ELUCIDATING THE DESIGN SPACE OF DIFFUSION-BASED GENERATIVE MODELS

TEAM MEMBERS:

SRIRAM DEGALA

KATHERINE SHAGALOV

INTRODUCTION

INTRODUCTION TO DIFFUSION MODELS

- Emergence of Diffusion Models:
  - A powerful framework for generating images.
  - Supports **unconditional** (no input) and **conditional** (input-based, e.g., text or labels) image generation.
  - Outperforms **GANs** (Generative Adversarial Networks) in certain scenarios.
- Applications Beyond Images:
  - Domains include:
    - Audio generation
    - Video generation
    - Image segmentation
    - Language translation

CHALLENGES AND CONTRIBUTIONS

- Challenges:
  - Dense theoretical literature focused on:
    - Sampling schedules
    - Training dynamics
    - Noise parameterization
  - Limited flexibility due to tightly coupled systems.
- Contributions:
  - Practical Focus:
    - Emphasis on tangible components and algorithms.
    - Explores interactions within the system for flexibility.
  - Sampling Process Improvements:
    - Optimized time discretization and higher-order methods (e.g., Runge-Kutta).
    - Reduced sampling steps for faster synthesis.
  - Training Dynamics:
    - Preconditioning inputs/outputs.
    - Improved noise distribution and augmentation techniques.

RESULTS

- Key Outcomes:
  - Significant improvements in image quality with record-breaking FID scores:
    - Achieved on datasets like CIFAR-10 with 1.79 and ImageNet (64×64 resolution) with 2.07 for class conditional.
  - Faster and more efficient sampling process.
- Encouraging Innovation:
  - Practical design space detailed for further exploration.
  - Open-source implementation and pre-trained models are available on GitHub.
- Vision:
  - Inspire targeted innovations in diffusion-based generative models.

EXPRESSING DIFFUSION MODELS IN A COMMON FRAMEWORK

- **Subtitle**: A Unified and Practical Perspective

- **Key Topics**: Data Distribution, Diffusion Process, ODE, and Modular Design

DATA DISTRIBUTION AND NOISE MODELING

- Key Concepts:
  - Data distribution: $p_{data}(x)$, standard deviation σ_{data}.

- Mollified distributions:
  - Adding Gaussian noise (σ) to data.
  - With noise levels ($\sigma_0 = \sigma_{\{max\}} > \sigma_1 > \ \ldots\ldots\ldots\ldots > \sigma_N$): Resembles pure Gaussian noise.

DIFFUSION PROCESS

- Process Overview:
  - Starts with a noisy image.
  - Iteratively removes noise to generate clean data.

- Intermediate Steps:
  - At each noise level σ_i, samples $x_i \sim p(x; \sigma)$.
  - Endpoint matches the original data distribution.

PROBABILITY FLOW ODE AND DENOISING SCORE MATCHING

- Probability Flow Ordinary Differential Equation (ODE):

$$dx = -\dot{\sigma}(t)\sigma(t)\nabla_x \log p\big(\boldsymbol{x}, \sigma(t)\big)\, dt.$$

  - Describes noise evolution over time.
  - Uses the score function $\nabla_x \log p(\boldsymbol{x}, \sigma)$ to guide denoising.

- Denoising Score Matching: $\nabla_x \log p(\boldsymbol{x}, \sigma) = \frac{D(x;\sigma) - x}{\sigma^2}$.
  - $D(x;\sigma)$: Neural Network predicts denoised outputs.

TRAJECTORY CURVATURE AND SAMPLING

- Trajectory Curvature:
  - Introduce scaling functions $s(t)$ for stability and accuracy:

$$dx = [\frac{s\cdot(t)}{s(t)}x \ - \frac{s(t)^2 \ \sigma\cdot(t)}{\sigma(t)}\nabla_x \log p(\frac{x}{s(t)}; \sigma(t))]\, dt\ .$$

- Discretization for Sampling:
  - Convert continuous ODE into finite steps.
  - Integration schemes (Euler, Runge-Kutta) impact efficiency and quality.

IMPROVEMENTS TO DETERMINISTIC SAMPLING IN DIFFUSION MODELS

- Subtitle: Enhancing Quality and Reducing Computational Costs

BACKGROUND AND MOTIVATION

- What is Deterministic Sampling?
  - A method for generating data by solving an Ordinary Differential Equation (ODE).
  - Produces consistent outputs without randomness.
- Challenges:
  - Computationally expensive due to high Number of Function Evaluations (NFE).
  - Quality of generated data (e.g., images) depends on accurate sampling.
- Goals:
  - Reduce NFE for faster sampling.
  - Maintain or improve image quality (e.g., lower Fréchet Inception Distance (FID)).

