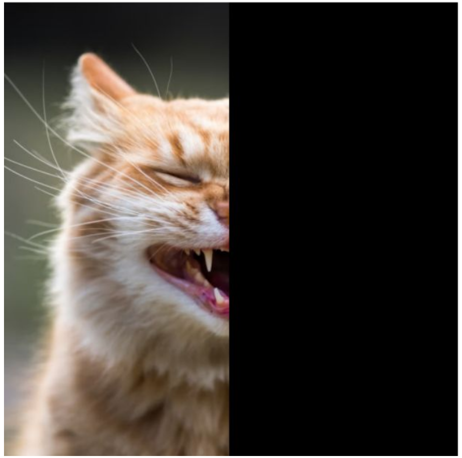
Fast constrained sampling in pre-trained diffusion models

Alexandros Graikos[1], Nebojsa Jojic[2], Dimitris Samaras[1]

[1] Stony Brook University [2] Microsoft Research

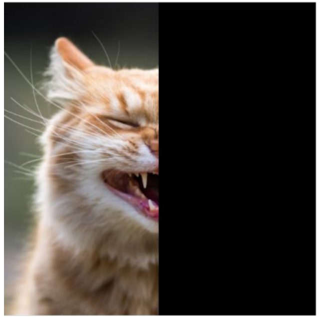
Constrained Sampling

Task:
Using the Stable Diffusion text-to-image model, inpaint the missing half



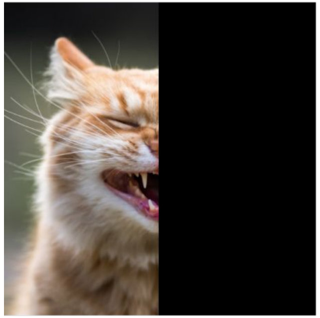
Constrained Sampling

Task:
Using the Stable Diffusion text-to-image model, inpaint the missing half



Constrained Sampling

Task:
Using the Stable Diffusion text-to-image model, inpaint the missing half



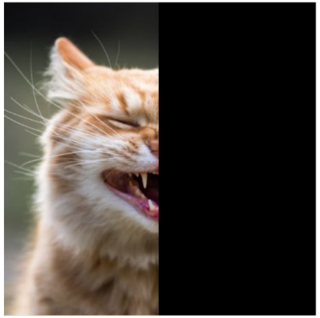
Outline of algorithm for sampling under constraints

Sampling under constraints with a diffusion model

Input: Diffusion model $\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t)$, schedule α_i, β_i, step size s, constraint $C(\boldsymbol{x}_0, \boldsymbol{y})$, learning rate λ, optimization steps K.
$\boldsymbol{x}_T \sim N(\boldsymbol{0}, \boldsymbol{I})$
for $t = T, T-s, T-2s, \ldots, s$ **do**
  // Optimize latent for constraint
  for $i = 1, 2, \ldots, K$ **do**
    // Compute error
    // **non-linear**: gradient descent | **linear**: closed-form
    $\boldsymbol{e} = \nabla_{\hat{\boldsymbol{x}}_0} C(\hat{\boldsymbol{x}}_0, \boldsymbol{y})$ or $\boldsymbol{e} = \text{Solver}_C(\hat{\boldsymbol{x}}_0, \boldsymbol{y})$
    // Compute update direction and perform step
    $\boldsymbol{e}_t = \mathcal{F}(\boldsymbol{J}, \boldsymbol{e})$
    $\boldsymbol{x}_t = \boldsymbol{x}_t - \lambda \boldsymbol{e}_t$
  end for
  // Run diffusion step
  $\boldsymbol{z}_t \sim N(\boldsymbol{0}, \boldsymbol{I})$
  $\boldsymbol{\epsilon}_t = \frac{1}{\sqrt{1-\alpha_{t-s}}}\boldsymbol{x}_t - \frac{\sqrt{\alpha_{t-s}}}{\sqrt{1-\alpha_{t-s}}}\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t)$
  $\boldsymbol{x}_{t-s} = \sqrt{\alpha_{t-s}}\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t) + \sqrt{1-\alpha_{t-s}-\beta_{t-s}^2}\boldsymbol{\epsilon}_t + \beta_{t-s}\boldsymbol{z}_t$
end for
Return: $\boldsymbol{x}_0$

Constrained Sampling

Task:
Using the Stable Diffusion text-to-image model, inpaint the missing half



Outline of algorithm for sampling under constraints

Solution:

1 Given a diffusion model $\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t)$ we compute how well the current prediction fits the constraint

Sampling under constraints with a diffusion model

Input: Diffusion model $\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t)$, schedule α_i, β_i, step size s, constraint $C(\boldsymbol{x}_0, \boldsymbol{y})$, learning rate λ, optimization steps K.

