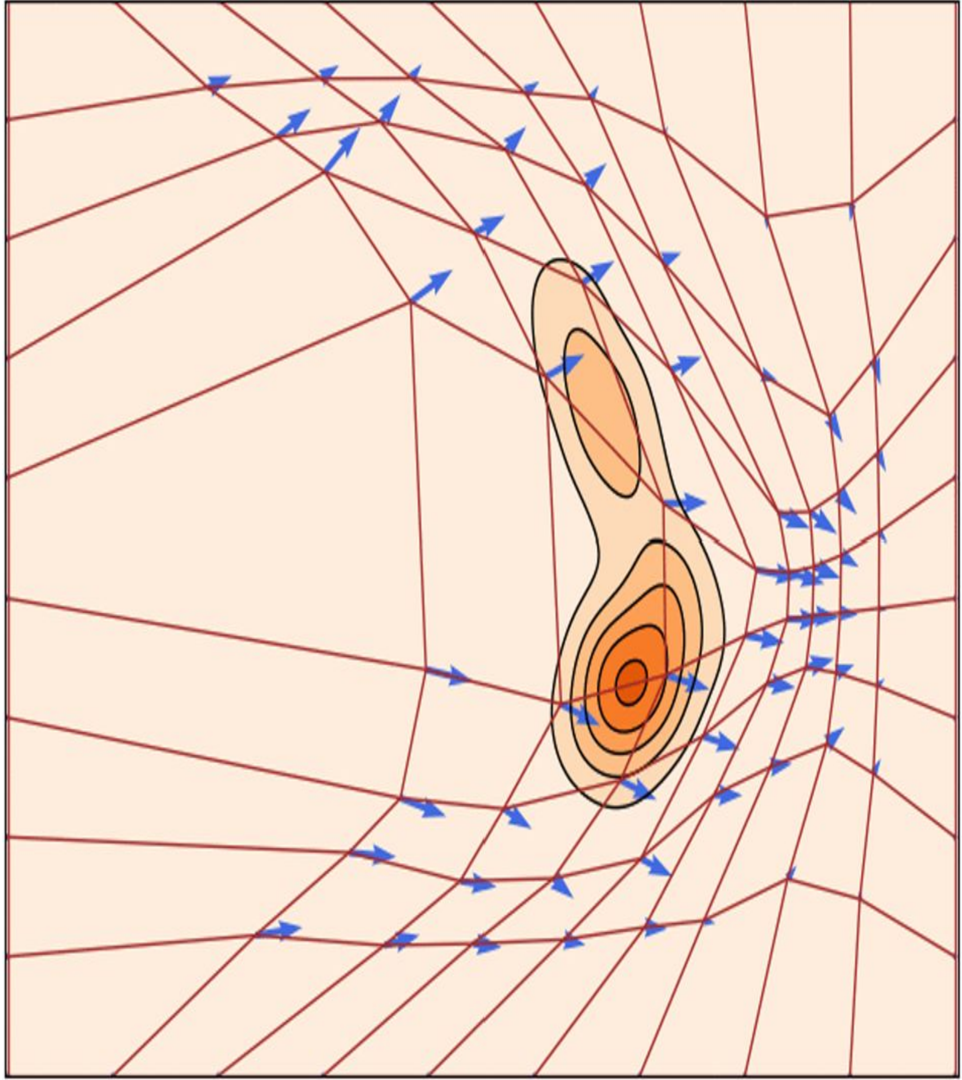
Flow Based Models

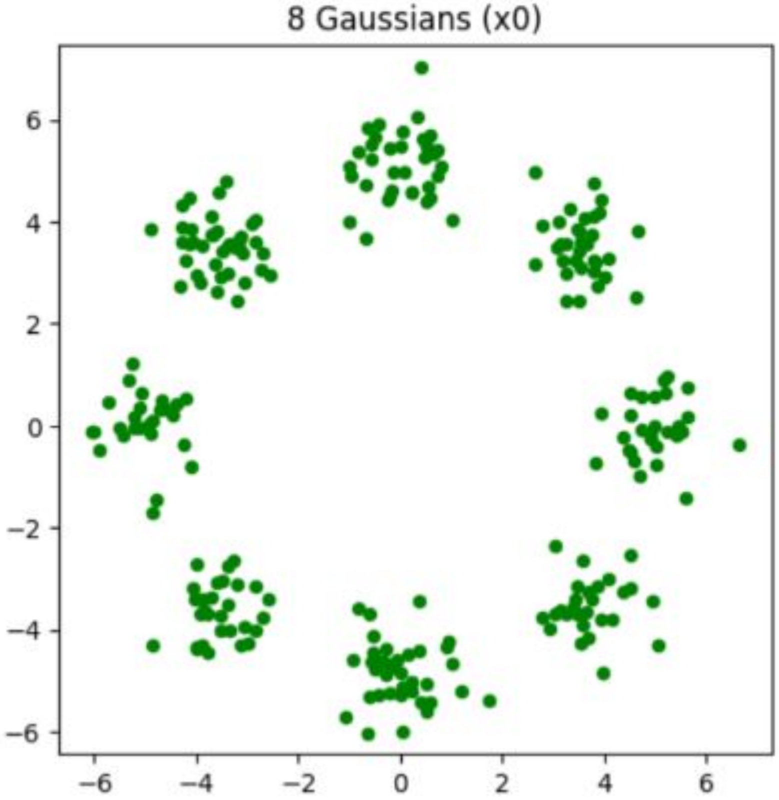
James Iorio
Mentored by: Sebastian Gutierrez-Hernandez