UNIFIED FRAMEWORK AND EVALUATION

- Unified Framework:
  - Standardizes formulas across different diffusion models:
    - VP (Variance Preserving), VE (Variance Exploding), DDIM (Denoising Diffusion Implicit Models).
- Evaluation Process:
  - Test improvements using pre-trained models: VP, VE, and ADM (ImageNet).
  - Metrics:
    - **Frechet Inception Distance (FID)**: Measures image quality.
    - **Number fo Function Evaluations (NFE)**: Evaluates computational cost.

KEY IMPROVEMENTS

1. Higher-Order Integrators:
    1. Replace Euler's method with Heun's 2nd-order method for better accuracy.
    2. Reduces truncation errors with only a slight increase in computation.
2. Noise Schedule Optimization:
    1. Noise steps σ are refined for smaller truncation errors.
    2. Focuses on shortening steps near σ_{min} where errors impact quality the most.
3. Trajectory Curvature Refinement:
    1. Simplifies ODE trajectories by setting $\sigma(t) = t$ and $s(t) = 1$.
    2. Reduces curvature and ensures linear trajectories, improving stability.

PSEUDO CODE: HEUN'S 2ND ORDER METHOD

Algorithm 1 Deterministic sampling using Heun's 2nd order method with arbitrary $\sigma(t)$ and $s(t)$.

procedure HEUNSAMPLER($D_\theta(\boldsymbol{x};\sigma)$, $\sigma(t)$, $s(t)$, $t_{i\in\{0,\dots,N\}}$)

  sample $\boldsymbol{x}_0 \sim \mathcal{N}\big(\mathbf{0},\ \sigma^2(t_0)\, s^2(t_0)\, \mathbf{I}\big)$ ▷ Generate initial sample at t_0

  for $i \in \{0,\dots,N-1\}$ **do** ▷ Solve Eq. 4 over N time steps

    $\boldsymbol{d}_i \leftarrow \left(\frac{\dot\sigma(t_i)}{\sigma(t_i)} + \frac{\dot s(t_i)}{s(t_i)}\right)\boldsymbol{x}_i - \frac{\dot\sigma(t_i)s(t_i)}{\sigma(t_i)} D_\theta\left(\frac{\boldsymbol{x}_i}{s(t_i)};\sigma(t_i)\right)$ ▷ Evaluate $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ at t_i

    $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + (t_{i+1}-t_i)\boldsymbol{d}_i$ ▷ Take Euler step from t_i to t_{i+1}

    if $\sigma(t_{i+1}) \neq 0$ **then** ▷ Apply 2nd order correction unless σ goes to zero

      $\boldsymbol{d}'_i \leftarrow \left(\frac{\dot\sigma(t_{i+1})}{\sigma(t_{i+1})} + \frac{\dot s(t_{i+1})}{s(t_{i+1})}\right)\boldsymbol{x}_{i+1} - \frac{\dot\sigma(t_{i+1})s(t_{i+1})}{\sigma(t_{i+1})} D_\theta\left(\frac{\boldsymbol{x}_{i+1}}{s(t_{i+1})};\sigma(t_{i+1})\right)$ ▷ Eval. $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ at t_{i+1}

      $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + (t_{i+1}-t_i)\left(\frac{1}{2}\boldsymbol{d}_i + \frac{1}{2}\boldsymbol{d}'_i\right)$ ▷ Explicit trapezoidal rule at t_{i+1}

  return $\boldsymbol{x}_N$ ▷ Return noise-free sample at t_N

TRAJECTORY CURVATURE AND NOISE SCHEDULE

- **Trajectory curvature** in the solution of the ODE used for deterministic sampling in diffusion models is influenced by the choice of the **noise schedule** $\sigma(t)$ and the **scaling function** $s(t)$.
- These functions influence the curvature of the ODE solution trajectory $\frac{dx}{dt}$, which in turn affects the accuracy and efficiency of sampling.
- **Optimal Choices:** As author argues that setting $\sigma(t) = t \; and \; s(t) = 1$

