Posterior Sampling Methods for Imaging Inverse Problems

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Matthew Bendel, B.S.

Graduate Program in Electrical and Computer Engineering

The Ohio State University

2025

Dissertation Committee:

Prof. Philip Schniter, Advisor

Prof. Rizwan Ahmad, Advisor

Prof. Emre Ertin

Abstract

In image recovery, one seeks to reconstruct an image from degraded measurements that are distorted, incomplete, and/or noise-corrupted. Often times, image recovery is posed as finding a single "best" reconstruction, which is known as *point estimation*. Due to the ill-posed nature of the problem, however, there can exist many images that are consistent with the measurements and the prior knowledge of the true image. In the Bayesian framework, posterior sampling can be used to explore this vast solution space by generating many probable reconstructions. This allows for uncertainty quantification and/or navigation of the perception-distortion tradeoff.

In the first two chapters of the dissertation, we focus on posterior sampling with conditional generative adversarial networks (cGANs). Typically, cGANs are regarded as producing high quality samples that have low diversity. Therefore, in the first chapter of the dissertation, we propose rcGAN which solves the lack-of-diversity issue via a novel regularization that is comprised of a supervised-L1 loss plus an adaptively weighted standard-deviation (SD) reward. We apply rcGAN to box inpainting and magnetic resonance (MR) image recovery, demonstrating its advantages over existing cGANs and contemporary diffusion methods.

In the second chapter of the dissertation we propose pcaGAN, an improvement over rcGAN with a novel regularization that aims for correctness in the K principal components of the posterior covariance matrix in addition to the posterior mean

and trace-covariance. We demonstrate pcaGAN's effectiveness on MNIST denoising, large-scale random inpainting, and MRI recovery, and show its advantages over other cGANs (including rcGAN), as well as contemporary diffusion methods.

For the third chapter of the dissertation, we shift our focus to diffusion models due to their unmatched versatility. Doing so requires approximating the gradient of the measurement-conditional score function in the diffusion reverse process. We show that the approximations produced by existing methods are relatively poor, especially early in the reverse process, and so we propose a new approach that iteratively reestimates and "renoises" the estimate several times per diffusion step. This iterative approach, which we call Fast Iterative REnoising (FIRE), injects colored noise that is shaped to ensure that the pre-trained diffusion model always sees white noise, in accordance with how it was trained. We then embed FIRE into the DDIM reverse process and show that the resulting "DDfire" offers state-of-the-art accuracy and runtime on box inpainting, Gaussian and motion deblurring, and 4x super-resolution.

Finally, we will propose some future research directions and share concluding thoughts.

To the love of my life, Allison.

Acknowledgments

This work was supported in part by the National Institutes of Health under grant R01-EB029957. I am deeply grateful for this support, which made much of this research possible.

Earning a Ph.D. has been one of the most challenging yet rewarding experiences of my life. Along the way, I have been surrounded by people who supported me through difficulties and celebrated with me in moments of triumph. I want to take this opportunity to recognize those who shared this journey with me and to express my heartfelt gratitude.

First and foremost, I thank my advisors, Professors Philip Schniter and Rizwan Ahmad, for their outstanding mentorship, guidance, and encouragement. I feel incredibly fortunate to have learned from two brilliant researchers who consistently pushed me to grow, while showing patience when I stumbled. Your lessons will stay with me long after this dissertation.

I am also grateful to Professor Emre Ertin for serving on both my candidacy and dissertation committees, and to Professor Kiryung Lee for his service on my candidacy committee as well as the excellent instruction he provided in two of my courses. I extend thanks to all the professors I studied under at OSU over the past 9.5 years—your teaching has shaped both my research and who I am today.

To my labmates—Jeff, Saurav, and Xuan—thank you for your generosity, candor, and collaboration. I am especially indebted to Saurav for his contributions to our "DDfire" paper, including work completed even after graduation, and for his guidance through candidacy and defense.

I was fortunate to spend an internship at Descript, where I had an extraordinary experience. Thank you to my manager, Jose, and the entire generative models team—Xingzhe, Stephen, Mithilesh, Sumukh, and Alejandro. I learned so much in a short time (including the essential truth that all bugs are sampling bugs) and am proud of the work we accomplished together.

Beyond academia, I owe immense gratitude to my friends and family. To all of you: your encouragement carried me through difficult times and made this journey far richer.

Josh, my brother—you have been with me at every step, from Club Penguin to college roommates to standing beside me as my best man. I am grateful every day for our bond and proud to be your brother.

Kyle, my best friend—your presence made the Ph.D. years bearable and often joyful. From Warzone shifts to Elden Ring marathons to iceberg movie lists, you've filled these years with memories I will always treasure.

Alex and Anthony, my friends for nearly two decades—you've been constants through every phase of life. Alex, your loyalty and kindness inspire me. Anthony, your focus and drive continue to amaze me. I cannot imagine this journey without either of you.

To Devin, Jay, and Nate—it has been a joy growing our friendships in recent years. Each of you has left a lasting mark on this chapter of my life.

I have also been blessed with the support of extended family. Aunt Tiff, Justin, Natalie, Aunt Lisa, and Connor — thank you. Aunt Tiff, I especially cherish our growing closeness during my Ph.D., from movie nights to many research conversations you patiently endured.

To my immediate family — Nana, Papa, Rick, and Adam—your love and encouragement have been foundational. Nana and Papa, you have always been more than grandparents—you are additional parents, and I am profoundly lucky to have you. Rick, you are the father I always wanted; your presence and guidance mean the world to me. And Adam, I am proud of the man you are becoming and grateful to have you as my brother.

I also thank my wonderful mother, Stephanie. You have always believed in me, always pushed me to do my best, and always been there when I needed you most. This accomplishment is as much yours as it is mine.

In more recent years, I have been grateful to see my family grow. To my in-laws, Kelly, Jeremy, and Abby, thank you for welcoming me so warmly and for your unwavering support.

Finally, to my wife, Allison—you are the love of my life and my greatest source of strength. Through every challenge of this Ph.D., you stood by me, and every joy was brighter because you were there. Earning this degree is one of my proudest achievements, but marrying you will always be the greatest. I love you more than words can say, and I look forward to a lifetime together.

Vita

August 2017 - May 2021	B.S. Electrical and Computer Engineering, The Ohio State University, Columbus, USA
August 2021 - present	Graduate Research Associate, Dept. of Electrical and Computer Engineering, The Ohio State University, Columbus, USA
May 2025 - August 2025	Applied Research Scientist Intern, Descript, San Francisco, USA
August 2025 - present	Graduate Teaching Associate, Dept. of Electrical and Computer Engineering, The Ohio State University, Columbus, USA

Publications

Research Publications

M. Bendel, S. K. Shastri, R. Ahmad, and P. Schniter "Solving Inverse Problems using Diffusion with Iterative Colored Renoising," *Transactions on Machine Learning Research*, August 2025.

M. Bendel, R. Ahmad, and P. Schniter, "pcaGAN: Improving Posterior-Sampling cGANs via Principal Component Regularization," in *Proc. Neural Info. Process. Syst. Conf.*, December 2024.

M. Bendel, R. Ahmad, and P. Schniter, "Mask-agnostic posterior sampling MRI via conditional GANs with guided reconstruction," in *Proc. NeurIPS Workshop on Deep Inverse Problems*, December 2023.

M. Bendel, R. Ahmad, and P. Schniter, "A regularized conditional GAN for posterior sampling in inverse problems," in *Proc. Neural Info. Process. Syst. Conf.*, December 2023.

Fields of Study

Major Field: Electrical and Computer Engineering

Table of Contents

Page

List of Tables

List of Figures

Chapter 1: Introduction

In image recovery, the goal is to recover the true image $\boldsymbol{x}$ from noisy/distorted/incomplete measurements $\boldsymbol{y} = \mathcal{M}(\boldsymbol{x})$. This arises in, e.g., linear inverse problems such as denoising, deblurring, inpainting, and magnetic resonance imaging (MRI) where $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{w}$ for $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \sigma_{\boldsymbol{y}}^2 \boldsymbol{I})$, as well as in non-linear inverse problems like phase-retrieval and image-to-image translation.

In much of the literature, image recovery is posed as point estimation, where the goal is to return a single best estimate $\widehat{\boldsymbol{x}}$. However, there are several shortcomings of this approach. First, it's not clear how to define "best," since L2- or L1-minimizing $\widehat{\boldsymbol{x}}$ are often regarded as too blurry, while efforts to make $\widehat{\boldsymbol{x}}$ perceptually pleasing can sacrifice agreement with the true image $\boldsymbol{x}$ and cause hallucinations [8, 35, 59, 10, 30].

Another major limitation with point estimation is that it's unclear how certain one can be about the recovered $\widehat{\boldsymbol{x}}$. Quantifying the uncertainty in $\widehat{\boldsymbol{x}}$ is especially important in medical applications such as MRI, where a diagnosis must be made based on the measurements $\boldsymbol{y}$. Rather than simply reporting our best guess of whether a pathology is present or absent based on $\widehat{\boldsymbol{x}}$, we might want to report the probability that the pathology is present (given all available data).

To address the limitations of point estimation, *posterior-sampling*-based image recovery [7, 99, 90, 98, 28, 93, 79, 37, 4, 111, 110, 9, 97, 85, 39, 87, 86, 48, 17, 96] aims

to generate $P \geq 1$ samples $\{\widehat{\boldsymbol{x}}_i\}_{i=1}^P$ from the posterior distribution $p_{\mathsf{x}|\mathsf{y}}(\cdot|\boldsymbol{y})$. Posterior sampling facilitates numerous strategies to quantify the uncertainty in estimating $\boldsymbol{x}$, or any function of $\boldsymbol{x}$, from $\boldsymbol{y}$ [2, 52]. It also can help with visualizing uncertainty and increasing robustness to adversarial attacks [66]. That said, the design of accurate and computationally-efficient posterior samplers remains an open problem. The recent literature has focused on conditional variational autoencoders (cVAEs) [28, 93, 79], conditional normalizing flows (cNFs) [7, 99, 90, 98], conditional generative adversarial networks (cGANs) [37, 4, 111, 110, 9], and Langevin/score/diffusion-based generative models [97, 85, 39, 87, 86, 48, 17, 96].

In the first two chapters of the dissertation, we focus on posterior sampling cGANs. We first propose rcGAN in Chapter 2, a novel cGAN regularization framework that enforces correctness in the generated $\boldsymbol{y}$-conditional mean *and* trace-covariance using L1 regularization plus a correctly weighted standard-deviation (SD) reward. We prove the correctness of the proposed regularization for the simple Gaussian case, and empirically demonstrate rcGAN's performance on accelerated MR image reconstrution and box inpainting, outperforming both cGAN and diffusion competitors.

We then propose pcaGAN in Chapter 3, an extension of rcGAN that encourages correctness in the K principal components of the $\boldsymbol{y}$-conditional covariance matrix, as well as the $\boldsymbol{y}$-conditional mean and trace covariance when sampling from the posterior. pcaGAN is inspired, in part, by a separate line of work where Nehme et al. [61] trained a Neural Posterior Principal Components (NPPC) network to directly estimate the eigenvectors and eigenvalues of the y-conditional covariance matrix, which allows powerful insights into the nature of uncertainty in an inverse problem. We demonstrate the effectiveness of pcaGAN on denoising, random large-scale inpainting,

and accelerated MR image recovery. There, we show that pcaGAN outperforms many contemporary diffusion and cGAN competitors, including rcGAN. We also demonstrate that pcaGAN recovers the principal components more accurately than NPPC using approximately the same runtime.

In Chapter 4, we shift our focus to diffusion-based methods. Diffusion modeling has emerged as a powerful approach to generate samples from a complex distribution p_0 [78, 84, 34, 87, 82]. Recently, diffusion has also been used to solve inverse problems [22], where the goal is to recover $\boldsymbol{x}_0 \sim p_0$ from measurements $\boldsymbol{y}$ in an unsupervised manner. There, a diffusion model is trained to generate samples from p_0 and, at test time, the reverse process is modified to incorporate knowledge of the measurements $\boldsymbol{y}$, with the goal of sampling from the posterior distribution $p(\boldsymbol{x}_0|\boldsymbol{y})$.

When implementing the reverse process, the main challenge is approximating the conditional score function $\nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{x}_t|\boldsymbol{y})$ at each step t, where $\boldsymbol{x}_t$ is an additive-white-Gaussian-noise (AWGN) corrupted version of $\boldsymbol{x}_0 \in \mathbb{R}^d$, and $\boldsymbol{y} \in \mathbb{R}^m$ is treated as a draw from a likelihood function $p(\boldsymbol{y}|\boldsymbol{x}_0)$. (See Sec. 4.1 for more details). A common approach uses Tweedie's formula [29] to write

$$\nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{x}_t|\boldsymbol{y}) = \frac{\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_t, \boldsymbol{y}\} - \boldsymbol{x}_t}{\sigma_t^2} \tag{1.1}$$

and then approximates the conditional denoiser $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_t, \boldsymbol{y}\}$ (e.g., [48, 96, 114, 20]).

In Chapter 4, we aim to improve the approximation of $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_t, \boldsymbol{y}\}$ at each step t. In particular, we propose an iterative approach to approximating $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_t, \boldsymbol{y}\}$ that we call Fast Iterative REnoising (FIRE). FIRE is like a plug-and-play (PnP) algorithm (see the PnP survey [5]) in that it iterates unconditional denoising with linear estimation from $\boldsymbol{x}_t$ and $\boldsymbol{y}$. We then embed FIRE into the DDIM diffusion reverse process [82], yielding the "DDfire" posterior sampler. Here, we show that

DDfire outperforms SOTA diffusion competitors in most metrics for a variety of linear inverse problems.

Finally, we conclude the dissertation by proposing some future research directions based on applying our cGAN regularization ideas from Chapters 2 and 3 in the context of training direct diffusion bridge (DDB) models [55, 23].

Chapter 2: A Regularized Conditional GAN for Posterior Sampling in Image Recovery Problems

In this chapter, we discuss rcGAN, a regularized conditional GAN for posterior sampling in inverse problems. Our proposed cGAN tackles the lack-of-diversity issue that often plagues continuous-conditioned GANs using a novel regularization that consists of supervised-ℓ_1 loss plus an adaptively weighted standard-deviation (SD) reward. The content of this chapter appears in "A regularized Conditional GAN for Posterior Sampling in Image Recovery Problems," which was published at the 37th annual conference on Neural Information Processing Systems (NeurIPS).

2.1 Background

In this chapter, we build on the Wasserstein cGAN framework from [4]. The goal is to design a generator network $G_{\boldsymbol{\theta}} : \mathcal{Z} \times \mathcal{Y} \to \mathcal{X}$ such that, for fixed $\boldsymbol{y}$, the random variable $\widehat{\boldsymbol{x}} = G_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{y})$ induced by $\boldsymbol{z} \sim p_{\mathsf{z}}$ has a distribution that best matches the posterior $p_{\mathsf{x}|\mathsf{y}}(\cdot|\boldsymbol{y})$ in Wasserstein-1 distance. Here, $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ denote the sets of $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{z}$, respectively, and $\boldsymbol{z}$ is drawn independently of $\boldsymbol{y}$.

The Wasserstein-1 distance can be expressed as

$$W_1(p_{\mathsf{x}|\mathsf{y}}(\cdot, \boldsymbol{y}), p_{\widehat{\mathsf{x}}|\mathsf{y}}(\cdot, \boldsymbol{y})) = \sup_{D \in L_1} \mathbb{E}_{\mathsf{x}|\mathsf{y}}\{D(\boldsymbol{x}, \boldsymbol{y})\} - \mathbb{E}_{\widehat{\mathsf{x}}|\mathsf{y}}\{D(\widehat{\boldsymbol{x}}, \boldsymbol{y})\}, \tag{2.1}$$

where L_1 denotes functions that are 1-Lipschitz with respect to their first argument and $D : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ is a "critic" or "discriminator" that tries to distinguish between true $\boldsymbol{x}$ and generated $\widehat{\boldsymbol{x}}$ given $\boldsymbol{y}$. Since we want the method to work for typical values of $\boldsymbol{y}$, we define a loss by taking an expectation of (2.1) over $\boldsymbol{y} \sim p_{\mathsf{y}}$. Since the expectation commutes with the supremum in (2.1), we have [4]

$$\mathbb{E}_{\mathsf{y}}\{W_1(p_{\mathsf{x}|\mathsf{y}}(\cdot,\boldsymbol{y}), p_{\widehat{\mathsf{x}}|\mathsf{y}}(\cdot,\boldsymbol{y}))\} = \sup_{D\in L_1} \mathbb{E}_{\mathsf{x},\mathsf{y}}\{D(\boldsymbol{x},\boldsymbol{y})\} - \mathbb{E}_{\widehat{\mathsf{x}},y}\{D(\widehat{\boldsymbol{x}},\boldsymbol{y})\} \tag{2.2}$$

$$= \sup_{D\in L_1} \mathbb{E}_{\mathsf{x},\mathsf{z},\mathsf{y}}\{D(\boldsymbol{x},\boldsymbol{y}) - D(G_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{y}),\boldsymbol{y})\}. \tag{2.3}$$

In practice, D is implemented by a neural network $D_{\boldsymbol{\phi}}$ with parameters $\boldsymbol{\phi}$, and $(\boldsymbol{\theta},\boldsymbol{\phi})$ are trained by alternately minimizing

$$\mathcal{L}_{\mathsf{adv}}(\boldsymbol{\theta},\boldsymbol{\phi}) \triangleq \mathbb{E}_{\mathsf{x},\mathsf{z},\mathsf{y}}\{D_{\boldsymbol{\phi}}(\boldsymbol{x},\boldsymbol{y}) - D_{\boldsymbol{\phi}}(G_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{y}),\boldsymbol{y})\} \tag{2.4}$$

with respect to $\boldsymbol{\theta}$ and minimizing $-\mathcal{L}_{\mathsf{adv}}(\boldsymbol{\theta},\boldsymbol{\phi}) + \mathcal{L}_{\mathsf{gp}}(\boldsymbol{\phi})$ with respect to $\boldsymbol{\phi}$, where $\mathcal{L}_{\mathsf{gp}}(\boldsymbol{\phi})$ is a gradient penalty that is used to encourage $D_{\boldsymbol{\phi}} \in L_1$ [31]. Furthemore, the expectation over $\boldsymbol{x}$ and $\boldsymbol{y}$ in (2.4) is replaced in practice by a sample average over the training examples $\{(\boldsymbol{x}_t,\boldsymbol{y}_t)\}_{t=1}^T$.

One of the main challenges with the cGAN framework in image recovery problems is that, for each measurement example $\boldsymbol{y}_t$, there is only a single image example $\boldsymbol{x}_t$. Thus, with the previously described training methodology, there is no incentive for the generator to produce diverse samples $G(\boldsymbol{z},\boldsymbol{y})|_{\boldsymbol{z}\sim p_{\mathsf{z}}}$ for a fixed $\boldsymbol{y}$. This can lead to the generator learning to ignore the code vector $\boldsymbol{z}$, which causes a form of "mode collapse."

Although issues with stability and mode collapse are also present in *unconditional* GANs (uGANs) or discretely conditioned cGANs [58], the causes are fundamentally different than in continuously conditioned cGANs like ours. With continuously

conditioned cGANs, there is only *one* example of a valid $\boldsymbol{x}_t$ for each given $\boldsymbol{y}_t$, whereas with uGANs there are many $\boldsymbol{x}_t$ and with discretely conditioned cGANs there are many $\boldsymbol{x}_t$ for each conditioning class. As a result, most strategies that are used to combat mode-collapse in uGANs [76, 43, 112] are not well suited to cGANs. For example, mini-batch discrimination strategies like MBSD [42], where the discriminator aims to distinguish a mini-batch of true samples $\{\boldsymbol{x}_t\}$ from a mini-batch of generated samples $\{\widehat{\boldsymbol{x}}_t\}$, don't work with cGANs because the posterior statistics are very different than the prior statistics.

To combat mode collapse in cGANs, Adler & Öktem [4] proposed to use a three-input discriminator $D_{\boldsymbol{\phi}}^{\mathsf{adler}} : \mathcal{X} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ and replace $\mathcal{L}_{\mathsf{adv}}$ from (2.4) with the loss

$$\mathcal{L}_{\mathsf{adv}}^{\mathsf{adler}}(\boldsymbol{\theta}, \boldsymbol{\phi}) \triangleq \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\mathsf{z}_2,\mathsf{y}} \big\{ \tfrac{1}{2} D_{\boldsymbol{\phi}}^{\mathsf{adler}}(\boldsymbol{x}, G_{\boldsymbol{\theta}}(\boldsymbol{z}_1, \boldsymbol{y}), \boldsymbol{y}) + \tfrac{1}{2} D_{\boldsymbol{\phi}}^{\mathsf{adler}}(G_{\boldsymbol{\theta}}(\boldsymbol{z}_2, \boldsymbol{y}), \boldsymbol{x}, \boldsymbol{y}) - D_{\boldsymbol{\phi}}^{\mathsf{adler}}(G_{\boldsymbol{\theta}}(\boldsymbol{z}_1, \boldsymbol{y}), G_{\boldsymbol{\theta}}(\boldsymbol{z}_2, \boldsymbol{y}), \boldsymbol{y}) \big\}, \tag{2.5}$$

which rewards variation between the first and second inputs to $D_{\boldsymbol{\phi}}^{\mathsf{adler}}$. They then proved that minimizing $\mathcal{L}_{\mathsf{adv}}^{\mathsf{adler}}$ in place of $\mathcal{L}_{\mathsf{adv}}$ does not compromise the Wasserstein cGAN objective, i.e., $\arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathsf{adv}}^{\mathsf{adler}}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{\mathsf{adv}}(\boldsymbol{\theta}, \boldsymbol{\phi})$. As we show in Sec. 2.3, this approach does prevent complete mode collapse, but it leaves much room for improvement.

2.2 Approach

2.2.1 Proposed Regularization: Supervised-ℓ_1 Plus SD Reward

We now propose a novel cGAN regularization framework, which we dub "rcGAN." To train the generator, we propose to solve

$$\arg\min_{\boldsymbol{\theta}}\{\beta_{\mathsf{adv}}\mathcal{L}_{\mathsf{adv}}(\boldsymbol{\theta},\boldsymbol{\phi})+\mathcal{L}_{1,\mathsf{SD},P_{\mathsf{rc}}}(\boldsymbol{\theta},\beta_{\mathsf{SD}})\} \tag{2.6}$$

with appropriately chosen $\beta_{\mathsf{adv}},\beta_{\mathsf{SD}}>0$ and $P_{\mathsf{rc}}\geq 2$, where the regularizer

$$\mathcal{L}_{1,\mathsf{SD},P_{\mathsf{rc}}}(\boldsymbol{\theta},\beta_{\mathsf{SD}}) \triangleq \mathcal{L}_{1,P_{\mathsf{rc}}}(\boldsymbol{\theta})-\beta_{\mathsf{SD}}\mathcal{L}_{\mathsf{SD},P_{\mathsf{rc}}}(\boldsymbol{\theta}) \tag{2.7}$$

is constructed from the P_{rc}-sample supervised-ℓ_1 loss and standard-deviation (SD) reward terms

$$\mathcal{L}_{1,P_{\mathsf{rc}}}(\boldsymbol{\theta}) \triangleq \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_\mathsf{P},\mathsf{y}}\left\{\|\boldsymbol{x}-\widehat{\boldsymbol{x}}_{(P_{\mathsf{rc}})}\|_1\right\} \tag{2.8}$$

$$\mathcal{L}_{\mathsf{SD},P_{\mathsf{rc}}}(\boldsymbol{\theta}) \triangleq \sqrt{\tfrac{\pi}{2P_{\mathsf{rc}}(P_{\mathsf{rc}}-1)}}\sum_{i=1}^{P_{\mathsf{rc}}}\mathbb{E}_{\mathsf{z}_1,\ldots,\mathsf{z}_\mathsf{P},\mathsf{y}}\left\{\|\widehat{\boldsymbol{x}}_i-\widehat{\boldsymbol{x}}_{(P_{\mathsf{rc}})}\|_1\right\}, \tag{2.9}$$

and where $\{\widehat{\boldsymbol{x}}_i\}$ denote the generated samples and $\widehat{\boldsymbol{x}}_{(P)}$ their P-sample average:

$$\widehat{\boldsymbol{x}}_i \triangleq G_{\boldsymbol{\theta}}(\boldsymbol{z}_i,\boldsymbol{y}), \qquad \widehat{\boldsymbol{x}}_{(P)} \triangleq \tfrac{1}{P}\sum_{i=1}^{P}\widehat{\boldsymbol{x}}_i. \tag{2.10}$$

The use of supervised-ℓ_1 loss and SD reward in (2.7) is not heuristic. As shown in Proposition 1, it encourages the samples $\{\widehat{\boldsymbol{x}}_i\}$ to match the true posterior in both mean and covariance.

Proposition 1. *Suppose $P_{\mathsf{rc}}\geq 2$ and $\boldsymbol{\theta}$ has complete control over the $\boldsymbol{y}$-conditional mean and covariance of $\widehat{\boldsymbol{x}}_i$. Then the parameters $\boldsymbol{\theta}_*=\arg\min_{\boldsymbol{\theta}}\mathcal{L}_{1,\mathsf{SD},P_{\mathsf{rc}}}(\boldsymbol{\theta},\beta^{\mathcal{N}}_{\mathsf{SD}})$ with*

$$\beta^{\mathcal{N}}_{\mathsf{SD}} \triangleq \sqrt{\tfrac{2}{\pi P_{\mathsf{rc}}(P_{\mathsf{rc}}+1)}} \tag{2.11}$$

yield generated statistics

$$\mathbb{E}_{\mathsf{z_i|y}}\{\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta}_*)|\boldsymbol{y}\} = \mathbb{E}_{\mathsf{x|y}}\{\boldsymbol{x}|\boldsymbol{y}\} = \widehat{\boldsymbol{x}}_{\mathsf{mmse}} \tag{2.12a}$$

$$\mathrm{Cov}_{\mathsf{z_i|y}}\{\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta}_*)|\boldsymbol{y}\} = \mathrm{Cov}_{\mathsf{x|y}}\{\boldsymbol{x}|\boldsymbol{y}\} \tag{2.12b}$$

when the elements of $\widehat{\boldsymbol{x}}_i$ and $\boldsymbol{x}$ are independent Gaussian conditioned on $\boldsymbol{y}$. Thus, minimizing $\mathcal{L}_{1,\mathsf{SD},P_{\mathsf{rc}}}$ encourages the $\boldsymbol{y}$-conditional mean and covariance of $\widehat{\boldsymbol{x}}_i$ to match those of the true $\boldsymbol{x}$.

See App. C.1 for a proof. In imaging applications, $\widehat{\boldsymbol{x}}_i$ and $\boldsymbol{x}$ may not be independent Gaussian conditioned on $\boldsymbol{y}$, and so the value of β_{SD} in (2.11) may not be appropriate. Thus we propose a method to automatically tune β_{SD} in Sec. 2.2.2.

Figure 2.1 shows a toy example with parameters $\boldsymbol{\theta} = [\mu, \sigma]^\top$, generator $G_{\boldsymbol{\theta}}(z, y) = \mu + \sigma z$, and $z \sim \mathcal{N}(0, 1)$, giving generated posterior $p_{\mathsf{x|y}}(x|y) = \mathcal{N}(x; \mu, \sigma^2)$. Assuming the true $p_{\mathsf{x|y}}(x|y) = \mathcal{N}(x; \mu_0, \sigma_0^2)$, Figs. 2.1(a)-(b) show that, by minimizing the proposed $\mathcal{L}_{1,\mathsf{SD},P_{\mathsf{rc}}}(\boldsymbol{\theta}, \beta_{\mathsf{SD}}^{\mathcal{N}})$ regularization over $\boldsymbol{\theta} = [\mu, \sigma]^\top$ for any $P_{\mathsf{rc}} \geq 2$, we recover the true $\boldsymbol{\theta}_0 = [\mu_0, \sigma_0]^\top$. They also show that the cost function steepens as P_{rc} decreases, which agrees with our empirical finding that $P_{\mathsf{rc}} = 2$ tends to work best in practice.

We note that regularizing a cGAN with supervised-ℓ_1 loss alone is not new; see, e.g., [37]. In fact, the use of supervised-ℓ_1 loss is often preferred over ℓ_2 in image recovery because it results in sharper, more visually pleasing results [109]. But regularizing a cGAN using supervised-ℓ_1 loss *alone* can push the generator towards mode collapse, for reasons described below. For example, in [37], ℓ_1-induced mode collapse led the authors to use dropout to induce generator variation, instead of random $\boldsymbol{z}_i$.

Why not supervised-ℓ_2 regularization? One may wonder: Why regularize using supervised-ℓ_1 loss plus an SD reward in (2.7) and not a more conventional choice like

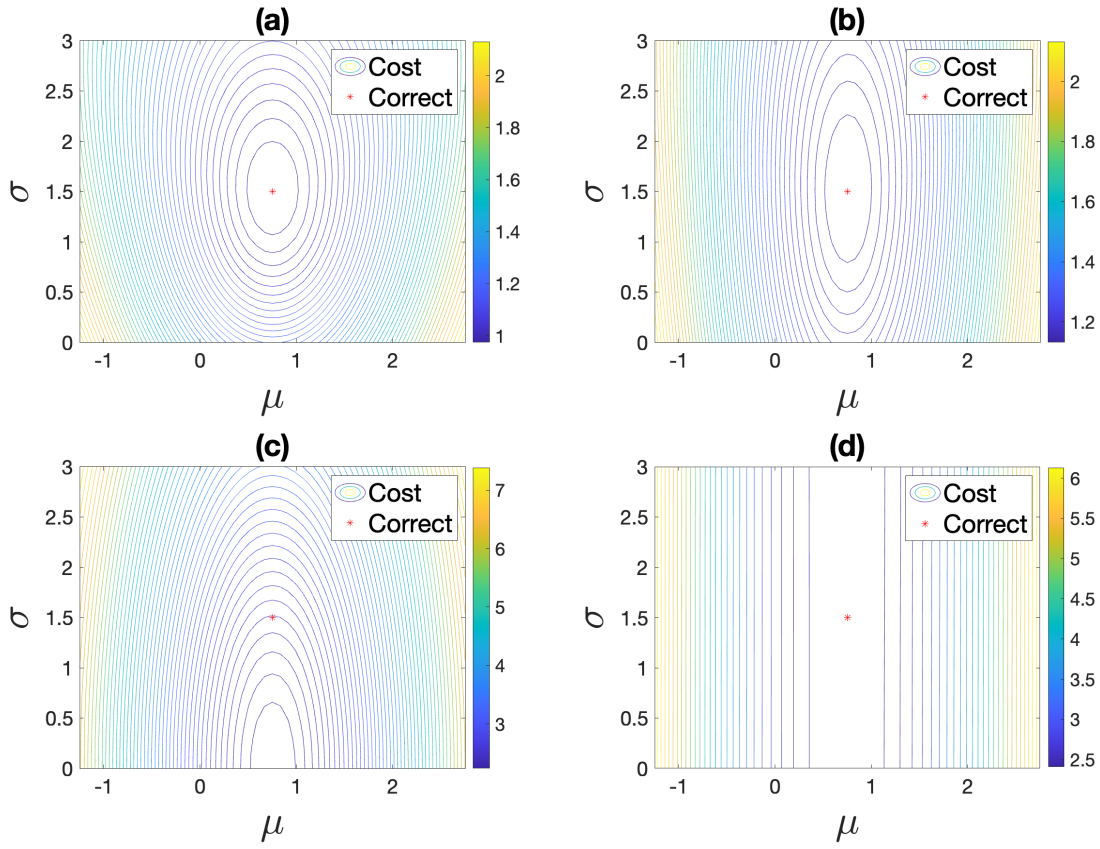


Figure 2.1: The contours show the regularizer value versus $\boldsymbol{\theta}=[\mu,\sigma]^\top$ for four different regularizers: (a) supervised-ℓ_1 plus SD reward with $\beta_{\mathsf{SD}}=\beta_{\mathsf{SD}}^{\mathcal{N}}$ at $P_{\mathsf{rc}}=2$, (b) supervised-ℓ_1 plus SD reward with $\beta_{\mathsf{SD}}=\beta_{\mathsf{SD}}^{\mathcal{N}}$ at $P_{\mathsf{rc}}=8$, (c) supervised-ℓ_2 at $P_{\mathsf{rc}}=8$, and (d) supervised-ℓ_2 plus variance reward at $P_{\mathsf{rc}}=8$. The red star shows the true posterior parameters $[\mu_0,\sigma_0]^\top$.

supervised-ℓ_2 loss plus a variance reward, or even supervised-ℓ_2 loss alone? We start by discussing the latter.

The use of supervised-ℓ_2 regularization in a cGAN can be found in [37]. In this case, to train the generator, one would aim to solve $\arg\min_{\boldsymbol{\theta}}\{\mathcal{L}_{\mathsf{adv}}(\boldsymbol{\theta},\boldsymbol{\phi})+\lambda\mathcal{L}_2(\boldsymbol{\theta})\}$ with some $\lambda>0$ and

$$\mathcal{L}_2(\boldsymbol{\theta}) \triangleq \mathbb{E}_{\mathsf{x},\mathsf{y}}\left\{\|\boldsymbol{x}-\mathbb{E}_{\mathsf{z}}\{G_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{y})\}\|_2^2\right\}. \tag{2.13}$$

Ohayon et al. [64] revived this idea for the explicit purpose of fighting mode collapse. In practice, however, the $\mathbb{E}_{\mathsf{z}}$ term in (2.13) must be implemented by a finite-sample average, giving

$$\mathcal{L}_{2,P_{\mathsf{rc}}}(\boldsymbol{\theta}) \triangleq \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_{\mathsf{P}},\mathsf{y}}\left\{\left\|\boldsymbol{x} - \tfrac{1}{P_{\mathsf{rc}}}\sum_{i=1}^{P_{\mathsf{rc}}} G_{\boldsymbol{\theta}}(\boldsymbol{z}_i,\boldsymbol{y})\right\|_2^2\right\} \tag{2.14}$$

for some $P_{\mathsf{rc}} \geq 2$. For example, Ohayon's implementation [65] used $P_{\mathsf{rc}} = 8$. As we show in Proposition 2, $\mathcal{L}_{2,P_{\mathsf{rc}}}$ *induces* mode collapse rather than prevents it.

Proposition 2. *Say P_{rc} is finite and $\boldsymbol{\theta}$ has complete control over the $\boldsymbol{y}$-conditional mean and covariance of $\widehat{\boldsymbol{x}}_i$. Then the parameters $\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_{2,P_{\mathsf{rc}}}(\boldsymbol{\theta})$ yield generated statistics*

$$\mathbb{E}_{\mathsf{z}_i|\mathsf{y}}\{\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta}_*)|\boldsymbol{y}\} = \mathbb{E}_{\mathsf{x}|\mathsf{y}}\{\boldsymbol{x}|\boldsymbol{y}\} = \widehat{\boldsymbol{x}}_{\mathsf{mmse}} \tag{2.15a}$$

$$\mathrm{Cov}_{\mathsf{z}_i|\mathsf{y}}\{\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta}_*)|\boldsymbol{y}\} = \mathbf{0}. \tag{2.15b}$$

Thus, minimizing $\mathcal{L}_{2,P_{\mathsf{rc}}}$ encourages mode collapse.

The proof (see App. C.2) follows from the bias-variance decomposition of (2.14), i.e.,

$$\begin{aligned}
&\mathcal{L}_{2,P_{\mathsf{rc}}}(\boldsymbol{\theta}) \\
&= \mathbb{E}_{\mathsf{y}}\left\{\|\widehat{\boldsymbol{x}}_{\mathsf{mmse}} - \mathbb{E}_{\mathsf{z}_i|\mathsf{y}}\{\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta})|\boldsymbol{y}\}\|_2^2 + \tfrac{1}{P_{\mathsf{rc}}}\,\mathrm{tr}[\mathrm{Cov}_{\mathsf{z}_i|\mathsf{y}}\{\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta})|\boldsymbol{y}\}]\right. \\
&\left. + \mathbb{E}_{\mathsf{x}|\mathsf{y}}\{\|\boldsymbol{e}_{\mathsf{mmse}}\|_2^2|\boldsymbol{y}\}\right\},
\end{aligned} \tag{2.16}$$

where $\boldsymbol{e}_{\mathsf{mmse}} \triangleq \boldsymbol{x} - \boldsymbol{x}_{\mathsf{mmse}}$ is the MMSE error. Figure 2.1(c) shows that $\mathcal{L}_{2,P_{\mathsf{rc}}}$ regularization causes mode collapse in the toy example, and Sec. 2.3.2 shows that it causes mode collapse in MRI.

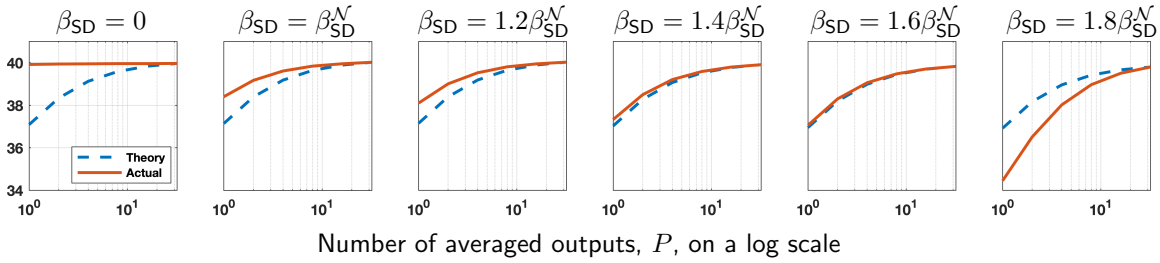


Figure 2.2: Example PSNR of $\widehat{\boldsymbol{x}}_{(P)}$ versus P, the number of averaged outputs, for several training β_{SD} and MRI recovery at $R = 4$. Also shown is the theoretical behavior for true-posterior samples.