$\boldsymbol{x}_T \sim N(\mathbf{0}, \boldsymbol{I})$

for $t = T, T-s, T-2s, \ldots, s$ **do**

  // Optimize latent for constraint

  for $i = 1, 2, \ldots, K$ **do**

    // Compute error

    // **non-linear**: gradient descent | **linear**: closed-form

    $\boldsymbol{e} = \nabla_{\hat{\boldsymbol{x}}_0} C(\hat{\boldsymbol{x}}_0, \boldsymbol{y})$ or $\boldsymbol{e} = \text{Solver}_C(\hat{\boldsymbol{x}}_0, \boldsymbol{y})$

    // Compute update direction and perform step

    $\boldsymbol{e}_t = \mathcal{F}(\boldsymbol{J}, \boldsymbol{e})$

    $\boldsymbol{x}_t = \boldsymbol{x}_t - \lambda \boldsymbol{e}_t$

  end for

  // Run diffusion step

  $\boldsymbol{z}_t \sim N(\mathbf{0}, \boldsymbol{I})$

  $\boldsymbol{\epsilon}_t = \frac{1}{\sqrt{1-\alpha_{t-s}}}\boldsymbol{x}_t - \frac{\sqrt{\alpha_{t-s}}}{\sqrt{1-\alpha_{t-s}}}\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t)$

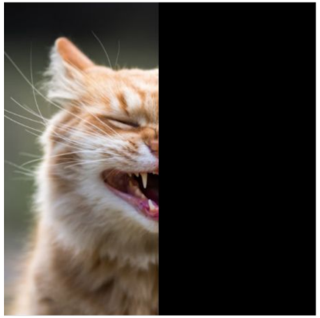
  $\boldsymbol{x}_{t-s} = \sqrt{\alpha_{t-s}}\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t) + \sqrt{1 - \alpha_{t-s} - \beta_{t-s}^2}\boldsymbol{\epsilon}_t + \beta_{t-s}\boldsymbol{z}_t$

end for

Return: $\boldsymbol{x}_0$

Constrained Sampling

Task:
Using the Stable Diffusion text-to-image model, inpaint the missing half



Outline of algorithm for sampling under constraints

Solution:

1. Given a diffusion model $\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t)$ we compute how well the current prediction fits the constraint

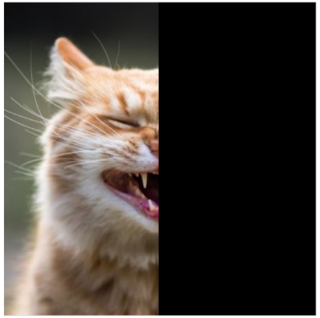
2. We transform the error from the output space ($\boldsymbol{e}$: clean image) to the input space ($\boldsymbol{e}_t$: noisy image) using the denoiser Jacobian $\boldsymbol{J}$

Sampling under constraints with a diffusion model

Input: Diffusion model $\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t)$, schedule α_i, β_i, step size s, constraint $C(\boldsymbol{x}_0, \boldsymbol{y})$, learning rate λ, optimization steps K.
$\boldsymbol{x}_T \sim N(\mathbf{0}, \boldsymbol{I})$
for $t = T, T-s, T-2s, \ldots, s$ **do**
    // Optimize latent for constraint
    for $i = 1, 2, \ldots, K$ **do**
        // Compute error
        // **non-linear**: gradient descent | **linear**: closed-form
        $\boldsymbol{e} = \nabla_{\hat{\boldsymbol{x}}_0} C(\hat{\boldsymbol{x}}_0, \boldsymbol{y})$ or $\boldsymbol{e} = \text{Solver}_C(\hat{\boldsymbol{x}}_0, \boldsymbol{y})$
        // Compute update direction and perform step
        $\boldsymbol{e}_t = \mathcal{F}(\boldsymbol{J}, \boldsymbol{e})$
        $\boldsymbol{x}_t = \boldsymbol{x}_t - \lambda \boldsymbol{e}_t$
    end for
    // Run diffusion step
    $\boldsymbol{z}_t \sim N(\mathbf{0}, \boldsymbol{I})$
    $\boldsymbol{\epsilon}_t = \frac{1}{\sqrt{1-\alpha_{t-s}}} \boldsymbol{x}_t - \frac{\sqrt{\alpha_{t-s}}}{\sqrt{1-\alpha_{t-s}}} \hat{\boldsymbol{x}}_0(\boldsymbol{x}_t)$
    $\boldsymbol{x}_{t-s} = \sqrt{\alpha_{t-s}} \hat{\boldsymbol{x}}_0(\boldsymbol{x}_t) + \sqrt{1 - \alpha_{t-s} - \beta_{t-s}^2} \boldsymbol{\epsilon}_t + \beta_{t-s} \boldsymbol{z}_t$
end for
Return: $\boldsymbol{x}_0$

