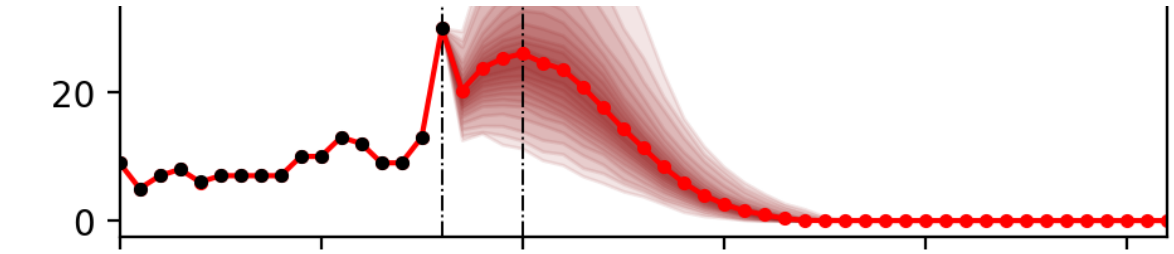

150
100
50
0

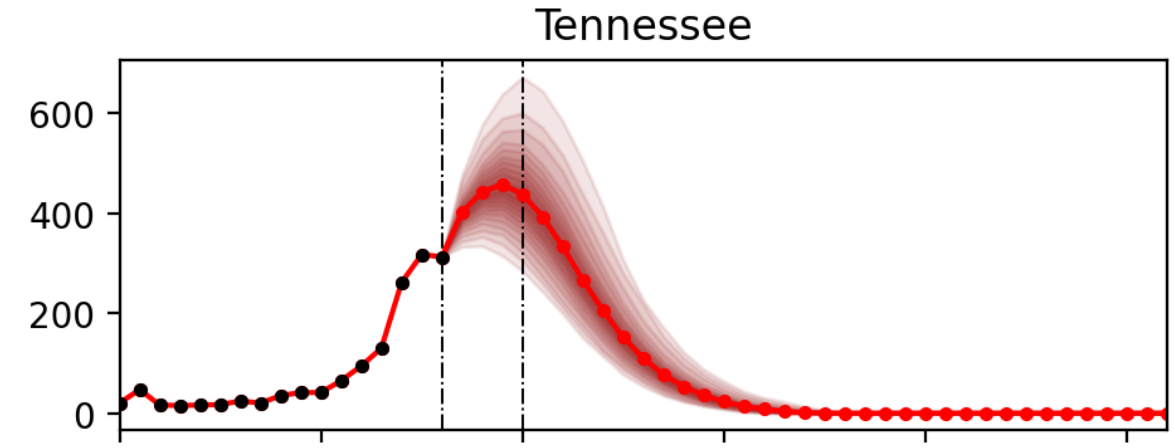

Pennsylvania
1500
1000
500
0

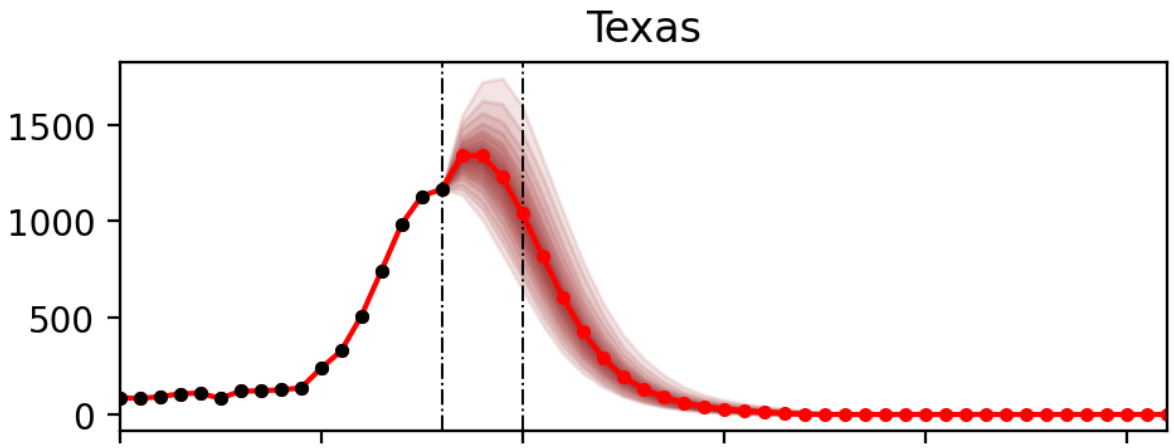

Rhode Island
30
20
10
0

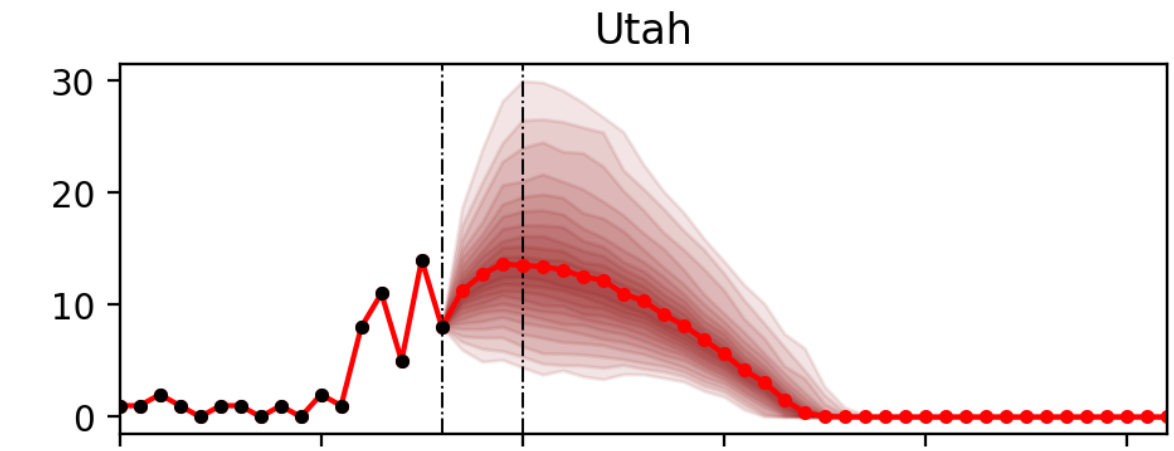

South Carolina
600
400
200
0

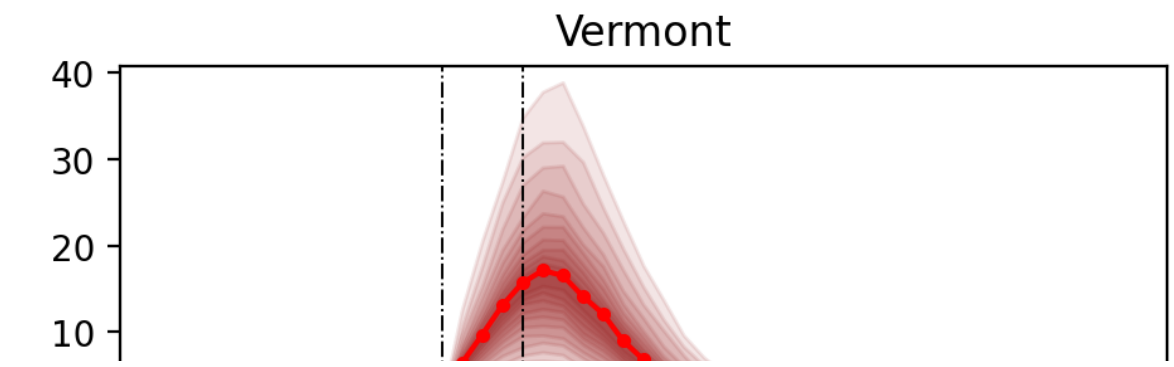

South Dakota
40


20
0

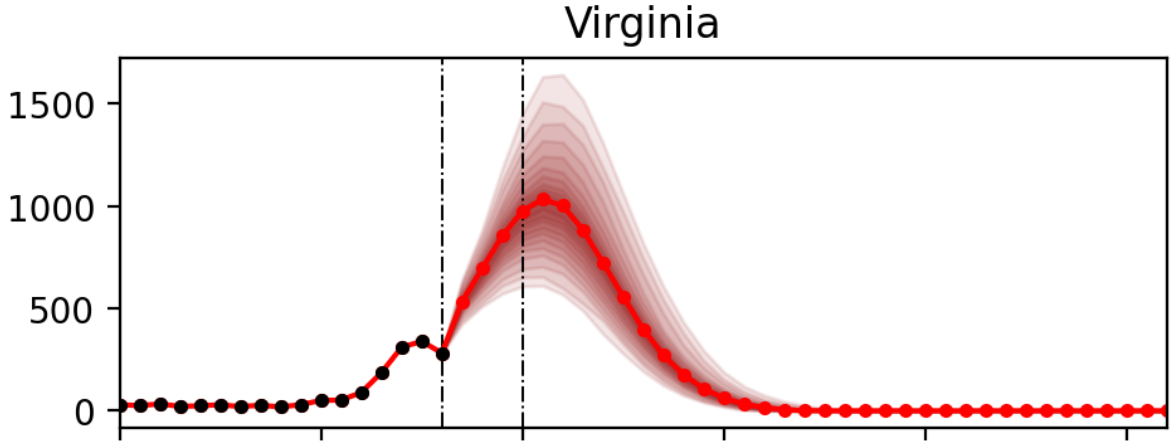

Tennessee
600
400
200
0

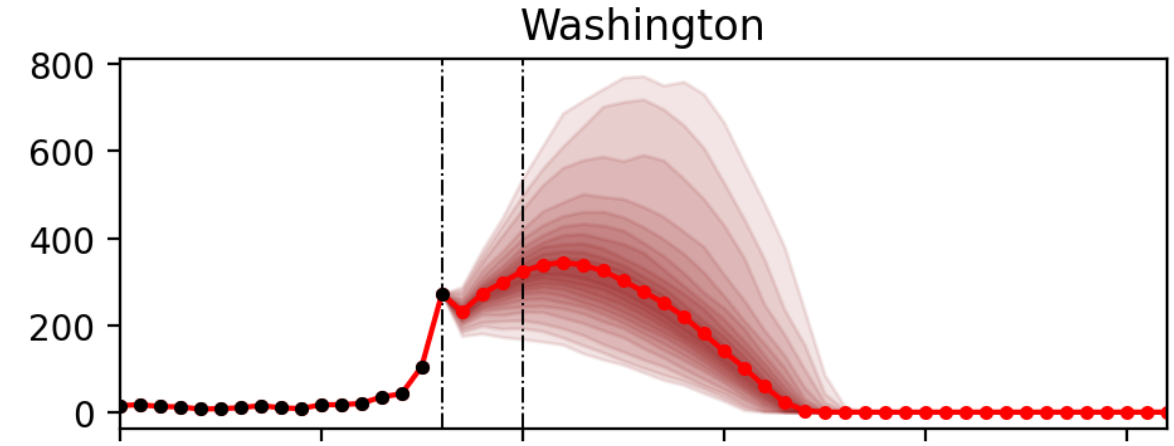

Texas
1500
1000
500
0

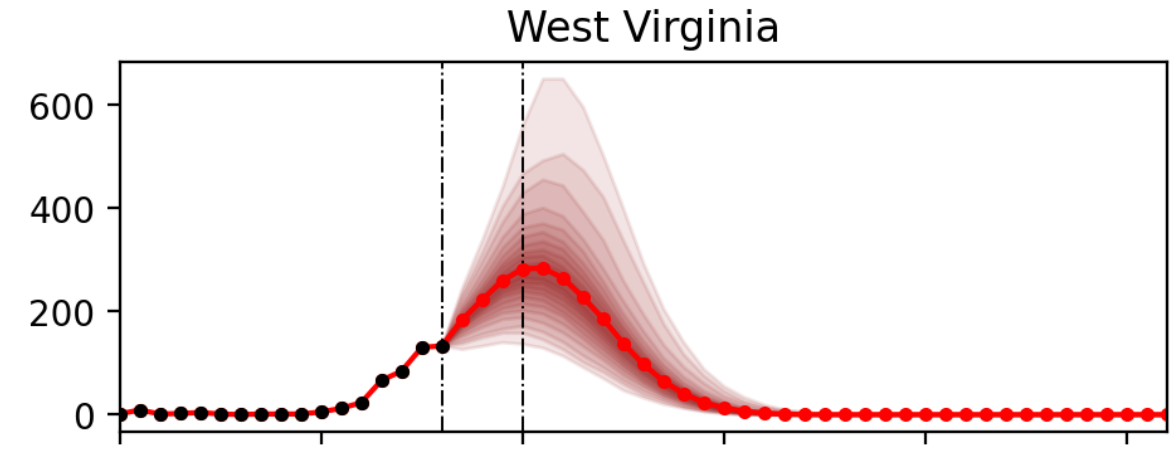

Utah
30
20
10
0

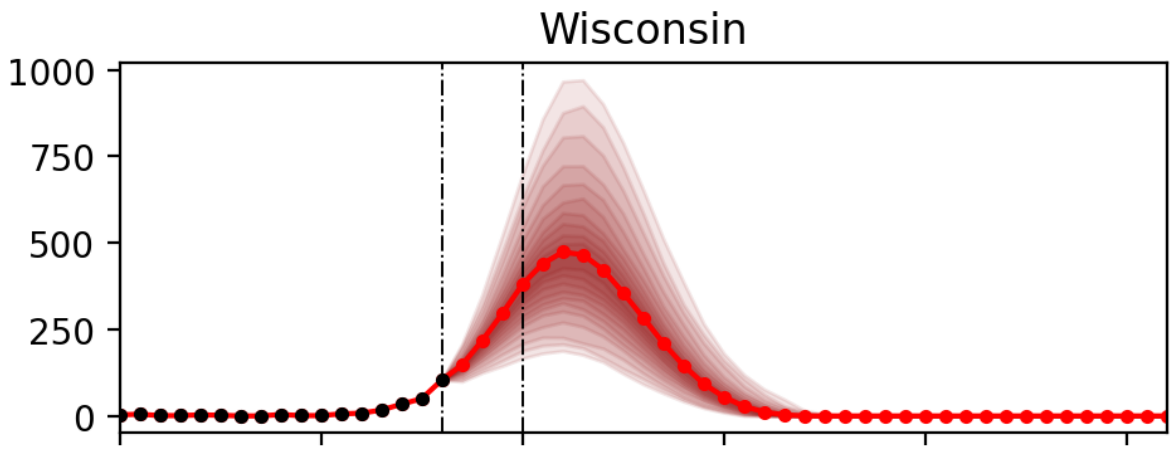

Vermont
40
30
20
10


0

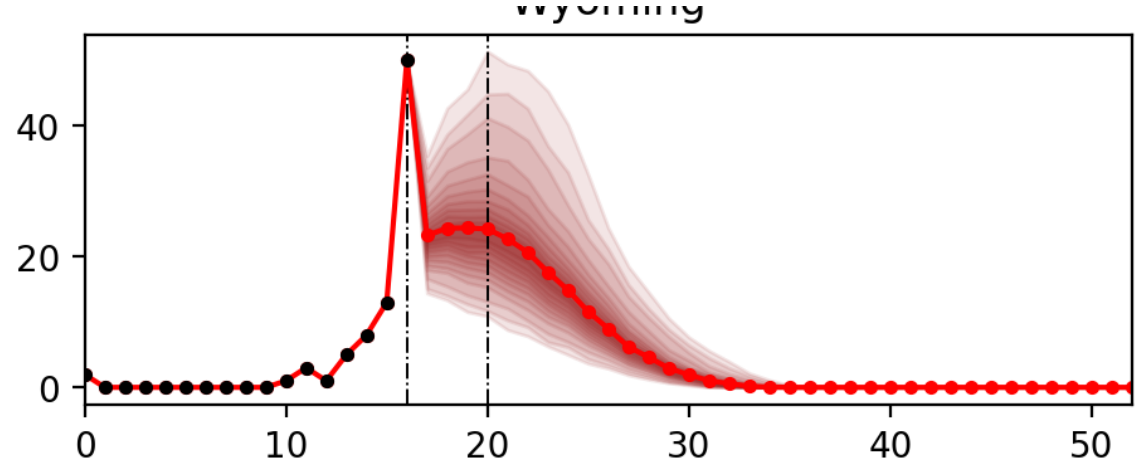

Virginia
1500
1000
500
0


Washington
800
600
400
200
0


West Virginia
600
400
200
0


Wisconsin
1000
750
500
250
0


Wyoming



In [66]:
```
## todo: line must be at gt-1
```

Data Export

In [67]:
```
forecast_date = datetime.date(2022,11,21)