Why not supervised ℓ_2 plus a variance reward? To mitigate $\mathcal{L}_{2,P_{\mathsf{rc}}}$'s incentive for mode-collapse, the second term in (2.16) could be canceled using a variance reward, giving

$$\mathcal{L}_{2,\mathsf{var},P_{\mathsf{rc}}}(\boldsymbol{\theta}) \triangleq \mathcal{L}_{2,P_{\mathsf{rc}}}(\boldsymbol{\theta}) - \tfrac{1}{P_{\mathsf{rc}}}\mathcal{L}_{\mathsf{var},P_{\mathsf{rc}}}(\boldsymbol{\theta}) \tag{2.17}$$

$$\text{with } \mathcal{L}_{\mathsf{var},P_{\mathsf{rc}}}(\boldsymbol{\theta}) \triangleq \tfrac{1}{P_{\mathsf{rc}}-1}\textstyle\sum_{i=1}^{P_{\mathsf{rc}}} \mathbb{E}_{\mathsf{z}_1,\ldots,\mathsf{z}_P,\mathsf{y}}\{\|\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta}) - \widehat{\boldsymbol{x}}_{(P)}(\boldsymbol{\theta})\|_2^2\}. \tag{2.18}$$

since App. C.3 shows that $\mathcal{L}_{\mathsf{var},P_{\mathsf{rc}}}(\boldsymbol{\theta})$ is an unbiased estimator of the posterior trace-covariance:

$$\mathcal{L}_{\mathsf{var},P_{\mathsf{rc}}}(\boldsymbol{\theta}) = \mathbb{E}_{\mathsf{y}}\{\mathrm{tr}[\mathrm{Cov}_{\mathsf{z}_i|\mathsf{y}}\{\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta})|\boldsymbol{y}\}]\} \quad \text{for any } P_{\mathsf{rc}} \geq 2. \tag{2.19}$$

However, the resulting $\mathcal{L}_{2,\mathsf{var},P_{\mathsf{rc}}}$ regularizer in (2.17) does not encourage the generated covariance to match the *true* posterior covariance, unlike the proposed $\mathcal{L}_{1,\mathsf{SD},P_{\mathsf{rc}}}$ regularizer in (2.7) (recall Proposition 1). For the toy example, this behavior is visible in Fig. 2.1(d).

2.2.2 Auto-tuning of SD Reward Weight β_{SD}

We now propose a method to auto-tune β_{SD} in (2.7) for a given training dataset. Our approach is based on the observation that larger values of β_{SD} tend to yield

samples $\widehat{\boldsymbol{x}}_i$ with more variation. But more variation is not necessarily better; we want samples with the correct amount of variation. To assess the correct amount of variation, we compare the expected ℓ_2 error of the P-sample average $\widehat{\boldsymbol{x}}_{(P)}$ to that of $\widehat{\boldsymbol{x}}_{(1)}$. When $\{\widehat{\boldsymbol{x}}_i\}$ are true-posterior samples, these errors follow a particular relationship, as established by Proposition 3 below (see App. C.4 for a proof).

Proposition 3. *Say $\widehat{\boldsymbol{x}}_i \sim p_{\mathsf{x}|\mathsf{y}}(\cdot|\boldsymbol{y})$ are independent samples of the true posterior and, for any $P \geq 1$, their P-sample average is $\widehat{\boldsymbol{x}}_{(P)} \triangleq \frac{1}{P}\sum_{i=1}^{P} \widehat{\boldsymbol{x}}_i$. Then*

$$\mathcal{E}_P \triangleq \mathbb{E}\{\|\widehat{\boldsymbol{x}}_{(P)} - \boldsymbol{x}\|_2^2|\boldsymbol{y}\} = \tfrac{P+1}{P}\mathcal{E}_{\mathsf{mmse}}, \quad \textit{which implies} \quad \tfrac{\mathcal{E}_1}{\mathcal{E}_P} = \tfrac{2P}{P+1}. \tag{2.20}$$

Experimentally we find that $\mathcal{E}_1/\mathcal{E}_P$ grows with the SD reward weight β_{SD}. (See Fig. 2.2.) Thus, we propose to adjust β_{SD} so that the observed SNR-gain ratio $\mathcal{E}_1/\mathcal{E}_{P_{\mathsf{val}}}$ matches the correct ratio $(2P_{\mathsf{val}})/(P_{\mathsf{val}}+1)$ from (2.20), for some $P_{\mathsf{val}} \geq 2$ that does not need to match P_{rc}. (We use $P_{\mathsf{val}} = 8$ in Sec. 2.3.) In particular, at each training epoch τ, we approximate $\mathcal{E}_{P_{\mathsf{val}}}$ and $\mathcal{E}_1$ as follows:

$$\widehat{\mathcal{E}}_{P_{\mathsf{val}},\tau} \triangleq \tfrac{1}{V}\textstyle\sum_{v=1}^{V} \|\frac{1}{P_{\mathsf{val}}}\sum_{i=1}^{P_{\mathsf{val}}} G_{\boldsymbol{\theta}_\tau}(\boldsymbol{z}_{i,v}, \boldsymbol{y}_v) - \boldsymbol{x}_v\|_2^2 \tag{2.21}$$

$$\widehat{\mathcal{E}}_{1,\tau} \triangleq \tfrac{1}{V}\textstyle\sum_{v=1}^{V} \|G_{\boldsymbol{\theta}_\tau}(\boldsymbol{z}_{1,v}, \boldsymbol{y}_v) - \boldsymbol{x}_v\|_2^2, \tag{2.22}$$

with validation set $\{(\boldsymbol{x}_v, \boldsymbol{y}_v)\}_{v=1}^{V}$ and i.i.d. codes $\{\boldsymbol{z}_{i,v}\}_{i=1}^{P_{\mathsf{val}}}$. We update β_{SD} using gradient descent:

$$\beta_{\mathsf{SD},\tau+1} = \beta_{\mathsf{SD},\tau} - \mu_{\mathsf{SD}} \cdot \big([\widehat{\mathcal{E}}_{1,\tau}/\widehat{\mathcal{E}}_{P_{\mathsf{val}},\tau}]_{\mathrm{dB}} - [2P_{\mathsf{val}}/(P_{\mathsf{val}}+1)]_{\mathrm{dB}}\big)\beta_{\mathsf{SD}}^{\mathcal{N}} \quad \text{for} \quad \tau = 0, 1, 2, \ldots \tag{2.23}$$

with $\beta_{\mathsf{SD},0} = \beta_{\mathsf{SD}}^{\mathcal{N}}$, some $\mu_{\mathsf{SD}} > 0$, and $[x]_{\mathrm{dB}} \triangleq 10\log_{10}(x)$.

2.3 Numerical Experiments

2.3.1 Conditional Fréchet Inception Distance

As previously stated, our goal is to train a generator $G_{\boldsymbol{\theta}}$ so that, for typical fixed values of $\boldsymbol{y}$, the generated distribution $p_{\widehat{\mathsf{x}}|\mathsf{y}}(\cdot|\boldsymbol{y})$ matches the true posterior $p_{\mathsf{x}|\mathsf{y}}(\cdot|\boldsymbol{y})$. It is essential to have a quantitative metric for evaluating performance with respect to this goal. For example, it is not enough that the generated samples are "accurate" in the sense of being close to the ground truth, nor is it enough that they are "diverse" according to some heuristically chosen metric.

We quantify posterior-approximation quality using the Conditional Fréchet Inception Distance (CFID) [80], a computationally efficient approximation to the conditional Wasserstein distance

$$\text{CWD} \triangleq \mathbb{E}_{\mathsf{y}}\{W_2(p_{\mathsf{x}|\mathsf{y}}(\cdot,\boldsymbol{y}), p_{\widehat{\mathsf{x}}|\mathsf{y}}(\cdot,\boldsymbol{y}))\}. \tag{2.24}$$

In (2.24), $W_2(p_{\mathsf{a}}, p_{\mathsf{b}})$ denotes the Wasserstein-2 distance between distributions p_{a} and p_{b}, defined as

$$W_2(p_{\mathsf{a}}, p_{\mathsf{b}}) \triangleq \min_{p_{\mathsf{a},\mathsf{b}} \in \Pi(p_{\mathsf{a}}, p_{\mathsf{b}})} \mathbb{E}_{\mathsf{a},\mathsf{b}}\{\|\boldsymbol{a}-\boldsymbol{b}\|_2^2\}, \tag{2.25}$$

where $\Pi(p_{\mathsf{a}}, p_{\mathsf{b}}) \triangleq \left\{p_{\mathsf{a},\mathsf{b}} : p_{\mathsf{a}} = \int p_{\mathsf{a},\mathsf{b}}\,\mathrm{d}\boldsymbol{b} \text{ and } p_{\mathsf{b}} = \int p_{\mathsf{a},\mathsf{b}}\,\mathrm{d}\boldsymbol{a}\right\}$ denotes the set of joint distributions $p_{\mathsf{a},\mathsf{b}}$ with prescribed marginals p_{a} and p_{b}. Similar to how FID [33]—a popular uGAN metric—is computed, CFID approximates CWD (2.24) as follows: i) the random vectors $\boldsymbol{x}$, $\widehat{\boldsymbol{x}}$, and $\boldsymbol{y}$ are replaced by low-dimensional embeddings $\underline{\boldsymbol{x}}$, $\underline{\widehat{\boldsymbol{x}}}$, and $\underline{\boldsymbol{y}}$, generated by the convolutional layers of a deep network, and ii) the embedding distributions $p_{\underline{\mathsf{x}}|\underline{\mathsf{y}}}$ and $p_{\underline{\widehat{\mathsf{x}}}|\underline{\mathsf{y}}}$ are approximated by multivariate Gaussians. More details on CFID are given in App. D.1.

2.3.2 MRI Experiments

We consider parallel MRI recovery, where the goal is to recover a complex-valued multicoil image $\boldsymbol{x}$ from zero-filled measurements $\boldsymbol{y}$ (see App. A.1 for details).

Data. For training data $\{\boldsymbol{x}_t\}$, we used the first 8 slices of all fastMRI [102] T2 brain training volumes with at least 8 coils, cropping to 384×384 pixels and compressing to 8 virtual coils [108], yielding 12 200 training images. Using the same procedure, 2 376 testing and 784 validation images were obtained from the fastMRI T2 brain testing volumes. From the 2 376 testing images, 72 were randomly selected to evaluate the Langevin technique [39] in order to limit its sample generation to 6 days. To create the measurement $\boldsymbol{y}_t$, we transformed $\boldsymbol{x}_t$ to the Fourier domain, sampled using pseudo-random GRO patterns [41] at acceleration $R = 4$ and $R = 8$, and Fourier-transformed the zero-filled k-space measurements back to the (complex, multicoil) image domain.

Architecture. We use a U-Net [74] for our generator and a standard CNN for our discriminator, along with data-consistency as in App. A.2. Architecture and training details are given in App. D.2.

Competitors. We compare our cGAN to the Adler and Öktem's cGAN [4], Ohayon et al.'s pscGAN [64], Jalal et al.'s fastMRI Langevin approach [39], and Sriram et al.'s E2E-VarNet [89]. The cGAN from [4] uses generator loss $\beta_{\mathsf{adv}}\mathcal{L}_{\mathsf{adv}}^{\mathsf{adler}}(\boldsymbol{\theta}, \boldsymbol{\phi})$ and discriminator loss $-\mathcal{L}_{\mathsf{adv}}^{\mathsf{adler}}(\boldsymbol{\theta}, \boldsymbol{\phi}) + \alpha_1 \mathcal{L}_{\mathsf{gp}}(\boldsymbol{\phi}) + \alpha_2 \mathcal{L}_{\mathsf{drift}}(\boldsymbol{\phi})$, while the cGAN from [64] uses generator loss $\beta_{\mathsf{adv}}\mathcal{L}_{\mathsf{adv}}(\boldsymbol{\theta}, \boldsymbol{\phi}) + \mathcal{L}_{2,P}(\boldsymbol{\theta})$ and discriminator loss $-\mathcal{L}_{\mathsf{adv}}(\boldsymbol{\theta}, \boldsymbol{\phi}) + \alpha_1 \mathcal{L}_{\mathsf{gp}}(\boldsymbol{\phi}) + \alpha_2 \mathcal{L}_{\mathsf{drift}}(\boldsymbol{\phi})$. Each used the value of β_{adv} specified in the original paper. All cGANs used the same generator and discriminator architectures, except that [4] used extra discriminator input channels to facilitate the 3-input loss (2.5). For the fastMRI

Langevin approach [39], we did not modify the authors' implementation in [38] except to use the GRO sampling mask. For the E2E-VarNet [89], we use the same training procedure and hyperparameters outlined in [39] except that we use the GRO sampling mask.

Testing. To evaluate performance, we converted the multicoil outputs $\widehat{\boldsymbol{x}}_i$ to complex-valued images using SENSE-based coil combining [70] with ESPIRiT-estimated [94] coil sensitivity maps, as described in App. A.1. Magnitude images were used to compute performance metrics.

Table 2.1: Average MRI results at $R \in \{4, 8\}$. Tested with VGG-16 features. CFID^1, FID, and APSD used 72 test samples and $P = 32$, CFID^2 used 2 376 test samples and $P = 8$, and CFID^3 used all 14 576 samples and $P = 1$

	$R = 4$						$R = 8$					
Model	CFID^1↓	CFID^2↓	CFID^3↓	FID↓	APSD	Time (4)↓	CFID^1↓	CFID^2↓	CFID^3↓	FID↓	APSD	Time (4)↓
E2E-VarNet (Sriram et al. [89])	7.47	6.99	6.61	8.84	0.0	310ms	7.82	6.81	6.31	8.40	0.0	316ms
Langevin (Jalal et al. [39])	5.29	-	-	6.12	5.9e-6	14 min	7.34	-	-	14.32	7.6e-6	14 min
cGAN (Adler & Öktem [4])	6.39	4.27	3.82	5.25	3.9e-6	**217 ms**	10.10	6.30	5.72	10.77	7.7e-6	**217 ms**
pscGAN (Ohayon et al. [64])	4.06	3.27	2.95	6.45	7.2e-8	**217 ms**	6.04	4.59	4.27	11.05	7.7e-7	**217 ms**
rcGAN	**3.10**	**1.54**	**1.29**	**3.75**	3.8e-6	**217 ms**	**4.87**	**2.23**	**1.79**	**7.72**	7.6e-6	**217 ms**

Results. Table 2.1 shows CFID, FID, APSD $\triangleq$ $(\frac{1}{NP}\sum_{i=1}^{P}\|\widehat{\boldsymbol{x}}_{(P)} - \widehat{\boldsymbol{x}}_i\|^2)^{1/2}$, and 4-sample generation time at $R \in \{4, 8\}$. (C)FID was computed using VGG-16 (not Inception-v3) to better align with radiologists' perceptions [46]. As previously described, the Langevin method was evaluated using only 72 test images. Because CFID is biased at small sample sizes [80], we re-evaluated the other methods using all 2 376 test images, and again using all 14 576 training and test images. Table 2.1 shows that our approach gave significantly better CFID and FID than the competitors. Also, the APSD of Ohayon et al.'s pscGAN was an order-of-magnitude smaller than the

Table 2.2: Average PSNR, SSIM, LPIPS, and DISTS of $\widehat{\boldsymbol{x}}_{(P)}$ versus P for $R = 4$ MRI

	PSNR↑						SSIM↑					
Model	P=1	P=2	P=4	P=8	P=16	P=32	P=1	P=2	P=4	P=8	P=16	P=32
E2E-VarNet (Sriram et al. [89])	**39.93**	-	-	-	-	-	**0.9641**	-	-	-	-	-
Langevin (Jalal et al. [39])	36.04	37.02	37.65	37.99	38.17	38.27	0.8989	0.9138	0.9218	0.9260	0.9281	0.9292
cGAN (Adler & Öktem [4])	35.63	36.64	37.24	37.56	37.73	37.82	0.9330	0.9445	0.9478	0.9480	0.9477	0.9473
pscGAN (Ohayon et al. [64])	39.44	39.46	39.46	39.47	39.47	39.47	0.9558	0.9546	0.9539	0.9535	0.9533	0.9532
rcGAN	36.96	38.14	38.84	39.24	39.44	39.55	0.9440	0.9526	0.9544	0.9542	0.9537	0.9533
	LPIPS↓						DISTS↓					
Model	P=1	P=2	P=4	P=8	P=16	P=32	P=1	P=2	P=4	P=8	P=16	P=32
E2E-VarNet (Sriram et al. [89])	0.0316	-	-	-	-	-	0.0859	-	-	-	-	-
Langevin (Jalal et al. [39])	0.0545	0.0394	0.0336	0.0320	0.0317	0.0316	0.1116	0.0921	0.0828	0.0793	0.0781	0.0777
cGAN (Adler & Öktem [4])	0.0285	0.0255	0.0273	0.0298	0.0316	0.0327	0.0972	0.0857	0.0878	0.0930	0.0967	0.0990
pscGAN (Ohayon et al. [64])	0.0245	0.0247	0.0248	0.0249	0.0249	0.0249	0.0767	0.0790	0.0801	0.0807	0.0810	0.0811
rcGAN	0.0175	**0.0164**	0.0188	0.0216	0.0235	0.0245	**0.0546**	0.0563	0.0667	0.0755	0.0809	0.0837

others, indicating mode collapse. The cGANs generated samples 3 800 times faster than the Langevin approach from [39].

Tables 2.2 and 2.3 show PSNR, SSIM, LPIPS [107], and DISTS [27] for the P-sample average $\widehat{\boldsymbol{x}}_{(P)}$ at $P \in \{1, 2, 4, 8, 16, 32\}$ and $R \in \{4, 8\}$, respectively. While the E2E-VarNet achieves the best PSNR at $R \in \{4, 8\}$ and the best SSIM at $R = 4$, the proposed rcGAN achieves the best LPIPS and DISTS performances at $R \in \{4, 8\}$ when $P = 2$ and the best SSIM at $R = 8$ when $P = 8$. The P dependence can be explained by the perception-distortion tradeoff [11]: as P increases, $\widehat{\boldsymbol{x}}_{(P)}$ transitions from better perceptual quality to lower ℓ_2 distortion. PSNR favors $P \to \infty$ (e.g., ℓ_2 optimality) while the other metrics favor particular combinations of perceptual quality and distortion. The plots in Appendices E.1.1 and E.1.2 show zoomed-in versions of $\widehat{\boldsymbol{x}}_{(P)}$ that visually demonstrate the perception-distortion tradeoff at $P \in \{1, 2, 4, 32\}$: smaller P yield sharper images with more variability from the ground truth, while larger P yield smoother reconstructions.

Table 2.3: Average PSNR, SSIM, LPIPS, and DISTS of $\widehat{\boldsymbol{x}}_{(P)}$ versus P for $R = 8$ MRI

	PSNR↑						SSIM↑					
Model	P=1	P=2	P=4	P=8	P=16	P=32	P=1	P=2	P=4	P=8	P=16	P=32
E2E-VarNet (Sriram et al. [89])	**36.49**	-	-	-	-	-	0.9220	-	-	-	-	-
Langevin (Jalal et al. [39])	32.17	32.83	33.45	33.74	33.83	33.90	0.8725	0.8919	0.9031	0.9091	0.9120	0.9137
cGAN (Adler & Öktem [4])	31.31	32.31	32.92	33.26	33.42	33.51	0.8865	0.9045	0.9103	0.9111	0.9102	0.9095
pscGAN (Ohayon et al. [64])	34.89	34.90	34.90	34.90	34.91	34.92	0.9222	0.9217	0.9213	0.9211	0.9211	0.9210
rcGAN	32.32	33.67	34.53	35.01	35.27	35.42	0.9030	0.9199	0.9252	**0.9257**	0.9251	0.9246
	LPIPS↓						DISTS↓					
Model	P=1	P=2	P=4	P=8	P=16	P=32	P=1	P=2	P=4	P=8	P=16	P=32
E2E-VarNet (Sriram et al. [89])	0.0575	-	-	-	-	-	0.1253	-	-	-	-	-
Langevin (Jalal et al. [39])	0.0769	0.0619	0.0579	0.0589	0.0611	0.0611	0.1341	0.1136	0.1086	0.1119	0.1175	0.1212
cGAN (Adler & Öktem [4])	0.0698	0.0614	0.0623	0.0667	0.0704	0.0727	0.1407	0.1262	0.1252	0.1291	0.1334	0.1361
pscGAN (Ohayon et al. [64])	0.0532	0.0536	0.0539	0.0540	0.0534	0.0540	0.1128	0.1143	0.1151	0.1155	0.1157	0.1158
rcGAN	0.0418	**0.0379**	0.0421	0.0476	0.0516	0.0539	0.0906	**0.0877**	0.0965	0.1063	0.1135	0.1177

Figure 2.3 shows zoomed versions of two posterior samples $\widehat{\boldsymbol{x}}_i$, as well as $\widehat{\boldsymbol{x}}_{(P)}$, at $P = 32$ and $R = 8$. The posterior samples show meaningful variations for the proposed method, essentially no variation for pscGAN, and vertical or horizontal reconstruction artifacts for Adler & Öktem's cGAN and the Langevin method, respectively. The $\widehat{\boldsymbol{x}}_{(P)}$ plots show that these artifacts are mostly suppressed by sample averaging with large P.

Figure 2.4 shows examples of $\widehat{\boldsymbol{x}}_{(P)}$, along with the corresponding pixel-wise absolute errors $|\widehat{\boldsymbol{x}}_{(P)} - \boldsymbol{x}|$ and pixel-wise SD images $(\frac{1}{P}\sum_{i=1}^{P}(\widehat{\boldsymbol{x}}_{(P)} - \widehat{\boldsymbol{x}}_i)^2)^{1/2}$, for $P = 32$ and $R = 8$. The absolute-error image for the Langevin technique looks more diffuse than those of the other methods in the brain region. The fact that it is brighter in the air region (i.e., near the edges) is a consequence of minor differences in sensitivity-map estimation. The pixel-wise SD images show a lack of variability for the E2E-VarNet, which does not generate posterior samples, as well as pscGAN, due to mode collapse. The Langevin pixel-wise SD images show localized hot-spots that appear to be reconstruction artifacts.

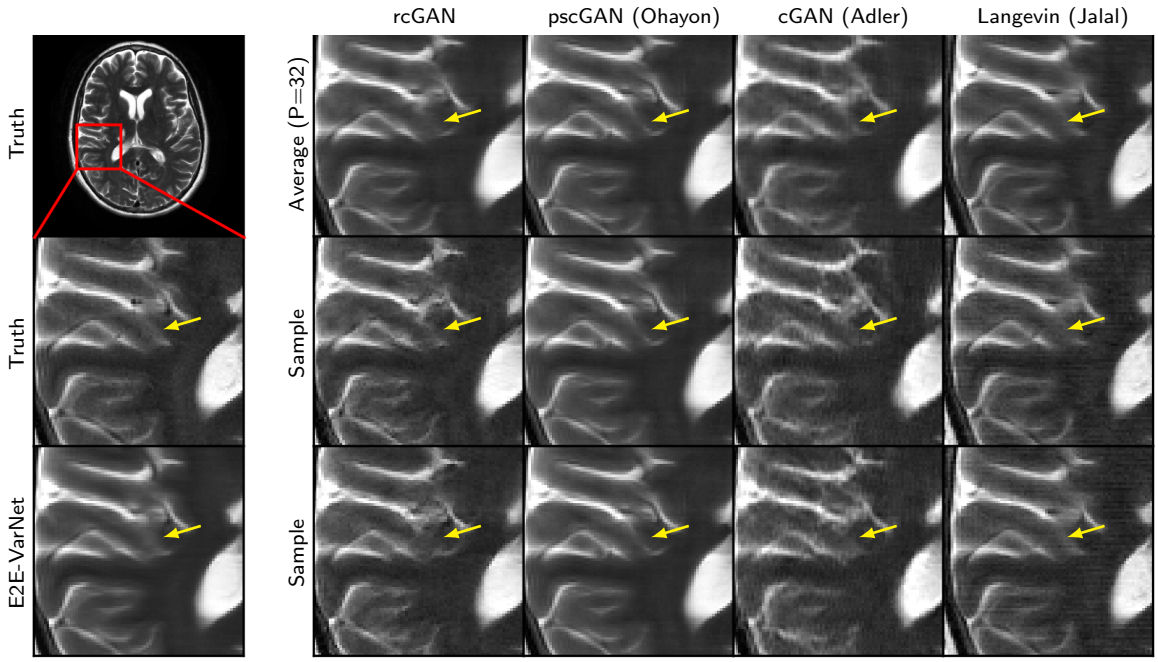


Figure 2.3: Example $R = 8$ MRI reconstructions. Arrows show meaningful variations across samples.

Appendices E.1.1 and E.1.2 show other example MRI recoveries with zoomed pixel-wise SD images at $R = 4$ and $R = 8$, respectively.

2.3.3 Inpainting Experiments

In this section, our goal is to complete a large missing square in a face image.

Data. We used 256×256 CelebA-HQ face images [42] and a centered 128×128 missing square. We randomly split the dataset, yielding 27 000 training, 2 000 validation, and 1 000 testing images.

Architecture. For our cGAN, we use the CoModGAN generator and discriminator from [111] with our proposed $\mathcal{L}_{1,\mathsf{SD},P_{\mathsf{rc}}}$ regularization. Unlike [111], we do not use MBSD [42] at the discriminator.

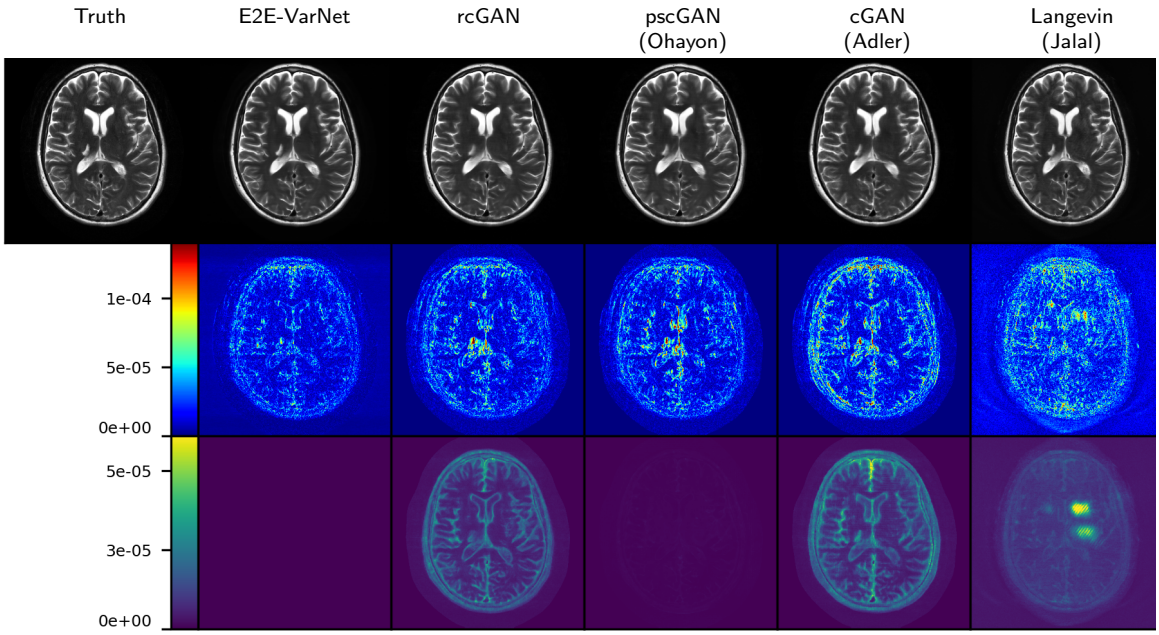


Figure 2.4: Example $R = 8$ MRI reconstructions with $P = 32$. Row one: P-sample average $\widehat{\boldsymbol{x}}_{(P)}$. Row two: pixel-wise absolute error $|\widehat{\boldsymbol{x}}_{(P)} - \boldsymbol{x}|$. Row three: pixel-wise SD $(\frac{1}{P}\sum_{i=1}^{P}(\widehat{\boldsymbol{x}}_i - \widehat{\boldsymbol{x}}_{(P)})^2)^{1/2}$.

Training/validation/testing. We use the same general training and testing procedure described in Sec. 2.3.2, but with $\beta_{\mathsf{adv}} = 5\text{e-}3$, $n_{\mathsf{batch}} = 100$, and 110 epochs of cGAN training. Running PyTorch on a server with 4 Tesla A100 GPUs, each with 82 GB of memory, the cGAN training takes approximately 2 days. FID was evaluated on 1 000 test images using $P = 32$ samples per measurement. To avoid the bias that would result from evaluating CFID on only 1 000 images (see Sec. 2.3.4), CFID was evaluated on all 30 000 images with $P = 1$.

Competitors. We compare with the state-of-the-art CoModGAN [111] and Score-based SDE [87] approaches. For CoModGAN, we use the PyTorch implementation from [103]. CoModGAN differs from our cGAN only in its use of MBSD and lack of $\mathcal{L}_{1,\mathsf{SD},P_{\mathsf{rc}}}$ regularization. For Song et al.'s SDE, we use the authors' implementation

Table 2.4: Average results for inpainting: FID was computed from 1 000 test images with $P = 32$, while CFID was computed from all 30 000 images with $P = 1$

Model	CFID↓	FID↓	Time (128)↓
Score-SDE (Song et al. [87])	5.11	7.92	48 min
CoModGAN (Zhao et al. [111])	5.29	8.50	**217 ms**
rcGAN	**4.69**	**7.45**	**217 ms**

from [88] with their pretrained weights and the settings they suggested for the 256×256 CelebA-HQ dataset.

Results. Table 2.4 shows test CFID, FID, and 128-sample generation time. The table shows that our approach gave the best CFID and FID, and that the cGANs generated samples 13 000 times faster than the score-based method. Figure 2.5 shows an example of five generated samples for the three methods under test. The samples are all quite good, although a few generated by CoModGAN and the score-based technique have minor artifacts. Some samples generated by our technique show almond-shaped eyes, demonstrating fairness. Additional examples are given in App. E.1.3.

2.3.4 CFID Decomposition

In this section, we investigate the small-sample bias effects of CFID, which have been previously noted in [80]. To do this, we write the CFID from (D.1) as a sum of two terms: a term that quantifies the conditional-mean error and a term that

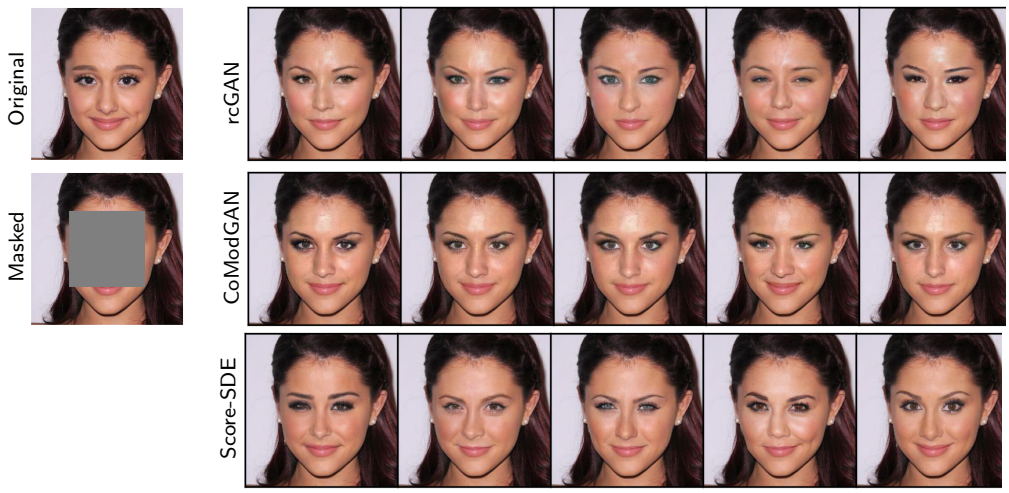


Figure 2.5: Example of inpainting a 128×128 square on a 256×256 resolution CelebA-HQ image.

quantifies the conditional-covariance error:

$$\mathrm{CFID} = \mathrm{CFID}_{\mathsf{mean}} + \mathrm{CFID}_{\mathsf{cov}} \tag{2.26}$$

$$\mathrm{CFID}_{\mathsf{mean}} \triangleq \mathbb{E}_{\mathsf{y}}\{\|\boldsymbol{\mu}_{\underline{\mathsf{x}}|\underline{\mathsf{y}}} - \boldsymbol{\mu}_{\underline{\widehat{\mathsf{x}}}|\underline{\mathsf{y}}}\|_2^2\} \tag{2.27}$$

$$\mathrm{CFID}_{\mathsf{cov}} \triangleq \mathrm{tr}\left[\boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{x}}|\underline{\mathsf{y}}} + \boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\widehat{\mathsf{x}}}|\underline{\mathsf{y}}} - 2\big(\boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{x}}|\underline{\mathsf{y}}}^{1/2}\boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\widehat{\mathsf{x}}}|\underline{\mathsf{y}}}\boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{x}}|\underline{\mathsf{y}}}^{1/2}\big)^{1/2}\right]. \tag{2.28}$$

To verify that (2.28) quantifies the error in $\boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\widehat{\mathsf{x}}}|\underline{\mathsf{y}}}$, notice that (2.28) equals zero when $\boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\widehat{\mathsf{x}}}|\underline{\mathsf{y}}} = \boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{x}}|\underline{\mathsf{y}}}$ and is otherwise positive (by Cauchy Schwarz).

In Table 2.5, we report $\mathrm{CFID}_{\mathsf{mean}}$ and $\mathrm{CFID}_{\mathsf{cov}}$ for the MRI and inpainting experiments, in addition to the total CFID (also shown in Tables 3.2 and 3.5). As before, we computed CFID on three test sets for each experiment, which contained 72, 2 376, and 14 576 samples respectively for MRI, and 1000, 3000, and 30 000 samples respectively for inpainting. Due to the slow sample-generation time of the Langevin/score-based methods [39, 87], we did not have the computational resources to evaluate them on all datasets, and that's why certain table entries are blank.

Table 2.5: The mean and covariance components of CFID, along with the total CFID, for the generative models in the MRI and inpainting experiments. For the MRI experiment, CFID^1 used 72 test samples and $P = 32$, CFID^2 used 2 376 test samples and $P = 8$, and CFID^3 used all 14 576 samples and $P = 1$. For the inpainting experiment, CFID^1 used 1 000 test images and $P = 32$, CFID^2 used 3 000 test and validation images and $P = 8$, and CFID^3 used all 30 000 images and $P = 1$.

Model	$\text{CFID}^1_{\mathsf{mean}}\downarrow$	$\text{CFID}^1_{\mathsf{cov}}\downarrow$	$\text{CFID}^1\downarrow$	$\text{CFID}^2_{\mathsf{mean}}\downarrow$	$\text{CFID}^2_{\mathsf{cov}}\downarrow$	$\text{CFID}^2\downarrow$	$\text{CFID}^3_{\mathsf{mean}}\downarrow$	$\text{CFID}^3_{\mathsf{cov}}\downarrow$	$\text{CFID}^3\downarrow$
					$R = 4$ MRI				
Langevin (Jalal [39])	1.89	3.40	5.29	-	-	-	-	-	-
cGAN (Adler [4])	3.12	3.27	6.39	2.79	1.48	4.27	2.71	1.10	3.82
pscGAN (Ohayon [64])	1.94	2.12	4.06	2.27	1.00	3.27	2.29	0.66	2.95
rcGAN	**0.98**	**2.12**	**3.10**	**0.86**	**0.68**	**1.54**	**0.86**	**0.43**	**1.29**
					$R = 8$ MRI				
Langevin (Jalal [39])	2.61	4.73	7.34	-	-	-	-	-	-
cGAN (Adler [4])	5.00	5.10	10.10	4.16	2.14	6.30	4.09	1.63	5.72
pscGAN (Ohayon [64])	2.73	3.31	6.04	3.07	1.52	4.59	3.30	0.97	4.27
rcGAN	**1.55**	**3.32**	**4.87**	**1.24**	**0.99**	**2.23**	**1.17**	**0.62**	**1.79**
					Inpainting				
Score SDE (Song [87])	0.97	**38.69**	**39.66**	-	-	-	0.90	**4.21**	5.11
CoModGAN (Zhao [111])	0.42	41.21	41.63	0.35	25.39	25.74	0.32	4.98	5.29
rcGAN	**0.32**	39.41	39.73	**0.25**	**22.32**	**22.58**	**0.24**	4.45	**4.69**

For both MRI experiments, Table 2.5 shows our method outperforming the competing methods in both the mean and covariance components of CFID (and thus the total CFID) for all sample sizes. And, in the inpainting experiment, Table 2.5 shows our method outperforming CoModGAN in both the mean and covariance components (and thus the total CFID) for all sample sizes.

For the inpainting experiment, Table 2.5 shows our method outperforming the score-based approach in total CFID on the 3000- and 30 000-sample test sets but not on the 1000-sample test set. However, we now argue that the 1000-sample inpainting experiment is heavily affected by small-sample bias, and therefore untrustworthy. Looking at the mean component of CFID (i.e., $\text{CFID}^1_{\mathsf{mean}}$, $\text{CFID}^2_{\mathsf{mean}}$, and $\text{CFID}^3_{\mathsf{mean}}$) across the inpainting experiments, we see that the values are relatively small and stable with sample size. But looking at the covariance component of CFID (i.e.,

$\text{CFID}^1_{\mathsf{cov}}$, $\text{CFID}^2_{\mathsf{cov}}$, and $\text{CFID}^3_{\mathsf{cov}}$) across the inpainting experiments, we see that the values are large and heavily dependent on sample size. For the 1000-sample inpainting experiment, the total CFID is dominated by the covariance component and thus strongly affected by small-sample bias. Consequently, for the 1000-sample inpainting experiment, the total CFID is not trustworthy.

2.4 Conclusion

In this chapter, we proposed a novel regularization framework for image-recovery cGANs that consists of supervised-ℓ_1 loss plus an appropriately weighted standard-deviation reward. For the case of an independent Gaussian posterior, we proved that our regularization yields generated samples that agree with the true-posterior samples in both mean and covariance. We also established limitations for alternatives based on supervised-ℓ_2 regularization with or without a variance reward. For practical datasets, we proposed a method to auto-tune our standard-deviation reward weight.

Experiments on parallel MRI and large-scale face inpainting showed our proposed method outperforming all cGAN and score-based competitors in CFID, which measures posterior-approximation quality, as well as other metrics like FID, PSNR, SSIM, LPIPS, and DISTS. Furthermore, it generates samples thousands of times faster than Langevin/score-based approaches.