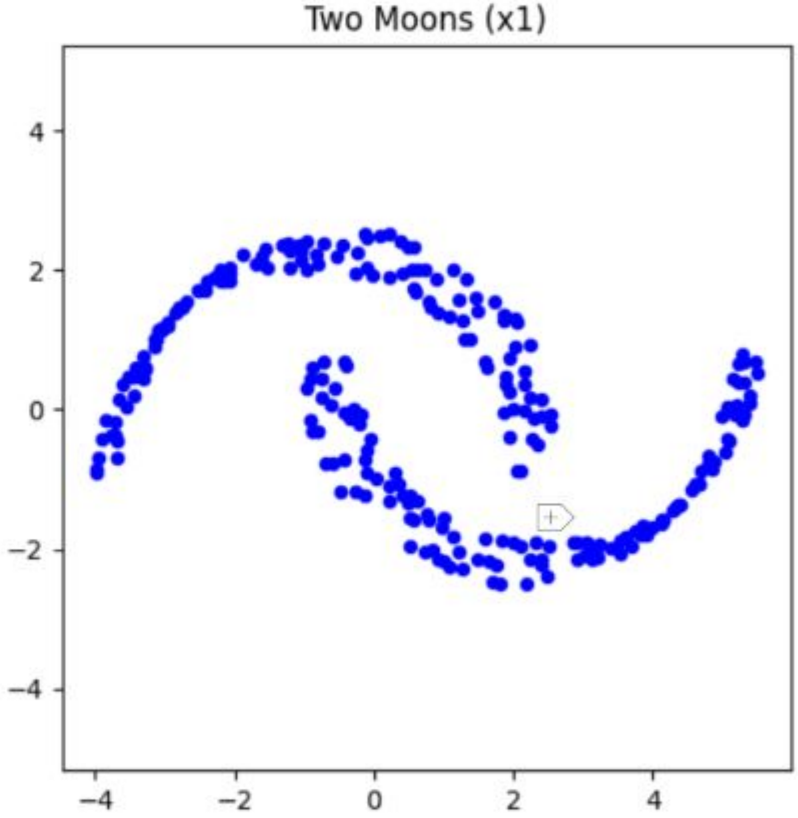


What's Covered

- Components of Flow Models
  - Couplings
  - Sigma
  - Paths
- Image Generation Example
- Challenges