$$\frac{dx}{dt} = x - \frac{D(x;t)}{t}$$

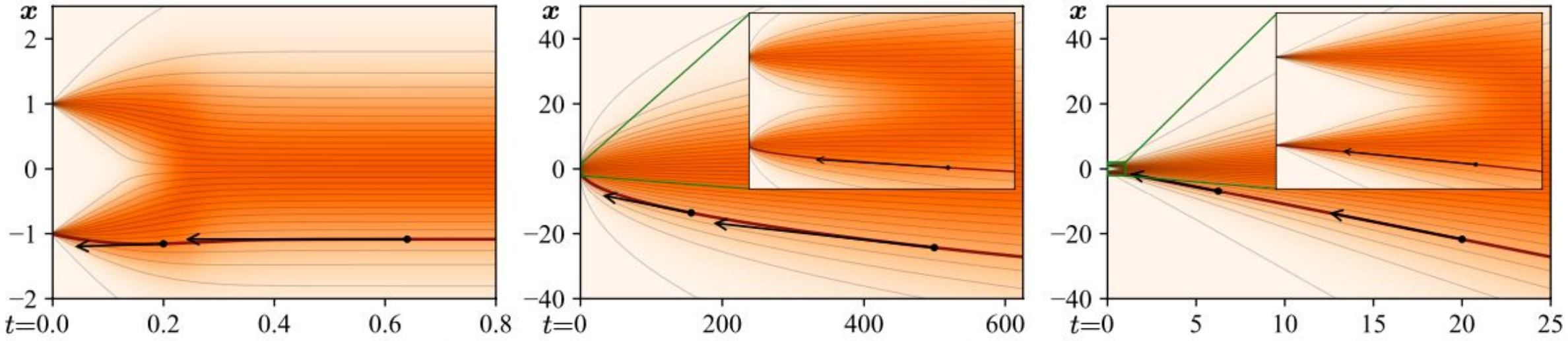
where x: data sample

$D(x; t)$: Denoised estimate of x at time t.

IMPACT ON TRAJECTORIES

- Linear Behavior:
  - At large and small noise levels ($\sigma(t) \to \infty \; and \; \sigma(t) \to 0$), the trajectories become approximately linear.
  - Nonlinear behavior (high curvature) is limited to a narrow range of intermediate noise levels.

At any point $x \; and \; t$, a single Euler step to $t = 0$ directly yields the denoised image $D_\theta(x; t)$.



(a) Variance preserving ODE [49] (b) Variance exploding ODE [49] (c) DDIM [47] / Our ODE

A 1D sketch of ODE curvature shows:

(a) Variance Preserving (VP) ODE trajectories flatten at large σ,

(b) Variance Exploding (VE) ODE has extreme curvature near data,

(c) DDIM schedules produce near-linear trajectories pointing towards the mean of data and the data manifold as $\sigma \to 0$.

STOCHASTIC SAMPLING IN DIFFUSION MODELS

- Subtitle: Exploring Diversity in Generative Modeling

WHAT IS STOCHASTIC SAMPLING?

- Definition:
  - **Stochastic sampling** refers to a process in diffusion-based generative models where randomness is introduced during the sampling process to generate diverse outputs. This approach leverages **Stochastic Differential Equations (SDEs)** to model how noise is gradually removed from an initial noisy input, transitioning it into an image.
- Process:
  - Starts with noisy data (Gaussian noise).
  - Gradually refines the noise into meaningful data, guided by the score function.

THE SDE EQUATION

- $dx_{\pm} = -\sigma^{\cdot}(t)\sigma(t)\nabla_x log\ p\big(x;\sigma(t)\big)dt \pm \beta(t)\sigma(t)^2\ \nabla_x log\ p\big(x;\sigma(t)\big)dt + \sqrt{2\beta(t)}\sigma(t)d\omega_t$
- Components of the SDE:
- Drift Terms:

i. $-\sigma^{\cdot}(t)\sigma(t)\nabla_x log\ p\big(x;\sigma(t)\big)dt$

  - **Deterministic Term**: Guides the sample x toward regions of higher data density by following the gradient of the log-probability (score function).
  - **Role**: Ensures the sample transitions from noisy states toward the clean data distribution.
  - $\sigma^{\cdot}(t)$: derivative of time w.r.t time.
  - $\nabla_x log\ p\big(x;\sigma(t)\big)$: **score function**, which points toward the denser regions of the data distribution.

ii. $\pm\beta(t)\sigma(t)^2\,\nabla_x log\; p\big(x;\sigma(t)\big)$

- $\beta(t)$: function adjusts the deterministic drift.
- Role: Modulates the effect of the score function on the trajectory, influencing how quickly or slowly the sample evolves.
- $\beta(t) = \frac{\sigma\cdot(t)}{\sigma(t)}$

- Diffusion Term:
  $\sqrt{(2\beta(t))}\sigma(t)d\omega_t$
- Stochastic Term: Introduces randomness into the trajectory via the Wiener process $d\omega_t$.
- $d\omega_t$: A Wiener process (random Brownian motion), introducing stochasticity into the trajectory..
- Role: Adds variability to the sampling process, ensuring diversity in the generated samples by allowing exploration of different paths on the data manifold.