Limitations. We acknowledge several limitations of our work in this chapter. First, while we focused on how to build a fast and accurate posterior sampler, it's not yet clear how to best exploit the resulting posterior samples in each given application. For example, in MRI, where the posterior distribution has the potential to help assess uncertainty in image recovery, it's still not quite clear how to best convey

uncertainty information to radiologists (e.g., they may not gain much from pixel-wise SD images). More work is needed on this front. Second, we acknowledge that, because radiologists are risk-averse, more studies are needed before they will feel comfortable incorporating generative deep-learning-based methods into the clinical workflow. Third, we acknowledge that the visual quality of our $R = 8$ MRI reconstructions falls below clinical standards. Fourth, some caution is needed when interpreting our CFID, FID, and DISTS perceptual metrics because the VGG-16 backbone used to compute them was trained on ImageNet data. Although there is some evidence that the resulting DISTS metric correlates well with radiologists' perceptions [46], there is also evidence that ImageNet-trained features may discard information that is diagnostically relevant in medical imaging [51]. Thus our results need to be validated with a pathology-centric radiologist study before they can be considered relevant to clinical practice.

Chapter 3: Improving Posterior-Sampling cGANs via Principal Component Regularization

In this chapter, we discuss pcaGAN, an extension to the rcGAN method described in Chapter 2. In addition to encouraging correctness in the posterior mean and trace-covariance, pcaGAN also encourages correctness in the K principal components of the $\boldsymbol{y}$-conditional covariance matrix. The content of this chapter appears in "Improving Posterior-Sampling cGANs via Principal Component Regularization," which was published at the 38th annual conference on Neural Information Processing Systems (NeurIPS).

3.1 Background

In this chapter, we build on the rcGAN regularization framework from Chapter 2, which itself builds on the cGAN framework from [4], as detailed in Section 2.1. Furthermore, in Chapter 2, we proposed to regularize the generator $G_{\boldsymbol{\theta}}$ in a way that encourages correct posterior means and trace-covariances, i.e.,

$$\boldsymbol{\mu}_{\widehat{\mathsf{x}}|\mathsf{y}} = \boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}} \qquad \text{for } \boldsymbol{\mu}_{\widehat{\mathsf{x}}|\mathsf{y}} \triangleq \mathbb{E}\{\widehat{\boldsymbol{x}}|\boldsymbol{y}\} \qquad \text{and } \boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}} \triangleq \mathbb{E}\{\boldsymbol{x}|\boldsymbol{y}\} \tag{3.1}$$

$$\operatorname{tr}(\boldsymbol{\Sigma}_{\widehat{\mathsf{x}}|\mathsf{y}}) = \operatorname{tr}(\boldsymbol{\Sigma}_{\mathsf{x}|\mathsf{y}}) \qquad \text{for } \boldsymbol{\Sigma}_{\widehat{\mathsf{x}}|\mathsf{y}} \triangleq \operatorname{Cov}\{\widehat{\boldsymbol{x}}|\boldsymbol{y}\} \qquad \text{and } \boldsymbol{\Sigma}_{\mathsf{x}|\mathsf{y}} \triangleq \operatorname{Cov}\{\boldsymbol{x}|\boldsymbol{y}\}. \tag{3.2}$$

To do this, we replaced $\mathcal{L}_{\mathsf{adv}}(\boldsymbol{\theta}, \boldsymbol{\phi})$ in Section 2.2 with the regularized adversarial loss

$$\mathcal{L}_{\mathsf{rcGAN}}(\boldsymbol{\theta}, \boldsymbol{\phi}) \triangleq \beta_{\mathsf{adv}} \mathcal{L}_{\mathsf{adv}}(\boldsymbol{\theta}, \boldsymbol{\phi}) + \mathcal{L}_{1,P_{\mathsf{rc}}}(\boldsymbol{\theta}) - \beta_{\mathsf{SD}} \mathcal{L}_{\mathsf{SD},P_{\mathsf{rc}}}(\boldsymbol{\theta}), \tag{3.3}$$

which involves the P_{rc}-sample supervised-ℓ_1 loss and standard-deviation (SD) reward terms

$$\mathcal{L}_{1,P}(\boldsymbol{\theta}) \triangleq \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_\mathsf{P},\mathsf{y}} \left\{ \|\boldsymbol{x} - \widehat{\boldsymbol{x}}_{(P)}\|_1 \right\} \tag{3.4}$$

$$\mathcal{L}_{\mathsf{SD},P}(\boldsymbol{\theta}) \triangleq \textstyle\sum_{i=1}^{P} \mathbb{E}_{\mathsf{z}_1,\ldots,\mathsf{z}_\mathsf{P},\mathsf{y}} \left\{ \|\widehat{\boldsymbol{x}}_i - \widehat{\boldsymbol{x}}_{(P)}\|_1 \right\}, \tag{3.5}$$

where typically $P_{\mathsf{rc}} = 2$. Recall, $\{\widehat{\boldsymbol{x}}_i\}$ are the generated samples and $\widehat{\boldsymbol{x}}_{(P)}$ is their P-sample average:

$$\widehat{\boldsymbol{x}}_i \triangleq G_{\boldsymbol{\theta}}(\boldsymbol{z}_i, \boldsymbol{y}) \text{ for } i = 1, \ldots, P_{\mathsf{rc}} \quad \text{and} \quad \widehat{\boldsymbol{x}}_{(P)} \triangleq \tfrac{1}{P} \textstyle\sum_{i=1}^{P} \widehat{\boldsymbol{x}}_i. \tag{3.6}$$

The reward weight β_{SD} in (3.3) is then automatically adjusted to accomplish (3.2) during training.

3.2 Approach

Whereas rcGAN aimed for correctness in the posterior mean and posterior trace-covariance statistics, our proposed pcaGAN *also* aims for correctness in the K principal components of the posterior covariance matrix $\boldsymbol{\Sigma}_{\widehat{\mathsf{x}}|\mathsf{y}}$, where K is user-selectable. To do this, pcaGAN adds two additional regularization terms to the rcGAN objective:

$$\mathcal{L}_{\mathsf{pcaGAN}}(\boldsymbol{\theta}, \boldsymbol{\phi}) \triangleq \mathcal{L}_{\mathsf{rcGAN}}(\boldsymbol{\theta}, \boldsymbol{\phi}) + \beta_{\mathsf{pca}} \mathcal{L}_{\mathsf{evec}}(\boldsymbol{\theta}) + \beta_{\mathsf{pca}} \mathcal{L}_{\mathsf{eval}}(\boldsymbol{\theta}) \tag{3.7}$$

$$\mathcal{L}_{\mathsf{evec}}(\boldsymbol{\theta}) \triangleq -\mathbb{E}_{\mathsf{y}} \left\{ \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_\mathsf{P}|\mathsf{y}} \left\{ \textstyle\sum_{k=1}^{K} [\widehat{\boldsymbol{v}}_k^\top (\boldsymbol{x} - \boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}})]^2 \big| \boldsymbol{y} \right\} \right\} \tag{3.8}$$

$$\mathcal{L}_{\mathsf{eval}}(\boldsymbol{\theta}) \triangleq \mathbb{E}_{\mathsf{y}} \left\{ \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_\mathsf{P}|\mathsf{y}} \left\{ \textstyle\sum_{k=1}^{K} \left(1 - \lambda_k / \widehat{\lambda}_k\right)^2 \big| \boldsymbol{y} \right\} \right\}. \tag{3.9}$$

Here, $\{(\widehat{\boldsymbol{v}}_k, \widehat{\lambda}_k)\}_{k=1}^K$ denote the principal eigenvectors and eigenvalues of the $\boldsymbol{\theta}$-dependent generated covariance matrix $\boldsymbol{\Sigma}_{\widehat{\mathsf{x}}|\mathsf{y}}$ and $\{(\boldsymbol{v}_k, \lambda_k)\}_{k=1}^K$ denote the principal eigenvectors

and eigenvalues of the true covariance matrix $\mathbf{\Sigma}_{\mathsf{x}|\mathsf{y}}$. Because (3.8) is the classical PCA objective [40], minimizing $\mathcal{L}_{\mathsf{evec}}(\boldsymbol{\theta})$ over $\boldsymbol{\theta}$ will drive the generated principal eigenvector $\widehat{\boldsymbol{v}}_k$ towards the true principal eigenvector $\boldsymbol{v}_k$ for each $k = 1, \ldots, K$. Likewise, minimizing $\mathcal{L}_{\mathsf{eval}}(\boldsymbol{\theta})$ over $\boldsymbol{\theta}$ will drive the generated principal eigenvalue $\widehat{\lambda}_k$ towards the true principal eigenvalue λ_k for each $k = 1, \ldots, K$. Based on our experiments, putting $\widehat{\lambda}_k$ in the denominator works better than the numerator and the squared error in (3.9) works better than an absolute value.

Algorithm 1 pcaGAN generator-training iteration

Require: number of estimated eigen-components K, number of samples used for eigenvector and eigenvalue regularization P_{pca}, number of samples used for rcGAN regularization P_{rc}, epoch at which to involve eigenvector regularization E_{evec}, epoch at which to involve eigenvalue regularization E_{eval}, lazy update period M, adversarial loss weight β_{adv}, std regularization weight β_{SD}, eigenvector and eigenvalue regularization weight β_{pca}, training batch $\{(\boldsymbol{x}_b, \boldsymbol{y}_b)\}_{b=1}^{B}$, current model parameters $\boldsymbol{\theta}$, current training epoch e, the current training step s

$\mathcal{L}(\boldsymbol{\theta}) \leftarrow 0$

for $b = 1, \ldots, B$ **do**
  $\boldsymbol{z}_i \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ for $i = 1, \ldots, P_{\mathsf{rc}}$
  $\widehat{\boldsymbol{x}}_i \leftarrow G_{\boldsymbol{\theta}}(\boldsymbol{z}_i, \boldsymbol{y}_b)$ for $i = 1, \ldots, P_{\mathsf{rc}}$
  $\mathcal{L}(\boldsymbol{\theta}) \leftarrow \mathcal{L}(\boldsymbol{\theta}) - \beta_{\mathsf{adv}} \sum_{i=1}^{P_{\mathsf{rc}}} D_{\boldsymbol{\phi}}(\widehat{\boldsymbol{x}}_i, \boldsymbol{y}_b)$
  $\widehat{\boldsymbol{x}}_{(P_{\mathsf{rc}})} = \frac{1}{P_{\mathsf{rc}}} \sum_{i=1}^{P_{\mathsf{rc}}} \widehat{\boldsymbol{x}}_i$
  $\mathcal{L}(\boldsymbol{\theta}) \leftarrow \mathcal{L}(\boldsymbol{\theta}) + \|\boldsymbol{x}_b - \widehat{\boldsymbol{x}}_{(P_{\mathsf{rc}})}\|_1 - \beta_{\mathsf{SD}} \sum_{i=1}^{P_{\mathsf{rc}}} \mathbb{E}_{\mathsf{z}_1,\ldots,\mathsf{z}_{\mathsf{P}},\mathsf{y}} \left\{ \|\widehat{\boldsymbol{x}}_i - \widehat{\boldsymbol{x}}_{(P_{\mathsf{rc}})}\|_1 \right\}$

  if $e \geq E_{\mathsf{evec}}$ and $s \bmod M = 0$ **then**
    $\boldsymbol{z}_j \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ for $j = 1, \ldots, P_{\mathsf{pca}}$
    $\widehat{\boldsymbol{x}}_j \leftarrow G_{\boldsymbol{\theta}}(\boldsymbol{z}_j, \boldsymbol{y}_b)$ for $j = 1, \ldots, P_{\mathsf{pca}}$
    $\widehat{\boldsymbol{\mu}} \leftarrow \texttt{StopGrad}(\frac{1}{P_{\mathsf{pca}}} \sum_{j=1}^{P_{\mathsf{pca}}} \widehat{\boldsymbol{x}}_j)$
    $\widehat{\boldsymbol{U}} \widehat{\boldsymbol{S}} \widehat{\boldsymbol{V}}^\top \leftarrow \mathrm{SVD}([\widehat{\boldsymbol{x}}_1 - \widehat{\boldsymbol{\mu}}, \ldots, \widehat{\boldsymbol{x}}_{P_{\mathsf{pca}}} - \widehat{\boldsymbol{\mu}}]^\top)$
    $\widehat{\boldsymbol{v}}_k \leftarrow [\widehat{\boldsymbol{V}}]_{:,k}$ for $k = 1, \ldots, K$
    $\mathcal{L}(\boldsymbol{\theta}) \leftarrow \mathcal{L}(\boldsymbol{\theta}) - \beta_{\mathsf{pca}} \sum_{k=1}^{K} [\widehat{\boldsymbol{v}}_k^\top (\boldsymbol{x}_b - \widehat{\boldsymbol{\mu}})]^2$
  end if

  if $e \geq E_{\mathsf{eval}}$ and $s \bmod M = 0$ **then**
    $\widehat{\lambda}_k \leftarrow [\widehat{\boldsymbol{S}}]_{kk}^2$ for $k = 1, \ldots, K$
    $\widetilde{\boldsymbol{X}} \leftarrow [\boldsymbol{x}_b - \widehat{\boldsymbol{\mu}}, \widehat{\boldsymbol{x}}_1 - \widehat{\boldsymbol{\mu}}, \widehat{\boldsymbol{x}}_2 - \widehat{\boldsymbol{\mu}}, \ldots, \widehat{\boldsymbol{x}}_{P_{\mathsf{pca}}} - \widehat{\boldsymbol{\mu}}]^\top$
    $\mathcal{L}(\boldsymbol{\theta}) \leftarrow \mathcal{L}(\boldsymbol{\theta}) + \beta_{\mathsf{pca}} \sum_{k=1}^{K} \left(1 - \frac{1}{\widehat{\lambda}_k} \texttt{StopGrad}(\frac{1}{P_{\mathsf{pca}}+1} \|\widehat{\boldsymbol{v}}_k^\top \widetilde{\boldsymbol{X}}\|^2)\right)^2$
  end if
end for

$\boldsymbol{\theta} \leftarrow \mathrm{Adam}(\boldsymbol{\theta}, \nabla \mathcal{L}(\boldsymbol{\theta}))$

In practice, the expectations in (3.8)-(3.9) are replaced by sample averages over the training data. In the typical case that the training data includes only a single image

$\boldsymbol{x}_t$ for each measurement vector $\boldsymbol{y}_t$, the quantities $\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}$ and $\{\lambda_k\}$ in (3.8)-(3.9) are unknown and non-trivial to estimate for each $\boldsymbol{y}_t$. Hence, when training the pcaGAN, we approximate them with learned quantities. This must be done carefully, however. For example, if $\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}$ in (3.8) was simply replaced by the $\boldsymbol{\theta}$-dependent quantity $\boldsymbol{\mu}_{\widehat{\mathsf{x}}|\mathsf{y}}$, then minimizing $\mathcal{L}_{\mathsf{evec}}(\boldsymbol{\theta})$ over $\boldsymbol{\theta}$ would encourage $\boldsymbol{\mu}_{\widehat{\mathsf{x}}|\mathsf{y}}$ to become overly large in order to drive $\mathcal{L}_{\mathsf{evec}}(\boldsymbol{\theta})$ to a large negative value.

Algorithm 1 details our proposed approach to training the pcaGAN. In particular, it describes the steps used to perform a single update of the generator parameters $\boldsymbol{\theta}$ based on the training batch $\{(\boldsymbol{x}_b, \boldsymbol{y}_b)\}_{b=1}^{B}$. Before diving into the details, we offer a brief summary of Algorithm 1. For the initial epochs, the rcGAN objective $\mathcal{L}_{\mathsf{rcGAN}}$ alone is optimized, which allows the generated posterior mean $\boldsymbol{\mu}_{\widehat{\mathsf{x}}|\mathsf{y}}$ to converge to the vicinity of $\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}$. Starting at E_{evec} epochs, the $\mathcal{L}_{\mathsf{evec}}(\boldsymbol{\theta})$ regularization from (3.8) is added, but with $\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}$ approximated as $\texttt{StopGrad}(\boldsymbol{\mu}_{\widehat{\mathsf{x}}|\mathsf{y}})$. The use of `StopGrad` forces $\mathcal{L}_{\mathsf{evec}}(\boldsymbol{\theta})$ to be minimized by manipulating the eigenvectors $\{\widehat{\boldsymbol{v}}_k\}_{k=1}^{K}$ and not the generated posterior mean $\boldsymbol{\mu}_{\widehat{\mathsf{x}}|\mathsf{y}}$. These eigenvectors are computed using an SVD of centered approximate-posterior samples. To reduce the computational burden imposed by this SVD, a "lazy regularization" [45] approach is adopted, which computes $\mathcal{L}_{\mathsf{evec}}(\boldsymbol{\theta})$ only once every M training steps. Training proceeds in this manner until the eigenvectors $\{\widehat{\boldsymbol{v}}_k\}$ converge. Starting at E_{eval} epochs, the $\mathcal{L}_{\mathsf{eval}}(\boldsymbol{\theta})$ regularization from (3.9) is added, but with λ_k approximated as

$$\lambda_k \approx \texttt{StopGrad}\big(\tfrac{1}{1+P_{\mathsf{pca}}}\big\|\widehat{\boldsymbol{v}}_k^\top[\boldsymbol{x}_b - \boldsymbol{\mu}_{\widehat{\mathsf{x}}|\mathsf{y}}, \widehat{\boldsymbol{x}}_1 - \boldsymbol{\mu}_{\widehat{\mathsf{x}}|\mathsf{y}}, \ldots, \widehat{\boldsymbol{x}}_{P_{\mathsf{pca}}} - \boldsymbol{\mu}_{\widehat{\mathsf{x}}|\mathsf{y}}]\big\|^2\big), \tag{3.10}$$

where `StopGrad` is used so that the optimization focuses on $\{\widehat{\lambda}_k\}$. The rationale behind (3.10) is that, when $\widehat{\boldsymbol{v}}_k = \boldsymbol{v}_k$ and $\boldsymbol{\mu}_{\widehat{\mathsf{x}}|\mathsf{y}} = \boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}$, the terms $[\widehat{\boldsymbol{v}}_k^\top(\boldsymbol{x}_b - \boldsymbol{\mu}_{\widehat{\mathsf{x}}|\mathsf{y}})]^2$ and $[\widehat{\boldsymbol{v}}_k^\top(\widehat{\boldsymbol{x}}_j - \boldsymbol{\mu}_{\widehat{\mathsf{x}}|\mathsf{y}})]^2$ $\forall j$ all equal λ_k in $\boldsymbol{y}_b$-conditional expectation. This expectation is

approximated using a $(1 + P_{\mathsf{pca}})$-term sample average in (3.10) via the squared norm. The eigenvalues $\{\widehat{\lambda}_k\}$ in (3.9) are computed using the previously described SVD and the regularization schedule is again M-lazy.

We now provide additional details on Algorithm 1. After the loss is initialized in line 1, the following steps are executed for each measurement vector $\boldsymbol{y}_b$ in the batch. First, approximate posterior samples $\{\widehat{\boldsymbol{x}}_i\}_{i=1}^{P_{\mathsf{rc}}}$ are generated in line 5, where $P_{\mathsf{rc}} = 2$ as done in Chapter 2. Using these samples, the adversarial component of the loss is added in line 6 and the rcGAN regularization is added in line 8. Starting at epoch E_{evec}, lines 11-16 are executed whenever the training iteration is a multiple of M. Nominally, E_{evec} is set where the validation PSNR of $\widehat{\boldsymbol{\mu}}$ (an empirical approximation of $\boldsymbol{\mu}_{\widehat{\mathsf{x}}|\mathsf{y}}$) stabilizes and $M = 100$. Within those lines, samples $\{\widehat{\boldsymbol{x}}_j\}_{j=1}^{P_{\mathsf{pca}}}$ are generated in line 12 (where nominally $P_{\mathsf{pca}} = 10K$), their sample mean is computed in line 13, and the SVD of the centered samples is computed in line 14. The top K right singular vectors are then extracted in line 15 in order to construct the $\mathcal{L}_{\mathsf{evec}}(\boldsymbol{\theta})$ regularization, which is added to the overall generator loss $\mathcal{L}(\boldsymbol{\theta})$ in line 16. Starting at epoch E_{eval}, where nominally $E_{\mathsf{eval}} = E_{\mathsf{evec}} + 25$, lines 20-22 are executed whenever the training iteration is a multiple of M. In line 20, the top K eigenvalues $\{\widehat{\lambda}_k\}$ are constructed from the previously computed singular values and, in line 22, the $\mathcal{L}_{\mathsf{eval}}(\boldsymbol{\theta})$ regularization is constructed and added to the overall training loss. The construction of $\mathcal{L}_{\mathsf{eval}}(\boldsymbol{\theta})$ was previously described around (3.10). Finally, once the losses for all batch elements have been incorporated, the gradient $\nabla\mathcal{L}(\boldsymbol{\theta})$ is computed using back-propagation and a gradient-descent step of $\boldsymbol{\theta}$ is performed using the Adam optimizer [50] in line 26.

3.3 Numerical Experiments

We now present experiments with Gaussian data, MNIST denoising, MRI, and FFHQ face inpainting. Additional implementation and training details for each experiment are provided in Appendix D.3.

3.3.1 Recovering Synthetic Gaussian Data

Here our goal is to recover $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathsf{x}}, \boldsymbol{\Sigma}_{\mathsf{x}}) \in \mathbb{R}^d$ from $\boldsymbol{y} = \boldsymbol{M}\boldsymbol{x} + \boldsymbol{w} \in \mathbb{R}^d$, where $\boldsymbol{M}$ masks $\boldsymbol{x}$ at even indices and noise $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$ is independent of $\boldsymbol{x}$ with $\sigma^2 = 0.001$. Since $\boldsymbol{x}$ and $\boldsymbol{y}$ are jointly Gaussian, the posterior is Gaussian with $\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}} = \boldsymbol{\mu}_{\mathsf{x}} + \boldsymbol{\Sigma}_{\mathsf{x}\mathsf{y}}\boldsymbol{\Sigma}_{\mathsf{y}}^{-1}(\boldsymbol{y} - \boldsymbol{\mu}_{\mathsf{y}})$ and $\boldsymbol{\Sigma}_{\mathsf{x}|\mathsf{y}} = \boldsymbol{\Sigma}_{\mathsf{x}} - \boldsymbol{\Sigma}_{\mathsf{x}\mathsf{y}}\boldsymbol{\Sigma}_{\mathsf{y}}^{-1}\boldsymbol{\Sigma}_{\mathsf{y}\mathsf{x}}$, where $\boldsymbol{\mu}_{\mathsf{x}}, \boldsymbol{\mu}_{\mathsf{y}}, \boldsymbol{\Sigma}_{\mathsf{x}}, \boldsymbol{\Sigma}_{\mathsf{y}}$ are marginal and $\boldsymbol{\Sigma}_{\mathsf{x}\mathsf{y}}, \boldsymbol{\Sigma}_{\mathsf{y}\mathsf{x}}$ are joint statistics.

We generate random $\boldsymbol{\mu}_{\mathsf{x}} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and $\boldsymbol{\Sigma}_{\mathsf{x}}$ with half-normal eigenvalues λ_k (see additional details in App. D.3.1), and consider a sequence of problem sizes $d = 10, 20, 30, \ldots, 100$. For each d, we generate 70 000 training, 20 000 validation, and 10 000 test samples. The generator and discriminator are simple multilayer perceptrons (see App. D.3.2) trained for 100 epochs with $K = d$, $E_{\mathsf{evec}} = 10$, $\beta_{\mathsf{adv}} = 10^{-5}$, and $\beta_{\mathsf{pca}} = 10^{-2}$.

Competitors. We compare the proposed pcaGAN to rcGAN and NPPC [61]. rcGAN uses the same generator and discriminator architectures as pcaGAN and is trained according to (3.3) with $\beta_{\mathsf{adv}} = 10^{-5}$ and $P_{\mathsf{rc}} = 2$. For NPPC, we use the authors' implementation [62] with $K = d$ and some minor modifications to work with vector data. To evaluate performance, we use the Wasserstein-2 (W2) distance between $p_{\mathsf{x}|\mathsf{y}}$ and $\widehat{p_{\mathsf{x}|\mathsf{y}}}$, which in the Gaussian case reduces to

$$\mathcal{W}_2(p_{\mathsf{x}|\mathsf{y}}, \widehat{p_{\mathsf{x}|\mathsf{y}}}) = \|\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}} - \widehat{\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}}\|_2^2 + \operatorname{tr}\left[\boldsymbol{\Sigma}_{\mathsf{x}|\mathsf{y}} + \widehat{\boldsymbol{\Sigma}_{\mathsf{x}|\mathsf{y}}} - 2(\boldsymbol{\Sigma}_{\mathsf{x}|\mathsf{y}}^{1/2}\widehat{\boldsymbol{\Sigma}_{\mathsf{x}|\mathsf{y}}}\boldsymbol{\Sigma}_{\mathsf{x}|\mathsf{y}}^{1/2})^{1/2}\right]. \tag{3.11}$$

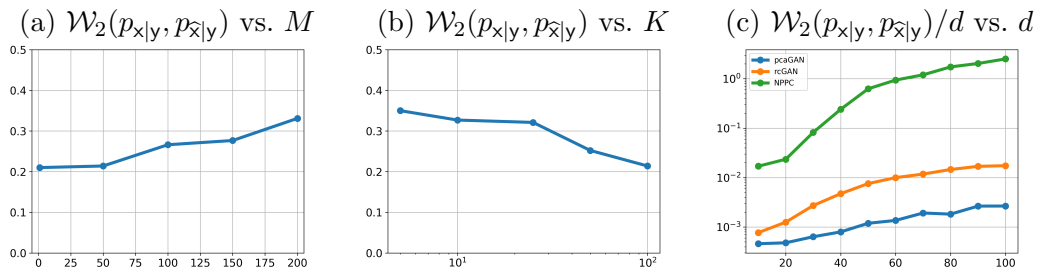


Figure 3.1: Gaussian experiment. Wasserstein-2 distance versus (a) lazy update period M for pcaGAN with $d = 100 = K$, (b) estimated eigen-components K for pcaGAN with $d = 100$ and $M = 100$, and (c) problem dimension d for all methods under test with $K = d$ and $M = 100$.

For the cGANs, we compute $\widehat{\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}}$ and $\widehat{\boldsymbol{\Sigma}_{\mathsf{x}|\mathsf{y}}}$ empirically from $10d$ samples, while for NPPC we use the conditional mean, eigenvalues, and eigenvectors returned by the approach.

Results. Figure 3.1a examines the impact of the lazy update period M on pcaGAN's W2 distance at $d = 100$ with $K = d$. Based on this figure, to balance performance with training overhead, we set $M = 100$ for all future experiments. Figure 3.1b examines the impact of K on W2 distance for the pcaGAN with $d = 100$. It shows that using $K < d$ causes a relatively mild increase in W2 distance, as expected due to the half-normal distribution on the true eigenvalues λ_k. Figure 3.1c shows that the proposed pcaGAN outperforms rcGAN and NPPC in W2 distance for all problem sizes d.

3.3.2 MNIST Denoising

Now our goal is to recover an MNIST digit $\boldsymbol{x} \in [0,1]^{28\times 28}$ from noisy measurements $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{w}$ with $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. We randomly split the MNIST training fold into 50 000 training and 10 000 validation images, and we use the entire MNIST test set

for testing. For pcaGAN and rcGAN, we use a U-Net [74] generator and the encoder portion of the same U-Net followed by one dense layer as the discriminator. pcaGAN was trained for 125 epochs with $E_{\mathsf{evec}} = 25$, $\beta_{\mathsf{adv}} = 10^{-5}$, $\beta_{\mathsf{pca}} = 10^{-1}$, and $K \in \{5, 10\}$.

Competitors. We again compare the proposed pcaGAN to rcGAN and NPPC. For rcGAN we used the same generator and discriminator architectures as pcaGAN and trained according to (3.3) with $\beta_{\mathsf{adv}} = 10^{-5}$ and $P_{\mathsf{rc}} = 2$. For NPPC, we used the authors' MNIST implementation from [62].

Following the NPPC paper [61], we evaluate performance using root MSE (rMSE) $\mathbb{E}_{\mathsf{x},\mathsf{y}}\{\|\boldsymbol{x} - \widehat{\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}}\|_2\}$ and Residual Error Magnitude (REM_5) $\mathbb{E}_{\mathsf{x},\mathsf{y}}\left\{\left\|(\boldsymbol{I} - \widehat{\boldsymbol{V}}_5\widehat{\boldsymbol{V}}_5^\top)\boldsymbol{e}\right\|_2\right\}$, where $\boldsymbol{e} = \boldsymbol{x} - \widehat{\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}}$ and $\widehat{\boldsymbol{V}}_5$ is an $28^2 \times 5$ matrix whose kth column equals the kth principal eigenvector $\widehat{\boldsymbol{v}}_k$. For the cGANs, we use $\widehat{\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}} = \widehat{\boldsymbol{x}}_{(P)}$ and compute $\{\widehat{\boldsymbol{v}}_k\}$ from the SVD of a matrix of centered samples $\{\widehat{\boldsymbol{x}}_i\}_{i=1}^P$, both with $P = 100$. For NPPC, we use the conditional means and eigenvectors returned by the approach. For performance evaluation, we also consider Conditional Fréchet Inception Distance (CFID) [80] with InceptionV3 features. CFID is analogous to Fréchet Inception Distance (FID) [33] but applies to conditional distributions (see Appendix D.1 for more details).

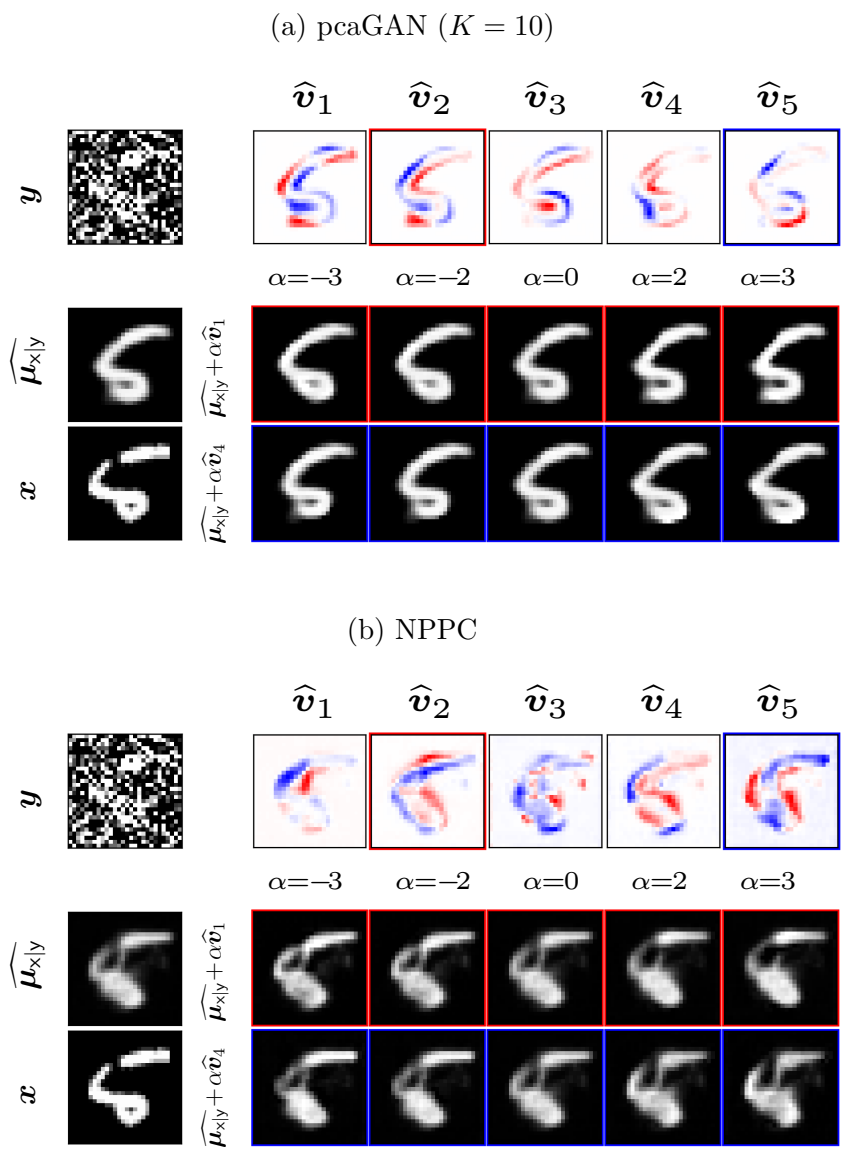


Figure 3.2: For (a) pcaGAN and (b) NPPC, this figure shows the true image $\boldsymbol{x}$, noisy measurements $\boldsymbol{y}$, the conditional mean $\widehat{\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}}$, principal eigenvectors $\{\widehat{\boldsymbol{v}}_k\}$, and two perturbations of $\widehat{\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}}$.

Results. Table 3.1 shows rMSE, REM_5, CFID, and the reconstruction time for a batch of 128 images on the test fold. (NPPC does not generate image samples and

Table 3.1: Average MNIST denoising results.

Model	rMSE↓	REM_5↓	CFID↓	Time(128)↓
NPPC (Nehme et al. [61])	**3.94**	3.63	–	**112 ms**
rcGAN	4.04	3.41	63.44	118 ms
pcaGAN ($K = 5$)	4.02	3.31	61.48	118 ms
pcaGAN ($K = 10$)	4.02	**3.25**	**60.16**	118 ms

so CFID does not apply.) The table shows that the proposed pcaGAN wins in all metrics, except for rMSE where NPPC wins.

This is not surprising because NPPC computes $\widehat{\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}}$ using a dedicated network trained to minimize MSE loss. NPPC also generates its eigenvectors slightly quicker than pcaGAN generates samples. Table 3.1 also shows that pcaGAN performance improves as K increases from 5 to 10, despite the fact that REM_5 uses only the top 5 eigenvectors. Figure 3.2 shows examples of the 5 principal eigenvectors and posterior mean learned by pcaGAN and NPPC. The eigenvectors of pcaGAN are more structured and less noisy than those of NPPC. Figure 3.2 also shows $\widehat{\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}} + \alpha \boldsymbol{v}_k$ for $\alpha \in [-3, 3]$ and $k \in \{1, 4\}$. Additional figures can be found in App. E.2.1.

3.3.3 Accelerated MRI

We now consider accelerated MRI, where the goal is to recover a complex-valued multicoil image $\boldsymbol{x}$ from masked frequency-domain (i.e., "k-space") measurements $\boldsymbol{y}$. To build the image data $\{\boldsymbol{x}_t\}$, we follow the approach in Chapter 2, which uses the first 8 slices of all fastMRI [102] T2 brain volumes with at least 8 coils, crops to 384×384 pixels, and compresses to 8 virtual coils [108]. This yields 12 200 training, 2 376 testing, and 784 validation images. To create each $\boldsymbol{y}_t$, we transform $\boldsymbol{x}_t$ to the

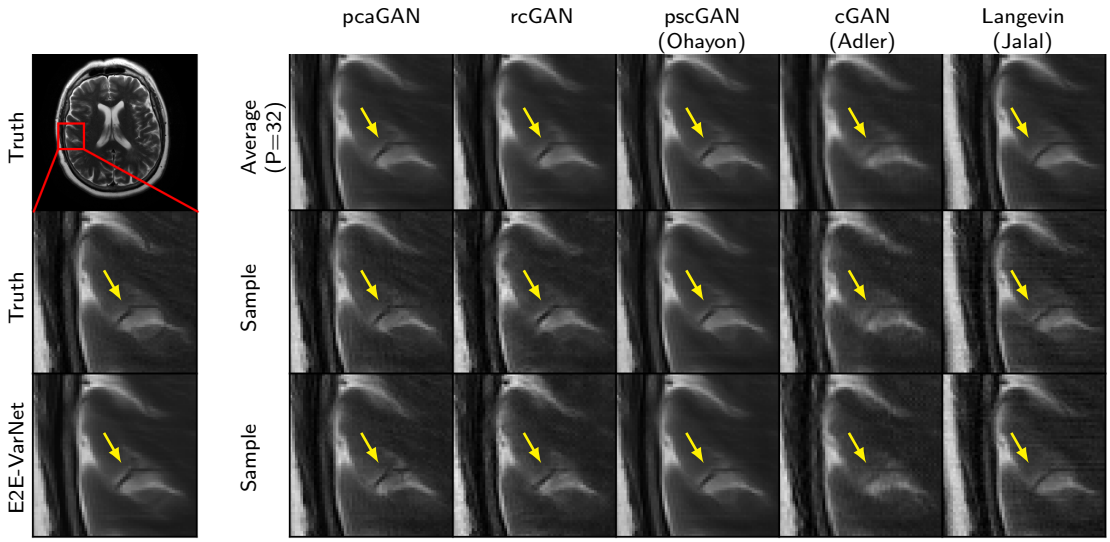


Figure 3.3: Example MRI recoveries at $R = 8$. Arrows highlight meaningful variations.

k-space, subsample using the Cartesian GRO mask [41] at accelerations $R = 4$ and $R = 8$, and transform the zero-filled k-space measurements back to the image domain.

We train pcaGAN for 100 epochs with $K = 1$, $E_{\mathsf{evec}} = 25$, $\beta_{\mathsf{adv}} = 10^{-5}$, and $\beta_{\mathsf{pca}} = 10^{-2}$ and select the final model using validation CFID computed with VGG-16 features.

Competitors. We compare the proposed pcaGAN to rcGAN, pscGAN [64], Adler & Öktem's cGAN [4], the Langevin approach [39], and the E2E-VarNet [89]. All cGANs use the generator and discriminator architectures as rcGAN and enforce data-consistency [81]. For rcGAN and the Langevin approach, we did not modify the authors' implementation from [38] except to use the GRO sampling mask. For E2E-VarNet, we use the GRO mask, hyperparameters, and training procedure from [39].