Constrained Sampling

Task:
Using the Stable Diffusion text-to-image model, inpaint the missing half



Outline of algorithm for sampling under constraints

Sampling under constraints with a diffusion model

Input: Diffusion model $\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t)$, schedule α_i, β_i, step size s, constraint $C(\boldsymbol{x}_0, \boldsymbol{y})$, learning rate λ, optimization steps K.
$\boldsymbol{x}_T \sim N(\mathbf{0}, \boldsymbol{I})$
for $t = T, T-s, T-2s, \dots, s$ **do**
    // Optimize latent for constraint
    for $i = 1, 2, \dots, K$ **do**
        // Compute error
        // **non-linear**: gradient descent | **linear**: closed-form
        $\boldsymbol{e} = \nabla_{\hat{\boldsymbol{x}}_0} C(\hat{\boldsymbol{x}}_0, \boldsymbol{y})$ or $\boldsymbol{e} = \text{Solver}_C(\hat{\boldsymbol{x}}_0, \boldsymbol{y})$
        // Compute update direction and perform step
        $\boldsymbol{e}_t = \mathcal{F}(\boldsymbol{J}, \boldsymbol{e})$
        $\boldsymbol{x}_t = \boldsymbol{x}_t - \lambda \boldsymbol{e}_t$
    end for
    // Run diffusion step
    $\boldsymbol{z}_t \sim N(\mathbf{0}, \boldsymbol{I})$
    $\boldsymbol{\epsilon}_t = \frac{1}{\sqrt{1-\alpha_{t-s}}} \boldsymbol{x}_t - \frac{\sqrt{\alpha_{t-s}}}{\sqrt{1-\alpha_{t-s}}} \hat{\boldsymbol{x}}_0(\boldsymbol{x}_t)$
    $\boldsymbol{x}_{t-s} = \sqrt{\alpha_{t-s}} \hat{\boldsymbol{x}}_0(\boldsymbol{x}_t) + \sqrt{1 - \alpha_{t-s} - \beta_{t-s}^2} \boldsymbol{\epsilon}_t + \beta_{t-s} \boldsymbol{z}_t$
end for
Return: $\boldsymbol{x}_0$

Solution:

1. Given a diffusion model $\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t)$ we compute how well the current prediction fits the constraint
2. We transform the error from the output space ($\boldsymbol{e}$: clean image) to the input space ($\boldsymbol{e}_t$: noisy image) using the denoiser Jacobian $\boldsymbol{J}$
3. We update the diffusion latent with the computed noisy image update direction

Constrained Sampling

2 We transform the error from the output space ($\boldsymbol{e}$: clean image) to the input space ($\boldsymbol{e}_t$: noisy image) using the denoiser Jacobian $\boldsymbol{J}$

// Compute update direction and perform step

$$\boldsymbol{e}_t = \mathcal{F}(\boldsymbol{J}, \boldsymbol{e})$$

$$\boldsymbol{x}_t = \boldsymbol{x}_t - \lambda \boldsymbol{e}_t$$

Constrained Sampling

2 We transform the error from the output space ($\boldsymbol{e}$: clean image) to the input space ($\boldsymbol{e}_t$: noisy image) using the denoiser Jacobian $\boldsymbol{J}$

// Compute update direction and perform step

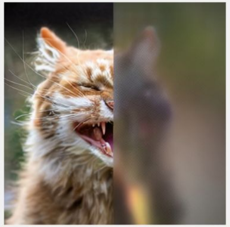
$$\boldsymbol{e}_t = \mathcal{F}(\boldsymbol{J}, \boldsymbol{e})$$

$$\boldsymbol{x}_t = \boldsymbol{x}_t - \lambda \boldsymbol{e}_t$$

"Local" Descent

$$\boldsymbol{x}'_t = \boldsymbol{x}_t - \lambda \boldsymbol{I} \boldsymbol{e}$$

✓ **Cheap**:
Constraint enforced locally
✗ **Inaccurate**:
Fails at capturing long-range dependencies



MPGD [2]
Time:18s
Memory: 9GB

Constrained Sampling

2 We transform the error from the output space ($\boldsymbol{e}$: clean image) to the input space ($\boldsymbol{e}_t$: noisy image) using the denoiser Jacobian $\boldsymbol{J}$