INTERPRETATION OF THE SDE

- The **deterministic terms** (first two terms) ensure th at sample x moves toward the data manifold in a structured manner.
- The **stochastic term** (last term) introduces noise at each step, allowing for the exploration of diverse trajectories.
- The $\pm$ introduces flexibility, where the specific sign and formulation may depend on the specific SDE variant or the desired properties of the sampling process

ROLE IN STOCHASTIC SAMPLING

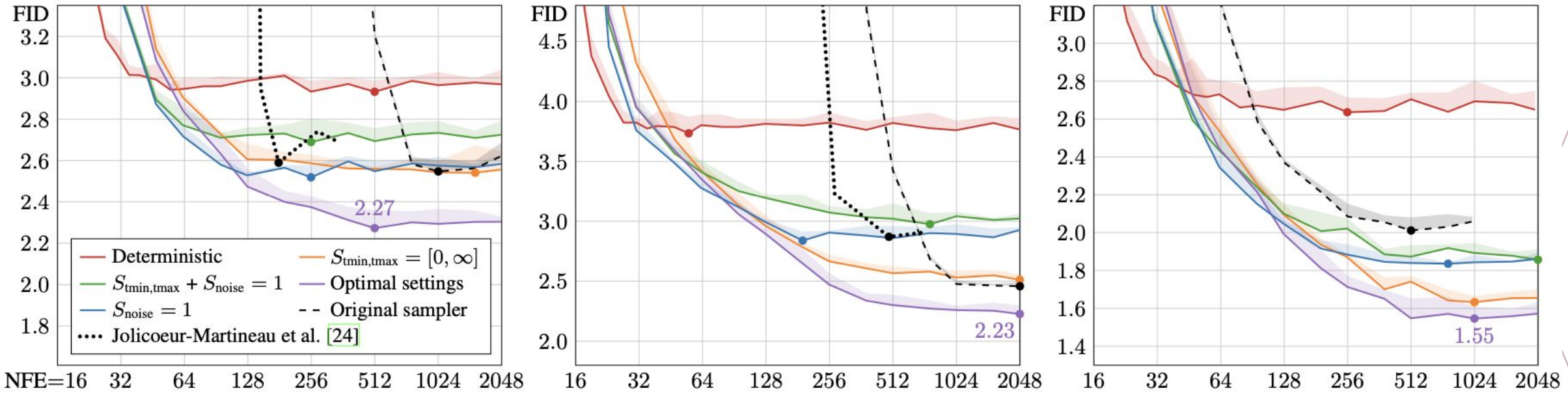
- Forward Process:
  - In the forward diffusion process, noise is progressively added to the data, leading to a distribution close to Gaussian noise.
- Reverse Process:
  - In the reverse sampling process, noise is removed step-by-step using the SDE, guided by the score function.

PSEUDO CODE: STOCHASTIC SAMPLING

Algorithm 2 Our stochastic sampler with $\sigma(t) = t$ and $s(t) = 1$.

procedure StochasticSampler($D_\theta(\boldsymbol{x};\sigma)$, $t_{i\in\{0,\dots,N\}}$, $\gamma_{i\in\{0,\dots,N-1\}}$, S_{noise})
  sample $\boldsymbol{x}_0 \sim \mathcal{N}\big(\mathbf{0},\ t_0^2\,\mathbf{I}\big)$
  for $i \in \{0,\dots,N-1\}$ **do**
    sample $\boldsymbol{\epsilon}_i \sim \mathcal{N}\big(\mathbf{0},\ S_{\text{noise}}^2\,\mathbf{I}\big)$
    $\hat{t}_i \leftarrow t_i + \gamma_i t_i$ ▷ Select temporarily increased noise level $\hat{t}_i$
    $\hat{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i + \sqrt{\hat{t}_i^2 - t_i^2}\,\boldsymbol{\epsilon}_i$ ▷ Add new noise to move from t_i to $\hat{t}_i$
    $\boldsymbol{d}_i \leftarrow \big(\hat{\boldsymbol{x}}_i - D_\theta(\hat{\boldsymbol{x}}_i;\hat{t}_i)\big)/\hat{t}_i$ ▷ Evaluate $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ at $\hat{t}_i$
    $\boldsymbol{x}_{i+1} \leftarrow \hat{\boldsymbol{x}}_i + (t_{i+1} - \hat{t}_i)\boldsymbol{d}_i$ ▷ Take Euler step from $\hat{t}_i$ to t_{i+1}
    if $t_{i+1} \neq 0$ **then**
      $\boldsymbol{d}'_i \leftarrow \big(\boldsymbol{x}_{i+1} - D_\theta(\boldsymbol{x}_{i+1};t_{i+1})\big)/t_{i+1}$ ▷ Apply 2nd order correction
      $\boldsymbol{x}_{i+1} \leftarrow \hat{\boldsymbol{x}}_i + (t_{i+1} - \hat{t}_i)\big(\frac{1}{2}\boldsymbol{d}_i + \frac{1}{2}\boldsymbol{d}'_i\big)$
  return $\boldsymbol{x}_N$