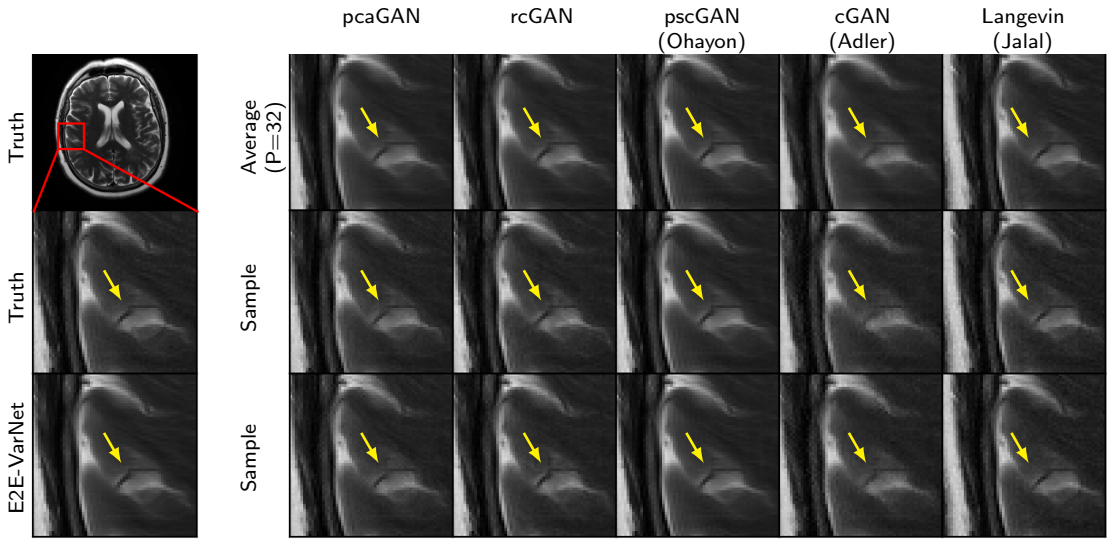


Figure 3.4: Example MRI recoveries at $R = 4$. Arrows highlight meaningful variations.

Following Chapter 2, we convert the multicoil outputs $\widehat{\boldsymbol{x}}_i$ to complex-valued images using SENSE-based coil combining [70] with ESPIRiT-estimated [94] coil sensitivity maps, and compute performance on magnitude images. All feature-based metrics (CFID, FID, LPIPS, DISTS) were computed with AlexNet features to show that pcaGAN does not overfit to the VGG-16 features used for validation. It was shown in [3] that image-quality metrics computed using ImageNet-trained feature generators like AlexNet and VGG-16 perform comparably to metrics computed using MRI-trained feature generators in terms of correlation with radiologists' scores.

Results. Table 3.2 shows CFID, FID, APSD $\triangleq (\frac{1}{P}\sum_{i=1}^{P}\frac{1}{N}\|\widehat{\boldsymbol{x}}_{(P)} - \widehat{\boldsymbol{x}}_i\|^2)^{1/2}$, and 4-sample generation time for the methods under test. Due to its slow sample-generation time, we evaluate the CFID, FID, and APSD of the Langevin technique [39] using the 72-image test from Chapter 2. But due to the bias of CFID at small sample sizes [80], we evaluate the other methods using all 2 376 test images (CFID^2) and again

Table 3.2: Average MRI results at acceleration $R \in \{4, 8\}$

	$R=4$						$R=8$					
Model	CFID^1↓	CFID^2↓	CFID^3↓	FID↓	APSD	Time (4)↓	CFID^1↓	CFID^2↓	CFID^3↓	FID↓	APSD	Time (4)↓
E2E-VarNet (Sriram et al. [89])	16.08	13.07	10.26	38.88	0.0	310ms	36.86	29.90	23.82	44.04	0.0	316ms
Langevin (Jalal et al. [39])	33.05	-	-	31.43	5.9e-6	14 min	48.59	-	-	52.62	7.6e-6	14 min
cGAN (Adler & Öktem [4])	19.00	12.05	7.00	29.77	3.9e-6	**217 ms**	59.94	40.24	26.10	31.81	7.7e-6	**217 ms**
pscGAN (Ohayon et al. [64])	13.74	10.56	7.53	37.28	7.2e-8	**217 ms**	39.67	31.81	24.06	43.39	7.7e-7	**217 ms**
rcGAN	9.71	5.27	1.69	25.62	3.8e-6	**217 ms**	24.04	13.20	3.83	28.43	7.6e-6	**217 ms**
pcaGAN	**8.78**	**4.48**	**1.29**	**25.02**	4.4e-6	**217 ms**	**21.65**	**11.47**	**3.21**	**28.35**	6.5e-6	**217 ms**

Table 3.3: Average PSNR, SSIM, LPIPS, and DISTS of $\widehat{\boldsymbol{x}}_{(P)}$ versus P for MRI at $R = 8$

	PSNR↑						SSIM↑					
Model	P=1	P=2	P=4	P=8	P=16	P=32	P=1	P=2	P=4	P=8	P=16	P=32
E2E-VarNet (Sriram et al. [89])	**36.49**	-	-	-	-	-	0.9220	-	-	-	-	-
Langevin (Jalal et al. [39])	32.17	32.83	33.45	33.74	33.83	33.90	0.8725	0.8919	0.9031	0.9091	0.9120	0.9137
cGAN (Adler & Öktem [4])	31.31	32.31	32.92	33.26	33.42	33.51	0.8865	0.9045	0.9103	0.9111	0.9102	0.9095
pscGAN (Ohayon et al. [64])	34.89	34.90	34.90	34.90	34.91	34.92	0.9222	0.9217	0.9213	0.9211	0.9211	0.9210
rcGAN	32.32	33.67	34.53	35.01	35.27	35.42	0.9030	0.9199	0.9252	0.9257	0.9251	0.9246
pcaGAN	33.28	34.47	35.20	35.61	35.82	35.94	0.9136	0.9257	**0.9283**	0.9275	0.9262	0.9253
	LPIPS↓						DISTS↓					
Model	P=1	P=2	P=4	P=8	P=16	P=32	P=1	P=2	P=4	P=8	P=16	P=32
E2E-VarNet (Sriram et al. [89])	0.0575	-	-	-	-	-	0.1253	-	-	-	-	-
Langevin (Jalal et al. [39])	0.0769	0.0619	0.0579	0.0589	0.0611	0.0611	0.1341	0.1136	0.1086	0.1119	0.1175	0.1212
cGAN (Adler & Öktem [4])	0.0698	0.0614	0.0623	0.0667	0.0704	0.0727	0.1407	0.1262	0.1252	0.1291	0.1334	0.1361
pscGAN (Ohayon et al. [64])	0.0532	0.0536	0.0539	0.0540	0.0534	0.0540	0.1128	0.1143	0.1151	0.1155	0.1157	0.1158
rcGAN	0.0418	0.0379	0.0421	0.0476	0.0516	0.0539	0.0906	0.0877	0.0965	0.1063	0.1135	0.1177
pcaGAN	0.0358	**0.0344**	0.0391	0.0442	0.0479	0.0499	0.0804	**0.0799**	0.0920	0.1026	0.1099	0.1144

using all 14 576 training and test images (CFID^3). Table 3.2 shows that pcaGAN yields better CFID and FID than the competitors. All cGANs generated samples 3–4 orders-of-magnitude faster than the Langevin approach [39].

Table 3.3 shows PSNR, SSIM, LPIPS [107], and DISTS [27] for the P-sample average $\widehat{\boldsymbol{x}}_{(P)}$ at $P \in \{1, 2, 4, 8, 16, 32\}$ and $R = 8$. It has been shown that DISTS correlates particularly well with radiologist scores [47]. The E2E-VarNet achieves the best PSNR, but the proposed cGAN achieves the best LPIPS and DISTS when $P = 2$

Table 3.4: Average PSNR, SSIM, LPIPS, and DISTS of $\widehat{\boldsymbol{x}}_{(P)}$ versus P for $R = 4$ MRI

	PSNR↑						SSIM↑					
Model	P=1	P=2	P=4	P=8	P=16	P=32	P=1	P=2	P=4	P=8	P=16	P=32
E2E-VarNet (Sriram et al. [89])	**39.93**	-	-	-	-	-	**0.9641**	-	-	-	-	-
Langevin (Jalal et al. [39])	36.04	37.02	37.65	37.99	38.17	38.27	0.8989	0.9138	0.9218	0.9260	0.9281	0.9292
cGAN (Adler & Öktem [4])	35.63	36.64	37.24	37.56	37.73	37.82	0.9330	0.9445	0.9478	0.9480	0.9477	0.9473
pscGAN (Ohayon et al. [64])	39.44	39.46	39.46	39.47	39.47	39.47	0.9558	0.9546	0.9539	0.9535	0.9533	0.9532
rcGAN	36.96	38.14	38.84	39.24	39.44	39.55	0.9440	0.9526	0.9544	0.9542	0.9537	0.9533
pcaGAN	37.32	38.43	39.11	39.47	39.67	39.77	0.9463	0.9541	0.9557	0.9553	0.9546	0.9542
	LPIPS↓						DISTS↓					
Model	P=1	P=2	P=4	P=8	P=16	P=32	P=1	P=2	P=4	P=8	P=16	P=32
E2E-VarNet (Sriram et al. [89])	0.0316	-	-	-	-	-	0.0859	-	-	-	-	-
Langevin (Jalal et al. [39])	0.0545	0.0394	0.0336	0.0320	0.0317	0.0316	0.1116	0.0921	0.0828	0.0793	0.0781	0.0777
cGAN (Adler & Öktem [4])	0.0285	0.0255	0.0273	0.0298	0.0316	0.0327	0.0972	0.0857	0.0878	0.0930	0.0967	0.0990
pscGAN (Ohayon et al. [64])	0.0245	0.0247	0.0248	0.0249	0.0249	0.0249	0.0767	0.0790	0.0801	0.0807	0.0810	0.0811
rcGAN	0.0175	0.0164	0.0188	0.0216	0.0235	0.0245	0.0546	0.0563	0.0667	0.0755	0.0809	0.0837
pcaGAN	0.0164	**0.0159**	0.0188	0.0214	0.0231	0.0242	**0.0542**	0.0548	0.0662	0.0754	0.0811	0.0843

and the best SSIM when $P = 8$. This P-dependence is related to the perception-distortion trade-off [11] and consistent with that reported in Chapter 2.

Table 3.4 shows PSNR, SSIM, LPIPS [107], and DISTS [27] for the P-sample average $\widehat{\boldsymbol{x}}_{(P)}$ at $P \in \{1, 2, 4, 8, 16, 32\}$ for $R = 4$. In this case, the E2E-VarNet attains the best PSNR and SSIM, while pcaGAN performs best in LPIPS and DISTS when $P = 2$ and $P = 1$, respectively. These results, in conjunction with the $R = 8$ results discussed in Sec. 3.3.3, show that pcaGAN yields a notable improvement over rcGAN in all metrics.

Figure 3.3 shows zoomed versions of two recoveries $\widehat{\boldsymbol{x}}_i$ and the sample average $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 32$ at $R = 8$. Similarly, Figure 3.4 shows zoomed versions of two recoveries $\widehat{\boldsymbol{x}}_i$ and the sample average $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 32$ at $R = 4$. Appendices E.2.2 and E.2.3 show additional plots of $\widehat{\boldsymbol{x}}_{(P)}$ that visually demonstrate the perception-distortion trade-off.

Table 3.5: Average FFHQ inpainting results.

Model	CFID↓	FID↓	LPIPS↓	Time (40 samples)↓
DPS (Chung et al. [17])	7.26	2.00	0.1245	14 min
DDNM (Wang et al. [96])	11.30	3.63	0.1409	30 s
DDRM (Kawar et al. [48])	13.17	5.36	0.1587	5 s
pscGAN (Ohayon et al. [64])	18.44	8.40	0.1716	**325 ms**
CoModGAN (Zhao et al. [111])	7.85	2.23	0.1290	**325 ms**
rcGAN	7.51	2.12	0.1262	**325 ms**
pcaGAN	**7.08**	**1.98**	**0.1230**	**325 ms**

3.3.4 Large-Scale Inpainting

Our final goal is to inpaint a face image with a large randomly generated masked region. For this task, we use 256×256 FFHQ face images [44] and the mask generation procedure from [111]. We randomly split the FFHQ training fold into 45 000 training and 5 000 validation images, and we use the remaining 20 000 images for testing.

For pcaGAN, we use CoModGAN's [111] generator and discriminator architecture and train for 100 epochs using $K = 2$, $E_{\mathsf{evec}} = 25$, $\beta_{\mathsf{adv}} = 5 \times 10^{-3}$, and $\beta_{\mathsf{pca}} = 10^{-3}$.

Competitors. We compare with CoModGAN [111], pscGAN [64], rcGAN, and state-of-the-art diffusion methods DDRM (20 NFEs) [48], DDNM (100 NFEs) [96], and DPS (1000 NFEs) [17]. CoModGAN, pscGAN, and rcGAN differ from pcaGAN only in generator regularization and CoModGAN's use of discriminator MBSD [42]. For DDNM, DDRM, and DPS, we use the authors' implementations from [95], [49], and [16] with mask generation from [111]. FID and CFID were evaluated on our 20 000 image test set with $P = 1$.

Results. Table 3.5 shows test CFID, FID, LPIPS, and 40-sample generation time. The table shows that the proposed pcaGAN wins in CFID, FID and LPIPS, and that the four cGANs generate samples 3–4 orders-of-magnitude faster than DPS. Figure 3.5

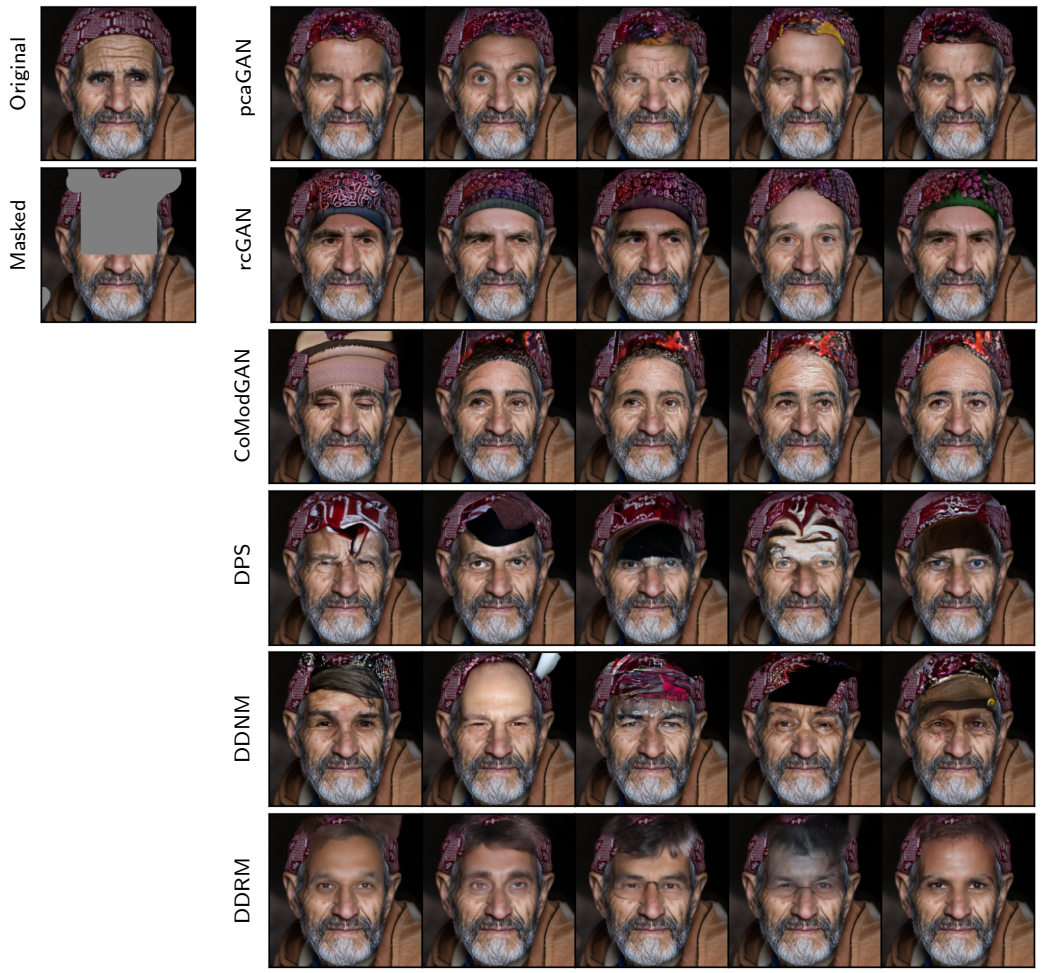


Figure 3.5: Example of inpainting a randomly generated mask on a 256×256 FFHQ face image.

shows five generated samples for each method under test, along with the true and masked image. pcaGAN shows better subjective quality than the competitors, as well as good diversity. Additional figures can be found in App. E.2.4.

3.4 Discussion

When training a cGAN, the overall goal is that the samples $\{\widehat{\boldsymbol{x}}_i\}$ generated from a particular $\boldsymbol{y}$ accurately represent the true posterior $p_{\mathsf{x}|\mathsf{y}}(\cdot|\boldsymbol{y})$. Achieving this goal is

challenging when training from paired data $\{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}$, because such datasets provide only one example of $\boldsymbol{x}$ for each given $\boldsymbol{y}$. Early methods like [37, 4, 111] focused on providing *some* variation among $\{\widehat{\boldsymbol{x}}_i\}$, but did not aim for the correct variation. rcGAN focused on providing the correct *amount* of variation by enforcing $\mathrm{tr}(\boldsymbol{\Sigma}_{\widehat{\mathsf{x}}|\mathsf{y}}) = \mathrm{tr}(\boldsymbol{\Sigma}_{\mathsf{x}|\mathsf{y}})$, and the proposed pcaGAN goes farther by encouraging $\boldsymbol{\Sigma}_{\widehat{\mathsf{x}}|\mathsf{y}}$ and $\boldsymbol{\Sigma}_{\mathsf{x}|\mathsf{y}}$ to agree along K principal directions. Our experiments demonstrate that pcaGAN yields a notable improvement over rcGAN and outperforms contemporary diffusion approaches like DPS [17].

PCA principles have also been used in *unconditional* GANs, where the goal is to train a generator $G_{\boldsymbol{\theta}}$ that turns codes $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ into outputs $\widehat{\boldsymbol{x}} = G_{\boldsymbol{\theta}}(\boldsymbol{z})$ that match the true marginal distribution p_{x} from which the training samples $\{\boldsymbol{x}_t\}$ are drawn. For example, the eigenGAN from [32] aims to train in such a way that semantic attributes are learned (without supervision) and can be independently controlled by manipulating individual entries of $\boldsymbol{z}$. But their goal is clearly different from ours.

3.5 Conclusion

In this chapter, we proposed pcaGAN, a novel image-recovery cGAN that enforces correctness in the K principal components of the conditional covariance matrix $\boldsymbol{\Sigma}_{\widehat{\mathsf{x}}|\mathsf{y}}$, as well as in the conditional mean $\boldsymbol{\mu}_{\widehat{\mathsf{x}}|\mathsf{y}}$ and trace-covariance $\mathrm{tr}(\boldsymbol{\Sigma}_{\widehat{\mathsf{x}}|\mathsf{y}})$. Experiments with synthetic Gaussian data showed pcaGAN outperforming both rcGAN and NPPC [61] in Wasserstein-2 distance across a range of problem sizes. Experiments on MNIST denoising, accelerated multicoil MRI, and large-scale image inpainting showed pcaGAN outperforming several other cGANs and diffusion models in CFID, FID,

PSNR, SSIM, LPIPS, and DISTS metrics. Furthermore, pcaGAN generates samples 3–4 orders-of-magnitude faster than the tested diffusion models. The proposed pcaGAN thus provides fast and accurate posterior sampling for image recovery problems, enabling uncertainty quantification, fairness in recovery, and easy navigation of the perception/distortion trade-off.

Limitations. We acknowledge several limitations of our work in this chapter. First, generating $P_{\mathsf{pca}} = 10K$ samples during training can impose a burden on memory when $\boldsymbol{x}$ is high dimensional. In the multicoil MRI experiment, $\boldsymbol{x} \in \mathbb{R}^d$ for $d = 2.4\text{e}6$, which limited us to $K = 1$ at batch size 2. Second, although our focus is on designing a fast and accurate posterior sampler, more work is needed on how to best use the generated samples across different applications. Using them to compute rigorous uncertainty intervals seems like a promising direction [6, 92, 60]. Third, the application to MRI is preliminary; additional tuning and validation is needed before it can be considered for clinical practice.

Chapter 4: Solving Inverse Problems using Diffusion with Iterative Colored Renoising

In this chapter, we discuss DDfire, a novel diffusion inverse solver. We show that the approximations produced by existing methods for the posterior score $\nabla_{\boldsymbol{x}_t} p_t(\boldsymbol{x}_t|\boldsymbol{y})$ are relatively poor, especially early in the reverse process. We propose a new approach that iteratively "renoises" the estimate several times per diffusion step. This iterative approach, which we call Fast Iterative REnoising (FIRE), injects colored noise such that the pre-trained diffusion model always sees white noise, in accordance with how it was trained. We leverage FIRE in the DDIM reverse process and show that the resulting "DDfire" offers state-of-the-art accuracy and runtime on several linear inverse problems. The content of this chapter appears in "Solving Inverse Problems using Diffusion with Iterative Colored Renoising," which was published in Transactions on Machine Learning Research (TMLR) in 2025.

4.1 Background

Given training data drawn from distribution p_0, diffusion models corrupt the data with ever-increasing amounts of noise and then learn to reverse that process in a way that can generate new samples from p_0. In this chapter, we assume the variance-exploding (VE) diffusion formulation [87], whereas Appendix B.1 provides

details on the variance-preserving (VP) formulation, including DDPM [34] and DDIM [82].

The VE diffusion forward process can be written as a stochastic differential equation (SDE) $\mathrm{d}\boldsymbol{x} = \sqrt{\mathrm{d}[\sigma^2(t)]/\,\mathrm{d}t}\,\mathrm{d}\boldsymbol{w}$ over t from 0 to T, where $\sigma^2(t)$ is a variance schedule and $\mathrm{d}\boldsymbol{w}$ is the standard Wiener process (SWP) [87]. The corresponding reverse process runs the SDE $\mathrm{d}\boldsymbol{x} = -\sigma^2(t)\nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{x})\,\mathrm{d}t + \sqrt{\mathrm{d}[\sigma^2(t)]/\,\mathrm{d}t}\,\mathrm{d}\overline{\boldsymbol{w}}$ backwards over t from T to 0, where $p_t(\cdot)$ is the marginal distribution of $\boldsymbol{x}$ at t and $\mathrm{d}\overline{\boldsymbol{w}}$ is the SWP run backwards. The "score function" $\nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{x})$ can be approximated using a deep neural network (DNN) $\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}, t)$ trained via denoising score matching [36].

In practice, time is discretized to $t \in \{0, 1, \ldots, T\}$, yielding the SMLD from [84], whose forward process, $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \sqrt{\sigma_{t+1}^2 - \sigma_t^2}\boldsymbol{w}_t$, with i.i.d $\{\boldsymbol{w}_t\} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and $\sigma_0^2 = 0$, implies that

$$\boldsymbol{x}_t = \boldsymbol{x}_0 + \sigma_t \boldsymbol{\epsilon}_t, \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}) \tag{4.1}$$

for all $t \in \{0, 1, \ldots, T\}$. The SMLD reverse process then uses i.i.d $\{\boldsymbol{n}_t\} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ in

$$\boldsymbol{x}_t = \boldsymbol{x}_{t+1} + (\sigma_{t+1}^2 - \sigma_t^2)\nabla_{\boldsymbol{x}} \ln p_{t+1}(\boldsymbol{x}_{t+1}) + \sqrt{\frac{\sigma_t^2(\sigma_{t+1}^2 - \sigma_t^2)}{\sigma_{t+1}^2}}\boldsymbol{n}_{t+1}. \tag{4.2}$$

To exploit side information about $\boldsymbol{x}_0$, such as the measurements $\boldsymbol{y}$ in an inverse problem, one can simply replace $p_t(\cdot)$ with $p_t(\cdot|\boldsymbol{y})$ in the above equations [87]. However, most works aim to avoid training a $\boldsymbol{y}$-dependent approximation of the conditional score function $\nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{x}_t|\boldsymbol{y})$. Rather, they take an "unsupervised" approach, where $\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) \approx \nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{x}_t)$ is learned during training but $\boldsymbol{y}$ is presented only at inference [22]. In this case, approximating $\nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{x}_t|\boldsymbol{y})$ is the key technical challenge.

There are two major approaches to approximate $\nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{x}_t|\boldsymbol{y})$. The first uses the Bayes' rule to write $\nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{x}_t|\boldsymbol{y}) = \nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{x}_t) + \nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{y}|\boldsymbol{x}_t)$ and then replaces $\nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{x}_t)$ with the score approximation $\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$. But the remaining term, $\nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{y}|\boldsymbol{x}_t)$, is intractable because $p_t(\boldsymbol{y}|\boldsymbol{x}_t) = \int p(\boldsymbol{y}|\boldsymbol{x}_0) p(\boldsymbol{x}_0|\boldsymbol{x}_t) \, \mathrm{d}\boldsymbol{x}_0$ with unknown $p(\boldsymbol{x}_0|\boldsymbol{x}_t)$, and so several approximations have been proposed. For example, DPS [17] uses $p(\boldsymbol{x}_0|\boldsymbol{x}_t) \approx \delta(\boldsymbol{x}_0 - \widehat{\boldsymbol{x}}_{0|t})$, where $\widehat{\boldsymbol{x}}_{0|t}$ is the approximation of $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_t\}$ computed from $\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t)$ using Tweedie's formula:

$$\widehat{\boldsymbol{x}}_{0|t} = \boldsymbol{x}_t + \sigma_t^2 \boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t). \tag{4.3}$$

Similarly, ΠGDM [83] uses $p(\boldsymbol{x}_0|\boldsymbol{x}_t) \approx \mathcal{N}(\boldsymbol{x}_0; \widehat{\boldsymbol{x}}_{0|t}, \zeta_t \boldsymbol{I})$ with some ζ_t. However, a drawback to both approaches is that they require backpropagation through $\boldsymbol{s}_{\boldsymbol{\theta}}(\cdot, t)$, which increases the cost of generating a single sample. In Fig. 4.4, we show that DDfire offers a 1.5× speedup over DPS at an equal number of NFEs.

The second major approach to approximating $\nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{x}_t|\boldsymbol{y})$ uses (1.1) with $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_t, \boldsymbol{y}\}$ approximated by a quantity that we'll refer to as $\widehat{\boldsymbol{x}}_{0|t,\boldsymbol{y}}$. For example, with AWGN-corrupted linear measurements

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_0 + \sigma_{\mathsf{w}} \boldsymbol{w} \in \mathbb{R}^m, \quad \boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \tag{4.4}$$

DDNM [96] approximates $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_t, \boldsymbol{y}\}$ by first computing $\widehat{\boldsymbol{x}}_{0|t}$ from (4.3) and then performing the hard data-consistency step $\widehat{\boldsymbol{x}}_{0|t,\boldsymbol{y}} = \boldsymbol{A}^+\boldsymbol{y} + (\boldsymbol{I} - \boldsymbol{A}^+\boldsymbol{A})\widehat{\boldsymbol{x}}_{0|t}$, where $(\cdot)^+$ is the pseudo-inverse. DDS [20] and DiffPIR [114] instead use the soft data-consistency step $\widehat{\boldsymbol{x}}_{0|t,\boldsymbol{y}} = \arg\min_{\boldsymbol{x}} \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|^2 + \gamma_t \|\boldsymbol{x} - \widehat{\boldsymbol{x}}_{0|t}\|^2$ with some $\gamma_t > 0$. DDRM [48] is a related technique that requires a singular value decomposition (SVD), which is prohibitive in many applications.

There are other ways to design posterior samplers. For example, [53] and [39] use Langevin dynamics. RED-diff [57] and SNORE [72] inject white noise into the RED algorithm [73], whose regularizer's gradient equals the score function [71]. [13, 21, 100, 101, 105] use Markov-chain Monte Carlo (MCMC) in the diffusion reverse process.

A key shortcoming of the aforementioned approaches is that their conditional-score approximations are not very accurate, especially early in the reverse process. For the methods that approximate $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_t,\boldsymbol{y}\}$, we can assess the approximation quality both visually and via mean-square error (MSE) or PSNR, since the exact $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_t,\boldsymbol{y}\}$ minimizes MSE given $\boldsymbol{x}_t$ and $\boldsymbol{y}$. For the methods that approximate $\nabla_{\boldsymbol{x}}\ln p_t(\boldsymbol{x}_t|\boldsymbol{y})$, we can compute their equivalent conditional-denoiser approximations using

$$\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_t,\boldsymbol{y}\} = \boldsymbol{x}_t + \sigma_t^2\nabla_{\boldsymbol{x}}\ln p_t(\boldsymbol{x}_t|\boldsymbol{y}), \tag{4.5}$$

which follows from (1.1). Figure 4.1 shows $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_t,\boldsymbol{y}\}$-approximations from the DDRM [48], DiffPIR [114], DPS [17], and DAPS [105] solvers at times 25%, 50%, and 75% through their reverse processes for noisy box inpainting with $\sigma_{\mathsf{w}} = 0.05$. The approximations show unwanted artifacts, especially early in the reverse process.

4.2 Approach

In this chapter, we aim to accurately approximate the conditional denoiser $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_t,\boldsymbol{y}\}$ at each step of the diffusion reverse process.

4.2.1 Fast Iterative REnoising (FIRE)

In this section, we describe the FIRE algorithm, which approximates $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{r}_{\mathsf{init}},\boldsymbol{y}\}$ assuming $\boldsymbol{y}$ from (4.4) and $\boldsymbol{r}_{\mathsf{init}} = \boldsymbol{x}_0 + \sigma_{\mathsf{init}}\boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon}\sim\mathcal{N}(\boldsymbol{0},\boldsymbol{I})$ and some $\sigma_{\mathsf{init}} > 0$.

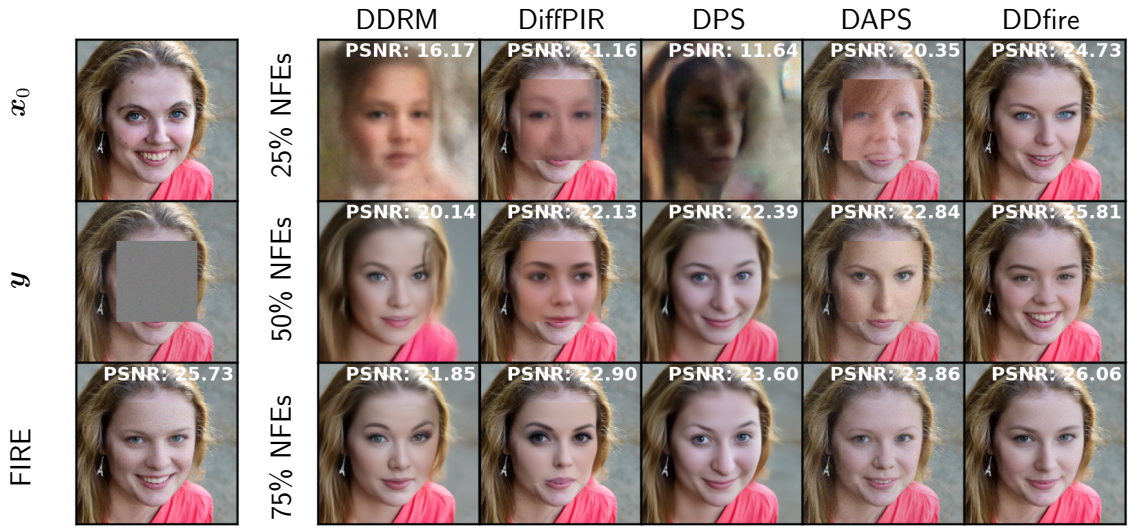


Figure 4.1: Left column: True $\boldsymbol{x}_0$, noisy box inpainting $\boldsymbol{y}$, and 50-iteration FIRE approximation of $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{y}\}$. Other columns: Approximations of $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_t, \boldsymbol{y}\}$ at different t (as measured by % NFEs). Note the over-smoothing with DDRM and DPS. Additionally, note the cut-and-paste artifacts of DiffPIR and DAPS.

FIRE performs half-quadratic splitting (HQS) PnP with a scheduled denoising variance σ^2, similar to DPIR from [106], but injects colored noise $\boldsymbol{c}$ to ensure that the error in the denoiser input $\boldsymbol{r}$ remains white. It is beneficial for the denoiser to see white input error during inference, because it is trained to remove white input error. The basic FIRE algorithm iterates the following steps $N \geq 1$ times, after initializing $\boldsymbol{r} \leftarrow \boldsymbol{r}_{\mathsf{init}}$ and $\sigma \leftarrow \sigma_{\mathsf{init}}$:

S1) Denoise $\boldsymbol{r}$ assuming AWGN of variance σ^2, giving $\overline{\boldsymbol{x}}$.

S2) MMSE estimate $\boldsymbol{x}_0$ given $\boldsymbol{y}$ from (4.4) and the prior $\boldsymbol{x}_0 \sim \mathcal{N}(\overline{\boldsymbol{x}}, \nu\boldsymbol{I})$ with some $\nu > 0$, giving $\widehat{\boldsymbol{x}}$.

S3) Update the denoising variance σ^2 via $\sigma^2 \leftarrow \sigma^2/\rho$ with some $\rho > 1$,

S4) Update $\boldsymbol{r} \leftarrow \widehat{\boldsymbol{x}}+\boldsymbol{c}$ using colored Gaussian noise $\boldsymbol{c}$ created to ensure $\mathrm{Cov}\{\boldsymbol{r}-\boldsymbol{x}_0\} = \sigma^2\boldsymbol{I}$.

S1)-S3) are essentially DPIR with regularization strength controlled by ν and a geometric denoising schedule with rate controlled by ρ, while S4) injects colored noise. In contrast, other renoising PnP approaches like SNORE [72] inject white noise. Section 4.3.1 examines the effect of removing $\boldsymbol{c}$ or replacing it with white noise. Next we provide details and enhancements of the basic FIRE algorithm.

In the sequel, we use "$\boldsymbol{d}(\boldsymbol{x},\sigma)$" to denote a neural-net approximation of the conditional-mean denoiser $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}\}$ of $\boldsymbol{x} = \boldsymbol{x}_0 + \sigma\boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0},\boldsymbol{I})$. Given a score function approximation $\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x},t) \approx \nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{x})$ as discussed in Sec. 4.1, the denoiser can be constructed via (4.3) as

$$\boldsymbol{d}(\boldsymbol{x},\sigma) = \boldsymbol{x} + \sigma^2\boldsymbol{s}_{\boldsymbol{\theta}}(\boldsymbol{x},t) \text{ with } t \text{ such that } \sigma_t = \sigma. \tag{4.6}$$

When FIRE estimates $\boldsymbol{x}_0$ from the measurements $\boldsymbol{y}$ and the denoiser output $\overline{\boldsymbol{x}}$, it employs a Gaussian approximation of the form $\boldsymbol{x}_0 \sim \mathcal{N}(\overline{\boldsymbol{x}},\nu\boldsymbol{I})$, similar to DDS, DiffPIR, and prox-based PnP algorithms. But it differs in that ν is explicitly estimated. The Gaussian approximation $\boldsymbol{x}_0 \sim \mathcal{N}(\overline{\boldsymbol{x}},\nu\boldsymbol{I})$ is equivalent to

$$\boldsymbol{x}_0 = \overline{\boldsymbol{x}} + \sqrt{\nu}\boldsymbol{e}, \quad \boldsymbol{e} \sim \mathcal{N}(\boldsymbol{0},\boldsymbol{I}). \tag{4.7}$$

Suppose $\boldsymbol{x}_0 = \overline{\boldsymbol{x}} + \sqrt{\nu_0}\boldsymbol{e}$ with $\boldsymbol{e} \sim \mathcal{N}(\boldsymbol{0},\boldsymbol{I})$, where ν_0 denotes the true error variance. Then (4.4) and (4.7) imply

$$\begin{aligned}
\mathbb{E}\{\|\boldsymbol{y} - \boldsymbol{A}\overline{\boldsymbol{x}}\|^2\} &= \mathbb{E}\{\|\boldsymbol{A}\boldsymbol{x}_0 + \sigma_{\mathsf{w}}\boldsymbol{w} - \boldsymbol{A}\boldsymbol{x}_0 + \sqrt{\nu_0}\boldsymbol{A}\boldsymbol{e}\|^2\} \\
&= \mathbb{E}\{\|\sigma_{\mathsf{w}}\boldsymbol{w} + \sqrt{\nu_0}\boldsymbol{A}\boldsymbol{e}\|^2\} \\
&= m\sigma_{\mathsf{w}}^2 + \nu_0\|\boldsymbol{A}\|_F^2,
\end{aligned} \tag{4.8}$$

assuming independence between $\boldsymbol{e}$ and $\boldsymbol{w}$. Consequently, an unbiased estimate of ν_0 can be constructed as

$$\big(\|\boldsymbol{y} - \boldsymbol{A}\overline{\boldsymbol{x}}\|^2 - m\sigma_{\mathsf{w}}^2\big)/\|\boldsymbol{A}\|_F^2 \triangleq \nu. \tag{4.9}$$

Figure 4.3 shows that, in practice, the estimate (4.9) accurately tracks the true error variance $\|\boldsymbol{x}_0 - \overline{\boldsymbol{x}}\|^2/d$.