// Compute update direction and perform step

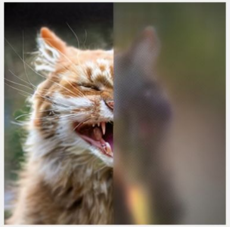
$$\boldsymbol{e}_t = \mathcal{F}(\boldsymbol{J}, \boldsymbol{e})$$

$$\boldsymbol{x}_t = \boldsymbol{x}_t - \lambda \boldsymbol{e}_t$$

"Local" Descent

$$\boldsymbol{x}_t' = \boldsymbol{x}_t - \lambda \boldsymbol{I} \boldsymbol{e}$$

✓ **Cheap**:
Constraint enforced locally
✗ **Inaccurate**:
Fails at capturing long-range dependencies



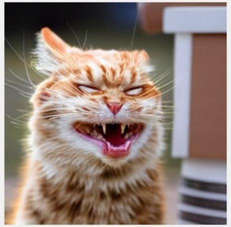
MPGD [2]
Time:18s
Memory: 9GB

Gradient Descent

$$\boldsymbol{x}_t' = \boldsymbol{x}_t - \lambda \boldsymbol{J}^T \boldsymbol{e}$$

✗ **Expensive**:
Backpropagation is slow & memory-heavy
✓ **Accurate**:
Constraint backpropagated through network — captures long-range effects

PSLD [1]
Time: 8min
Memory: 17GB



FreeDoM [3]
Time: 1min
Memory: 17GB

Constrained Sampling

2 We transform the error from the output space ($\boldsymbol{e}$: clean image) to the input space ($\boldsymbol{e}_t$: noisy image) using the denoiser Jacobian $\boldsymbol{J}$

// Compute update direction and perform step

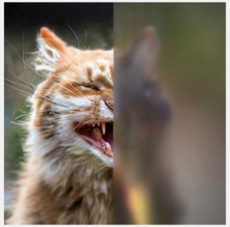
$$\boldsymbol{e}_t = \mathcal{F}(\boldsymbol{J}, \boldsymbol{e})$$

$$\boldsymbol{x}_t = \boldsymbol{x}_t - \lambda \boldsymbol{e}_t$$

"Local" Descent

$$\boldsymbol{x}_t' = \boldsymbol{x}_t - \lambda \boldsymbol{I}\boldsymbol{e}$$

✓ **Cheap**:
Constraint enforced locally
✗ **Inaccurate**:
Fails at capturing long-range dependencies



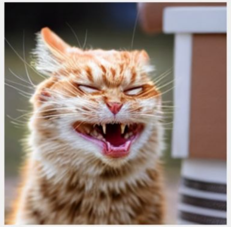
MPGD [2]
Time:18s
Memory: 9GB

Gradient Descent

$$\boldsymbol{x}_t' = \boldsymbol{x}_t - \lambda \boldsymbol{J}^T \boldsymbol{e}$$

✗ **Expensive**:
Backpropagation is slow & memory-heavy
✓ **Accurate**:
Constraint backpropagated through network — captures long-range effects

PSLD [1]
Time: 8min
Memory: 17GB



FreeDoM [3]
Time: 1min
Memory: 17GB

Inexact Newton (Ours)

$$\boldsymbol{x}_t' = \boldsymbol{x}_t - \lambda \boldsymbol{J}\boldsymbol{e}$$

✓ **Cheap**:
No backpropagation
✓ **Accurate**:
Captures long-range dependencies

Ours
Time: 15s
Memory: 9GB

Inexact Newton for constrained sampling

Why is this a Newton method?

- We need to solve the system: $\boldsymbol{J}\boldsymbol{e}_t = \boldsymbol{e}$

  Newton-Gauss (single-step): $\boldsymbol{e}_t = \boldsymbol{J}^{-1}\boldsymbol{e}$

- To avoid computing $\mathbf{J}^{-1}$ we can iteratively solve with inexact steps [5]. We propose the step

Inexact Newton (multiple steps): $\boldsymbol{e}_t' = \boldsymbol{e}_t - \lambda \boldsymbol{J}\boldsymbol{e}$

Inexact Newton for constrained sampling

Why is this a Newton method?

- We need to solve the system: $\boldsymbol{J}\boldsymbol{e}_t = \boldsymbol{e}$

  Newton-Gauss (single-step): $\boldsymbol{e}_t = \boldsymbol{J}^{-1}\boldsymbol{e}$

- To avoid computing $\mathbf{J}^{-1}$ we can iteratively solve with inexact steps [5]. We propose the step

Inexact Newton (multiple steps): $\boldsymbol{e}_t' = \boldsymbol{e}_t - \lambda \boldsymbol{J}\boldsymbol{e}$

Why is it cheaper than gradient descent?