forecast_date_str=str(datetime.date(2022,11,21))
team_abbrv = "UNC_IDD-InfluPaint"

target_dates = pd.date_range(forecast_date, forecast_date + datetime.time

target_dict= dict(zip(
    target_dates,
    [f"{n} wk ahead inc flu hosp" for n in range(1,5)]))

print(target_dates)
#pd.DataFrame(colums=["forecast_date","target_end_date","location","type"
df_list=[]
for qt in flusight_quantiles:
    a =  pd.DataFrame(data.apply_transform_inv(np.quantile(fluforecasts[:
              columns= flusetup.locations, index=pd.date_range(flusetup.f
    a["US"] = a.sum(axis=1)
    a = pd.melt(a.reset_index(names="target_end_date"),id_vars="target_en
    a["quantile"] = '{:<.3f}'.format(qt)

    df_list.append(a)
```

```
DatetimeIndex(['2022-11-26', '2022-12-03', '2022-12-10', '2022-12-17'], d
type='datetime64[ns]', freq='W-SAT')
```

In [68]:
```
df = pd.concat(df_list)
df["forecast_date"] = forecast_date_str
df["type"] = "quantile"
df["target"] = df["target_end_date"].map(target_dict)
df = df[["forecast_date","target_end_date","location","type","quantile","
df
```

Out[68]:

	forecast_date	target_end_date	location	type	quantile	value	target
0	2022-11-21	2022-11-26	01	quantile	0.010	256.965056	1 wk ahead inc flu hosp
1	2022-11-21	2022-12-03	01	quantile	0.010	229.520571	2 wk ahead inc flu hosp
2	2022-11-21	2022-12-10	01	quantile	0.010	185.982130	3 wk ahead inc flu hosp
3	2022-11-21	2022-12-17	01	quantile	0.010	139.111691	4 wk ahead inc flu hosp
4	2022-11-21	2022-11-26	02	quantile	0.010	11.651919	1 wk ahead inc flu hosp
...	...	...	...	...	...	...	...
203	2022-11-21	2022-12-17	56	quantile	0.990	51.405429	4 wk ahead inc flu hosp
204	2022-11-21	2022-11-26	US	quantile	0.990	15642.632342	1 wk ahead inc flu hosp