Components of Flow Models: Coupling

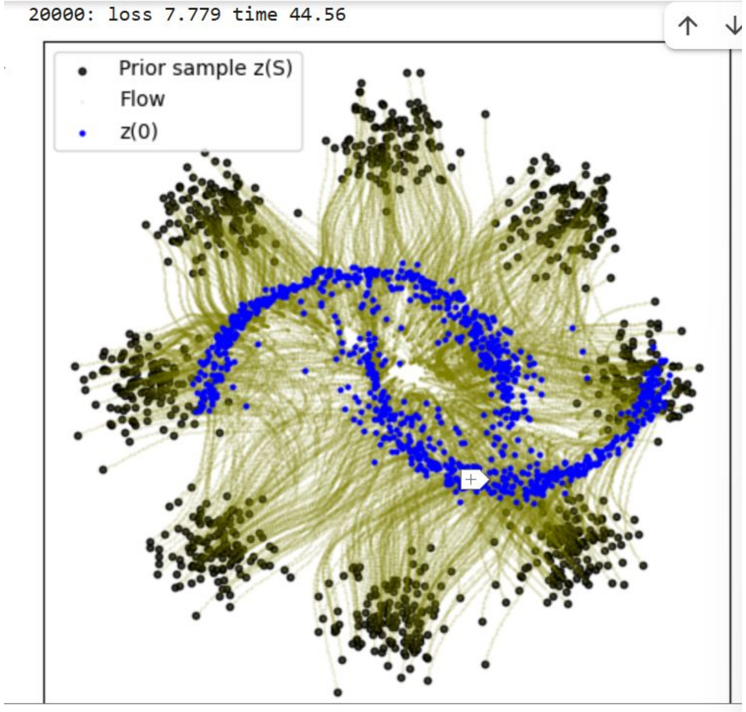


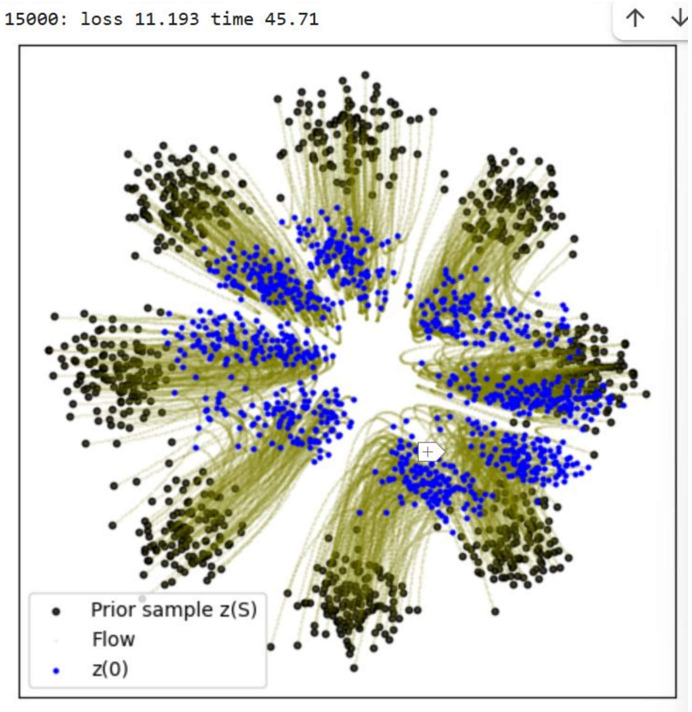


How we define endpoint pairs for flow models

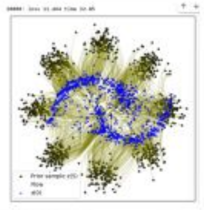
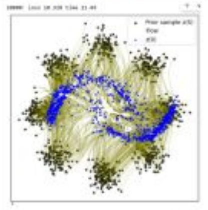
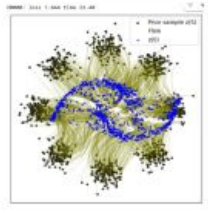
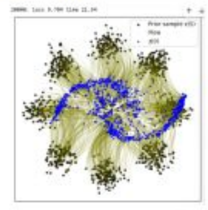
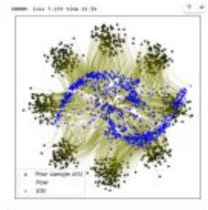
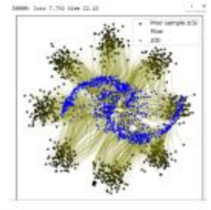
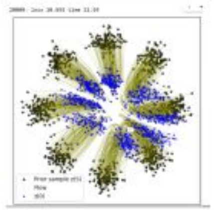
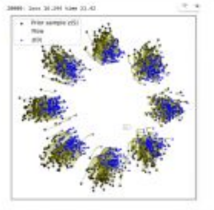
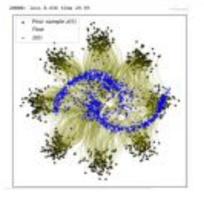
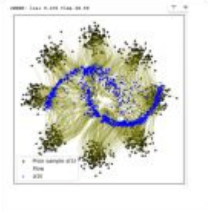