- The Jacobian-vector product can be computed with two forward passes.

$$\boldsymbol{J}\boldsymbol{e} \approx \frac{\hat{\boldsymbol{x}}_0(\boldsymbol{x}_t + \delta \boldsymbol{e}) - \hat{\boldsymbol{x}}_0(\boldsymbol{x}_t)}{\delta}$$

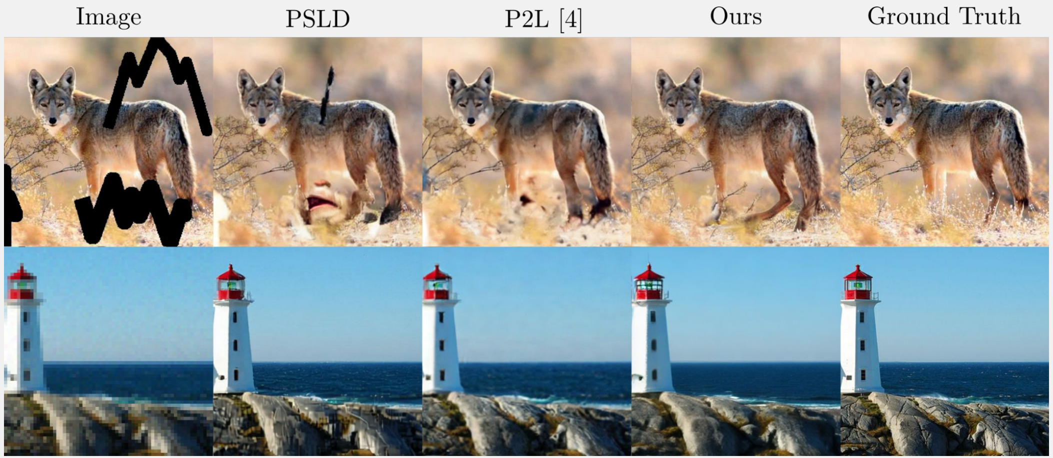
- This is cheaper than backpropagation (gradient descent)

Results - Linear constraints

Method	Inpaint (Free-form) PSNR ↑	LPIPS ↓	FID ↓	Time
P2L$^{+\text{ captions}}$	21.99	0.229	32.82	30m
LDPS	21.54	0.332	46.72	6m
PSLD	20.92	0.251	40.57	8m
Ours	21.73	0.258	19.39	15s

Method	Super-res (×8) PSNR ↑	LPIPS ↓	FID ↓	Time
P2L$^{+\text{ captions}}$	23.38	0.386	51.81	30m
LDPS	23.21	0.475	61.09	6m
PSLD	23.17	0.471	60.81	8m
Ours	24.26	0.455	60.99	1m
Ours$^{+\text{ captions}}$	24.95	0.405	44.74	1m

Results - Linear constraints

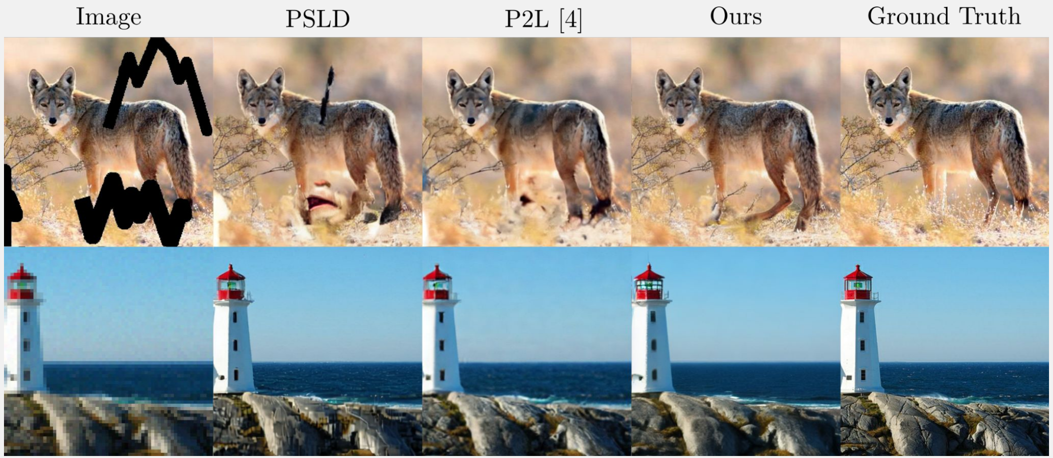


Method	Inpaint (Free-form) PSNR ↑	LPIPS ↓	FID ↓	Time
P2L+ captions	21.99	0.229	32.82	30m
LDPS	21.54	0.332	46.72	6m
PSLD	20.92	0.251	40.57	8m
Ours	21.73	0.258	19.39	15s