Under (4.4) and (4.7), the MMSE estimate of $\boldsymbol{x}_0$ from $\boldsymbol{y}$ and $\overline{\boldsymbol{x}}$ can be written as [69]

$$\begin{aligned}\widehat{\boldsymbol{x}} &\triangleq \arg\min_{\boldsymbol{x}} \left\{ \frac{1}{2\sigma_{\mathsf{w}}^2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|^2 + \frac{1}{2\nu}\|\boldsymbol{x} - \overline{\boldsymbol{x}}\|^2 \right\} \\ &= \left(\boldsymbol{A}^\top\boldsymbol{A} + \frac{\sigma_{\mathsf{w}}^2}{\nu}\boldsymbol{I}\right)^{-1}\left(\boldsymbol{A}^\top\boldsymbol{y} + \frac{\sigma_{\mathsf{w}}^2}{\nu}\overline{\boldsymbol{x}}\right).\end{aligned} \tag{4.10}$$

Equation (4.10) can be computed using conjugate gradients (CG) or, if practical, the SVD $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^\top$ via

$$\widehat{\boldsymbol{x}} = \boldsymbol{V}\left(\boldsymbol{S}^\top\boldsymbol{S} + \frac{\sigma_{\mathsf{w}}^2}{\nu}\boldsymbol{I}\right)^{-1}\left(\boldsymbol{S}^\top\boldsymbol{U}^\top\boldsymbol{y} + \frac{\sigma_{\mathsf{w}}^2}{\nu}\boldsymbol{V}^\top\overline{\boldsymbol{x}}\right). \tag{4.11}$$

In any case, from (4.7) and (4.10), the error in $\widehat{\boldsymbol{x}}$ can be written as

$$\begin{aligned}\widehat{\boldsymbol{x}} - \boldsymbol{x}_0 &= \left(\boldsymbol{A}^\top\boldsymbol{A} + \frac{\sigma_{\mathsf{w}}^2}{\nu}\boldsymbol{I}\right)^{-1}\left(\boldsymbol{A}^\top[\boldsymbol{A}\boldsymbol{x}_0 + \sigma_{\mathsf{w}}\boldsymbol{w}] + \frac{\sigma_{\mathsf{w}}^2}{\nu}[\boldsymbol{x}_0 - \sqrt{\nu}\boldsymbol{e}]\right) - \boldsymbol{x}_0 \\ &= \left(\boldsymbol{A}^\top\boldsymbol{A} + \frac{\sigma_{\mathsf{w}}^2}{\nu}\boldsymbol{I}\right)^{-1}\left(\sigma_{\mathsf{w}}\boldsymbol{A}^\top\boldsymbol{w} - \frac{\sigma_{\mathsf{w}}^2}{\sqrt{\nu}}\boldsymbol{e}\right),\end{aligned} \tag{4.12}$$

and so the covariance of the error in $\widehat{\boldsymbol{x}}$ can be written as

$$\begin{aligned}\mathrm{Cov}\{\widehat{\boldsymbol{x}} - \boldsymbol{x}_0\} &= \left(\boldsymbol{A}^\top\boldsymbol{A} + \frac{\sigma_{\mathsf{w}}^2}{\nu}\boldsymbol{I}\right)^{-1}\left(\sigma_{\mathsf{w}}^2\boldsymbol{A}^\top\boldsymbol{A} + \frac{\sigma_{\mathsf{w}}^4}{\nu}\boldsymbol{I}\right)\left(\boldsymbol{A}^\top\boldsymbol{A} + \frac{\sigma_{\mathsf{w}}^2}{\nu}\boldsymbol{I}\right)^{-1} \\ &= \left(\frac{1}{\sigma_{\mathsf{w}}^2}\boldsymbol{A}^\top\boldsymbol{A} + \frac{1}{\nu}\boldsymbol{I}\right)^{-1} \triangleq \boldsymbol{C}.\end{aligned} \tag{4.13}$$

From (4.13) we see that the error in $\widehat{\boldsymbol{x}}$ can be strongly colored. For example, in the case of inpainting, where $\boldsymbol{A}$ is formed from rows of the identity matrix, the error

variance in the masked pixels equals ν, while the error variance in the unmasked pixels equals $(1/\sigma_{\mathsf{w}}^2 + 1/\nu)^{-1} \leq \sigma_{\mathsf{w}}^2$. These two values may differ by many orders of magnitude. Since most denoisers are trained to remove white noise with a specified variance of σ^2, direct denoising of $\widehat{\boldsymbol{x}}$ performs poorly, as we show in Sec. 4.3.1.

To circumvent the issues that arise from colored denoiser-input error, we propose to add "complementary" colored Gaussian noise $\boldsymbol{c} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$ to $\widehat{\boldsymbol{x}}$ so that the resulting $\boldsymbol{r} = \widehat{\boldsymbol{x}} + \boldsymbol{c}$ has an error covariance of $\sigma^2\boldsymbol{I}$, i.e., white error. This requires that

$$\begin{aligned}\boldsymbol{\Sigma} &= \sigma^2\boldsymbol{I} - \boldsymbol{C} \\ &= \sigma^2\boldsymbol{I} - \left(\frac{1}{\sigma_{\mathsf{w}}^2}\boldsymbol{V}\boldsymbol{S}^\top\boldsymbol{S}\boldsymbol{V}^\top + \frac{1}{\nu}\boldsymbol{I}\right)^{-1} \\ &= \boldsymbol{V}\,\mathrm{Diag}(\boldsymbol{\lambda})\boldsymbol{V}^\top \text{ for } \lambda_i = \sigma^2 - \frac{1}{s_i^2/\sigma_{\mathsf{w}}^2 + 1/\nu}\end{aligned} \tag{4.14}$$

for $s_i^2 \triangleq [\boldsymbol{S}^\top\boldsymbol{S}]_{i,i}$. By setting $\sigma^2 \geq \nu$, we ensure that $\lambda_i \geq 0\ \forall i$, needed for $\boldsymbol{\Sigma}$ to be a valid covariance matrix. In the case that the SVD is practical to implement, we can generate $\boldsymbol{c}$ using

$$\boldsymbol{c} = \boldsymbol{V}\,\mathrm{Diag}(\boldsymbol{\lambda})^{1/2}\boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}). \tag{4.15}$$

In the absence of an SVD, we propose to approximate $\boldsymbol{\Sigma}$ by

$$\widehat{\boldsymbol{\Sigma}} \triangleq (\sigma^2 - \nu)\boldsymbol{I} + \xi\boldsymbol{A}^\top\boldsymbol{A} \tag{4.16}$$

with some $\xi \geq 0$. Note that $\widehat{\boldsymbol{\Sigma}}$ agrees with $\boldsymbol{\Sigma}$ in the nullspace of $\boldsymbol{A}$ (i.e., when $s_n = 0$) for any ξ. By choosing

$$\xi = \frac{1}{s_{\mathsf{max}}^2}\left(\nu - \frac{1}{s_{\mathsf{max}}^2/\sigma_{\mathsf{w}}^2 + 1/\nu}\right), \tag{4.17}$$

$\widehat{\boldsymbol{\Sigma}}$ will also agree with $\boldsymbol{\Sigma}$ in the strongest measured subspace (i.e., when $s_n = s_{\mathsf{max}}$). Without an SVD, s_{max} can be computed using the power iteration [68]. Finally,

$\boldsymbol{c} \sim \mathcal{N}(\boldsymbol{0}, \widehat{\boldsymbol{\Sigma}})$ can be generated via

$$\boldsymbol{c} = \left[\sqrt{\sigma^2 - \nu}\boldsymbol{I} \quad \sqrt{\xi}\boldsymbol{A}^\top\right]\boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}) \in \mathbb{R}^{d+m}. \tag{4.18}$$

Figure D.1 shows a close agreement between the ideal and approximate renoised error spectra in practice. Next, we provide the main theoretical result on FIRE.

Theorem 4. *Suppose that, for any input $\boldsymbol{r} = \boldsymbol{x}_0 + \sigma\boldsymbol{\epsilon}$ with $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, the denoiser output $\boldsymbol{d}(\boldsymbol{r}, \sigma)$ has white Gaussian error with known variance $\nu < \sigma^2$ and independent of the noise $\boldsymbol{w}$ in (4.4). Then if initialized using $\boldsymbol{r}_{\mathsf{init}} = \boldsymbol{x}_0 + \sigma_{\mathsf{init}}\boldsymbol{\epsilon}$ with arbitrarily large but finite σ_{init} and $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, there exists a $\rho > 1$ under which the FIRE iteration S1)-S4) converges to the true $\boldsymbol{x}_0$.*

Appendix C.5 provides a proof. Note that a key assumption of Theorem 4 is that the denoiser output error is white and Gaussian. Because this may not hold in practice, we propose to replace S1) with a "stochastic denoising" step (4.20), in which AWGN is explicitly added to the denoiser output. As the AWGN variance increases, the denoiser output becomes closer to white and Gaussian but its signal-to-noise ratio (SNR) degrades. To balance these competing objectives, we propose to add AWGN with variance approximately equal to that of the raw-denoiser output error. We estimate the latter quantity from the denoiser input variance σ^2 by training a predictor of the form

$$\widehat{\nu}_{\boldsymbol{\phi}}(\sigma) \approx \mathbb{E}\{\|\boldsymbol{d}(\boldsymbol{x}_0 + \sigma\boldsymbol{\epsilon}, \sigma) - \boldsymbol{x}_0\|^2/d\}, \tag{4.19}$$

where the expectation is over $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and validation images $\boldsymbol{x}_0 \sim p_0$. Recall that d is the dimension of $\boldsymbol{x}_0$. In our experiments, $\widehat{\nu}_{\boldsymbol{\phi}}(\cdot)$ is implemented using a lookup

Algorithm 2 FIRE: $\widehat{\boldsymbol{x}} = \mathsf{FIRE}_{\mathsf{SLM}}(\boldsymbol{y}, \boldsymbol{A}, \sigma_{\mathsf{w}}, \boldsymbol{r}_{\mathsf{init}}, \sigma_{\mathsf{init}}, N, \rho)$

Require: $\boldsymbol{d}(\cdot,\cdot), \boldsymbol{y}, \boldsymbol{A}, s_{\mathsf{max}}, \sigma_{\mathsf{w}}, N, \rho > 1, \boldsymbol{r}_{\mathsf{init}}, \sigma_{\mathsf{init}}$. Also $\boldsymbol{A} = \boldsymbol{U}\,\mathrm{Diag}(\boldsymbol{s})\boldsymbol{V}^\top$ if using SVD.

$\boldsymbol{r} = \boldsymbol{r}_{\mathsf{init}}$ and $\sigma = \sigma_{\mathsf{init}}$ ▷ Initialize
for $n = 1, \dots, N$ **do**
  $\overline{\boldsymbol{x}} \leftarrow \boldsymbol{d}(\boldsymbol{r}, \sigma) + \sqrt{\widehat{\nu}_{\boldsymbol{\phi}}(\sigma)}\boldsymbol{v}, \quad \boldsymbol{v} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ ▷ Stochastic denoising
  $\nu \leftarrow (\|\boldsymbol{y} - \boldsymbol{A}\overline{\boldsymbol{x}}\|^2 - \sigma_{\mathsf{w}}^2 m)/\|\boldsymbol{A}\|_F^2$ ▷ Error variance of $\overline{\boldsymbol{x}}$
  $\widehat{\boldsymbol{x}} \leftarrow \arg\min_{\boldsymbol{x}} \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|^2/\sigma_{\mathsf{w}}^2 + \|\boldsymbol{x} - \overline{\boldsymbol{x}}\|^2/\nu$ ▷ Estimate $\boldsymbol{x}_0 \sim \mathcal{N}(\overline{\boldsymbol{x}}, \nu\boldsymbol{I})$ from $\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{A}\boldsymbol{x}_0, \sigma_{\mathsf{w}}^2\boldsymbol{I})$
  $\sigma^2 \leftarrow \max\{\sigma^2/\rho, \nu\}$ ▷ Decrease target variance
  if have SVD **then**
    $\lambda_i \leftarrow \sigma^2 - (s_i^2/\sigma_{\mathsf{w}}^2 + 1/\nu)^{-1}, \quad i = 1, \dots, d$
    $\boldsymbol{c} \leftarrow \boldsymbol{V}\,\mathrm{Diag}(\boldsymbol{\lambda})^{1/2}\boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ ▷ Colored Gaussian noise
  else
    $\xi \leftarrow \big(\nu - (s_{\mathsf{max}}^2/\sigma_{\mathsf{w}}^2 + 1/\nu)^{-1}\big)/s_{\mathsf{max}}^2$
    $\boldsymbol{c} \leftarrow \big[\sqrt{\sigma^2 - \nu}\boldsymbol{I} \quad \sqrt{\xi}\boldsymbol{A}^\top\big]\,\boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ ▷ Colored Gaussian noise
  end if
  $\boldsymbol{r} \leftarrow \widehat{\boldsymbol{x}} + \boldsymbol{c}$ ▷ Renoise so that $\mathrm{Cov}\{\boldsymbol{r} - \boldsymbol{x}_0\} = \sigma^2\boldsymbol{I}$
end for
return $\widehat{\boldsymbol{x}}$

table. The stochastic denoising step is then

$$\overline{\boldsymbol{x}} = \boldsymbol{d}(\boldsymbol{x}, \sigma) + \sqrt{\widehat{\nu}_{\boldsymbol{\phi}}(\sigma)}\boldsymbol{v}, \quad \boldsymbol{v} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}). \tag{4.20}$$

Algorithm 2 summarizes the FIRE algorithm for (4.4). In App. D.4.1, we describe a minor enhancement to Alg. 2 that speeds up the MMSE estimation step when CG is used.

4.2.2 Putting FIRE into Diffusion

Sections 4.2.1 detailed the FIRE algorithm for (4.4). There, the FIRE algorithm approximates $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{r}, \boldsymbol{y}\}$ given the measurements $\boldsymbol{y}$ and the side-information $\boldsymbol{r} = \boldsymbol{x}_0 + \sigma\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Thus, recalling the discussion in Sec. 4.1, FIRE can

be used in the SMLD reverse process as an approximation of $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_t, \boldsymbol{y}\}$ by setting $\boldsymbol{r} = \boldsymbol{x}_t$ and $\sigma = \sigma_t$.

Instead of using SMLD for the diffusion reverse process, however, we use DDIM from [82], which can be considered as a generalization of SMLD. In the sequel, we distinguish the DDIM quantities by writing them with subscript k. As detailed in App. B.3, DDIM is based on the model

$$\boldsymbol{x}_k = \boldsymbol{x}_0 + \sigma_k \boldsymbol{\epsilon}_k, \quad \boldsymbol{\epsilon}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \tag{4.21}$$

for $k = 1, \ldots, K$, where $\{\sigma_k^2\}_{k=1}^K$ is a specified sequence of variances. The DDIM reverse process iterates

$$\boldsymbol{x}_{k-1} = h_k \boldsymbol{x}_k + g_k \, \mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_k, \boldsymbol{y}\} + \varsigma_k \boldsymbol{n}_k \tag{4.22}$$

$$\varsigma_k = \eta_{\mathsf{ddim}} \sqrt{\frac{\sigma_{k-1}^2(\sigma_k^2 - \sigma_{k-1}^2)}{\sigma_k^2}}, \quad h_k = \sqrt{\frac{\sigma_{k-1}^2 - \varsigma_k^2}{\sigma_k^2}}, \quad g_k = 1 - h_k \tag{4.23}$$

over $k = K, \ldots, 2, 1$, starting from $\boldsymbol{x}_K \sim \mathcal{N}(\boldsymbol{0}, \sigma_K^2 \boldsymbol{I})$, using i.i.d $\{\boldsymbol{n}_k\}_{k=1}^K \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and some $\eta_{\mathsf{ddim}} \geq 0$. When $\eta_{\mathsf{ddim}} = 1$ and $K = T$, DDIM reduces to SMLD. But when $\eta_{\mathsf{ddim}} = 0$, the DDIM reverse process (4.22) is deterministic and can be considered as a discretization of the probability-flow ODE [82], which can outperform SMLD when the number of discretization steps K is small [15].

For a specified number K of DDIM steps (which we treat as a tuning parameter), we set the DDIM variances $\{\sigma_k^2\}_{k=1}^K$ as the geometric sequence

$$\sigma_k^2 = \sigma_{\mathsf{min}}^2 \Big(\frac{\sigma_{\mathsf{max}}^2}{\sigma_{\mathsf{min}}^2} \Big)^{\frac{k-1}{K-1}}, \quad k = 1, \ldots, K \tag{4.24}$$

for some σ_{min}^2 and σ_{max}^2 that are typically chosen to match the minimum and maximum variances used to train the denoiser $\boldsymbol{d}(\cdot, \cdot)$ or score approximation $\boldsymbol{s}_{\boldsymbol{\theta}}(\cdot, \cdot)$. So for example, if $\boldsymbol{s}_{\boldsymbol{\theta}}(\cdot, \cdot)$ was trained over the DDPM steps $t \in \{1, \ldots, T\}$ for $T = 1000$,

then we would set $\sigma^2_{\mathsf{min}} = (1-\overline{\alpha}_1)/\overline{\alpha}_1$ and $\sigma^2_{\mathsf{max}} = (1-\overline{\alpha}_{1000})/\overline{\alpha}_{1000}$ with $\overline{\alpha}_t$; see (B.7) for additional details.

Next we discuss how we set the FIRE iteration schedule $\{N_k\}_{k=1}^K$ and variance-decrease-factor $\rho > 1$. In doing so, we have two main goals:

G1) Ensure that, at every DDIM step k, the denoiser's output-error variance is at most ν_{thresh} at the final FIRE iteration, where ν_{thresh} is some value to be determined.

G2) Meet a fixed budget of $N_{\mathsf{tot}} \triangleq \sum_{k=1}^K N_k$ total NFEs.

Note that, because the denoiser's output-error variance increases monotonically with its input-error variance, we can rephrase G1) as

G1*) Ensure that, at every DDIM step k, the denoiser's input-error variance is at most $\sigma^2_{\mathsf{thresh}}$ at the final FIRE iteration, where $\sigma^2_{\mathsf{thresh}}$ is some value to be determined.

Although $\sigma^2_{\mathsf{thresh}}$ could be tuned directly, it's not the most convenient option because a good search range can be difficult to construct. Instead, we tune the fraction $\delta \in [0, 1)$ of DDIM steps k that use a single FIRE iteration (i.e., that use $N_k = 1$) and we set $\sigma^2_{\mathsf{thresh}}$ at the DDIM variance σ^2_k of the first reverse-process step k that uses a single FIRE iteration, i.e., $1 + \lfloor (K-1)\delta \rfloor \triangleq k_{\mathsf{thresh}}$. (Note that $k_{\mathsf{thresh}} = 1$ when $\delta = 0$ and $k_{\mathsf{thresh}} = K - 1$ for $\delta \approx 1$.) All subsequent[1] DDIM steps $k < k_{\mathsf{thresh}}$ will then automatically satisfy G1*) because σ^2_k decreases with k.

To ensure that the earlier DDIM steps $k > k_{\mathsf{thresh}}$ also satisfy G1*), we need that $\sigma^2_k/\rho^{N_k-1} \leq \sigma^2_{\mathsf{thresh}}$, since σ^2_k is the denoiser input-error variance at the first FIRE iteration and σ^2_k/ρ^{N_k-1} is the denoiser input-error variance at the last FIRE iteration.

[1]Recall that the reverse process counts backwards, i.e., $k = K, K-1, \ldots, 2, 1$.

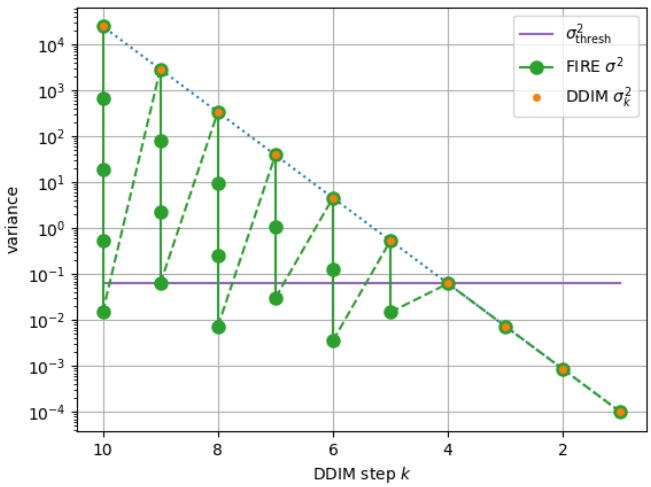


Figure 4.2: For an FFHQ denoiser: the geometric DDIM variances $\{\sigma_k^2\}_{k=1}^K$ versus DDIM step k for $K = 10$, the $\sigma^2_{\mathsf{thresh}}$ corresponding to a $\delta = 0.4$ fraction of single-FIRE-iteration DDIM steps, and the denoiser input variance σ^2 at each FIRE iteration of each DDIM step, for $N_{\mathsf{tot}} = 25$ total NFEs.

For a fixed $\rho > 1$, we can rewrite this inequality as

$$N_k \geq \frac{\ln \sigma_k^2 - \ln \sigma^2_{\mathsf{thresh}}}{\ln \rho} + 1 \triangleq \underline{N}_k. \tag{4.25}$$

Because N_k is a positive integer, it suffices to choose

$$N_k = \lceil \max\{1, \underline{N}_k\} \rceil \ \forall k. \tag{4.26}$$

Finally, ρ is chosen as the smallest value that meets the NFE budget G2) under (4.26). We find this value using bisection search. For a given k_{thresh}, a lower bound on the total NFEs is $k_{\mathsf{thresh}} \cdot 1 + (K - k_{\mathsf{thresh}}) \cdot 2$. The definition of k_{thresh} then implies that $N_{\mathsf{tot}} \geq K(2 - \delta) + \delta - 1$ and thus $K \leq (N_{\mathsf{tot}} + 1 - \delta)/(2 - \delta) \triangleq K_{\mathsf{min}}$.

In summary, for a budget of N_{tot} total NFEs, we treat the number of DDIM steps $K \in \{1, \ldots, K_{\mathsf{min}}\}$ and the fraction of single-FIRE-iteration steps $\delta \in [0, 1)$ as tuning parameters and, from them, compute $\{\sigma_k^2\}_{k=1}^K$, $\{N_k\}_{k=1}^K$, and ρ. Figure 4.2 shows an

Algorithm 3 DDfire

Require: $\boldsymbol{y}, \boldsymbol{A}, \sigma_{\mathsf{w}}$ or $p_{\mathsf{y}|\mathsf{z}}, \rho, \{\sigma_k\}_{k=1}^K, \{N_k\}_{k=1}^K, \eta_{\mathsf{ddim}} \geq 0$

$\boldsymbol{x}_K \sim \mathcal{N}(\boldsymbol{0}, \sigma_K^2 \boldsymbol{I})$

for $k = K, K-1, \ldots, 1$ **do**

$\quad \widehat{\boldsymbol{x}}_{0|k} = \mathsf{FIRE}(\boldsymbol{y}, \boldsymbol{A}, *, \boldsymbol{x}_k, \sigma_k, N_k, \rho)$ ▷ FIRE via Alg. 2

$\quad \varsigma_k = \eta_{\mathsf{ddim}} \sqrt{\dfrac{\sigma_{k-1}^2(\sigma_k^2 - \sigma_{k-1}^2)}{\sigma_k^2}}$

$\quad \boldsymbol{x}_{k-1} = \sqrt{\dfrac{\sigma_{k-1}^2 - \varsigma_k^2}{\sigma_k^2}} \boldsymbol{x}_k + \left(1 - \sqrt{\dfrac{\sigma_{k-1}^2 - \varsigma_k^2}{\sigma_k^2}}\right) \widehat{\boldsymbol{x}}_{0|k} + \varsigma_k \boldsymbol{n}_k, \quad \boldsymbol{n}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ ▷ DDIM update

end for

return $\widehat{\boldsymbol{x}}_{0|k}$

example. The pair (K, δ) can be tuned using cross-validation. Algorithm 3 details DDIM with the FIRE approximation of $\mathbb{E}\{\boldsymbol{x}_0 | \boldsymbol{x}_k, \boldsymbol{y}\}$, which we refer to as "DDfire."

4.2.3 Relation to Other Methods

To solve inverse problems, a number of algorithms have been proposed that iterate denoising (possibly score-based), data-consistency (hard or soft), and renoising. Such approaches are referred to as either plug-and-play, Langevin, or diffusion methods. (Recall the discussion in Section 4.1). While existing approaches use white renoising, the proposed DDfire uses colored renoising that whitens the denoiser input error.

When estimating $\boldsymbol{x}_0$ from the measurements $\boldsymbol{y}$ of (4.4), methods such as DDS, DiffPIR, DAPS, and SNORE use a Gaussian prior approximation of the form $\boldsymbol{x}_0 \sim \mathcal{N}(\overline{\boldsymbol{x}}, \nu \boldsymbol{I})$. But while they control ν with tuning parameters, DDfire explicitly estimates ν via (4.9). See Sec. 4.4 for a detailed comparison of DDfire to DDS, DiffPIR, and SNORE.

Table 4.1: DDfire ablation results for noisy FFHQ box inpainting with $\sigma_{\mathsf{w}} = 0.05$ at 1000 NFEs.

Method	PSNR↑	LPIPS↓	Runtime
DDfire	24.31	0.1127	34.37s
DDfire w/o renoising	18.48	0.2349	34.37s
DDfire w/o colored renoising	23.64	0.1553	34.37s
DDfire w/ stochastic denoising	24.30	0.1143	34.37s
DDfire w/o estimating ν	23.02	0.1755	34.37s
DDfire w/o CG early stopping	24.31	0.1127	52.12s
DDfire w/ SVD	24.31	0.1124	30.97s

4.3 Numerical Experiments

We use 256×256 FFHQ [44] and ImageNet [24] datasets with pretrained diffusion models from [17] and [26], respectively. As linear inverse problems, we consider box inpainting with a 128×128 mask, Gaussian deblurring using a 61×61 blur kernel with 3-pixel standard deviation, motion deblurring using a 61×61 blur kernel generated using [12] with intensity 0.5, and $4\times$ bicubic super-resolution. We compare to DDRM [48], DiffPIR [114], ΠGDM [83], DDS [20], DPS [17], RED-diff [57], and DAPS [105].

Unless specified otherwise, DDfire was configured as follows. For the linear inverse problems, CG is used (no SVD), 1000 NFEs are used without stochastic denoising, and the (K, δ) hyperparameters are tuned to minimize LPIPS [107] on a 100-sample validation set (see Table D.1). Appendix D.4 contains additional details on the implementation of DDfire and the competing methods.

4.3.1 Ablation Study

We first perform an ablation study on the DDfire design choices in Sec. 4.2 using noisy FFHQ box inpainting and a 100-image validation set. The results are summarized

in Table 4.1. We first see that both PSNR and LPIPS suffer significantly when FIRE is run without renoising (i.e., $\boldsymbol{c} = \boldsymbol{0}$ in line 14 in Alg. 2). Similarly, renoising using white noise (i.e., $\boldsymbol{c} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$ in line 14 of Alg. 2) gives noticeably worse PSNR and LPIPS than the proposed colored noise. Using stochastic denoising gives nearly identical performance to plain denoising (i.e., $\widehat{\nu}_{\phi}(\sigma) = 0$ in line 3 of Alg. 2), and so we use plain denoising by default with linear inverse problems. A more significant degradation results when the denoiser output-error variance ν is not adapted to $\overline{\boldsymbol{x}}$ in line 4 of Alg. 2 but set at the data-average value $\widehat{\nu}_{\boldsymbol{\phi}}(\sigma)$. On the other hand, when CG doesn't use early stopping (as described in App. D.4.1) the runtime increases without improving PSNR or LPIPS. Thus, we use early stopping by default. Finally, using an SVD instead of CG, which also avoids the noise approximation in (4.16), gives essentially identical PSNR and LPIPS but with a slightly faster runtime. Figure 4.4 shows another LPIPS/runtime comparison of the SVD and CG versions of DDfire.

4.3.2 Accuracy of σ^2 and ν

Figure 4.3 shows DDfire's σ^2 versus iteration i, for comparison to the true denoiser input variance $\|\boldsymbol{r} - \boldsymbol{x}_0\|_2^2/d$, and DDfire's ν, for comparison to the true denoiser output variance $\|\overline{\boldsymbol{x}}_0 - \boldsymbol{x}_0\|_2^2/d$, for 25 FIRE iterations with $\rho = 1.5$ for noisy 4× super-resolution at $t[k] = 1000$. We see that the DDfire estimates σ^2 and ν track the true error variances quite closely.

4.3.3 PSNR, LPIPS, and FID Results

For noisy linear inverse problems, Tables 4.2–4.3 show PSNR, LPIPS, and FID [33] on a 1000-sample test set for FFHQ and ImageNet data, respectively. DDRM

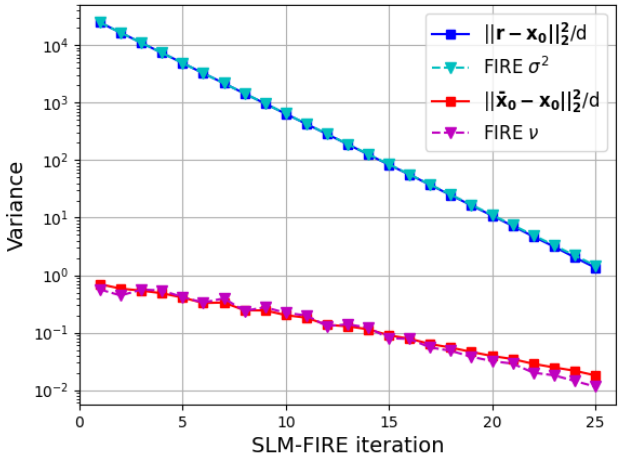


Figure 4.3: DDfire σ^2, true denoiser input variance $\|\boldsymbol{r} - \boldsymbol{x}_0\|_2^2/d$, DDfire ν, and true denoiser output variance $\|\overline{\boldsymbol{x}}_0 - \boldsymbol{x}_0\|_2^2/d$ vs. DDfire iteration for noisy $4\times$ super-resolution at $t[k] = 1000$ for a single validation sample $\boldsymbol{x}_0$.

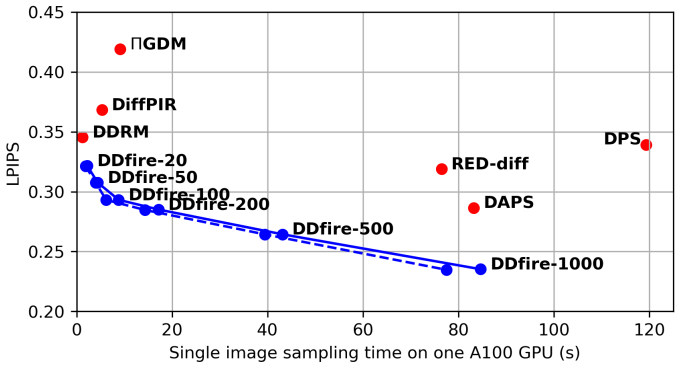


Figure 4.4: LPIPS vs. single image sampling time for noisy Gaussian deblurring on an A100 GPU. The evaluation used 1000 ImageNet images. Solid line: DDfire with CG for various numbers of NFEs. Dashed line: DDfire with SVD.

was not applied to motion deblurring due to the lack of an SVD. Tables 4.2–4.3 show that DDfire wins in most cases and otherwise performs well.

Table 4.2: Noisy FFHQ results with measurement noise standard deviation $\sigma_{\mathsf{w}} = 0.05$.

	Inpaint (box)			Deblur (Gaussian)			Deblur (Motion)			4× Super-resolution		
Model	PSNR↑	LPIPS↓	FID↓	PSNR↑	LPIPS↓	FID↓	PSNR↑	LPIPS↓	FID↓	PSNR↑	LPIPS↓	FID↓
DDRM	21.71	0.1551	40.61	25.35	0.2223	51.70	-	-	-	**27.32**	0.1864	45.82
DiffPIR	22.43	0.1883	31.98	24.56	0.2394	34.82	26.91	0.1952	26.67	24.89	0.2486	32.33
ΠGDM	21.41	0.2009	44.41	23.66	0.2525	45.34	25.14	0.2082	41.95	24.40	0.2520	51.41
DDS	20.28	0.1481	30.23	26.74	0.1648	25.47	27.52	0.1503	27.59	26.71	0.1852	27.09
DPS	22.54	0.1368	35.69	25.70	0.1774	25.18	26.74	0.1655	27.17	26.30	0.1850	27.38
RED-diff	23.58	0.1883	48.86	26.99	0.2081	38.82	16.47	0.5074	128.68	25.61	0.3569	70.86
DAPS	23.61	0.1415	31.51	26.97	0.1827	31.10	27.13	0.1718	30.74	26.91	0.1885	30.83
DDfire	**24.75**	**0.1101**	**25.26**	**27.10**	**0.1533**	**24.97**	**28.14**	**0.1374**	**26.12**	27.13	**0.1650**	**25.73**

Table 4.3: Noisy ImageNet results with measurement noise standard deviation $\sigma_{\mathsf{w}} = 0.05$.

	Inpaint (box)			Deblur (Gaussian)			Deblur (Motion)			4× Super-resolution		
Model	PSNR↑	LPIPS↓	FID↓	PSNR↑	LPIPS↓	FID↓	PSNR↑	LPIPS↓	FID↓	PSNR↑	LPIPS↓	FID↓
DDRM	18.24	0.2423	67.47	22.56	0.3454	68.78	-	-	-	**24.49**	0.2777	64.68
DiffPIR	18.03	0.2860	65.55	21.31	0.3683	56.35	24.36	0.2888	54.11	23.31	0.3383	63.48
ΠGDM	17.69	0.3303	86.36	20.87	0.4191	75.43	22.15	0.3591	70.91	21.25	0.4149	78.57
DDS	16.68	0.2222	63.07	23.14	0.2684	50.84	23.34	0.2674	50.08	23.03	0.3011	52.13
DPS	18.23	0.2314	59.10	21.30	0.3393	50.46	21.77	0.3307	80.27	23.38	0.2904	49.86
RED-diff	18.95	0.2909	108.88	23.45	0.3190	65.65	15.21	0.5647	198.74	22.99	0.3858	83.06
DAPS	19.99	0.2199	61.53	**23.91**	0.2863	56.87	24.58	0.2722	54.83	24.04	0.2729	55.54
DDfire	**20.39**	**0.1915**	**55.54**	23.71	**0.2353**	**50.05**	**24.59**	**0.2314**	**49.25**	23.58	**0.2629**	**49.67**

Fig. 4.5 shows image examples for inpainting, motion deblurring, Gaussian deblurring, and 4× super-resolution on ImageNet. Similarly, Fig. 4.6 shows image examples for inpainting, motion deblurring, Gaussian deblurring, and 4× super-resolution on FFHQ. In both cases, the zoomed regions show that DDfire did a better job recovering fine details.

4.3.4 Runtime Results

Figure 4.4 shows LPIPS vs. average runtime (in seconds on an A100 GPU) to generate a single image for noisy Gaussian deblurring on the 1000-sample ImageNet

Algorithm 4 DDS [20]

Require: $\boldsymbol{d}(\cdot,\cdot), \boldsymbol{y}, \boldsymbol{A}, \gamma_{\mathsf{dds}}, M_{\mathsf{cg}}, \{\sigma_k\}_{k=1}^K, \eta_{\mathsf{ddim}} \geq 0$

$\boldsymbol{x}_K \sim \mathcal{N}(\boldsymbol{0}, \sigma_K^2 \boldsymbol{I})$

for $k = K, K-1, \ldots, 1$ **do**

  $\overline{\boldsymbol{x}}_k = \boldsymbol{d}(\boldsymbol{x}_k, \sigma_k)$ ▷ Denoising

  $\widehat{\boldsymbol{x}}_{0|k} = \mathsf{CG}(\boldsymbol{A}^\top \boldsymbol{A} + \gamma_{\mathsf{dds}} \boldsymbol{I}, \boldsymbol{A}^\top \boldsymbol{y} + \gamma_{\mathsf{dds}} \overline{\boldsymbol{x}}_k, \overline{\boldsymbol{x}}_k, M_{\mathsf{cg}}) \approx \arg\min_{\boldsymbol{x}} \left\{ \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|^2 + \gamma_{\mathsf{dds}} \|\boldsymbol{x} - \overline{\boldsymbol{x}}_k\|^2 \right\}$

  $\varsigma_k = \eta_{\mathsf{ddim}} \sqrt{\dfrac{\sigma_{k-1}^2 (\sigma_k^2 - \sigma_{k-1}^2)}{\sigma_k^2}}$

  $\boldsymbol{x}_{k-1} = \sqrt{\dfrac{\sigma_{k-1}^2 - \varsigma_k^2}{\sigma_k^2}} \boldsymbol{x}_k + \left(1 - \sqrt{\dfrac{\sigma_{k-1}^2 - \varsigma_k^2}{\sigma_k^2}} \right) \widehat{\boldsymbol{x}}_{0|k} + \varsigma_k \boldsymbol{n}_k, \quad \boldsymbol{n}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ ▷ DDIM update

end for

return $\widehat{\boldsymbol{x}}_{0|k}$

test set. The figure shows that DDfire gives a better performance/complexity tradeoff than the competitors. It also shows that DDfire is approximately 1.5 times faster than DPS when both are run at 1000 NFEs, due to DPS's use of backpropagation.

4.4 Discussion

In this section, we compare DDfire to other renoising PnP schemes for the SLM that involve a proximal data-fidelity step or, equivalently, MMSE estimation of $\boldsymbol{x}_0$ under the Gaussian prior assumption (4.7): DDS, DiffPIR, and SNORE, which are detailed in Alg. 4, Alg. 5, and Alg. 6, respectively.

Comparing DDfire (see Alg. 2 and Alg. 3) to DDS, we see that both approaches use CG for approximate MMSE estimation under a Gaussian prior approximation and both use a DDIM diffusion update. But DDS uses the hyperparameters γ_{dds} and M_{cg} (the number CG iterations, which is usually small, such as four) to adjust the noise variance ν in the Gaussian prior approximation (4.7), while DDfire estimates ν

Algorithm 5 DiffPIR [114]

Require: $\boldsymbol{d}(\cdot,\cdot), \boldsymbol{y}, \boldsymbol{A}, \sigma_{\mathsf{w}}, \lambda_{\mathsf{diffpir}}, \{\sigma_k\}_{k=1}^K, \eta_{\mathsf{diffpir}}$

$\boldsymbol{x}_K \sim \mathcal{N}(\boldsymbol{0}, \sigma_K^2 \boldsymbol{I})$

for $k = K, K-1, \ldots, 1$ **do**

  $\overline{\boldsymbol{x}}_k = \boldsymbol{d}(\boldsymbol{x}_k, \sigma_k)$ ▷ Denoising

  $\widehat{\boldsymbol{x}}_{0|k} = \arg\min_{\boldsymbol{x}} \left\{ \frac{1}{2\sigma_{\mathsf{w}}^2} \|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|^2 + \frac{\lambda_{\mathsf{diffpir}}}{2\sigma_k^2} \|\boldsymbol{x} - \overline{\boldsymbol{x}}_k\|^2 \right\}$ ▷ Approximate MMSE estimation

  $\varrho_k = \sqrt{1 - \eta_{\mathsf{diffpir}}} \dfrac{\sigma_{k-1}}{\sigma_k}$

  $\boldsymbol{x}_{k-1} = \varrho_k \boldsymbol{x}_k + (1 - \varrho_k) \widehat{\boldsymbol{x}}_{0|k} + \sqrt{\eta_{\mathsf{diffpir}}} \sigma_{k-1} \boldsymbol{n}_k, \quad \boldsymbol{n}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ ▷ DDIM-like update

end for

return $\widehat{\boldsymbol{x}}_{0|k}$

at each iteration. Also, DDfire injects colored Gaussian noise to whiten the denoiser input error while DDS injects no noise outside of DDIM.