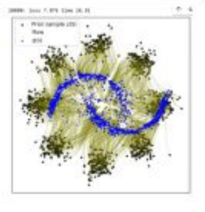
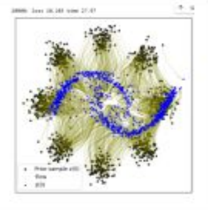
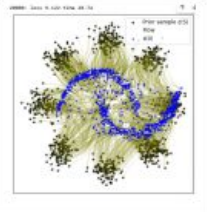
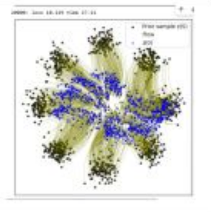
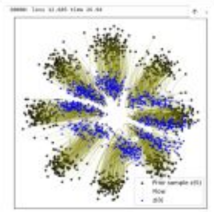
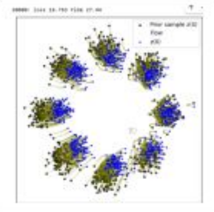
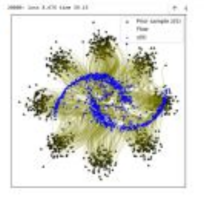
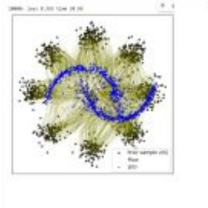
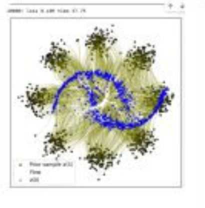
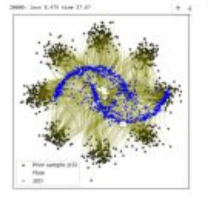
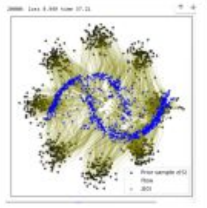
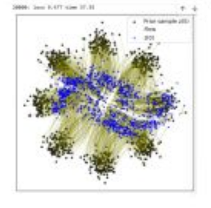
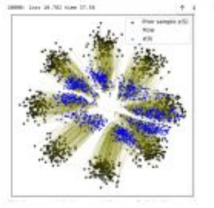
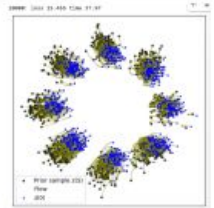
Components of Flow Models: Sigma