Method	Super-res (×8) PSNR ↑	LPIPS ↓	FID ↓	Time
P2L+ captions	23.38	0.386	51.81	30m
LDPS	23.21	0.475	61.09	6m
PSLD	23.17	0.471	60.81	8m
Ours	24.26	0.455	60.99	1m
Ours+ captions	24.95	0.405	44.74	1m

- **Inpainting**: Better consistency between the given and missing regions (FID ↓)

Results - Linear constraints

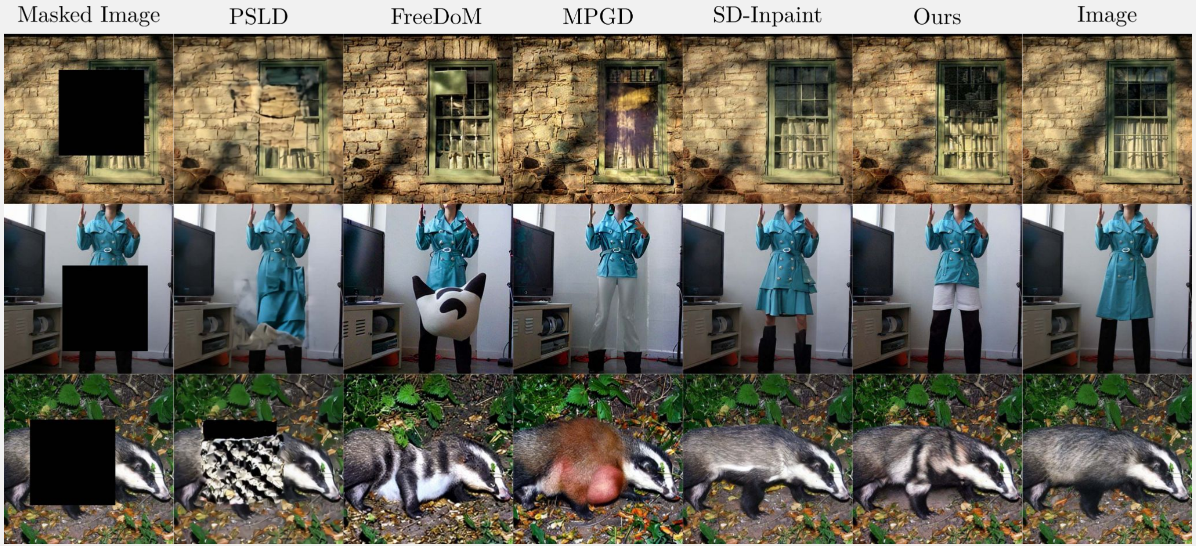


Method	Inpaint (Free-form) PSNR ↑	LPIPS ↓	FID ↓	Time
P2L+ captions	21.99	0.229	32.82	30m
LDPS	21.54	0.332	46.72	6m
PSLD	20.92	0.251	40.57	8m
Ours	21.73	0.258	19.39	15s

Method	Super-res (×8) PSNR ↑	LPIPS ↓	FID ↓	Time
P2L+ captions	23.38	0.386	51.81	30m
LDPS	23.21	0.475	61.09	6m
PSLD	23.17	0.471	60.81	8m
Ours	24.26	0.455	60.99	1m
Ours+ captions	24.95	0.405	44.74	1m

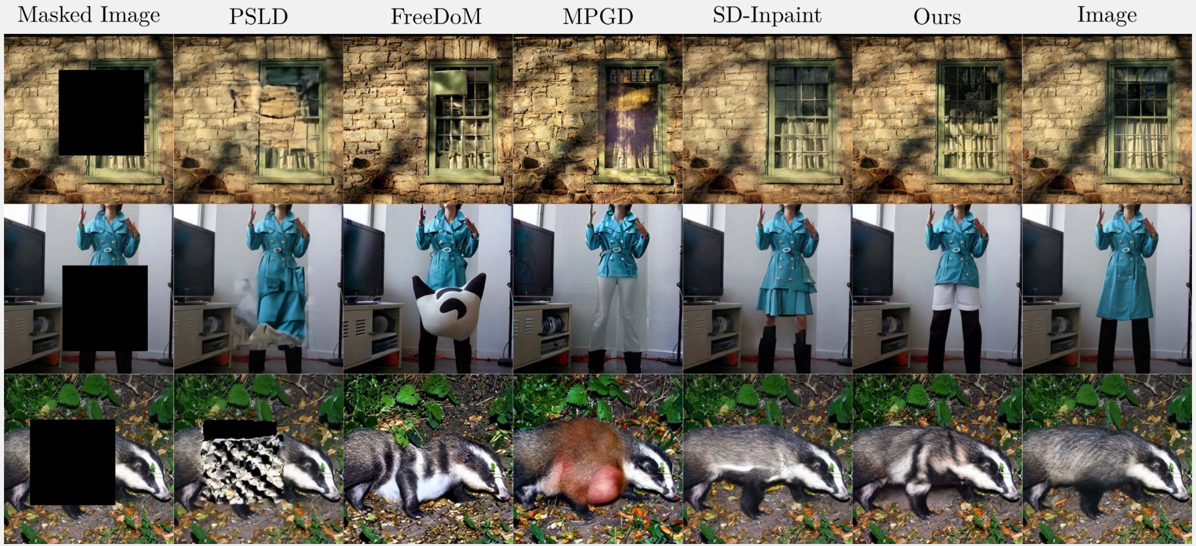
- **Inpainting**: Better consistency between the given and missing regions (FID ↓)
- **Super-resolution**: Similar quality at a fraction of the inference time