Comparing DDfire to DiffPIR, we see that both perform MMSE estimation under a Gaussian prior approximation and both use a DDIM-like diffusion update. But DiffPIR uses the hyperparameter $\lambda_{\mathsf{diffpir}}$ to adjust the noise variance ν in the Gaussian prior approximation (4.7), while DDfire estimates ν at each iteration. Also, DDfire injects colored Gaussian noise to whiten the denoiser input error while DiffPIR injects no noise outside of its DDIM-like step.

Comparing DDfire to SNORE, we see that both perform MMSE estimation under a Gaussian prior approximation and both use renoising. But SNORE uses white renoising while DDfire uses colored renoising to whiten the denoiser input error. Also, SNORE uses the hyperparameter δ_{snore} to adjust the noise variance ν in the Gaussian prior approximation (4.7), while DDfire estimates it at each iteration. Furthermore, since SNORE is based on the RED algorithm [73], its denoiser output is scaled and shifted. Finally, SNORE has many more tuning parameters than DDfire.

Algorithm 6 Annealed Proximal SNORE [72]

Require: $\boldsymbol{d}(\cdot,\cdot), \boldsymbol{y}, \boldsymbol{A}, \sigma_{\mathsf{w}}, \delta_{\mathsf{snore}}, M_{\mathsf{snore}}, \{\sigma_i\}_{i=1}^{M_{\mathsf{snore}}}, \{\alpha_i\}_{i=1}^{M_{\mathsf{snore}}}, \{K_i\}_{i=1}^{M_{\mathsf{snore}}}, \widehat{\boldsymbol{x}}_{\mathsf{init}}$

$\widehat{\boldsymbol{x}}_{0|K_{M_{\mathsf{snore}}}} = \widehat{\boldsymbol{x}}_{\mathsf{init}}$
for $i = M_{\mathsf{snore}}, M_{\mathsf{snore}} - 1 \ldots, 1$ **do**
  for $k = K_i, K_i - 1, \ldots, 1$ **do**
    $\boldsymbol{r}_k = \widehat{\boldsymbol{x}}_{0|k} + \sigma_i \boldsymbol{n}_{i,k}, \quad \boldsymbol{n}_{i,k} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ ▷ Renoising
    $\overline{\boldsymbol{x}}_k = \left(1 - \frac{\delta_{\mathsf{snore}}\alpha_i}{\sigma_i^2}\right)\widehat{\boldsymbol{x}}_{0|k} + \frac{\delta_{\mathsf{snore}}\alpha_i}{\sigma_i^2}\boldsymbol{d}(\boldsymbol{r}_k, \sigma_i)$ ▷ RED update
    $\widehat{\boldsymbol{x}}_{0|k-1} = \arg\min_{\boldsymbol{x}} \left\{ \frac{1}{2\sigma_{\mathsf{w}}^2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|^2 + \frac{1}{2\delta_{\mathsf{snore}}}\|\boldsymbol{x} - \overline{\boldsymbol{x}}_k\|^2 \right\}$ ▷ Approximate MMSE estimation
  end for
end for
return $\widehat{\boldsymbol{x}}_{0|k}$

4.5 Conclusion

In this chapter, we proposed the Fast Iterative Renoising (FIRE) algorithm, which can be interpreted as the HQS plug-and-play algorithm with a colored renoising step that aims to whiten the denoiser input error. Since the FIRE algorithm approximates the measurement-conditional denoiser $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_t, \boldsymbol{y}\}$, or equivalently the measurement-conditional score $\nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{x}_t|\boldsymbol{y})$, it can be readily combined with DDIM for diffusion posterior sampling, giving the "DDfire" algorithm. Experiments on box inpainting, Gaussian and motion deblurring, and 4× super-resolution with FFHQ and ImageNet images show DDfire outperforming DDRM, ΠGDM, DDS, DiffPIR, DPS, RED-diff, and DAPS in PSNR, LPIPS, and FID metrics in nearly all cases. Finally, DDfire offers fast inference, with better LPIPS-versus-runtime curves than the competitors.

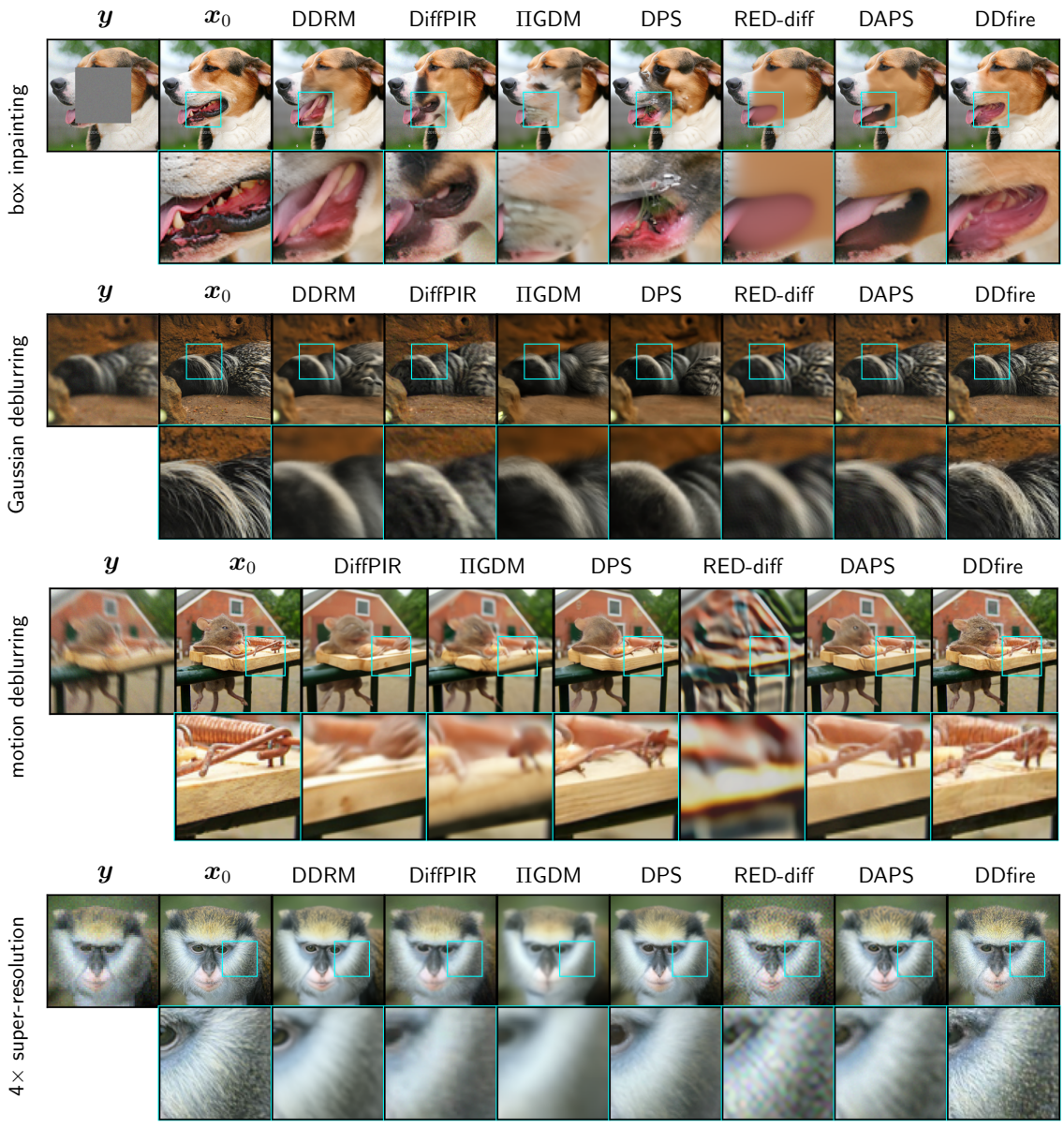


Figure 4.5: Example recoveries from noisy linear inverse problems with ImageNet images.

Figure 4.6: Example recoveries from noisy linear inverse problems with FFHQ images.

Chapter 5: Final Thoughts

5.1 Final Experiment

As a final exercise, we compare rcGAN (Chapter 2), pcaGAN (Chapter 3), and DDfire (Chapter 4) on accelerated multicoil MRI reconstruction at $R \in \{4, 8\}$, using the same training/validation/test splits and testing procedure described in Chapter 3. For DDfire, we trained a new EDM-style denoiser using the fastMRI-EDM [1] training code from [75]. At inference, the $\boldsymbol{A}$ matrix used by DDfire is of the form

$$\boldsymbol{A} = \boldsymbol{M}\boldsymbol{F}\boldsymbol{S}, \tag{5.1}$$

where $\boldsymbol{M} \in \mathbb{R}^{CM \times CN}$ is a subsampling operator, $\boldsymbol{F}^{CN \times CN}$ is a block-diagonal matrix where each $N \times N$ block contains the unitary 2D discrete Fourier transform, $\boldsymbol{S} \in \mathbb{C}^{CN \times N}$ is a block-diagonal matrix containing the sensitivity maps of the C measurement coils, and M, N are our single-coil measurement and image dimensions, respectively. However, our denoiser operates on the coil-combined image, not the multicoil measurements (or individual coil images, like our cGANs). We do not add any noise to the k-space, but the fastMRI data is inherently noisy, and so we set σ_y=0.01 and tune DDfire via a grid search, finding the LPIPS-minimizing choices of (K, δ) which were found to be $(50, 0.2)$ for both $R = 4$ and $R = 8$.

Table 5.1: Average MRI results at acceleration $R = 4$.

Model	PSNR↑	SSIM↑	LPIPS↓	DISTS↓	CFID↓	FID↓	APSD	Time (4)↓
E2E-VarNet (Sriram et al. [89])	**39.93**	**0.9641**	0.0316	0.0859	16.08	38.88	0.0	310 ms
rcGAN	39.55	0.9544	0.0164	0.0546	9.71	25.62	3.8e-6	**217 ms**
pcaGAN	39.77	0.9557	**0.0159**	**0.0542**	8.78	25.02	4.4e-5	**217 ms**
DDfire ($K = 500$)	38.87	0.9489	0.0288	0.0746	10.46	27.49	4.0e-6	1.5 min
DDfire ($K = 200$)	39.31	0.9531	0.0198	0.0710	8.76	24.90	2.3e-6	1.5 min
DDfire ($K = 100$)	39.44	0.9542	0.0191	0.0699	8.32	22.21	1.2e-6	1.5 min
DDfire ($K = 50$)	39.61	0.9562	0.0178	0.0672	**7.63**	**20.07**	9.2e-7	1.5 min

Table 5.2: Average MRI results at acceleration $R = 8$.

Model	PSNR↑	SSIM↑	LPIPS↓	DISTS↓	CFID↓	FID↓	APSD	Time (4)↓
E2E-VarNet (Sriram et al. [89])	36.49	0.9220	0.0575	0.1253	36.86	44.04	0.0	316 ms
rcGAN	35.42	0.9257	0.0379	0.0877	24.04	28.43	7.6e-6	**217 ms**
pcaGAN	35.94	0.9283	0.0344	0.0799	21.65	28.35	6.5e-5	**217 ms**
DDfire ($K = 500$)	33.99	0.9194	0.0480	0.0954	26.46	30.21	7.7e-6	1.5 min
DDfire ($K = 200$)	35.72	0.9384	0.0311	0.0823	16.62	25.66	7.4e-6	1.5 min
DDfire ($K = 100$)	36.27	0.9428	0.0277	0.0810	14.74	24.54	7.1e-6	1.5 min
DDfire ($K = 50$)	**37.18**	**0.9486**	**0.0251**	**0.0748**	**14.89**	**23.29**	7.0e-6	1.5 min

Table 5.1 shows test results for PSNR, SSIM, LPIPS, and DISTS for each method under test with optimal averaging constant P. For each method, we draw $P \in \{1, 2, 4, 8, 16, 32\}$ independent samples for each $\boldsymbol{y}$, average those P outputs to form a single estimate, compute the metric on that estimate, and repeat for every P. We then select the "optimal" P per method and metric, i.e., the P that maximizes PSNR/SSIM or minimizes LPIPS/DISTS, and report that best score in the table. The specific P values for each method/metric are listed in Table 5.3. We also list CFID, FID, APSD, and the reconstruction time of 4 samples. The E2E-VarNet still wins in PSNR and SSIM, while pcaGAN still wins in LPIPS and DISTS. However, DDfire performs best in CFID and FID.

Table 5.3: Optimal averaging constant P for each method/metric.

	$R = 4$				$R = 8$			
Model	PSNR	SSIM	LPIPS	DISTS	PSNR	SSIM	LPIPS	DISTS
rcGAN	32	4	2	1	32	8	2	2
pcaGAN	32	4	2	1	32	4	2	2
DDfire ($K = 500$)	32	8	2	1	32	8	4	2
DDfire ($K = 200$)	32	16	8	1	32	16	8	2
DDfire ($K = 100$)	32	32	8	1	32	32	16	2
DDfire ($K = 50$)	32	32	8	1	32	32	16	2

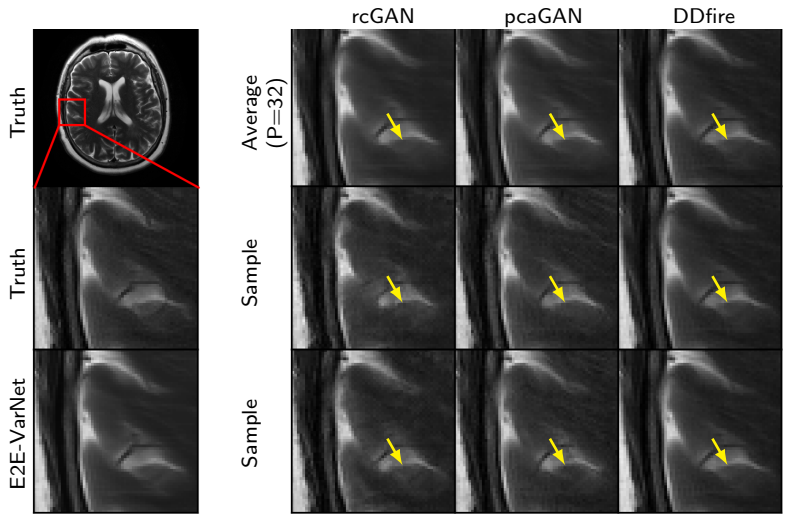


Figure 5.1: Example MRI recoveries at $R = 8$. Arrows highlight meaningful variations.

Table 5.2 similarly shows test results for PSNR, SSIM, LPIPS, and DISTS for each method under test with optimal averaging constant P, as well as CFID, FID, APSD, and the reconstruction time of 4 samples. There, we see that DDfire outperforms both rcGAN and pcaGAN in all metrics, however the cGANs are notably faster with respect to sample generation time. Even so, DDfire nets a significant speedup over the previously evaluated MRI diffusion method from Jalal et al. (see Chapters 2 and

3). DDfire also outperforms the E2E-VarNet from Sriram et al. in PSNR, unlike the cGANs. Figure 5.1 shows zoomed versions of two recoveries $\widehat{\boldsymbol{x}}_i$ and the sample average $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 32$ at $R = 8$, as well as the estimate from the E2E-VarNet. As shown in the figure, DDfire trades some sample diversity for higher fidelity—the arrowed structure appears in the ground truth and DDfire, but is missing or faint in rcGAN/pcaGAN.

In these experiments, DDfire's sample diversity is chiefly governed by the number of DDIM steps K. Our LPIPS-driven grid search yields a relatively small $K = 50$ (for both accelerations), which steers the sampler toward mode-seeking behavior: distortion is reduced, but between-sample variability (measured by APSD) is suppressed. The "optimal P" still shifts by metric, tending lower for perceptual scores (LPIPS/DISTS) and higher for distortion-style metrics (PSNR/SSIM). However, when computing SSIM and LPIPS, one may note that the optimal P values of DDfire are notably higher than those of rcGAN/pcaGAN at $K = 50$ (see Table 5.3). This is a consequence of lower sample diversity relative to the cGANs, which can be improved by increasing K. However, this comes at the expense of worse fidelity. Our experimental results, shown in Tables 5.1 and 5.2, verify this empirically and we also see P values in Table 5.3 that are more inline with what we may expect. In short, the LPIPS-based tuning explicitly prioritizes fidelity over diversity in DDfire, whereas the cGAN baselines balance these aims via their training objectives.

5.2 Potential Future Work

Similar to how diffusion models transition from noise realizations to clean images, direct diffusion bridges (DDBs) [55, 23] transition from corrupted images to clean

images (or, more generally, from one distribution of images to another). Using $\boldsymbol{x}_0$ to denote a clean image and $\boldsymbol{x}_1$ to denote a corrupted image, the DDB process generates intermediate samples $\boldsymbol{x}_t$ via

$$\boldsymbol{x}_t = (1-\alpha_t)\boldsymbol{x}_0 + \alpha_t \boldsymbol{x}_1 + \sigma_t \boldsymbol{z}_t, \quad \boldsymbol{z}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \tag{5.2}$$

for some increasing $\alpha_t \in [0,1]$ where $\alpha_0 = 0$ and $\alpha_1 = 1$, and $\sigma_t \geq 0$. Typically, we have $\sigma_t = \alpha_t \epsilon_t$ where ϵ_t is a non-increasing function of t. For a linear inverse problem, we have $\boldsymbol{x}_1 = \boldsymbol{A}^\top \boldsymbol{y}$ or $\boldsymbol{x}_1 = \boldsymbol{A}^+ \boldsymbol{y}$.

The value in adopting direct diffusion bridges (DDBs) over a generic diffusion prior is that the DDB explicitly embeds the degraded observation $\boldsymbol{y}$ (and implicitly $\boldsymbol{A}$) into the forward process in (5.2). This design keeps the trajectory near the measurement-consistent manifold throughout sampling and can accelerate convergence by lowering the burden on data-consistency projections or guidance. However, this problem-tailored approach loses generality: a DDB trained for a particular degradation operator $\boldsymbol{A}$ (or set of similar of $\boldsymbol{A}$) is not a drop-in solver for arbitrary inverse problems. So, we are trading off generality for improved posterior sampling performance.

We can derive an "ancestral sampling" strategy for DDB where [18], for any $s < t$, we have

$$\boldsymbol{x}_s = (1-\alpha_{s|t})\boldsymbol{x}_0 + \alpha_{s|t}\boldsymbol{x}_t + \sigma_{s|t}\boldsymbol{n}_{s|t}, \quad \boldsymbol{n}_{s|t} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}) \tag{5.3}$$

for $\boldsymbol{n}_{s|t}$ that is independent of $\boldsymbol{x}_0$ and $\boldsymbol{x}_t$, with $\alpha_{s|t} \triangleq \alpha_s/\alpha_t$ and $\sigma^2_{s|t} \triangleq \alpha_s^2(\epsilon_s^2 - \epsilon_t^2)$. In practice, we cannot actually compute $\boldsymbol{x}_s$ because $\boldsymbol{x}_0$ is unknown. Instead, we use an estimate of $\boldsymbol{x}_0$ computed using an approximation $\widehat{\boldsymbol{x}}_t$ of $\boldsymbol{x}_t$, yielding

$$\widehat{\boldsymbol{x}}_s = (1-\alpha_{s|t})G_{\boldsymbol{\theta}}(\widehat{\boldsymbol{x}}_t, t) + \alpha_{s|t}\widehat{\boldsymbol{x}}_t + \sigma_{s|t}\boldsymbol{n}_{s|t}, \quad \boldsymbol{n}_{s|t} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \tag{5.4}$$

where $G_{\boldsymbol{\theta}}$ is trained to estimate $\boldsymbol{x}_0$ from $\boldsymbol{x}_t$. The typical training objective for $G_{\boldsymbol{\theta}}$ is

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}_0,\boldsymbol{x}_1,\boldsymbol{z},t\sim p(t)} \left\{ \lambda_t ||G_{\boldsymbol{\theta}}(\boldsymbol{x}_t, t) - \boldsymbol{x}_0||_2^2 \right\}, \tag{5.5}$$

where $\boldsymbol{x}_t$ is computed according to (5.2), λ_t is some t-dependent weight, and $p(t)$ is a user-selected density (e.g., uniform).

Even though $G_{\boldsymbol{\theta}}$ is analogous to the denoiser/score-network used by diffusion models, it is something else entirely. In particular, it is trained with a specific forward model (i.e., $\boldsymbol{A}$) in mind. Consequently, we are free to train $G_{\boldsymbol{\theta}}$ in whatever way we see fit. To that end, one potential direction is to train $G_{\boldsymbol{\theta}}$ with the rcGAN/pcaGAN regularization described in Chapters 2 and 3 to improve the generator's statistics. The rcGAN regularization yields objective

$$\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}_0,\boldsymbol{x}_1,\boldsymbol{z},t\sim p(t)} \left\{ \frac{\lambda_t}{P_{\mathsf{rc}}} \sum_{i=1}^{P_{\mathsf{rc}}} ||G_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,i}, t) - \boldsymbol{x}_0||_2^2 \right\} + \beta \mathcal{L}_{1,\mathsf{SD},P_{\mathsf{rc}}}(\boldsymbol{\theta}, \beta_{\mathsf{SD}}) \tag{5.6}$$

where $\boldsymbol{x}_{t,i} = (1-\alpha_t)\boldsymbol{x}_0 + \alpha_t \boldsymbol{x}_1 + \sigma_t \boldsymbol{z}_{t,i}$, $\beta, \lambda_t > 0$, $P_{\mathsf{rc}} \geq 2$, and the regularizer expressions are defined in (2.7)-(3.5) with $\widehat{\boldsymbol{x}}_i = G_{\boldsymbol{\theta}}(\boldsymbol{x}_{t,i}, t)$. One must choose β and λ_t carefully in order to balance the effects of both terms.

Coupled with improved training of $G_{\boldsymbol{\theta}}$, one may also improve the reverse process with a "conditional DDB" (cDDB) approach. In particular, replace $\boldsymbol{x}_0$ in (5.3) with $\mathbb{E}\{\boldsymbol{x}_0|\widehat{\boldsymbol{x}}_t, \boldsymbol{y}\}$ instead of $\mathbb{E}\{\boldsymbol{x}_0|\widehat{\boldsymbol{x}}_t\}$. This can be accomplished with a DiffPIR-style [114] approach, computing $\mathbb{E}\{\boldsymbol{x}_0|\widehat{\boldsymbol{x}}_t, \boldsymbol{y}\}$ by

$$\arg\min_{\boldsymbol{x}} \frac{\gamma_{\mathsf{w}}}{2} ||\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}||^2 + \frac{\gamma_t}{2} ||\boldsymbol{x} - \widehat{\boldsymbol{x}}_0||^2, \tag{5.7}$$

where γ_{w} and γ_t are the noise precision in $\boldsymbol{y}$ and the precision of the error in $\widehat{\boldsymbol{x}}_0$ relative to the true $\boldsymbol{x}_0$, respectively. γ_t can be estimated as in DDfire. Note that with

the cGAN approach described above, one could choose to compute $\widehat{\boldsymbol{x}}_0$ in a perception-distortion tradeoff-optimal way at inference time via $\widehat{\boldsymbol{x}}_0 = \frac{1}{P}\sum_{i=1}^{P} G_{\boldsymbol{\theta}}(\boldsymbol{z}_i, \widehat{\boldsymbol{x}}_t, t)$ for some appropriate choice of $P \geq 1$.

Another option is to consider an unrolled approach where $G_{\boldsymbol{\theta}}$ is trained with a fixed number of reverse DDB steps in mind, but still leverages our cGAN regularization. Here, the statistical discrepancy between $\boldsymbol{x}_t$ and $\widehat{\boldsymbol{x}}_t$ can be better accounted for in the training process.

5.3 Conclusion

We proposed three methods for solving inverse problems via posterior sampling.

In Chapter 2 we developed a new regularization framework for image-recovery cGANs that combines a supervised ℓ_1 loss with a carefully weighted standard-deviation reward. For Gaussian posteriors, we proved that this approach produces samples matching the true posterior in both mean and covariance, and we identified the shortcomings of alternatives based on ℓ_2 losses. To make the method practical, we proposed an automatic procedure for tuning the variance-reward weight. On tasks such as parallel MRI reconstruction and large-scale face inpainting, our approach consistently outperformed state-of-the-art cGANs and score-based methods across CFID, FID, PSNR, SSIM, LPIPS, and DISTS. Importantly, it also enabled sampling thousands of times faster than Langevin or score-based approaches.

In Chapter 3 we then proposed pcaGAN, an image-recovery cGAN designed to enforce correctness in key statistical components of the conditional distribution: the mean, trace covariance, and the leading K principal components of the covariance matrix $\boldsymbol{\Sigma}_{\widehat{\mathsf{x}}|\mathsf{y}}$. Synthetic Gaussian experiments showed that pcaGAN achieved lower

Wasserstein-2 distances than rcGAN and NPPC across a range of dimensions. On real-world tasks—MNIST denoising, accelerated multicoil MRI, and large-scale inpainting—pcaGAN surpassed competing cGANs and diffusion models in CFID, FID, and standard perceptual/quality metrics. Moreover, pcaGAN achieved these results while sampling 3–4 orders of magnitude faster than diffusion-based competitors, making it well suited for uncertainty quantification, fairness in recovery, and balancing perception-distortion trade-offs.

In Chapter 4 we introduced the Fast Iterative Renoising (FIRE) algorithm, interpretable as a plug-and-play HQS method with a colored renoising step that whitens denoiser input error. Since FIRE approximates the measurement-conditional denoiser $\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_t, \boldsymbol{y}\}$—equivalently, the measurement-conditioned score—it can be seamlessly paired with DDIM to yield the DDfire algorithm for solving linear inverse problems. Across diverse experiments, including box inpainting, Gaussian/motion deblurring, and 4× super-resolution on FFHQ and ImageNet, DDfire consistently outperformed DDRM, ΠGDM, DDS, DiffPIR, DPS, RED-diff, and DAPS in PSNR, LPIPS, and FID. It also delivered superior LPIPS–runtime trade-offs, establishing itself as both fast and high-performing for posterior sampling in linear inverse problems.

Finally, in Section 5.1, we compared rcGAN, pcaGAN, and DDfire for accelerated multicoil MRI reconstruction at accelerations $R \in \{4, 8\}$ using the same data splits and testing protocol as Chapter 3. Quantitatively, DDfire outperforms both cGANs at $R = 8$ across PSNR, SSIM, LPIPS, DISTS, CFID, and FID, albeit with slower sample generation time, but it remains substantially faster than prior diffusion-based MRI methods. However, DDfire falls short of pcaGAN in PSNR, SSIM, LPIPS, and DISTS at $R = 4$ MRI. Qualitatively, DDfire yields higher-fidelity details at $R = 8$

(with some reduction in sample diversity), which align better with the ground truth than rcGAN/pcaGAN.

Bibliography

[1] Asad Aali, Marius Arvinte, Sidharth Kumar, Yamin Ishraq Arefeen, and Jonathan I Tamir. Robust multi-coil mri reconstruction via self-supervised denoising. *Magn. Reson. Med.*, 94:1859 – 1877, 2024.

[2] Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U Rajendra Acharya, et al. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. *Info. Fusion*, 76:243–297, 2021.

[3] Philip M Adamson, Arjun D Desai, Jeffrey Dominic, Christian Bluethgen, Jeff P. Wood, Ali B Syed, Robert D. Boutin, Kathryn J. Stevens, Shreyas Vasanawala, John M. Pauly, Akshay S Chaudhari, and Beliz Gunel. Using deep feature distances for evaluating MR image reconstruction quality. In *Proc. Neural Info. Process. Syst. Workshop*, 2023.

[4] Jonas Adler and Ozan Öktem. Deep Bayesian inversion. *arXiv:1811.05910*, 2018.

[5] R. Ahmad, C. A. Bouman, G. T. Buzzard, S. Chan, S. Liu, E. T. Reehorst, and P. Schniter. Plug and play methods for magnetic resonance imaging. *IEEE Signal Process. Mag.*, 37(1):105–116, March 2020.

[6] Anastasios N. Angelopoulos, Amit P. Kohli, Stephen Bates, Michael I. Jordan, Jitendra Malik, Thayer Alshaabi, Srigokul Upadhyayula, and Yaniv Romano. Image-to-image regression with distribution-free uncertainty quantification and applications in imaging. In *Proc. Intl. Conf. Mach. Learn.*, 2022.

[7] Lynton Ardizzone, Carsten Lüth, Jakob Kruse, Carsten Rother, and Ullrich Köthe. Guided image generation with conditional invertible neural networks. *arXiv:1907.02392*, 2019.

[8] Chinmay Belthangady and Loic A Royer. Applications, promises, and pitfalls of deep learning for fluorescence image reconstruction. *Nature Methods*, 16(12):1215–1225, 2019.

[9] Matthew Bendel, Rizwan Ahmad, and Philip Schniter. A regularized conditional GAN for posterior sampling in inverse problems. In *Proc. Neural Info. Process. Syst. Conf.*, 2023.

[10] Sayantan Bhadra, Varun A Kelkar, Frank J Brooks, and Mark A Anastasio. On hallucinations in tomographic image reconstruction. *IEEE Trans. Med. Imag.*, 40(11):3249–3260, 2021.

[11] Yochai Blau and Tomer Michaeli. The perception-distortion tradeoff. In *Proc. IEEE Conf. Comp. Vision Pattern Recog.*, pages 6228–6237, 2018.

[12] Levi Borodenko. motionblur. Downloaded from `https://github.com/LeviBorodenko/motionblur`, 2020.

[13] Charles A Bouman and Gregery T Buzzard. Generative plug and play: Posterior sampling for inverse problems. In *Proc. Allerton Conf. Commun. Control Comput.*, pages 1–7, 2023.

[14] Dongdong Chen and Mike E Davies. Deep decomposition learning for inverse imaging problems. In *Proc. Europ. Conf. Comp. Vision*, pages 510–526, 2020.

[15] Sitan Chen, Sinho Chewi, Holden Lee, Yuanzhi Li, Jianfeng Lu, and Adil Salim. The probability flow ODE is provably fast. In *Proc. Neural Info. Process. Syst. Conf.*, volume 36, pages 68552–68575, 2023.

[16] Hyungjin Chung, Jeongsol Kim, Michael T McCann, Marc L Klasky, and Jong Chul Ye. diffusion-posterior-sampling. `https://github.com/DPS2022/diffusion-posterior-sampling`, March 2023.

[17] Hyungjin Chung, Jeongsol Kim, Michael T McCann, Marc L Klasky, and Jong Chul Ye. Diffusion posterior sampling for general noisy inverse problems. In *Proc. Intl. Conf. Learn. Rep.*, 2023.

[18] Hyungjin Chung, Jeongsol Kim, and Jong Chul Ye. Direct diffusion bridge using data consistency for inverse problems. In *Proc. Neural Info. Process. Syst. Conf.*, volume 36, 2023.

[19] Hyungjin Chung, Suhyeon Lee, and Jong Chul Ye. DDS. `https://github.com/hyungjin-chung/DDS`, 2024.

[20] Hyungjin Chung, Suhyeon Lee, and Jong Chul Ye. Decomposed diffusion sampler for accelerating large-scale inverse problems. In *Proc. Intl. Conf. Learn. Rep.*, 2024.

[21] Florentin Coeurdoux, Nicolas Dobigeon, and Pierre Chainais. Plug-and-play split Gibbs sampler: Embedding deep generative priors in Bayesian inference. *IEEE Trans. Image Process.*, 33:3496–3507, 2024.

[22] Giannis Daras, Hyungjin Chung, Chieh-Hsin Lai, Yuki Mitsufuji, Jong Chul Ye, Peyman Milanfar, Alexandros G Dimakis, and Mauricio Delbracio. A survey on diffusion models for inverse problems. *arXiv:2410.00083*, 2024.

[23] Mauricio Delbracio and Peyman Milanfar. Inversion by direct iteration: An alternative to denoising diffusion for image restoration. *Trans. on Mach. Learn.*, June 2023.

[24] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. IEEE Conf. Comp. Vision Pattern Recog.*, pages 248–255, 2009.

[25] Puneesh Deora, Bhavya Vasudeva, Saumik Bhattacharya, and Pyari Mohan Pradhan. Structure preserving compressive sensing MRI reconstruction using generative adversarial networks. In *Proc. IEEE Conf. Comp. Vision Pattern Recog. Workshop*, pages 2211–2219, June 2020.

[26] Prafulla Dhariwal and Alex Nichol. Diffusion models beat GANs on image synthesis. In *Proc. Neural Info. Process. Syst. Conf.*, volume 34, pages 8780–8794, 2021.

[27] Keyan Ding, Kede Ma, Shiqi Wang, and Eero P Simoncelli. Image quality assessment: Unifying structure and texture similarity. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(5):2567–2581, 2020.

[28] Vineet Edupuganti, Morteza Mardani, Shreyas Vasanawala, and John Pauly. Uncertainty quantification in deep MRI reconstruction. *IEEE Trans. Med. Imag.*, 40(1):239–250, January 2021.

[29] B. Efron. Tweedie's formula and selection bias. *J. Am. Statist. Assoc.*, 106(496):1602–1614, 2011.

[30] Nina M Gottschling, Vegard Antun, Anders C Hansen, and Ben Adcock. The troublesome kernel—On hallucinations, no free lunches and the accuracy-stability trade-off in inverse problems. *arXiv:2001.01258*, 2023.

[31] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of Wasserstein GANs. In *Proc. Neural Info. Process. Syst. Conf.*, page 5769–5779, 2017.

[32] Zhenliang He, Meina Kan, and Shiguang Shan. EigenGAN: Layer-wise eigen-learning for GANs. In *Proc. IEEE Intl. Conf. Comput. Vis.*, pages 14408–14417, 2021.

[33] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *Proc. Neural Info. Process. Syst. Conf.*, volume 30, 2017.

[34] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Proc. Neural Info. Process. Syst. Conf.*, volume 33, pages 6840–6851, 2020.

[35] David P Hoffman, Isaac Slavitt, and Casey A Fitzpatrick. The promise and peril of deep learning in microscopy. *Nature Methods*, 18(2):131–132, 2021.

[36] Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. *J. Mach. Learn. Res.*, 6:695–709, 2005.

[37] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proc. IEEE Conf. Comp. Vision Pattern Recog.*, pages 1125–1134, 2017.

[38] Ajil Jalal, Marius Arvinte, Giannis Daras, Eric Price, Alex Dimakis, and Jonathan Tamir. csgm-mri-langevin. `https://github.com/utcsilab/csgm-mri-langevin`, 2021. Accessed: 2021-12-05.

[39] Ajil Jalal, Marius Arvinte, Giannis Daras, Eric Price, Alex Dimakis, and Jonathan Tamir. Robust compressed sensing MRI with deep generative priors. In *Proc. Neural Info. Process. Syst. Conf.*, 2021.

[40] I. T. Jolliffe. *Principal Component Analysis*, volume 2. Springer Verlag, New York, 2002.

[41] Mihir Joshi, Aaron Pruitt, Chong Chen, Yingmin Liu, and Rizwan Ahmad. Technical report (v1.0)–pseudo-random cartesian sampling for dynamic MRI. *arXiv:2206.03630*, 2022.

[42] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *Proc. Intl. Conf. Learn. Rep.*, 2018.

[43] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. In *Proc. Neural Info. Process. Syst. Conf.*, volume 33, pages 12104–12114, 2020.

[44] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proc. IEEE Conf. Comp. Vision Pattern Recog.*, pages 4396–4405, 2019.

[45] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *Proc. IEEE Conf. Comp. Vision Pattern Recog.*, pages 8110–8119, 2020.

[46] Segrey Kastryulin, Jamil Zakirov, Nicola Pezzotti, and Dmitry V. Dylov. Image quality assessment for magnetic resonance imaging. *arXiv:2203.07809*, 2022.

[47] Sergey Kastryulin, Jamil Zakirov, Nicola Pezzotti, and Dmitry V Dylov. Image quality assessment for magnetic resonance imaging. *IEEE Access*, 11:14154–14168, 2023.

[48] Bahjat Kawar, Michael Elad, Stefano Ermon, and Jiaming Song. Denoising diffusion restoration models. In *Proc. Neural Info. Process. Syst. Conf.*, 2022.

[49] Bahjat Kawar, Michael Elad, Stefano Ermon, and Jiaming Song. Denoising diffusion restoration models. Downloaded from `https://github.com/bahjat-kawar/ddrm`, May 2022.

[50] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. Intl. Conf. Learn. Rep.*, 2015.

[51] Tuomas Kynkäänniemi, Tero Karras, Miika Aittala, Timo Aila, and Jaakko Lehtinen. The role of imagenet classes in fréchet inception distance. *arXiv:2203.06026*, 2022.

[52] Benjamin Lambert, Florence Forbes, Senan Doyle, Harmonie Dehaene, and Michel Dojat. Trustworthy clinical AI solutions: A unified review of uncertainty quantification in deep learning models for medical image analysis. *Artificial Intelligence in Medicine*, page 102830, 2024.

[53] Rémi Laumont, Valentin De Bortoli, Andrés Almansa, Julie Delon, Alain Durmus, and Marcelo Pereyra. Bayesian imaging using plug & play priors: When Langevin meets Tweedie. *SIAM J. Imag. Sci.*, 15(2):701–737, 2022.

[54] Fred C Leone, Lloyd S Nelson, and RB Nottingham. The folded normal distribution. *Technometrics*, 3(4):543–550, 1961.

[55] Guan-Horng Liu, Arash Vahdat, De-An Huang, Evangelos Theodorou, Weili Nie, and Anima Anandkumar. I^2SB: Image-to-image Schrödinger bridge. In *Proc. Intl. Conf. Mach. Learn.*, pages 22042–22062, 2023.

[56] David G Luenberger and Yinyu Ye. *Linear and Nonlinear Programming*. Springer, 2016.