Overfitting vs. Underfitting

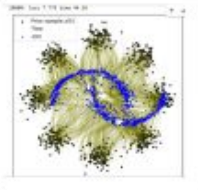
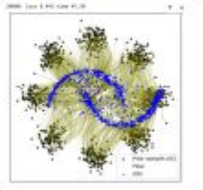
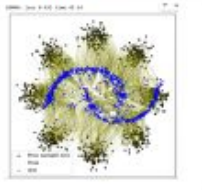
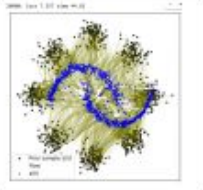
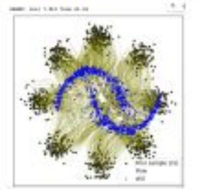
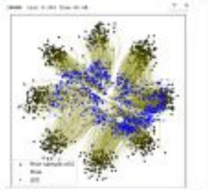
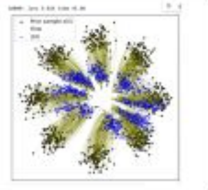
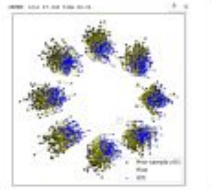
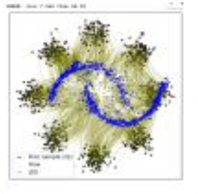
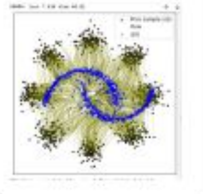
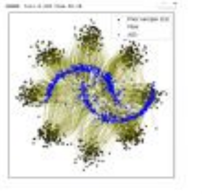
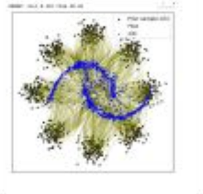
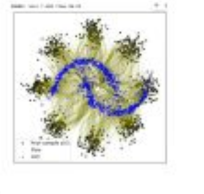
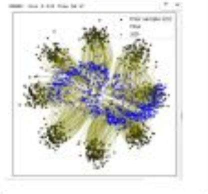
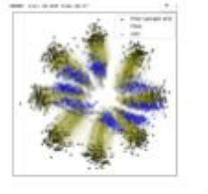
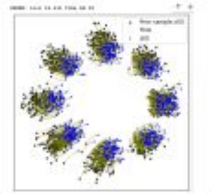
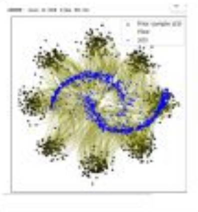
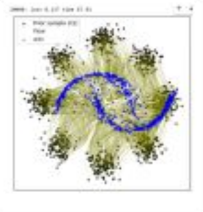
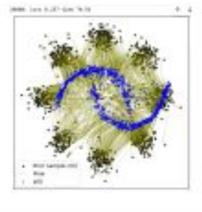
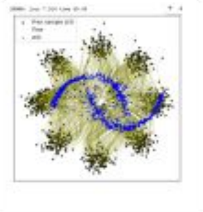
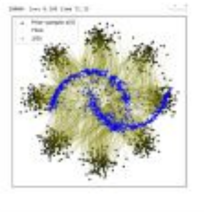
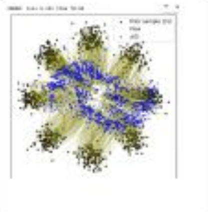
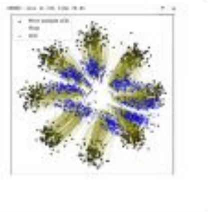
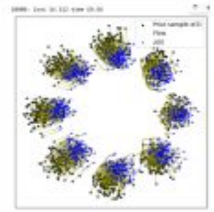