▷ $\gamma_i = \begin{cases} \min\left(\frac{S_{\text{churn}}}{N}, \sqrt{2}-1\right) & \text{if } t_i \in [S_{\text{tmin}}, S_{\text{tmax}}] \\ 0 & \text{otherwise} \end{cases}$



(a) Uncond. CIFAR-10, VP (b) Uncond. CIFAR-10, VE (c) Class-cond. ImageNet-64

Evaluate the stochastic sampler Algorithm by comparing its performance with different configurations of $S_{churn}, S_{tmin}, S_{tmax}, S_{noise}$. Also highlighting the lowest observed FID with dots.

S_{churn} (Sampling Churn): How much randomness is introduced overall.

S_{tmin} (Minimum Time Thresold): Control the range of noise application across time steps.

S_{tmax} (Maximum Time Thresold): Control the range of noise application across time steps.

S_{noise} (Noise Strength): Directly controls the strength of noise added at each step.

Hyperparameters used in stochastic samplers for diffusion models.

STOCHASTIC SAMPLING VS DETERMISTIC SAMPLING

Feature	Stochastic Sampling	Deterministic Sampling
Diversity	Diverse outputs due to randomness.	Consistent outputs with no randomness.
Process	Solves an SDE with stochastic noise.	Solves an ODE without noise.
Cost	Higher due to randomness evaluations	Lower computational cost.
Use Case	Best for exploring diverse outputs.	Suitable for scenarios needing consistency.

COMPARISONS OF TRAJECTORY SHAPES

- The text refers to a figure illustrating ODE trajectories for different models:
  - Variance Preserving (VP):
    - Trajectories flatten at large σ, with gradients pointing toward data only at small σ.
  - Variance Exploding (VE):
    - Extreme curvature throughout, especially near the data manifold.
  - DDIM/Proposed Schedule:
    - Linear trajectories at large and small σ, with high curvature only in a narrow intermediate range.
    - These trajectories are more efficient and align better with the denoiser output.

RESULTS AND PERFORMANCE GAINS

- Key Findings:
  - Heun's method achieves comparable FID with fewer NFEs.
  - Noise schedule $\sigma(t) = t$ performs significantly better for VP and VE models.
  - Improvements reduce NFE by:
    - **7.3×** for VP.
    - **300×** for VE.
    - **3.2×** for DDIM.
- Practical Efficiency:
  - Generates **26.3 CIFAR-10 images per second** on NVIDIA V100 GPU.

CONCLUSION

- The authors optimize the noise schedule ($\sigma(t) = t$) and scaling function ($s(t) = 1$) to reduce trajectory curvature and truncation errors.
- These choices significantly reduce computational costs (NFE) while maintaining or improving image quality (FID).
- The approach generalizes across various diffusion models, highlighting its practicality and robustness.

PRECONDITIONING AND TRAINING IN DIFFUSION MODELS

- **Subtitle**: Enhancing Stability, Efficiency, and Quality

PURPOSE OF PRECONDITIONING AND TRAINING

- Stabilize Training: Prevent instability due to large gradients or varying magnitudes across noise levels.
- Improve Convergence: Ensure the network learns effectively across all noise levels.
- Optimize Sampling: Make the reverse sampling process more efficient by conditioning the network outputs appropriately.