Results - Linear constraints



	Inpaint (Box)			
Method	PSNR ↑	LPIPS ↓	FID ↓	Time
LDPS	17.52	0.42	76.32	6m
PSLD	17.30	0.38	74.02	8m
FreeDoM	16.18	0.42	55.68	1m
Ours ($K = 5$, $\lambda = 0.5$)	18.30	0.30	42.01	15s
Ours ($K = 2$, $\lambda = 0.5$)	18.01	0.39	68.75	7s
Ours ($K = 5$, $\lambda = 1.0$)	17.48	0.32	47.20	15s
SD-Inpaint	19.05	0.28	32.93	4s

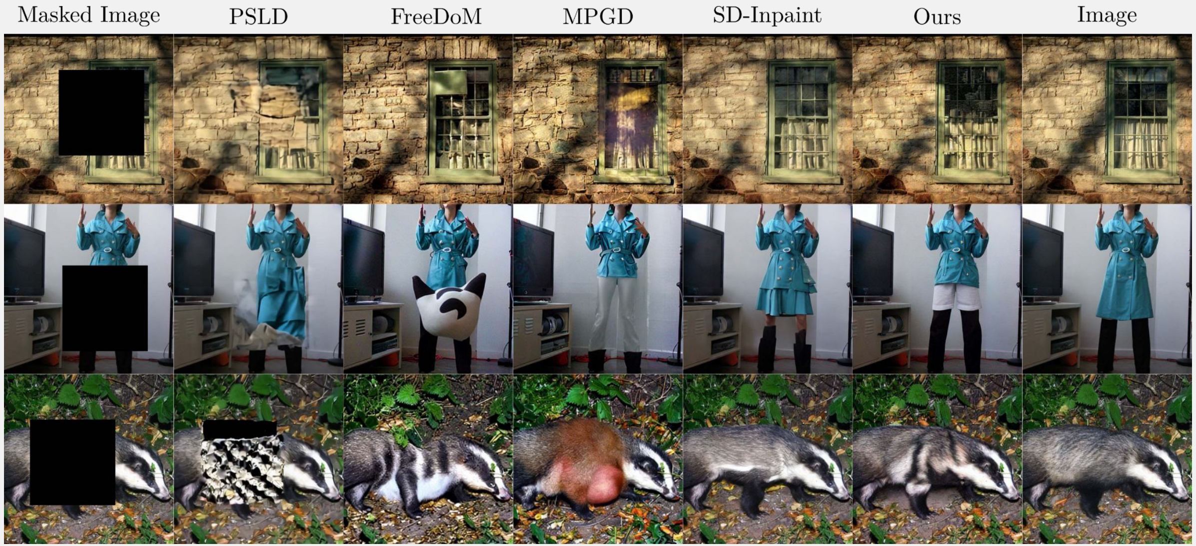
Results - Linear constraints



Method	Inpaint (Box)			
	PSNR ↑	LPIPS ↓	FID ↓	Time
LDPS	17.52	0.42	76.32	6m
PSLD	17.30	0.38	74.02	8m
FreeDoM	16.18	0.42	55.68	1m
Ours ($K = 5$, $\lambda = 0.5$)	18.30	0.30	42.01	15s
Ours ($K = 2$, $\lambda = 0.5$)	18.01	0.39	68.75	7s
Ours ($K = 5$, $\lambda = 1.0$)	17.48	0.32	47.20	15s
SD-Inpaint	19.05	0.28	32.93	4s