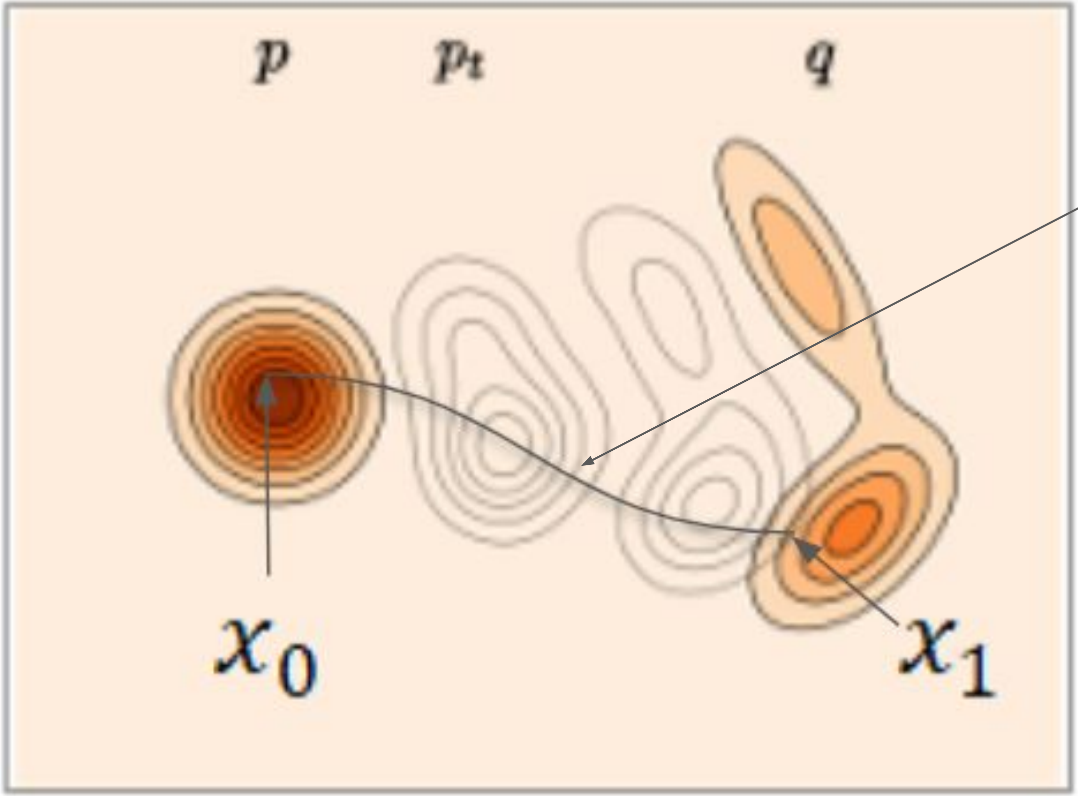
Exploration of Sigma

	σ = 0	σ = 0.0001	σ = 0.001	σ = 0.01	σ = 0.1	σ = 0.5	σ = 1	σ = 5
BS = 50								
BS = 100								
BS = 200								

Exploration of Sigma

	σ = 0	σ = 0.0001	σ = 0.001	σ = 0.01	σ = 0.1	σ = 0.5	σ = 1	σ = 5
BS = 256								
BS = 300								
BS = 500								