[57] Morteza Mardani, Jiaming Song, Jan Kautz, and Arash Vahdat. A variational perspective on solving inverse problems with diffusion models. In *Proc. Intl. Conf. Learn. Rep.*, 2024.

[58] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv:1411.1784*, 2014.

[59] Matthew J Muckley, Bruno Riemenschneider, Alireza Radmanesh, Sunwoo Kim, Geunu Jeong, Jingyu Ko, Yohan Jun, Hyungseob Shin, Dosik Hwang, Mahmoud Mostapha, et al. Results of the 2020 fastMRI challenge for machine learning MR image reconstruction. *IEEE Trans. Med. Imag.*, 40(9):2306–2317, 2021.

[60] Dominik Narnhofer, Andreas Habring, Martin Holler, and Thomas Pock. Posterior-variance-based error quantification for inverse problems in imaging. *SIAM J. Imag. Sci.*, 17(1):301–333, 2024.

[61] Elias Nehme, Omer Yair, and Tomer Michaeli. Uncertainty quantification via neural posterior principal components. In *Proc. Neural Info. Process. Syst. Conf.*, 2023.

[62] Elias Nehme, Omer Yair, and Tomer Michaeli. Uncertainty quantification via neural posterior principal components. Downloaded from `https://github.com/EliasNehme/NPPC`, January 2023.

[63] NVlabs. RED-diff. Downloaded from `https://github.com/NVlabs/RED-diff`, 2023.

[64] Guy Ohayon, Theo Adrai, Gregory Vaksman, Michael Elad, and Peyman Milanfar. High perceptual quality image denoising with a posterior sampling CGAN. In *Proc. IEEE Intl. Conf. Comput. Vis. Workshops*, volume 10, pages 1805–1813, 2021.

[65] Guy Ohayon, Theo Adrai, Gregory Vaksman, Michael Elad, and Peyman Milanfar. High perceptual quality image denoising with a posterior sampling CGAN. Downloaded from `https://github.com/theoad/pscgan`, July 2021.

[66] Guy Ohayon, Theo Joseph Adrai, Michael Elad, and Tomer Michaeli. Reasons for the superiority of stochastic estimators over deterministic ones: Robustness, consistency and perceptual quality. In *Proc. Intl. Conf. Mach. Learn.*, pages 26474–26494, 2023.

[67] Frank Ong and Michael Lustig. SigPy: A python package for high performance iterative reconstruction. In *Proc. Annu. Meeting ISMRM*, volume 4819, 2019.

[68] Beresford N Parlett. *The symmetric eigenvalue problem*. SIAM, 1998.

[69] H. V. Poor. *An Introduction to Signal Detection and Estimation*. Springer, New York, NY, USA, 2nd edition, 1994.

[70] Klaas P. Pruessmann, Markus Weiger, Markus B. Scheidegger, and Peter Boesiger. SENSE: Sensitivity encoding for fast MRI. *Magn. Reson. Med.*, 42(5):952–962, 1999.

[71] Edward T. Reehorst and Philip Schniter. Regularization by denoising: Clarifications and new interpretations. *IEEE Trans. Comput. Imag.*, 5(1):52–67, March 2019.

[72] Marien Renaud, Jean Prost, Arthur Leclaire, and Nicolas Papadakis. Plug-and-play image restoration with stochastic denoising regularization. In *Proc. Intl. Conf. Mach. Learn.*, 2024.

[73] Yaniv Romano, Michael Elad, and Peyman Milanfar. The little engine that could: Regularization by denoising (RED). *SIAM J. Imag. Sci.*, 10(4):1804–1844, 2017.

[74] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Proc. Intl. Conf. Med. Image Comput. Comput. Assist. Intervent.*, pages 234–241, 2015.

[75] Rohan Sanda, Asad Aali, Andrew Johnston, Eduardo Reis, Jonathan Singh, Gordon Wetzstein, and Sara Fridovich-Keil. Padis-mri: Patch-based diffusion for data-efficient, radiologist-preferred mri reconstruction. `https://github.com/voilalab/PaDIS-MRI`, 2025. Accessed: 2025-10-07.

[76] Edgar Schonfeld, Bernt Schiele, and Anna Khoreva. A U-Net based discriminator for generative adversarial networks. In *Proc. IEEE Conf. Comp. Vision Pattern Recog.*, pages 8207–8216, 2020.

[77] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

[78] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proc. Intl. Conf. Mach. Learn.*, pages 2256–2265, 2015.

[79] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Proc. Neural Info. Process. Syst. Conf.*, 2015.

[80] Michael Soloveitchik, Tzvi Diskin, Efrat Morin, and Ami Wiesel. Conditional Frechet inception distance. *arXiv:2103.11521*, 2021.

[81] Casper Kaae Sønderby, Jose Caballero, Lucas Theis, Wenzhe Shi, and Ferenc Huszár. Amortised MAP inference for image super-resolution. In *Proc. Intl. Conf. Learn. Rep.*, 2017.

[82] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *Proc. Intl. Conf. Learn. Rep.*, 2021.

[83] Jiaming Song, Arash Vahdat, Morteza Mardani, and Jan Kautz. Pseudoinverse-guided diffusion models for inverse problems. In *Proc. Intl. Conf. Learn. Rep.*, 2023.

[84] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *Proc. Neural Info. Process. Syst. Conf.*, 2019.

[85] Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. In *Proc. Neural Info. Process. Syst. Conf.*, 2020.

[86] Yang Song, Liyue Shen, Lei Xing, and Stefano Ermon. Solving inverse problems in medical imaging with score-based generative models. In *Proc. Intl. Conf. Learn. Rep.*, 2022.

[87] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *Proc. Intl. Conf. Learn. Rep.*, 2021.

[88] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. Downloaded from `https://github.com/yang-song/score_sde_pytorch`, October 2022.

[89] Anuroop Sriram, Jure Zbontar, Tullie Murrell, Aaron Defazio, C. Lawrence Zitnick, Nafissa Yakubova, Florian Knoll, and Patricia Johnson. End-to-end variational networks for accelerated MRI reconstruction. In *Proc. Intl. Conf. Med. Image Comput. Comput. Assist. Intervent.*, pages 64–73, 2020.

[90] He Sun and Katherine L Bouman. Deep probabilistic imaging: Uncertainty quantification and multi-modal solution characterization for computational imaging. In *Proc. AAAI Conf. Artificial Intell.*, volume 35, pages 2628–2637, 2021.

[91] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proc. IEEE Conf. Comp. Vision Pattern Recog.*, 2016.

[92] Jacopo Teneggi, Matthew Tivnan, J. Webster Stayman, and Jeremias Sulam. How to trust your diffusion model: A convex optimization approach to conformal risk control, 2023.

[93] Francesco Tonolini, Jack Radford, Alex Turpin, Daniele Faccio, and Roderick Murray-Smith. Variational inference for computational imaging inverse problems. *J. Mach. Learn. Res.*, 21(179):1–46, 2020.

[94] Martin Uecker, Peng Lai, Mark J Murphy, Patrick Virtue, Michael Elad, John M Pauly, Shreyas S Vasanawala, and Michael Lustig. ESPIRiT–an eigenvalue approach to autocalibrating parallel MRI: Where SENSE meets GRAPPA. *Magn. Reson. Med.*, 71(3):990–1001, 2014.

[95] Y. Wang. DDNM. Downloaded from https://github.com/wyhuai/DDNM, September 2023.

[96] Yinhuai Wang, Jiwen Yu, and Jian Zhang. Zero-shot image restoration using denoising diffusion null-space model. In *Proc. Intl. Conf. Learn. Rep.*, 2023.

[97] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proc. Intl. Conf. Mach. Learn.*, pages 681–688, 2011.

[98] Jeffrey Wen, Rizwan Ahmad, and Philip Schniter. A conditional normalizing flow for accelerated multi-coil MR imaging. In *Proc. Intl. Conf. Mach. Learn.*, 2023.

[99] Christina Winkler, Daniel Worrall, Emiel Hoogeboom, and Max Welling. Learning likelihoods with conditional normalizing flows. *arXiv:1912.00042*, 2019.

[100] Zihui Wu, Yu Sun, Yifan Chen, Bingliang Zhang, Yisong Yue, and Katherine Bouman. Principled probabilistic imaging using diffusion models as plug-and-play priors. In *Proc. Neural Info. Process. Syst. Conf.*, 2024.

[101] Xingyu Xu and Yuejie Chi. Provably robust score-based diffusion posterior sampling for plug-and-play image reconstruction. In *Proc. Neural Info. Process. Syst. Conf.*, 2024.

[102] Jure Zbontar, Florian Knoll, Anuroop Sriram, Matthew J. Muckley, Mary Bruno, Aaron Defazio, Marc Parente, Krzysztof J. Geras, Joe Katsnelson, Hersh Chandarana, Zizhao Zhang, Michal Drozdzal, Adriana Romero, Michael Rabbat, Pascal Vincent, James Pinkerton, Duo Wang, Nafissa Yakubova, Erich Owens, C. Lawrence Zitnick, Michael P. Recht, Daniel K. Sodickson, and Yvonne W. Lui. fastMRI: An open dataset and benchmarks for accelerated MRI. *arXiv:1811.08839*, 2018.

[103] Yu Zeng. co-mod-gan-pytorch. Downloaded from `https://github.com/zengxianyu/co-mod-gan-pytorch`, September 2022.

[104] Bingliang Zhang, Wenda Chu, Julius Berner, Chenlin Meng, Anima Anandkumar, and Yang Song. Improving diffusion inverse problem solving with decoupled noise annealing. Downloaded from `https://github.com/zhangbingliang2019/DAPS`, 2024.

[105] Bingliang Zhang, Wenda Chu, Julius Berner, Chenlin Meng, Anima Anandkumar, and Yang Song. Improving diffusion inverse problem solving with decoupled noise annealing. In *Proc. IEEE Conf. Comp. Vision Pattern Recog.*, 2025.

[106] Kai Zhang, Yawei Li, Wangmeng Zuo, Lei Zhang, Luc Van Gool, and Radu Timofte. Plug-and-play image restoration with deep denoiser prior. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(10):6360–6376, 2021.

[107] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proc. IEEE Conf. Comp. Vision Pattern Recog.*, pages 586–595, 2018.

[108] Tao Zhang, John M Pauly, Shreyas S Vasanawala, and Michael Lustig. Coil compression for accelerated imaging with Cartesian sampling. *Magn. Reson. Med.*, 69(2):571–582, 2013.

[109] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for image restoration with neural networks. *IEEE Trans. Comput. Imag.*, 3(1):47–57, March 2017.

[110] He Zhao, Huiqi Li, Sebastian Maurer-Stroh, and Li Cheng. Synthesizing retinal and neuronal images with generative adversarial nets. *Med. Image Analysis*, 49, 07 2018.

[111] Shengyu Zhao, Jonathan Cui, Yilun Sheng, Yue Dong, Xiao Liang, Eric I-Chao Chang, and Yan Xu. Large scale image completion via co-modulated generative adversarial networks. In *Proc. Intl. Conf. Learn. Rep.*, 2021.

[112] Zhengli Zhao, Sameer Singh, Honglak Lee, Zizhao Zhang, Augustus Odena, and Han Zhang. Improved consistency regularization for GANs. In *Proc. AAAI Conf. Artificial Intell.*, volume 35, pages 11033–11041, 2021.

[113] Yuanzhi Zhu, Kai Zhang, Jingyun Liang, Jiezhang Cao, Bihan Wen, Radu Timofte, and Luc Van Gool. Diffpir. Downloaded from `https://github.com/yuanzhi-zhu/DiffPIR`, July 2024.

[114] Yuanzhi Zhu, Kai Zhang, Jingyun Liang, Jiezhang Cao, Bihan Wen, Radu Timofte, and Luc Van Gool. Denoising diffusion models for plug-and-play image restoration. In *Proc. IEEE Conf. Comp. Vision Pattern Recog.*, pages 1219–1229, 2023.

Appendix A: Additional Details for Inverse Problems

A.1 MR Imaging Details

We now give details on magnetic resonance (MR) image recovery. Suppose that the goal is to recover the N-pixel MR image $\boldsymbol{i} \in \mathbb{C}^N$ from the multicoil measurements $\{\boldsymbol{k}_c\}_{c=1}^C$, where [70]

$$\boldsymbol{k}_c = \boldsymbol{M}\boldsymbol{F}\boldsymbol{S}_c\boldsymbol{i} + \boldsymbol{n}_c. \tag{A.1}$$

In (A.1), C refers to the number of coils, $\boldsymbol{k}_c \in \mathbb{C}^M$ are the measurements from the cth coil, $\boldsymbol{M} \in \mathbb{R}^{M \times N}$ is a sub-sampling operator containing rows from $\boldsymbol{I}_N$—the $N \times N$ identity matrix, $\boldsymbol{F} \in \mathbb{C}^{N \times N}$ is the unitary 2D discrete Fourier transform, $\boldsymbol{S}_c \in \mathbb{C}^{N \times N}$ is a diagonal matrix containing the sensitivity map of the cth coil, and $\boldsymbol{n}_c \in \mathbb{C}^M$ is noise. From (A.1), it can be seen that the MR measurements are collected in the spatial Fourier domain, otherwise known as the "k-space." The sensitivity maps $\{\boldsymbol{S}_c\}$ are estimated from $\{\boldsymbol{k}_c\}$ using ESPIRiT [94] (in our case via SigPy [67]), which yields maps with the property $\sum_{c=1}^C \boldsymbol{S}_c^{\mathsf{H}}\boldsymbol{S}_c = \boldsymbol{I}_N$. The ratio $R \triangleq \frac{N}{M}$ is known as the acceleration rate.

There are different ways that one could apply the generative posterior sampling framework to multicoil MR image recovery. One is to configure the generator to produce posterior samples $\widehat{\boldsymbol{i}}$ of the complex image $\boldsymbol{i}$. Another is to configure the

generator to produce posterior samples $\widehat{\boldsymbol{x}}$ of the stack $\boldsymbol{x} \triangleq [\boldsymbol{x}_1^\top, \ldots, \boldsymbol{x}_C^\top]^\top$ of "coil images" $\boldsymbol{x}_c \triangleq \boldsymbol{S}_c \boldsymbol{i}$ and later coil-combining them to yield a complex image estimate $\widehat{\boldsymbol{i}} \triangleq [\boldsymbol{S}_1^{\mathsf{H}}, \ldots, \boldsymbol{S}_C^{\mathsf{H}}]\widehat{\boldsymbol{x}}$. We take the latter approach. Furthermore, rather than feeding our generator with k-space measurements $\boldsymbol{k}_c$, we choose to feed it with aliased coil images $\boldsymbol{y}_c \triangleq \boldsymbol{F}^{\mathsf{H}} \boldsymbol{M}^\top \boldsymbol{k}_c$. Writing (A.1) in terms of the coil images, we obtain

$$\boldsymbol{y}_c = \boldsymbol{F}^{\mathsf{H}} \boldsymbol{M}^\top \boldsymbol{M} \boldsymbol{F} \boldsymbol{x}_c + \boldsymbol{w}_c, \tag{A.2}$$

where $\boldsymbol{w}_c \triangleq \boldsymbol{F}^{\mathsf{H}} \boldsymbol{M}^\top \boldsymbol{n}_c$. Then we can stack $\{\boldsymbol{y}_c\}$ and $\{\boldsymbol{w}_c\}$ column-wise into vectors $\boldsymbol{y}$ and $\boldsymbol{w}$, and set $\boldsymbol{A} = \boldsymbol{I}_C \otimes \boldsymbol{F}^{\mathsf{H}} \boldsymbol{M}^\top \boldsymbol{M} \boldsymbol{F} \in \mathbb{C}^{NC \times NC}$, to obtain the formulation $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{w}$.

To train our generator, we assume to have access to paired training examples $\{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}$, where $\boldsymbol{x}_t$ is a stack of coil images and $\boldsymbol{y}_t$ is the corresponding stack of kf-space coil measurements. The fastMRI multicoil dataset [102] provides $\{(\boldsymbol{x}_t, \boldsymbol{k}_t)\}$, from which we can easily obtain $\{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}$.

A.2 Data-Consistency for Inverse Problems

In this section, we describe a data-consistency procedure that can be optionally used when our cGAN is used to solve a *linear* inverse problem, i.e., to recover $\boldsymbol{x}$ from $\boldsymbol{y}$ under a model of the form

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{w}, \tag{A.3}$$

where $\boldsymbol{A}$ is a known linear operator and $\boldsymbol{w}$ is unknown noise. The motivation is that, in some applications, such as medical imaging or inpainting, the end user may feel comfortable knowing that the generated samples $\{\widehat{\boldsymbol{x}}_i\}$ are consistent with the

measurements $\boldsymbol{y}$ in that

$$\boldsymbol{y} = \boldsymbol{A}\widehat{\boldsymbol{x}}_i. \tag{A.4}$$

When (A.4) holds, $\boldsymbol{A}^+\boldsymbol{y} = \boldsymbol{A}^+\boldsymbol{A}\widehat{\boldsymbol{x}}_i$ must also hold, where $(\cdot)^+$ denotes the pseudo-inverse. The quantity $\boldsymbol{A}^+\boldsymbol{A}$ can be recognized as the orthogonal projection matrix associated with the row space of $\boldsymbol{A}$. So, (A.4) requires the component of $\widehat{\boldsymbol{x}}_i$ in the row space of $\boldsymbol{A}$ to equal $\boldsymbol{A}^+\boldsymbol{y}$, while placing no constraints on the component of $\widehat{\boldsymbol{x}}_i$ in the nullspace of $\boldsymbol{A}$. This suggests the following data-consistency procedure:

$$\widehat{\boldsymbol{x}}_i = (\boldsymbol{I} - \boldsymbol{A}^+\boldsymbol{A})\widehat{\boldsymbol{x}}_i^{\mathsf{raw}} + \boldsymbol{A}^+\boldsymbol{y}. \tag{A.5}$$

where $\widehat{\boldsymbol{x}}_i^{\mathsf{raw}}$ is the raw generator output. We note that a version of this idea for point estimation was proposed in [81].

The data-consistency procedure (A.5) ensures that any generative method will generate only the component of $\boldsymbol{x}$ that lies in the nullspace of $\boldsymbol{A}$. Consequently, (A.5) is admissible only when $\boldsymbol{A}$ has a non-trivial nullspace. Also, because no attempt is made to remove the noise $\boldsymbol{w}$ in $\boldsymbol{y}$, this approach is recommended only for low-noise applications. For high-noise applications, an extension based on the dual-decomposition approach [14] would be more appropriate, but we leave this to future work.

When applying (A.5) to the MRI formulation in App. A.1, we note that $\boldsymbol{A} = \boldsymbol{I}_C \otimes \boldsymbol{F}^{\mathsf{H}}\boldsymbol{M}^\top\boldsymbol{M}\boldsymbol{F}$ is an orthogonal projection matrix, and so $\boldsymbol{I} - \boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{I} - \boldsymbol{A} = \boldsymbol{I} \otimes \boldsymbol{F}^{\mathsf{H}}(\boldsymbol{I} - \boldsymbol{M}^\top\boldsymbol{M})\boldsymbol{F}$.

Appendix B: Additional Diffusion Details

B.1 VP Formulation

In the main text, we describe DDfire for the VE SDE formulation from [87] and the corresponding SMLD discretization from [84]. Here, we describe it for the VP SDE from [87] and the corresponding DDPM discretization from [34].

From [87], the general SDE forward process can be written as

$$\mathrm{d}\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x}, t)\,\mathrm{d}t + g(t)\,\mathrm{d}\boldsymbol{w} \tag{B.1}$$

for some choices of $\boldsymbol{f}(\cdot,\cdot)$ and $g(\cdot)$, where $\mathrm{d}\boldsymbol{w}$ is the standard Wiener process (i.e., Brownian motion). The reverse process can then be described by

$$\mathrm{d}\boldsymbol{x} = \big(\boldsymbol{f}(\boldsymbol{x}, t) - g^2(t)\nabla_{\boldsymbol{x}} \ln p_t(\boldsymbol{x})\big)\,\mathrm{d}t + g(t)\,\mathrm{d}\overline{\boldsymbol{w}}, \tag{B.2}$$

where $p_t(\cdot)$ is the distribution of $\boldsymbol{x}$ at time t and $\mathrm{d}\overline{\boldsymbol{w}}$ is the reverse Wiener process. In the VE-SDE, $\boldsymbol{f}(\boldsymbol{x}, t) = \boldsymbol{0}$ and $g(t) = \sqrt{\mathrm{d}[\sigma^2(t)]/\,\mathrm{d}t}$ for some variance schedule $\sigma^2(t)$, but in the VP-SDE, $\boldsymbol{f}(\boldsymbol{x}, t) = -\frac{1}{2}\beta(t)\boldsymbol{x}$ and $g(t) = \sqrt{\beta(t)}$ for some variance schedule $\beta(t)$. When discretized to $t \in \{0, 1, \ldots, T\}$, the VP forward process becomes

$$\widetilde{\boldsymbol{x}}_t = \sqrt{1-\beta_t}\widetilde{\boldsymbol{x}}_{t-1} + \sqrt{\beta_t}\widetilde{\boldsymbol{w}}_{t-1} \tag{B.3}$$

with i.i.d. $\{\widetilde{\boldsymbol{w}}_t\} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, so that

$$\widetilde{\boldsymbol{x}}_t = \sqrt{\overline{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\overline{\alpha}_t}\widetilde{\boldsymbol{\epsilon}}_t \tag{B.4}$$

with $\alpha_t \triangleq 1 - \beta_t$, $\overline{\alpha}_t = \prod_{s=1}^{t} \alpha_s$, and $\widetilde{\boldsymbol{\epsilon}}_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Throughout, we write the VP quantities with tildes to distinguish them from the VE quantities. The DDPM reverse process then takes the form

$$\widetilde{\boldsymbol{x}}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\big(\widetilde{\boldsymbol{x}}_t + \beta_t \nabla_{\boldsymbol{x}_t} \ln p(\widetilde{\boldsymbol{x}}_t)\big) + \Sigma_t \widetilde{\boldsymbol{n}}_t \quad \text{with} \quad \Sigma_t^2 \triangleq \frac{1-\overline{\alpha}_{t-1}}{1-\overline{\alpha}_t}\beta_t \tag{B.5}$$

and is typically initialized at $\widetilde{\boldsymbol{x}}_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. By rewriting (B.4) as

$$\frac{1}{\sqrt{\overline{\alpha}_t}}\widetilde{\boldsymbol{x}}_t = \boldsymbol{x}_0 + \sqrt{\frac{1-\overline{\alpha}_t}{\overline{\alpha}_t}}\widetilde{\boldsymbol{\epsilon}}_t \tag{B.6}$$

and comparing it to (4.1), we recognize the VP/VE relationships

$$\frac{1}{\sqrt{\overline{\alpha}_t}}\widetilde{\boldsymbol{x}}_t = \boldsymbol{x}_t \quad \text{and} \quad \frac{1-\overline{\alpha}_t}{\overline{\alpha}_t} = \sigma_t^2 \;\Leftrightarrow\; \overline{\alpha}_t = \frac{1}{1+\sigma_t^2}. \tag{B.7}$$

Furthermore, assuming that $\overline{\alpha}_T \ll 1$, the VP initialization $\widetilde{\boldsymbol{x}}_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is well approximated by the VE initialization $\boldsymbol{x}_T \sim \mathcal{N}(\mathbf{0}, \sigma_T^2 \boldsymbol{I})$.

B.2 DDIM Details for VP

The DDIM reverse process from [82] provides an alternative to the DDPM reverse process that offers a flexible level of stochasticity. When describing VP DDIM, we will write the quantities as $\widetilde{\boldsymbol{x}}_k, \widetilde{\boldsymbol{\epsilon}}_k, \widetilde{\boldsymbol{n}}_k, \overline{\alpha}_k$ to distinguish them from the corresponding VP DDPM quantities $\widetilde{\boldsymbol{x}}_t, \widetilde{\boldsymbol{\epsilon}}_t, \widetilde{\boldsymbol{n}}_t, \overline{\alpha}_t$, and we will write the total number of steps as K. Like (B.4), DDIM is built around the model

$$\widetilde{\boldsymbol{x}}_k = \sqrt{\overline{\alpha}_k}\boldsymbol{x}_0 + \sqrt{1-\overline{\alpha}_k}\widetilde{\boldsymbol{\epsilon}}_k, \quad \widetilde{\boldsymbol{\epsilon}}_k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}). \tag{B.8}$$

Adapting the first two equations from [82, App.D.3] to our notation, we have

$$\widetilde{\boldsymbol{x}}_{k-1} = \sqrt{\overline{\alpha}_{k-1}}\left(\frac{\widetilde{\boldsymbol{x}}_k - \sqrt{1-\overline{\alpha}_k}\,\mathbb{E}\{\widetilde{\boldsymbol{\epsilon}}_k|\widetilde{\boldsymbol{x}}_k, \boldsymbol{y}\}}{\sqrt{\overline{\alpha}_k}}\right) + \sqrt{1-\overline{\alpha}_{k-1}-\widetilde{\varsigma}_k^2}\;\mathbb{E}\{\widetilde{\boldsymbol{\epsilon}}_k|\widetilde{\boldsymbol{x}}_k, \boldsymbol{y}\} + \widetilde{\varsigma}_k\widetilde{\boldsymbol{n}}_k \tag{B.9}$$

$$\widetilde{\varsigma}_k \triangleq \eta_{\mathsf{ddim}}\sqrt{\frac{1-\overline{\alpha}_{k-1}}{1-\overline{\alpha}_k}}\sqrt{1-\frac{\overline{\alpha}_k}{\overline{\alpha}_{k-1}}} \tag{B.10}$$

with $\widetilde{\boldsymbol{n}}_k \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and tunable $\eta_{\mathsf{ddim}} \geq 0$. When $\eta_{\mathsf{ddim}} = 1$ and $K = T$, DDIM reduces to DDPM. But when $\eta_{\mathsf{ddim}} = 0$, the reverse process is deterministic. In fact, it can be considered a discretization of the probability flow ODE [82], which often works much better than the SDE when the number of discretization steps K is small. We now write (B.9) in a simpler form. Applying $\mathbb{E}\{\cdot|\widetilde{\boldsymbol{x}}_k, \boldsymbol{y}\}$ to both sides of (B.8) gives

$$\widetilde{\boldsymbol{x}}_k = \sqrt{\overline{\alpha}_k}\,\mathbb{E}\{\boldsymbol{x}_0|\widetilde{\boldsymbol{x}}_k, \boldsymbol{y}\} + \sqrt{1-\overline{\alpha}_k}\,\mathbb{E}\{\widetilde{\boldsymbol{\epsilon}}_k|\widetilde{\boldsymbol{x}}_k, \boldsymbol{y}\} \tag{B.11}$$

which implies

$$\mathbb{E}\{\widetilde{\boldsymbol{\epsilon}}_k|\widetilde{\boldsymbol{x}}_k, \boldsymbol{y}\} = \frac{\widetilde{\boldsymbol{x}}_k - \sqrt{\overline{\alpha}_k}\,\mathbb{E}\{\boldsymbol{x}_0|\widetilde{\boldsymbol{x}}_k, \boldsymbol{y}\}}{\sqrt{1-\overline{\alpha}_k}}, \tag{B.12}$$

and plugging (B.12) into (B.9) gives

$$\begin{aligned}
\widetilde{\boldsymbol{x}}_{k-1} &= \sqrt{\overline{\alpha}_{k-1}}\,\mathbb{E}\{\boldsymbol{x}_0|\widetilde{\boldsymbol{x}}_k, \boldsymbol{y}\} + \widetilde{\varsigma}_k\widetilde{\boldsymbol{n}}_k \\
&\quad + \sqrt{1-\overline{\alpha}_{k-1}-\widetilde{\varsigma}_k^2}\left(\frac{\widetilde{\boldsymbol{x}}_k - \sqrt{\overline{\alpha}_k}\,\mathbb{E}\{\boldsymbol{x}_0|\widetilde{\boldsymbol{x}}_k, \boldsymbol{y}\}}{\sqrt{1-\overline{\alpha}_k}}\right) && \text{(B.13)} \\
&= \sqrt{\overline{\alpha}_{k-1}}\,\mathbb{E}\{\boldsymbol{x}_0|\widetilde{\boldsymbol{x}}_k, \boldsymbol{y}\} + \widetilde{\varsigma}_k\widetilde{\boldsymbol{n}}_k \\
&\quad + \widetilde{h}_k\big(\widetilde{\boldsymbol{x}}_k - \sqrt{\overline{\alpha}_k}\,\mathbb{E}\{\boldsymbol{x}_0|\widetilde{\boldsymbol{x}}_k, \boldsymbol{y}\}\big) && \text{(B.14)} \\
&= \widetilde{h}_k\widetilde{\boldsymbol{x}}_k + \widetilde{g}_k\,\mathbb{E}\{\boldsymbol{x}_0|\widetilde{\boldsymbol{x}}_k, \boldsymbol{y}\} + \widetilde{\varsigma}_k\widetilde{\boldsymbol{n}}_k && \text{(B.15)}
\end{aligned}$$

for

$$\widetilde{h}_k \triangleq \sqrt{\frac{1-\overline{\alpha}_{k-1}-\widetilde{\varsigma}_k^2}{1-\overline{\alpha}_k}} \quad \text{and} \quad \widetilde{g}_k \triangleq \sqrt{\overline{\alpha}_{k-1}} - \widetilde{h}_k\sqrt{\overline{\alpha}_k}. \tag{B.16}$$

Thus the VP DDIM reverse process can be described by (B.10), (B.15), and (B.16) with $\widetilde{\boldsymbol{n}}_k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ $\forall k$ and initialization $\widetilde{\boldsymbol{x}}_K \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

B.3 DDIM Details for VE

We now provide the details for the VE version of DDIM. Starting with the VP DDIM reverse process (B.15), we can divide both sides by $\sqrt{\overline{\alpha}_{k-1}}$ to get

$$\frac{\widetilde{\boldsymbol{x}}_{k-1}}{\sqrt{\overline{\alpha}_{k-1}}} = \frac{\widetilde{h}_k\sqrt{\overline{\alpha}_k}}{\sqrt{\overline{\alpha}_{k-1}}}\frac{\widetilde{\boldsymbol{x}}_k}{\sqrt{\overline{\alpha}_k}} + \frac{\widetilde{g}_k}{\sqrt{\overline{\alpha}_{k-1}}}\,\mathbb{E}\{\boldsymbol{x}_0|\widetilde{\boldsymbol{x}}_k, \boldsymbol{y}\} + \frac{\widetilde{\varsigma}_k}{\sqrt{\overline{\alpha}_{k-1}}}\widetilde{\boldsymbol{n}}_k \tag{B.17}$$

and leveraging the VP-to-VE relationship (B.7) to write

$$\boldsymbol{x}_{k-1} = h_k\boldsymbol{x}_k + g_k\,\mathbb{E}\{\boldsymbol{x}_0|\boldsymbol{x}_k, \boldsymbol{y}\} + \varsigma_k\boldsymbol{n}_k \tag{B.18}$$

with

$$h_k = \frac{\widetilde{h}_k\sqrt{\overline{\alpha}_k}}{\sqrt{\overline{\alpha}_{k-1}}}, \quad g_k = \frac{\widetilde{g}_k}{\sqrt{\overline{\alpha}_{k-1}}}, \quad \varsigma_k = \frac{\widetilde{\varsigma}_k}{\sqrt{\overline{\alpha}_{k-1}}} \tag{B.19}$$

with $\boldsymbol{n}_k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ $\forall k$ and initialization $\boldsymbol{x}_K \sim \mathcal{N}(\mathbf{0}, \sigma_K^2\boldsymbol{I})$. Plugging $\widetilde{g}_k$ from (B.16) into (B.18), we find

$$g_k = \frac{\sqrt{\overline{\alpha}_{k-1}} - \widetilde{h}_k\sqrt{\overline{\alpha}_k}}{\sqrt{\overline{\alpha}_{k-1}}} = 1 - h_k. \tag{B.20}$$

Then plugging (B.10) into (B.18), we find

$$\varsigma_k = \frac{\eta_{\mathsf{ddim}}}{\sqrt{\overline{\alpha}_{k-1}}}\sqrt{\frac{1-\overline{\alpha}_{k-1}}{1-\overline{\alpha}_k}}\sqrt{1-\frac{\overline{\alpha}_k}{\overline{\alpha}_{k-1}}} \tag{B.21}$$

$$= \eta_{\mathsf{ddim}}\sqrt{\frac{1}{\overline{\alpha}_{k-1}}\frac{1-\overline{\alpha}_{k-1}}{1-\overline{\alpha}_k}\left(1-\frac{\overline{\alpha}_k}{\overline{\alpha}_{k-1}}\right)} \tag{B.22}$$

$$= \eta_{\mathsf{ddim}}\sqrt{\frac{1-\overline{\alpha}_{k-1}}{\overline{\alpha}_{k-1}}\frac{\overline{\alpha}_k}{1-\overline{\alpha}_k}\left(\frac{1}{\overline{\alpha}_k}-\frac{1}{\overline{\alpha}_{k-1}}\right)} \tag{B.23}$$

$$= \eta_{\mathsf{ddim}}\sqrt{\frac{\sigma_{k-1}^2}{\sigma_k^2}\left([1+\sigma_k^2]-[1+\sigma_{k-1}^2]\right)} \tag{B.24}$$

$$= \eta_{\mathsf{ddim}}\sqrt{\frac{\sigma_{k-1}^2(\sigma_k^2-\sigma_{k-1}^2)}{\sigma_k^2}}. \tag{B.25}$$

Finally, noting from (B.7), (B.18), and (B.25) that

$$\frac{\widetilde{\varsigma}_k^2}{1-\overline{\alpha}_{k-1}} = \varsigma_k^2 \frac{\overline{\alpha}_{k-1}}{1-\overline{\alpha}_{k-1}} = \frac{\varsigma_k^2}{\sigma_{k-1}^2} = \frac{\eta_{\mathsf{ddim}}^2}{\sigma_{k-1}^2} \frac{\sigma_{k-1}^2(\sigma_k^2 - \sigma_{k-1}^2)}{\sigma_k^2} \tag{B.26}$$

$$= \eta_{\mathsf{ddim}}^2 \left(1 - \frac{\sigma_{k-1}^2}{\sigma_k^2}\right), \tag{B.27}$$

we plug $\widetilde{h}_k$ from (B.16) into (B.18) to find

$$h_k = \sqrt{\frac{\overline{\alpha}_k}{\overline{\alpha}_{k-1}} \frac{1-\overline{\alpha}_{k-1}-\widetilde{\varsigma}_k^2}{1-\overline{\alpha}_k}} = \sqrt{\frac{\overline{\alpha}_k}{1-\overline{\alpha}_k} \frac{1-\overline{\alpha}_{k-1}-\widetilde{\varsigma}_k^2}{\overline{\alpha}_{k-1}}} \tag{B.28}$$

$$= \sqrt{\frac{\overline{\alpha}_k}{1-\overline{\alpha}_k} \frac{1-\overline{\alpha}_{k-1}}{\overline{\alpha}_{k-1}} \left(1 - \frac{\widetilde{\varsigma}_k^2}{1-\overline{\alpha}_{k-1}}\right)} = \sqrt{\frac{\sigma_{k-1}^2}{\sigma_k^2} \left(1 - \eta_{\mathsf{ddim}}^2 \left(1 - \frac{\sigma_{k-1}^2}{\sigma_k^2}\right)\right)} \tag{B.29}$$

$$= \sqrt{\frac{\sigma_{k-1}^2}{\sigma_k^2} \left(1 - \frac{\varsigma_k^2}{\sigma_{k-1}^2}\right)} = \sqrt{\frac{\sigma_{k-1}^2}{\sigma_k^2} - \frac{\varsigma_k^2}{\sigma_k^2}} = \sqrt{\frac{\sigma_{k-1}^2 - \varsigma_k^2}{\sigma_k^2}}. \tag{B.30}$$

The VE DDIM reverse process is summarized in (4.22)-(4.23).

Appendix C: Proofs and Derivations

C.1 Proof of Proposition 1

Here we prove Proposition 1. To begin, for an N-pixel image, we rewrite (2.8)-(2.9) as

$$\mathcal{L}_{1,P}(\boldsymbol{\theta}) = \sum_{j=1}^{N} \mathbb{E}_{\mathsf{y}} \left\{ \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \left\{ |x_j - \tfrac{1}{P} \sum_{i=1}^{P} \widehat{x}_{ij}| \; \middle| \boldsymbol{y} \right\} \right\} \tag{C.1}$$

$$\mathcal{L}_{\mathsf{SD},P}(\boldsymbol{\theta}) = \sum_{j=1}^{N} \mathbb{E}_{\mathsf{y}} \left\{ \mathbb{E}_{\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \left\{ \tfrac{\gamma_P}{P} \sum_{i=1}^{P} |\widehat{x}_{ij} - \tfrac{1}{P} \sum_{k=1}^{P} \widehat{x}_{kj}| \; \middle| \boldsymbol{y} \right\} \right\}, \tag{C.2}$$

where $x_j \triangleq [\boldsymbol{x}]_j$, $\widehat{x}_{ij} \triangleq [\widehat{\boldsymbol{x}}_i]_j$, and

$$\gamma_P \triangleq \sqrt{\tfrac{\pi P}{2(P-1)}}. \tag{C.3}$$

To simplify the notation in the sequel, we will consider an arbitrary fixed value of j and use the abbreviations

$$x_j \to X, \qquad\qquad \widehat{x}_{ij} \to \widehat{X}_i. \tag{C.4}$$

Recall that $\boldsymbol{x}$ and $\{\widehat{\boldsymbol{x}}_i\}$ are mutually independent when conditioned on $\boldsymbol{y}$ because the code vectors $\{\boldsymbol{z}_i\}$ are generated independently of both $\boldsymbol{x}$ and $\boldsymbol{y}$. In the context of Proposition 1, we also assume that the vector elements x_j and $\widehat{x}_{ij}$ are independent Gaussian when conditioned on $\boldsymbol{y}$. This implies that we can make the notational shift

$$p_{\mathsf{x}|\mathsf{y}}(x_j|\boldsymbol{y}) \to \mathcal{N}(X; \mu_0, \sigma_0^2), \qquad\qquad p_{\widehat{\mathsf{x}}|\mathsf{y}}(\widehat{x}_{ij}|\boldsymbol{y}) \to \mathcal{N}(\widehat{X}_i; \mu, \sigma^2), \tag{C.5}$$

where X and $\{\widehat{X}_i\}$ are mutually independent. With this simplified notation, we note that $[\widehat{\boldsymbol{x}}_{\mathsf{mmse}}]_j \to \mu_0$, and that mode collapse corresponds to $\sigma = 0$.