- **Box Inpainting**: Other methods fail at long-range consistency

Results - Linear constraints

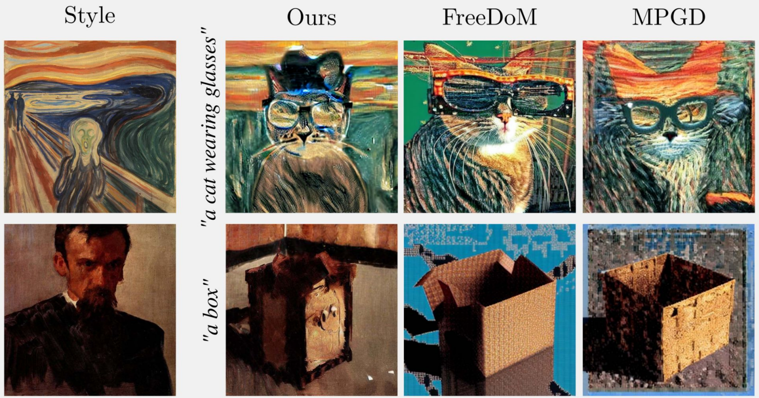


	Inpaint (Box)			
Method	PSNR ↑	LPIPS ↓	FID ↓	Time
LDPS	17.52	0.42	76.32	6m
PSLD	17.30	0.38	74.02	8m
FreeDoM	16.18	0.42	55.68	1m
Ours ($K = 5$, $\lambda = 0.5$)	18.30	0.30	42.01	15s
Ours ($K = 2$, $\lambda = 0.5$)	18.01	0.39	68.75	7s
Ours ($K = 5$, $\lambda = 1.0$)	17.48	0.32	47.20	15s
SD-Inpaint	19.05	0.28	32.93	4s

- **Box Inpainting**: Other methods fail at long-range consistency
- Our results are the closest to full fine-tuning (SD-Inpaint)

Results - Non-linear constraints

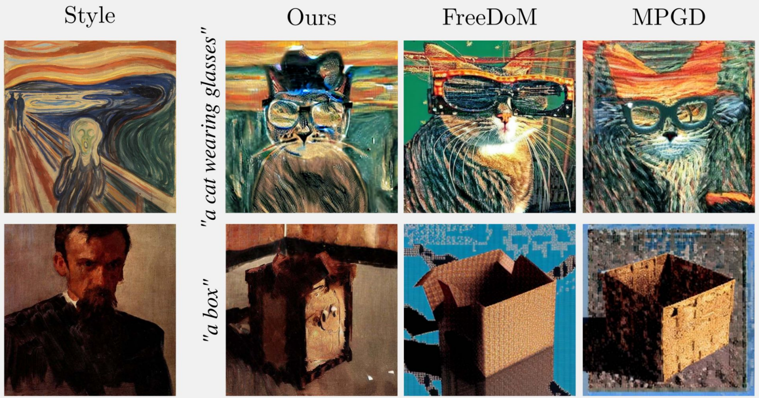
Style-guided generation



Method	Style Score ↓	CLIP ↑	Time
DDIM	761.0	31.61	9s
FreeDoM	498.08	30.14	80s
MPGD	441.00	26.61	50s
Ours ($w = 2$)	368.37	23.95	45s
Ours ($w = 5$)	310.96	24.57	45s
Ours$^{\text{OpenCLIP}}$ ($w = 5$)	434.45	25.94	45s

Results - Non-linear constraints

Style-guided generation



Method	Style Score ↓	CLIP ↑	Time
DDIM	761.0	31.61	9s
FreeDoM	498.08	30.14	80s
MPGD	441.00	26.61	50s
Ours ($w = 2$)	368.37	23.95	45s
Ours ($w = 5$)	310.96	24.57	45s
Ours$^{\text{OpenCLIP}}$ ($w = 5$)	434.45	25.94	45s

Style guidance: Better results with the same compute

Poster Session:

Thu 4 Dec 4:30 p.m. PST — 7:30 p.m. PST

Exhibit Hall C,D,E

GitHub:

arXiv:

References:
[1] Rout, Litu, et al. "Solving linear inverse problems provably via posterior sampling with latent diffusion models." NeurIPS 2023
[2] He, Yutong, et al. "Manifold Preserving Guided Diffusion." ICLR 2024
[3] Yu, Jiwen, et al. "Freedom: Training-free energy-guided conditional diffusion model." ICCV 2023
[4] Chung, Hyungjin, et al. "Prompt-tuning Latent Diffusion Models for Inverse Problems." ICML 2024
[5] Dembo, Ron S., Stanley C. Eisenstat, and Trond Steihaug. "Inexact newton methods." SIAM Journal on Numerical analysis 1982

Acknowledgements:
Part of this work was done during an internship at Microsoft Research Redmond. This research was also partially supported by NSF grants IIS-2123920, IIS-2212046.