Components of Flow Models: Paths

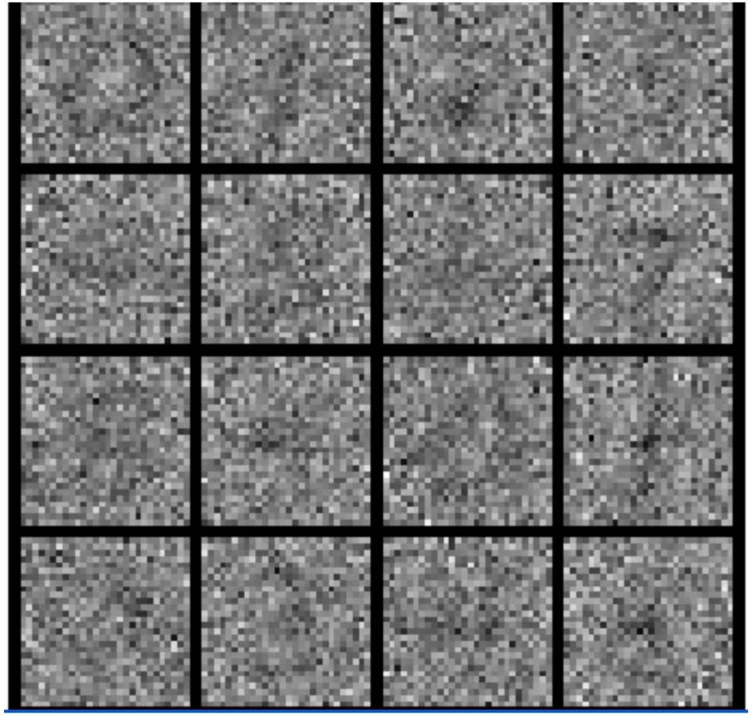


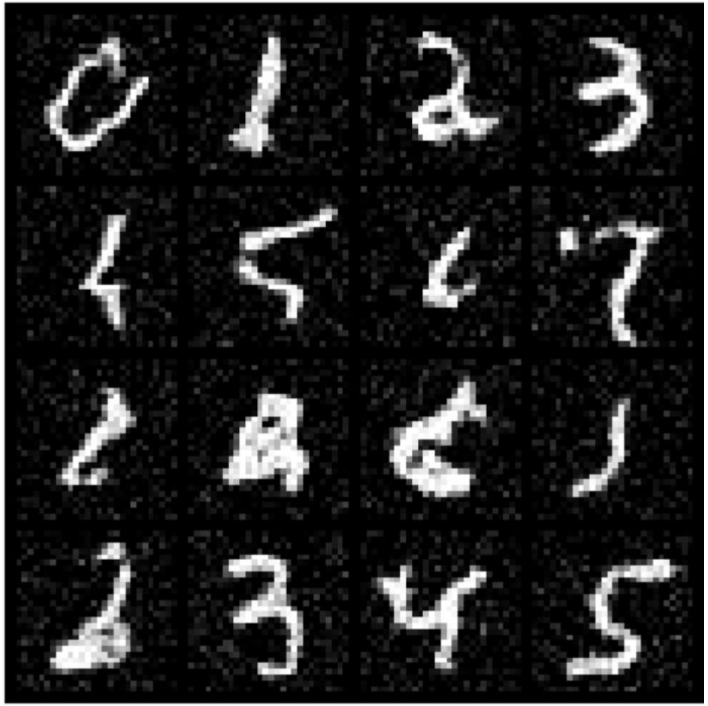
$$x_t = (1-t)x_0 + tx_1$$

Picture (Lipman et al., 2024, p.6)