COMPONENTS OF PRECONDITIONING

- Input Preconditioning:
  - The input x to the model (a combination of clean signal y and noise n) is scaled to ensure consistency across noise levels.
  - $x = y + n$
  - $c_{in}(\sigma) = \frac{1}{\sqrt{\sigma^2 + \sigma_{data}^2}}$
  - σ: $standard\ deviation\ for\ noise$
  - σ_{data}: $standard\ deviation\ of\ input\ data$
- Output Preconditioning:
  - The model's output is scaled to reduce sensitivity to noise levels.
  - $c_{out}(\sigma) = \sigma . \frac{\sigma_{data}}{\sqrt{\sigma_{data}^2 + \sigma^2}}$

- Skip Connections:
- A skip connection is added to propagate the input signal directly to the output.
  - $D_\theta(x, \sigma) = c_{skip}(\sigma).x + c_{out}(\sigma).F_\theta\big(c_{in}(\sigma).x, c_{noise}(\sigma)\big).$
  - $c_{skip}(\sigma)$: $Weighting\ factor\ for\ skip\ connection.$
  - $c_{noise}(\sigma)$: $Embedding\ of\ the\ noise\ level\ \sigma.$

TRAINING PROCESS

- **Training Objective**: The network is trained to minimize the squared error between the denoised output and the actual data:

$$L = E_{\sigma, y, n}\left[\lambda(\sigma). \left|\left|D_\theta(y + n; \sigma) - y\right|\right|_2^2\right]$$

- **Key Components**:
- **Noise Distribution ($\boldsymbol{\sigma}$)**: Noise levels are sampled from a **custom distribution** to prioritize noise levels where the network has the most impact.
- **Loss Weighting ($\lambda(\sigma)$)**: Adjusts the importance of different noise levels during training.

$$\lambda(\sigma) = \frac{\sigma_{data}}{\sigma^2 + \sigma_{data}^2}$$

ADVANTAGES OF PRECONDITIONING

- Improved Stability
  - Preconditioning ensures that inputs and outputs remain within a manageable range across all noise levels, avoiding exploding or vanishing gradients.
- Efficient Training
  - By focusing on noise levels where the model has the most impact, training becomes more effective and efficient.
- Robust Sampling
  - Preconditioned outputs ensure smoother transitions during the reverse sampling process, reducing errors and improving sample quality.

PRACTICAL IMPLEMENTATION

- **Custom Noise Schedules**: Train the model to prioritize noise levels that contribute most to the denoising process.
- **Preconditioning Factors**: Tune $c_{in}(\sigma)$, $c_{out}(\sigma)$, $c_{skip}(\sigma)$ for the dataset and task.

RESULTS

- Higher Quality Outputs: Lower Fréchet Inception Distance (FID) scores on benchmark datasets.
- Faster Sampling: The improved training framework reduces the number of function evaluations (NFE) required for high-quality results

CODING-RESULTS

- Key Outcomes:
  - Significant improvements in image quality with record-breaking FID scores:
    - Achieved on datasets like CIFAR-10 with 1.79 and ImageNet (64×64 resolution) with 2.07 for class conditional.
  - Faster and more efficient sampling process.
- Encouraging Innovation:
  - Practical design space detailed for further exploration.
  - Open-source implementation and pre-trained models are available on GitHub.
- Vision:
  - Inspire targeted innovations in diffusion-based generative models.

CODE EXECUTION

- Ran in Google Colab to utilize NVIDIA GPUs (T4 free, A100 paid)
- Cloned repository and ran example.py file
- For each of the 4 Datasets provided, load the best pretrained model and generate an 8x8 image grid of random images

generate.py flags for improvements:

- -- steps Number of sampling steps to take
  - Optimal is 18 steps for simpler datasets, up to 256 for complex datasets
  - Goal is to show low FID score with few sampling steps
- -- solver Which solver to use (Heun or Euler)
  - Use proposed Heun's 2nd order method for sampling

DATASETS USED

CIFAR-10: Dataset of 60k images of 10 labeled classes with 6k images of each class: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Used for conditional and unconditional models.

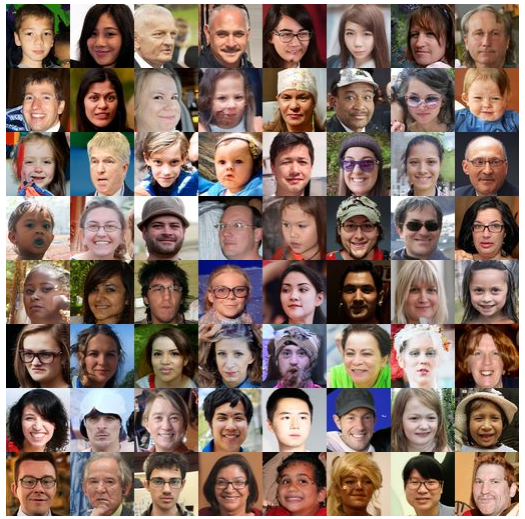
AFHQv2: Animal Faces High Quality Dataset. Consists of 15k images of animal faces of three domains: cat, dog, wildlife. Used for unconditional models.