205	2022-11-21	2022-12-03	US	quantile	0.990	19252.057281	2 wk ahead inc flu hosp
206	2022-11-21	2022-12-10	US	quantile	0.990	22141.435993	3 wk ahead inc flu hosp
207	2022-11-21	2022-12-17	US	quantile	0.990	23933.338259	4 wk ahead inc flu hosp

4784 rows × 7 columns

In [69]:

```
for col in df.columns:
    print(col)
    print(df[col].unique())
```

```
forecast_date
['2022-11-21']
target_end_date
['2022-11-26T00:00:00.000000000' '2022-12-03T00:00:00.000000000'
 '2022-12-10T00:00:00.000000000' '2022-12-17T00:00:00.000000000']
location
['01' '02' '04' '05' '06' '08' '09' '10' '11' '12' '13' '15' '16' '17'
 '18' '19' '20' '21' '22' '23' '24' '25' '26' '27' '28' '29' '30' '31'
 '32' '33' '34' '35' '36' '37' '38' '39' '40' '41' '42' '44' '45' '46'
 '47' '48' '49' '50' '51' '53' '54' '55' '56' 'US']
type
['quantile']
quantile
['0.010' '0.025' '0.050' '0.100' '0.150' '0.200' '0.250' '0.300' '0.350'
 '0.400' '0.450' '0.500' '0.550' '0.600' '0.650' '0.700' '0.750' '0.800'
 '0.850' '0.900' '0.950' '0.975' '0.990']
value
[  256.96505631   229.52057124   185.98212986 ... 19252.05728055
 22141.43599316 23933.3382586 ]
target
['1 wk ahead inc flu hosp' '2 wk ahead inc flu hosp'
 '3 wk ahead inc flu hosp' '4 wk ahead inc flu hosp']
```

In [70]:
```
assert sum(df["value"]<0) == 0
assert sum(df["value"].isna()) == 0
```

In [71]:
```
df.to_csv(f"{forecast_date_str}-{team_abbrv}.csv", index=False)
```

In [ ]:
```
```

In [72]:
```
a = a
```

In [ ]:
```
```

In [73]:
```
import requests
requests.post("https://ntfy.sh/chadi_modeling",
    data="Notebook finshed running !",
    headers={
        "Title": "Inpainting-diffusion",
        "Priority": "urgent",
        "Tags": "warning,tada"
    })
```

Out[73]:
```
<Response [200]>
```