Sample Image Generation

MLP vs. UNET Models





Hardest Part & What Made It Work?

```python
# Define device: Use GPU if available, else CPU
use_cuda = torch.cuda.is_available() # Check if CUDA-capable GPU is available
device = torch.device("cuda" if use_cuda else "cpu")

# Training Parameters:
batch_size = 32 # Images per batch
num_epochs = 10 # Number of times to loop through full training dataset

# Load MNIST Dataset:
trainset = datasets.MNIST(
    "../data", # Folder where dataset is saved/downloaded
    train=True, # Loads training set
    download=True,
    transform=transforms.Compose([ # Converts PIL images to tensors and normalizes pixel values to [-1, 1]
        transforms.ToTensor(), # Convert PIL Image to PyTorch tensor
        transforms.Normalize((0.5,), (0.5,)) # Normalize grayscale channel to mean=0.5, std=0.5
        ]),
    )

# Create a DataLoader to handle batching and shuffling
train_loader = torch.utils.data.DataLoader(
    trainset,
    batch_size=batch_size,
```

```python
# 3. CFM Class:

sigma = 0.1 # Noise scale
# Utility to convert tensors to images
to_pil = ToPILImage()

# Create Raw UNet Model:
raw_unet = UNetModel(
    dim=(1,28,28), # (channels, height, width) of input images
    num_channels=32, # Base number of feature maps
    num_res_blocks=1, # Number of residual blocks per level
    num_classes=10, # Number of conditional classes (10 for MNIST)
    class_cond=True # Whether model is conditioned on labels
)

# Wrap UNet in a nn.Module
class UNetWrapper(nn.Module):
  def __init__(self, net):
    super().__init__()
    self.net = net
  def forward(self, t, x, y=None): # Forward Pass for NeuralODE/CFM
  # t = current time step, x = imput image tensor, shape (batch, channels, H, W),  y = optional cond
  # Make sure input has channel dim
    if x.ndim == 3:
```

```python
# 4. Training Loop:
num_epochs = 10
num_ode_steps = 5  # Number of time steps per trajectory

# Main Training Loop:
for epoch in range(num_epochs):
  for i, (x1, y) in enumerate(train_loader): # Loop over batches from DataLoader
    optimizer.zero_grad() # Reset gradients

    # Move to device and ensure 4D shape
    x1 = x1.to(device) # Target image batch
    y = y.to(device) # Conditional labels
    if x1.ndim == 3:  # MNIST: (batch, H, W) -> (batch, 1, H, W)
        x1 = x1.unsqueeze(1)

    # Sample initial noise
    x0 = torch.randn_like(x1) # Same shape as x1

    # Sample discrete times for integration
    times = torch.linspace(0., 1., steps=num_ode_steps).to(device)

    # Initialize state with noise
    xt = x0
```

```python
    for t_j in times:
        t_batch = t_j.expand(x1.size(0))  # vectorized over batch

        # Sample target velocity from CFM
        _, xt_sample, ut_sample = FM.sample_location_and_conditional_flow(xt, x1)

        # Ensure 4D tensors
        xt_sample = xt_sample.unsqueeze(1) if xt_sample.ndim == 3 else xt_sample
        ut_sample = ut_sample.unsqueeze(1) if ut_sample.ndim == 3 else ut_sample

        # Mixed precision forward
        with autocast():
            vt = model(t_batch, xt_sample, y)
            step_loss = ((vt - ut_sample)**2).mean()
        traj_loss += step_loss

        # Euler step to update trajectory
        xt = xt + vt * dt

    traj_loss /= num_ode_steps
```

Questions?

References

Jaiswal, S. (2025, April 5). *Multilayer Perceptrons in Machine Learning: A Comprehensive Guide*. DataCamp. https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning - MLP Basics

Lipman, Y., Havasi, M., Holderrieth, P., Shaul, N., Le, M., Karrer, B., Chen, R. T. Q., Lopez-Paz, D., Ben-Hamu, H., & Gat, I. (2024). *Flow Matching Guide and Code*. FAIR at Meta, MIT CSAIL, Weizmann Institute of Science. https://arxiv.org/abs/2412.06264

Tong, A., Malkin, N., Huguet, G., Zhang, Y., Rector-Brooks, J., Fatras, K., Bengio, Y. (2023). *conditional-flow-matching: TorchCFM — Conditional Flow Matching library*. GitHub repository. https://github.com/atong01/conditional-flow-matching - For MLP Model

Tong, A. Y. (2023). *Flow_matching_tutorial.ipynb* [Jupyter Notebook]. GitHub. Retrieved December 1, 2025, from https://colab.research.google.com/github/atong01/conditional-flow-matching/blob/main/examples/2D_tutorials/Flow_matching_tutorial.ipynb - Main Examples

[JamesCFM.ipynb - Colab]