FFHQ: Flickr Faces High Quality Dataset. 70k images of human faces, taken from Flickr website. Images have faces of varying race, age, and with/without accessories such as glasses and hats. Data is unlabeled, used for unconditional models.

ImageNet64: Downsampled version of full ImageNet dataset. Dataset of 150k images of images labeled with the presence or absence of 1000 different items representing nouns from WordNet. Used for conditional and unconditional models.

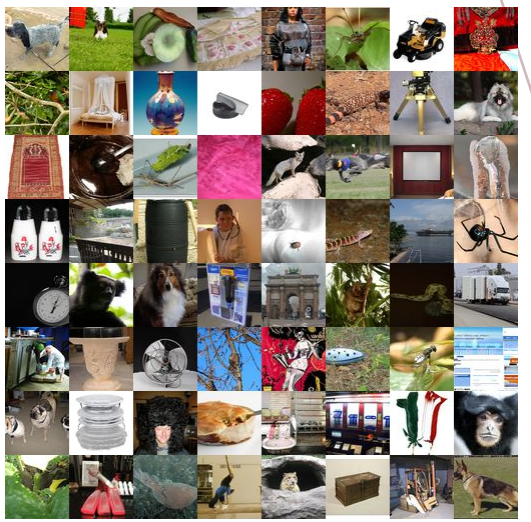
GENERATED IMAGES FROM PRE-TRAINED MODELS

CIFAR-10



FFHQ

AFHQv2



ImageNet64

FRÉCHET INCEPTION DISTANCE (FID)

- Metric used to measure image quality of images created by generative models
- Takes mean and covariance statistics of a large set of generated images and calculates the difference between the statistics of the training data or real images
- Lower FID = Higher image realism quality
- This code calculates statistics from 50,000 generated images. Researchers recommend calculating FID three times with different random seed flags and taking the lowest
  - Challenge: Long run times (3.5 hours per FID calculation before upgrade, 25 min per calculation after upgrade)

PRE-TRAINED MODEL FID RESULTS

CIFAR-10 Conditional: 1.84866

AFHQv2 Unconditional: 2.15764

FFHQ Unconditional: 2.06781

ImageNet Conditional: 2.27311

TESTING PERFORMED ON MNIST DATA

- Created new model on MNIST Dataset
  - Full dataset is 60k images of handwritten digits 0-9, labeled
- Downsampled to 600 images (1%) due to computational power limitations, ~2 hour training time

```
!torchrun --standalone --nproc_per_node=1 edm/train.py --outdir=training-runs \
    --data=mnist_dataset_downsampled --cond=1 --arch=ddpmpp --batch=256 --fp16=True \
    --duration=6 --lr=0.001
```

- `--batch=256` Optimal batch size of 256
- `--fp16=True` Enabled mixed-precision training, gave a significant increase in training speed (from 3s/kimg to 1s/kimg)
- `--duration=6` Train on 6 kimg (10 epochs)
- `--cond=1` Set class-conditional flag to True since classes are important for MNIST dataset

MNIST TEST DATA MODEL RESULTS

Images generated:

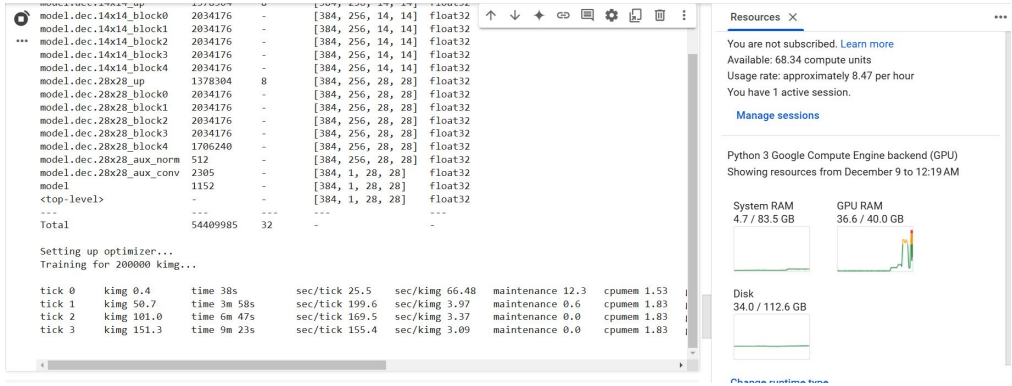
FIDs:

- default steps=18, solver=Heun: 8.34717, 8.32231, **8.29461**
- steps=18, solver=Euler: **10.2189**

CHALLENGES AND LIMITATIONS

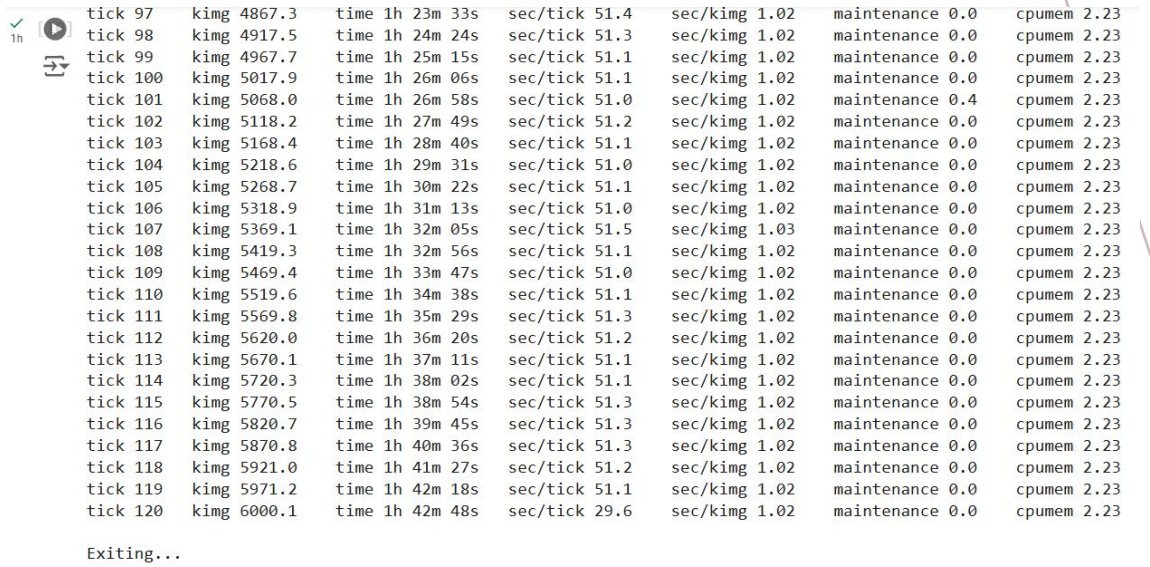
Hardware and computational power

- Had to purchase premium GPUs from Google Cloud, switched to NVIDIA A100
- Still, full dataset with optimized batch size of 256 and 200 kimg (~3 epochs) would have taken almost 7 days to train



NOTE ON MODEL PERFORMANCE TIME

Model	GPUs	Time	Options
cifar10-32x32-cond-vp	8xV100	~2 days	`--cond=1 --arch=ddpmpp`
cifar10-32x32-cond-ve	8xV100	~2 days	`--cond=1 --arch=ncsnpp`
cifar10-32x32-uncond-vp	8xV100	~2 days	`--cond=0 --arch=ddpmpp`
cifar10-32x32-uncond-ve	8xV100	~2 days	`--cond=0 --arch=ncsnpp`
ffhq-64x64-uncond-vp	8xV100	~4 days	`--cond=0 --arch=ddpmpp --batch=256 --cres=1,2,2,2 --lr=2e-4 --dropout=0.05 --augment=0.15`
ffhq-64x64-uncond-ve	8xV100	~4 days	`--cond=0 --arch=ncsnpp --batch=256 --cres=1,2,2,2 --lr=2e-4 --dropout=0.05 --augment=0.15`
afhqv2-64x64-uncond-vp	8xV100	~4 days	`--cond=0 --arch=ddpmpp --batch=256 --cres=1,2,2,2 --lr=2e-4 --dropout=0.25 --augment=0.15`
afhqv2-64x64-uncond-ve	8xV100	~4 days	`--cond=0 --arch=ncsnpp --batch=256 --cres=1,2,2,2 --lr=2e-4 --dropout=0.25 --augment=0.15`
imagenet-64x64-cond-adm	32xA100	~13 days	`--cond=1 --arch=adm --duration=2500 --batch=4096 --lr=1e-4 --ema=50 --dropout=0.10 --augment=0 --fp16=1 --ls=100 --tick=200`



Most of the provided pre-trained models took multiple days to train, with 8 GPUs. The ImageNet model took 13 days to train with 32 A100 GPUs. Since we do not have such resources or time, we had to simplify our own model significantly. With downsampling to 1% and training on only 10 epochs, our model took almost 2 hours to train, which restricted us from training/retraining fully many times with different fine tuning parameters.