Furthermore, if $\boldsymbol{\theta}$ can completely control (μ, σ), then (2.12) can be rewritten as

$$(\mu_*, \sigma_*) = \arg\min_{\mu,\sigma} \left\{ \mathcal{L}_{1,P}(\mu,\sigma) - \beta_{\mathsf{SD}} \mathcal{L}_{\mathsf{SD},P}(\mu,\sigma) \right\} \;\Rightarrow\; \begin{cases} \mu_* = \mu_0 \\ \sigma_* = \sigma_0 \end{cases} \tag{C.6}$$

with

$$\mathcal{L}_{1,P}(\mu,\sigma) = \mathbb{E}_{X,\widehat{X}_1,\ldots\widehat{X}_P}\{|X - \tfrac{1}{P}\textstyle\sum_{i=1}^P \widehat{X}_i|\} \tag{C.7}$$

$$\mathcal{L}_{\mathsf{SD},P}(\mu,\sigma) = \mathbb{E}_{\widehat{X}_1,\ldots\widehat{X}_P}\{\tfrac{\gamma_P}{P}\textstyle\sum_{i=1}^P |\widehat{X}_i - \tfrac{1}{P}\sum_{k=1}^P \widehat{X}_k|\}. \tag{C.8}$$

Although σ_* must be positive, it turns out that we do not need to enforce this in the optimization (C.6) because it will arise naturally.

To further analyze (C.7) and (C.8), we define

$$\widehat{\mu} \triangleq \tfrac{1}{P}\textstyle\sum_{i=1}^P \widehat{X}_i \tag{C.9}$$

$$\widehat{\sigma} \triangleq \tfrac{\gamma_P}{P}\textstyle\sum_{i=1}^P |\widehat{X}_i - \widehat{\mu}|. \tag{C.10}$$

The quantity $\widehat{\mu}$ can be recognized as the unbiased estimate of the mean μ of $\widehat{X}_i$, and we now show that $\widehat{\sigma}$ is an unbiased estimate of the SD σ of $\widehat{X}_i$ in the case that $\widehat{X}_i$ is Gaussian. To see this, first observe that the i.i.d. $\mathcal{N}(\mu,\sigma^2)$ property of $\{\widehat{X}_i\}$ implies that $\widehat{X}_i - \widehat{\mu} = (1-\frac{1}{P})\widehat{X}_i - \frac{1}{P}\sum_{k\neq i}\widehat{X}_k$ is Gaussian with mean zero and variance $(1-\frac{1}{P})^2\sigma^2 + \frac{P-1}{P^2}\sigma^2 = \frac{P-1}{P}\sigma^2$. The variable $|\widehat{X}_i - \widehat{\mu}|$ is thus half-normal distributed with mean $\sqrt{\frac{2(P-1)}{\pi P}\sigma^2}$ [54]. Because $\{\widehat{X}_i\}$ are i.i.d., the variable $\frac{1}{P}\sum_{i=1}^P |\widehat{X}_i - \widehat{\mu}|$ has the same mean as $|\widehat{X}_i - \widehat{\mu}|$. Finally, multiplying $\frac{1}{P}\sum_{i=1}^P |\widehat{X}_i - \widehat{\mu}|$ by γ_P yields $\widehat{\sigma}$ from (C.10), and multiplying its mean using the expression for γ_P from (C.3) implies

$$\mathbb{E}\{\widehat{\sigma}\} = \sigma, \tag{C.11}$$

and so $\widehat{\sigma}$ is an unbiased estimator of σ, the SD of $\widehat{X}_i$.

With the above definitions of $\widehat{\mu}$ and $\widehat{\sigma}$, the optimization cost in (C.6) can be written as

$$\mathcal{L}_{1,P}(\mu,\sigma) - \beta_{\mathsf{SD}}\mathcal{L}_{\mathsf{SD},P}(\mu,\sigma) = \mathbb{E}_{X,\widehat{X}_1,\ldots\widehat{X}_P}\left\{|X-\widehat{\mu}|\right\} - \beta_{\mathsf{SD}}\,\mathbb{E}_{\widehat{X}_1,\ldots\widehat{X}_P}\left\{\widehat{\sigma}\right\} \tag{C.12}$$

$$= \mathbb{E}_{X,\widehat{X}_1,\ldots\widehat{X}_P}\left\{|X-\widehat{\mu}|\right\} - \beta_{\mathsf{SD}}\sigma, \tag{C.13}$$

where in the last step we exploited the unbiased property of $\widehat{\sigma}$. To proceed further, we note that the i.i.d. Gaussian property of $\{\widehat{X}_i\}$ implies $\widehat{\mu} \sim \mathcal{N}(\mu, \sigma^2/P)$, after which the mutual independence of $\{\widehat{X}_i\}$ and X yields

$$X - \widehat{\mu} \sim \mathcal{N}(\mu_0 - \mu, \sigma_0^2 + \sigma^2/P). \tag{C.14}$$

Taking the absolute value of a Gaussian random yields a folded-normal random variable [54]. Using the mean and variance in (C.14), the expressions in [54] yield

$$\mathbb{E}_{X,\widehat{X}_1,\ldots\widehat{X}_P}\left\{|X-\widehat{\mu}|\right\} = \sqrt{\frac{2(\sigma_0^2+\sigma^2/P)}{\pi}}\exp\Big(-\frac{(\mu_0-\mu)^2}{2(\sigma_0^2+\sigma^2/P)}\Big) + (\mu_0-\mu)\,\mathrm{erf}\Big(\frac{\mu_0-\mu}{\sqrt{2(\sigma_0^2+\sigma^2/P)}}\Big). \tag{C.15}$$

Thus the optimization cost (C.13) can be written as

$$J(\mu,\sigma) = \sqrt{\frac{2(\sigma_0^2+\sigma^2/P)}{\pi}}\exp\Big(-\frac{(\mu-\mu_0)^2}{2(\sigma_0^2+\sigma^2/P)}\Big) + (\mu-\mu_0)\,\mathrm{erf}\Big(\frac{\mu-\mu_0}{\sqrt{2(\sigma_0^2+\sigma^2/P)}}\Big) - \beta_{\mathsf{SD}}\sigma. \tag{C.16}$$

Since $J(\cdot,\cdot)$ is convex, the minimizer $(\mu_*,\sigma_*) = \arg\min_{\mu,\sigma} J(\mu,\sigma)$ satisfies $\nabla J(\mu_*,\sigma_*) = (0,0)$. To streamline the derivation, we define

$$c \triangleq \sqrt{2(\sigma_0^2+\sigma^2/P)/\pi}, \qquad\qquad s \triangleq \sqrt{\sigma_0^2+\sigma^2/P} \tag{C.17}$$

so that

$$J(\mu,\sigma) = c\exp\Big(-\frac{(\mu-\mu_0)^2}{2s^2}\Big) + (\mu-\mu_0)\,\mathrm{erf}\,\Big(\frac{\mu-\mu_0}{\sqrt{2s^2}}\Big) - \beta_{\mathsf{SD}}\sigma. \tag{C.18}$$

Because c and s are invariant to μ, we get

$$\begin{aligned}\frac{\partial J(\mu,\sigma)}{\partial\mu} = -c\exp\Big(-\frac{(\mu-\mu_0)^2}{2s^2}\Big)\frac{\mu-\mu_0}{s^2} + \mathrm{erf}\,\Big(\frac{\mu-\mu_0}{\sqrt{2s^2}}\Big) \\ + (\mu-\mu_0)\frac{2}{\sqrt{\pi}}\exp\Big(-\frac{(\mu-\mu_0)^2}{2s^2}\Big),\end{aligned} \tag{C.19}$$

which equals zero if and only if $\mu = \mu_0$. Thus we have determined that $\mu_* = \mu_0$. Plugging $\mu_* = \mu_0$ into (C.16), we find

$$J(\mu_*,\sigma) = \sqrt{2(\sigma_0^2 + \sigma^2/P)/\pi} - \beta_{\mathsf{SD}}\sigma. \tag{C.20}$$

Taking the derivative with respect to σ, we get

$$\frac{\partial J(\mu_*,\sigma)}{\partial\sigma} = \sqrt{\frac{2}{\pi P(P\sigma_0^2/\sigma^2+1)}} - \beta_{\mathsf{SD}} \tag{C.21}$$

$$= \sqrt{\frac{2}{\pi P(P\sigma_0^2/\sigma^2+1)}} - \sqrt{\frac{2}{\pi P(P+1)}}, \tag{C.22}$$

where in the last step we applied the value of β_{SD} from (2.11). It can now be seen that $\frac{\partial J(\mu_*,\sigma)}{\partial\sigma} = 0$ if and only if $\sigma = \sigma_0$, which implies that $\sigma_* = \sigma_0$. Thus we have established (C.6), which completes the proof of Proposition 1.

C.2 Derivation of Proposition 2

Here we prove Proposition 2. To start, we establish some notation and conditional-mean properties:

$$\begin{array}{rll}
\widehat{\boldsymbol{x}}_{\mathsf{mmse}} \triangleq \mathbb{E}_{\mathsf{x}|\mathsf{y}}\{\boldsymbol{x}|\boldsymbol{y}\} & & \\
\boldsymbol{e}_{\mathsf{mmse}} \triangleq \boldsymbol{x} - \widehat{\boldsymbol{x}}_{\mathsf{mmse}}, & \mathbf{0} = \mathbb{E}_{\mathsf{x}|\mathsf{y}}\{\boldsymbol{e}_{\mathsf{mmse}}|\boldsymbol{y}\} & \\
\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta}) \triangleq G_{\boldsymbol{\theta}}(\boldsymbol{z}_i,\boldsymbol{y}), & \overline{\boldsymbol{x}}(\boldsymbol{\theta}) \triangleq \mathbb{E}_{\mathsf{z}_i|\mathsf{y}}\{\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta})|\boldsymbol{y}\} & \\
\widehat{\boldsymbol{x}}_{(P)}(\boldsymbol{\theta}) \triangleq \frac{1}{P}\sum_{i=1}^P \widehat{\boldsymbol{x}}_i(\boldsymbol{\theta}), & \overline{\boldsymbol{x}}(\boldsymbol{\theta}) = \mathbb{E}_{\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}}\{\widehat{\boldsymbol{x}}_{(P)}(\boldsymbol{\theta})|\boldsymbol{y}\} & \\
\boldsymbol{d}_i(\boldsymbol{\theta})) \triangleq \widehat{\boldsymbol{x}}_i(\boldsymbol{\theta}) - \overline{\boldsymbol{x}}(\boldsymbol{\theta}), & \mathbf{0} = \mathbb{E}_{\mathsf{z}_i|\mathsf{y}}\{\boldsymbol{d}_i(\boldsymbol{\theta})|\boldsymbol{y}\}\ \forall\boldsymbol{\theta} & \\
\boldsymbol{d}_{(P)}(\boldsymbol{\theta}) \triangleq \frac{1}{P}\sum_{i=1}^P \boldsymbol{d}_i(\boldsymbol{\theta}), & \mathbf{0} = \mathbb{E}_{\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}}\{\boldsymbol{d}_{(P)}(\boldsymbol{\theta})|\boldsymbol{y}\}\ \forall\boldsymbol{\theta} &
\end{array} \tag{C.23}$$

Our first step is to write (2.14) as

$$\mathcal{L}_{2,P}(\boldsymbol{\theta}) = \mathbb{E}_{\mathsf{y}} \left\{ \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \{ \|\boldsymbol{x} - \widehat{\boldsymbol{x}}_{(P)}(\boldsymbol{\theta})\|_2^2 | \boldsymbol{y} \} \right\}. \tag{C.24}$$

Leveraging the fact that $\widehat{\boldsymbol{x}}_{\mathsf{mmse}}$ and $\overline{\boldsymbol{x}}(\boldsymbol{\theta})$ are deterministic given $\boldsymbol{y}$, we write the inner term in (C.24) as

$$\begin{aligned}
&\mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \{ \|\boldsymbol{x} - \widehat{\boldsymbol{x}}_{(P)}(\boldsymbol{\theta})\|_2^2 | \boldsymbol{y} \} \\
&= \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \{ \|\widehat{\boldsymbol{x}}_{\mathsf{mmse}} + \boldsymbol{e}_{\mathsf{mmse}} - \overline{\boldsymbol{x}}(\boldsymbol{\theta}) - \boldsymbol{d}_{(P)}(\boldsymbol{\theta})\|_2^2 | \boldsymbol{y} \} && \text{(C.25)} \\
&= \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \{ \|\widehat{\boldsymbol{x}}_{\mathsf{mmse}} - \overline{\boldsymbol{x}}(\boldsymbol{\theta})\|_2^2 | \boldsymbol{y} \} \\
&\quad + 2 \operatorname{Re} \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \{ (\widehat{\boldsymbol{x}}_{\mathsf{mmse}} - \overline{\boldsymbol{x}}(\boldsymbol{\theta}))^{\mathsf{H}} (\boldsymbol{e}_{\mathsf{mmse}} - \boldsymbol{d}_{(P)}(\boldsymbol{\theta})) | \boldsymbol{y} \} \\
&\quad + \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \{ \|\boldsymbol{e}_{\mathsf{mmse}} - \boldsymbol{d}_{(P)}(\boldsymbol{\theta})\|_2^2 | \boldsymbol{y} \} && \text{(C.26)} \\
&= \|\widehat{\boldsymbol{x}}_{\mathsf{mmse}} - \overline{\boldsymbol{x}}(\boldsymbol{\theta})\|_2^2 + 2 \operatorname{Re} \Big[(\widehat{\boldsymbol{x}}_{\mathsf{mmse}} - \overline{\boldsymbol{x}}(\boldsymbol{\theta}))^{\mathsf{H}} \underbrace{\mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \{ (\boldsymbol{e}_{\mathsf{mmse}} - \boldsymbol{d}_{(P)}(\boldsymbol{\theta})) | \boldsymbol{y} \}}_{= \boldsymbol{0}} \Big] \\
&\quad + \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \{ \|\boldsymbol{e}_{\mathsf{mmse}} - \boldsymbol{d}_{(P)}(\boldsymbol{\theta})\|_2^2 | \boldsymbol{y} \} && \text{(C.27)} \\
&= \|\widehat{\boldsymbol{x}}_{\mathsf{mmse}} - \mathbb{E}_{\mathsf{z}_i|\mathsf{y}} \{ \widehat{\boldsymbol{x}}_i(\boldsymbol{\theta}) | \boldsymbol{y} \} \|_2^2 + \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \{ \|\boldsymbol{e}_{\mathsf{mmse}} - \boldsymbol{d}_{(P)}(\boldsymbol{\theta})\|_2^2 | \boldsymbol{y} \}. && \text{(C.28)}
\end{aligned}$$

where in (C.27) we used the fact that $\boldsymbol{d}_{(P)}$ and $\boldsymbol{e}_{\mathsf{mmse}}$ are both zero-mean when conditioned on $\boldsymbol{y}$. We now leverage the fact that $\{\boldsymbol{z}_i\}$ are independent of $\boldsymbol{x}$ and $\boldsymbol{y}$ to write

$$\begin{aligned}
&\mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \{ \|\boldsymbol{e}_{\mathsf{mmse}} - \boldsymbol{d}_{(P)}(\boldsymbol{\theta})\|_2^2 | \boldsymbol{y} \} \\
&= \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \{ \|\boldsymbol{e}_{\mathsf{mmse}}\|_2^2 | \boldsymbol{y} \} + 2 \operatorname{Re} \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \{ \boldsymbol{e}_{\mathsf{mmse}}^{\mathsf{H}} \boldsymbol{d}_{(P)}(\boldsymbol{\theta}) | \boldsymbol{y} \} \\
&\quad + \mathbb{E}_{\mathsf{x},\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \{ \|\boldsymbol{d}_{(P)}(\boldsymbol{\theta})\|_2^2 | \boldsymbol{y} \} && \text{(C.29)} \\
&= \mathbb{E}_{\mathsf{x}|\mathsf{y}} \{ \|\boldsymbol{e}_{\mathsf{mmse}}\|_2^2 | \boldsymbol{y} \} + 2 \operatorname{Re} \Big[\underbrace{\mathbb{E}_{\mathsf{x}|\mathsf{y}} \{ \boldsymbol{e}_{\mathsf{mmse}} | \boldsymbol{y} \}}_{= \boldsymbol{0}}^{\mathsf{H}} \underbrace{\mathbb{E}_{\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \{ \boldsymbol{d}_{(P)}(\boldsymbol{\theta}) | \boldsymbol{y} \}}_{= \boldsymbol{0}} \Big] \\
&\quad + \mathbb{E}_{\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}} \{ \|\boldsymbol{d}_{(P)}(\boldsymbol{\theta})\|_2^2 | \boldsymbol{y} \}. && \text{(C.30)}
\end{aligned}$$

Finally, we can leverage the fact that $\{\boldsymbol{z}_i\}$ are i.i.d. to write

$$\mathbb{E}_{\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}}\{\|\boldsymbol{d}_{(P)}(\boldsymbol{\theta})\|_2^2|\boldsymbol{y}\} = \mathbb{E}_{\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}}\{\|\tfrac{1}{P}\textstyle\sum_{i=1}^P \boldsymbol{d}_i(\boldsymbol{\theta})\|_2^2|\boldsymbol{y}\} \tag{C.31}$$

$$= \tfrac{1}{P^2}\textstyle\sum_{i=1}^P \mathbb{E}_{\mathsf{z}_i|\mathsf{y}}\{\|\boldsymbol{d}_i(\boldsymbol{\theta})\|_2^2|\boldsymbol{y}\} \tag{C.32}$$

$$= \tfrac{1}{P}\,\mathbb{E}_{\mathsf{z}_i|\mathsf{y}}\{\|\boldsymbol{d}_i(\boldsymbol{\theta})\|_2^2|\boldsymbol{y}\} \text{ for any } i \tag{C.33}$$

$$= \tfrac{1}{P}\,\mathbb{E}_{\mathsf{z}_i|\mathsf{y}}\{\mathrm{tr}[\boldsymbol{d}_i(\boldsymbol{\theta})\boldsymbol{d}_i(\boldsymbol{\theta})^{\mathsf{H}}]|\boldsymbol{y}\} \tag{C.34}$$

$$= \tfrac{1}{P}\,\mathrm{tr}\left[\,\mathbb{E}_{\mathsf{z}_i|\mathsf{y}}\{\boldsymbol{d}_i(\boldsymbol{\theta})\boldsymbol{d}_i(\boldsymbol{\theta})^{\mathsf{H}}\}|\boldsymbol{y}\}\right] \tag{C.35}$$

$$= \tfrac{1}{P}\,\mathrm{tr}\left[\,\mathrm{Cov}_{\mathsf{z}_i|\mathsf{y}}\{\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta})|\boldsymbol{y}\}\right]. \tag{C.36}$$

Combining (C.24), (C.28), (C.30), and (C.36), we get the bias-variance decomposition

$$\mathcal{L}_{2,P}(\boldsymbol{\theta}) = \mathbb{E}_{\mathsf{y}}\Big\{\|\widehat{\boldsymbol{x}}_{\mathsf{mmse}} - \mathbb{E}_{\mathsf{z}_i|\mathsf{y}}\{\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta})|\boldsymbol{y}\}\|_2^2 + \tfrac{1}{P}\,\mathrm{tr}\left[\,\mathrm{Cov}_{\mathsf{z}_i|\mathsf{y}}\{\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta})|\boldsymbol{y}\}\right] + \mathbb{E}_{\mathsf{x}|\mathsf{y}}\{\|\boldsymbol{e}_{\mathsf{mmse}}\|_2^2|\boldsymbol{y}\}\Big\}. \tag{C.37}$$

We now see that if $\boldsymbol{\theta}$ has complete control over the $\boldsymbol{y}$-conditional mean and covariance of $\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta})$, then minimizing (C.37) over $\boldsymbol{\theta}$ will cause

$$\mathbb{E}_{\mathsf{z}_i|\mathsf{y}}\{\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta})|\boldsymbol{y}\} = \widehat{\boldsymbol{x}}_{\mathsf{mmse}} \tag{C.38}$$

$$\mathrm{Cov}_{\mathsf{z}_i|\mathsf{y}}\{\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta})|\boldsymbol{y}\} = \boldsymbol{0}, \tag{C.39}$$

which proves Proposition 2.

C.3 Derivation of (2.19)

To show that the expression for $\mathcal{L}_{\mathsf{var},P}$ in (2.19) holds, we first rewrite (2.18) as

$$\mathcal{L}_{\mathsf{var},P}(\boldsymbol{\theta}) = \tfrac{1}{P-1}\textstyle\sum_{i=1}^P \mathbb{E}_{\mathsf{y}}\{\mathbb{E}_{\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}}\{\|\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta}) - \widehat{\boldsymbol{x}}_{(P)}(\boldsymbol{\theta})\|_2^2|\boldsymbol{y}\} \tag{C.40}$$

where the definitions from (C.23) imply

$$\begin{aligned}
&\mathbb{E}_{\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}}\{\|\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta}) - \widehat{\boldsymbol{x}}_{(P)}(\boldsymbol{\theta})\|_2^2|\boldsymbol{y}\} \\
&= \mathbb{E}_{\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}}\{\|\overline{\boldsymbol{x}}(\boldsymbol{\theta}) + \boldsymbol{d}_i(\boldsymbol{\theta}) - \boldsymbol{d}_{(P)}(\boldsymbol{\theta}) - \overline{\boldsymbol{x}}(\boldsymbol{\theta})\|_2^2|\boldsymbol{y}\} && \text{(C.41)} \\
&= \mathbb{E}_{\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}}\{\|\boldsymbol{d}_i(\boldsymbol{\theta}) - \tfrac{1}{P}\textstyle\sum_{j=1}^P \boldsymbol{d}_j(\boldsymbol{\theta})\|_2^2|\boldsymbol{y}\} && \text{(C.42)} \\
&= \mathbb{E}_{\mathsf{z}_1,\ldots,\mathsf{z}_P|\mathsf{y}}\{\|(1-\tfrac{1}{P})\boldsymbol{d}_i(\boldsymbol{\theta}) - \tfrac{1}{P}\textstyle\sum_{j\neq i} \boldsymbol{d}_j(\boldsymbol{\theta})\|_2^2|\boldsymbol{y}\} && \text{(C.43)} \\
&= (1-\tfrac{1}{P})^2\,\mathbb{E}_{\mathsf{z}_i|\mathsf{y}}\{\|\boldsymbol{d}_i(\boldsymbol{\theta})\|_2^2|\boldsymbol{y}\} + \tfrac{P-1}{P^2}\,\mathbb{E}_{\mathsf{z}_i|\mathsf{y}}\{\|\boldsymbol{d}_i(\boldsymbol{\theta})\|_2^2|\boldsymbol{y}\} && \text{(C.44)} \\
&= \tfrac{P-1}{P}\,\mathbb{E}_{\mathsf{z}_i|\mathsf{y}}\{\|\boldsymbol{d}_i(\boldsymbol{\theta})\|_2^2|\boldsymbol{y}\} \text{ for any } i. && \text{(C.45)}
\end{aligned}$$

For (C.44), we leveraged the zero-mean and i.i.d. nature of $\{\boldsymbol{d}_i(\boldsymbol{\theta})\}$ conditioned on $\boldsymbol{y}$. By plugging (C.45) into (C.40), we get

$$\begin{aligned}
\mathcal{L}_{\mathsf{var},P}(\boldsymbol{\theta}) &= \tfrac{1}{P}\textstyle\sum_{i=1}^P \mathbb{E}_{\mathsf{y}}\{\mathbb{E}_{\mathsf{z}_i|\mathsf{y}}\{\|\boldsymbol{d}_i(\boldsymbol{\theta})\|_2^2|\boldsymbol{y}\}\} && \text{(C.46)} \\
&= \mathbb{E}_{\mathsf{y}}\{\mathbb{E}_{\mathsf{z}_i|\mathsf{y}}\{\|\boldsymbol{d}_i(\boldsymbol{\theta})\|_2^2|\boldsymbol{y}\}\} \text{ for any } i && \text{(C.47)} \\
&= \mathbb{E}_{\mathsf{y}}\{\mathrm{tr}[\mathrm{Cov}_{\mathsf{z}_i|\mathsf{y}}\{\widehat{\boldsymbol{x}}_i(\boldsymbol{\theta})|\boldsymbol{y}\}]\}, && \text{(C.48)}
\end{aligned}$$

where (C.47) follows because $\{\boldsymbol{d}_i(\boldsymbol{\theta})\}$ are i.i.d. when conditioned on $\boldsymbol{y}$ and (C.48) follows from manipulations similar to those used for (C.36).

C.4 Proof of Proposition 3

Here we prove Proposition 3. Recall from (C.23) that $\widehat{\boldsymbol{x}}_{\mathsf{mmse}} \triangleq \mathbb{E}\{\boldsymbol{x}|\boldsymbol{y}\}$ and $\boldsymbol{e}_{\mathsf{mmse}} \triangleq \boldsymbol{x} - \widehat{\boldsymbol{x}}_{\mathsf{mmse}}$. To reduce clutter, we will abbreviate $\boldsymbol{e}_{\mathsf{mmse}}$ by $\boldsymbol{e}$ in this appendix. Also, for true-posterior samples $\widehat{\boldsymbol{x}}_i \sim p_{\mathsf{x}|\mathsf{y}}(\cdot|\boldsymbol{y})$, we define

$$\widehat{\boldsymbol{e}}_i \triangleq \widehat{\boldsymbol{x}}_i - \widehat{\boldsymbol{x}}_{\mathsf{mmse}}. \tag{C.49}$$

Then using $\widehat{\boldsymbol{x}}_{(P)} \triangleq \frac{1}{P}\sum_{i=1}^{P}\widehat{\boldsymbol{x}}_i$ and from $\mathcal{E}_P$ from (2.20), we have

$$\mathcal{E}_P = \mathbb{E}\{\|\widehat{\boldsymbol{x}}_{(P)} - \boldsymbol{x}\|^2|\boldsymbol{y}\} \tag{C.50}$$

$$= \mathbb{E}\{\|(\tfrac{1}{P}\textstyle\sum_{i=1}^{P}\widehat{\boldsymbol{x}}_i) - \boldsymbol{x}\|^2|\boldsymbol{y}\} \tag{C.51}$$

$$= \mathbb{E}\{\|\tfrac{1}{P}\textstyle\sum_{i=1}^{P}(\widehat{\boldsymbol{x}}_i - \boldsymbol{x})\|^2|\boldsymbol{y}\} \tag{C.52}$$

$$= \tfrac{1}{P^2}\,\mathbb{E}\{\|\textstyle\sum_{i=1}^{P}(\widehat{\boldsymbol{x}}_i - \widehat{\boldsymbol{x}}_{\mathsf{mmse}} + \widehat{\boldsymbol{x}}_{\mathsf{mmse}} - \boldsymbol{x})\|^2|\boldsymbol{y}\} \tag{C.53}$$

$$= \tfrac{1}{P^2}\,\mathbb{E}\{\|\textstyle\sum_{i=1}^{P}(\widehat{\boldsymbol{e}}_i - \boldsymbol{e})\|^2|\boldsymbol{y}\} \tag{C.54}$$

$$= \tfrac{1}{P^2}\,\mathbb{E}\{\textstyle\sum_{i=1}^{P}(\widehat{\boldsymbol{e}}_i - \boldsymbol{e})^{\mathsf{H}}\sum_{j=1}^{P}(\widehat{\boldsymbol{e}}_j - \boldsymbol{e})|\boldsymbol{y}\} \tag{C.55}$$

$$= \tfrac{1}{P^2}\textstyle\sum_{i=1}^{P}\sum_{j=1}^{P}\mathbb{E}\{(\widehat{\boldsymbol{e}}_i - \boldsymbol{e})^{\mathsf{H}}(\widehat{\boldsymbol{e}}_j - \boldsymbol{e})|\boldsymbol{y}\} \tag{C.56}$$

$$= \tfrac{1}{P^2}\textstyle\sum_{i=1}^{P}\mathbb{E}\{(\widehat{\boldsymbol{e}}_i - \boldsymbol{e})^{\mathsf{H}}(\widehat{\boldsymbol{e}}_i - \boldsymbol{e})|\boldsymbol{y}\} + \tfrac{1}{P^2}\textstyle\sum_{i=1}^{P}\sum_{j\neq i}\mathbb{E}\{(\widehat{\boldsymbol{e}}_i - \boldsymbol{e})^{\mathsf{H}}(\widehat{\boldsymbol{e}}_j - \boldsymbol{e})|\boldsymbol{y}\} \tag{C.57}$$

$$\begin{aligned} = \tfrac{1}{P^2}\textstyle\sum_{i=1}^{P}\big[\mathbb{E}\{\|\widehat{\boldsymbol{e}}_i\|^2|\boldsymbol{y}\} - 2\,\mathrm{Re}\,\mathbb{E}\{\widehat{\boldsymbol{e}}_i^{\mathsf{H}}\boldsymbol{e}|\boldsymbol{y}\} + \mathbb{E}\{\|\boldsymbol{e}\|^2|\boldsymbol{y}\}\big] \\ + \tfrac{1}{P^2}\textstyle\sum_{i=1}^{P}\sum_{j\neq i}\mathrm{Re}\,\big[\mathbb{E}\{\widehat{\boldsymbol{e}}_i^{\mathsf{H}}\widehat{\boldsymbol{e}}_j|\boldsymbol{y}\} - \mathbb{E}\{\widehat{\boldsymbol{e}}_i^{\mathsf{H}}\boldsymbol{e}|\boldsymbol{y}\} - \mathbb{E}\{\boldsymbol{e}^{\mathsf{H}}\widehat{\boldsymbol{e}}_j|\boldsymbol{y}\} + \mathbb{E}\{\|\boldsymbol{e}\|^2|\boldsymbol{y}\}\big] \end{aligned} \tag{C.58}$$

$$= \tfrac{1}{P^2}\textstyle\sum_{i=1}^{P}\mathbb{E}\{\|\widehat{\boldsymbol{e}}_i\|^2|\boldsymbol{y}\} + \tfrac{1}{P}\,\mathbb{E}\{\|\boldsymbol{e}\|^2|\boldsymbol{y}\} + \tfrac{P(P-1)}{P^2}\,\mathbb{E}\{\|\boldsymbol{e}\|^2|\boldsymbol{y}\}, \tag{C.59}$$

where certain terms vanished because the i.i.d. and zero-mean properties of $\{\boldsymbol{e}, \widehat{\boldsymbol{e}}_1, \ldots, \widehat{\boldsymbol{e}}_P\}$ imply

$$\mathbb{E}\{\widehat{\boldsymbol{e}}_i^{\mathsf{H}}\widehat{\boldsymbol{e}}_j|\boldsymbol{y}\} = \mathbb{E}\{\widehat{\boldsymbol{e}}_i|\boldsymbol{y}\}^{\mathsf{H}}\,\mathbb{E}\{\widehat{\boldsymbol{e}}_j|\boldsymbol{y}\} = 0 \tag{C.60}$$

$$\mathbb{E}\{\widehat{\boldsymbol{e}}_i^{\mathsf{H}}\boldsymbol{e}|\boldsymbol{y}\} = \mathbb{E}\{\widehat{\boldsymbol{e}}_i|\boldsymbol{y}\}^{\mathsf{H}}\,\mathbb{E}\{\boldsymbol{e}|\boldsymbol{y}\} = 0 \tag{C.61}$$

$$\mathbb{E}\{\boldsymbol{e}^{\mathsf{H}}\widehat{\boldsymbol{e}}_j|\boldsymbol{y}\} = \mathbb{E}\{\boldsymbol{e}|\boldsymbol{y}\}^{\mathsf{H}}\,\mathbb{E}\{\widehat{\boldsymbol{e}}_j|\boldsymbol{y}\} = 0. \tag{C.62}$$

Finally, note that $\mathbb{E}\{\|\boldsymbol{e}\|^2|\boldsymbol{y}\} = \mathcal{E}_{\mathsf{mmse}}$ from (C.23). Furthermore, because $\{\boldsymbol{x}, \widehat{\boldsymbol{x}}_1, \ldots, \widehat{\boldsymbol{x}}_P\}$ are independent samples of $p_{\mathsf{x}|\mathsf{y}}(\cdot|\boldsymbol{y})$ under the assumptions of Proposition 3, we have $\mathbb{E}\{\|\boldsymbol{e}\|^2|\boldsymbol{y}\} = \mathbb{E}\{\|\widehat{\boldsymbol{e}}_i\|^2|\boldsymbol{y}\}$ and so (C.59) becomes

$$\mathcal{E}_P = \frac{1}{P^2}\sum_{i=1}^{P}\mathcal{E}_{\mathsf{mmse}} + \frac{1}{P}\mathcal{E}_{\mathsf{mmse}} + \frac{P(P-1)}{P^2}\mathcal{E}_{\mathsf{mmse}} = \frac{P+1}{P}\mathcal{E}_{\mathsf{mmse}}. \tag{C.63}$$

This result holds for any $P \geq 1$, which implies the ratio

$$\frac{\mathcal{E}_1}{\mathcal{E}_P} = \frac{2P}{P+1}. \tag{C.64}$$

C.5 Proof of Theorem 4

To prove Theorem 4, we begin by writing the key FIRE steps with explicit iteration index $n \geq 1$:

$$\overline{\boldsymbol{x}}[n] = \boldsymbol{d}(\boldsymbol{r}[n], \sigma[n]) \tag{C.65}$$

$$\widehat{\boldsymbol{x}}[n] = \left(\boldsymbol{A}^\top \boldsymbol{A} + \frac{\sigma_{\mathsf{w}}^2}{\nu[n]}\boldsymbol{I}\right)^{-1}\left(\boldsymbol{A}^\top \boldsymbol{y} + \frac{\sigma_{\mathsf{w}}^2}{\nu[n]}\overline{\boldsymbol{x}}[n]\right) \tag{C.66}$$

$$\sigma^2[n+1] = \max\{\sigma^2[n]/\rho, \nu[n]\} \tag{C.67}$$

$$\lambda_i[n] = \sigma^2[n+1] - (s_i^2/\sigma_{\mathsf{w}}^2 + 1/\nu[n])^{-1}, \quad i = 1, \ldots, d \tag{C.68}$$

$$\boldsymbol{n}[n] = \boldsymbol{V}\,\mathrm{Diag}(\boldsymbol{\lambda}[n])^{1/2}\boldsymbol{\varepsilon}[n], \quad \boldsymbol{\varepsilon}[n] \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}) \tag{C.69}$$

$$\boldsymbol{r}[n+1] = \widehat{\boldsymbol{x}}[n] + \boldsymbol{n}[n] \tag{C.70}$$

Our proof uses induction. By the assumptions of the theorem, we know that there exists an iteration n (in particular $n = 1$) for which $\boldsymbol{r}[n] = \boldsymbol{x}_0 + \sigma[n]\boldsymbol{\epsilon}[n]$ with $\boldsymbol{\epsilon}[n] \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and finite $\sigma[n]$. Then due to the denoiser assumption, we know that $\overline{\boldsymbol{x}}[n] = \boldsymbol{x}_0 - \sqrt{\nu[n]}\boldsymbol{e}[n]$ with $\boldsymbol{e}[n] = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and known $\nu[n] < \sigma^2[n]$. We assume that this value of $\nu[n]$ is used in lines (C.66)-(C.68). Using these results and (4.4), we can rewrite (C.66) as

$$\widehat{\boldsymbol{x}}[n] = \left(\boldsymbol{A}^\top \boldsymbol{A} + \frac{\sigma_{\mathsf{w}}^2}{\nu[n]}\boldsymbol{I}\right)^{-1}\left(\boldsymbol{A}^\top\big(\boldsymbol{A}\boldsymbol{x}_0 + \sigma_{\mathsf{w}}\boldsymbol{w}\big) + \frac{\sigma_{\mathsf{w}}^2}{\nu[n]}\big(\boldsymbol{x}_0 - \sqrt{\nu[n]}\boldsymbol{e}[n]\big)\right) \tag{C.71}$$

$$= \boldsymbol{x}_0 + \left(\boldsymbol{A}^\top \boldsymbol{A} + \frac{\sigma_{\mathsf{w}}^2}{\nu[n]}\boldsymbol{I}\right)^{-1}\left(\sigma_{\mathsf{w}}\boldsymbol{A}^\top \boldsymbol{w} - \frac{\sigma_{\mathsf{w}}^2}{\sqrt{\nu[n]}}\boldsymbol{e}[n]\right) \sim \mathcal{N}(\boldsymbol{x}_0, \boldsymbol{C}[n]) \tag{C.72}$$

for

$$\boldsymbol{C}[n] \triangleq \left(\boldsymbol{A}^\top\boldsymbol{A} + \frac{\sigma_{\mathsf{w}}^2}{\nu[n]}\boldsymbol{I}\right)^{-1}\left(\sigma_{\mathsf{w}}^2\boldsymbol{A}^\top\boldsymbol{A} + \frac{\sigma_{\mathsf{w}}^4}{\nu[n]}\boldsymbol{I}\right)\left(\boldsymbol{A}^\top\boldsymbol{A} + \frac{\sigma_{\mathsf{w}}^2}{\nu[n]}\boldsymbol{I}\right)^{-1}$$
$$= \left(\frac{1}{\sigma_{\mathsf{w}}^2}\boldsymbol{A}^\top\boldsymbol{A} + \frac{1}{\nu[n]}\boldsymbol{I}\right)^{-1} \tag{C.73}$$

by leveraging the independent-Gaussian assumption on $\boldsymbol{e}[n]$. From this and (C.68)-(C.69), we can then deduce

$$\boldsymbol{n}[n] \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}[n]) \text{ with } \boldsymbol{\Sigma}[n] \triangleq \boldsymbol{V}\,\mathrm{Diag}(\boldsymbol{\lambda}[n])\boldsymbol{V}^\top = \sigma^2[n{+}1]\boldsymbol{I} - \boldsymbol{C}[n] \tag{C.74}$$

so that, from (C.70),

$$\boldsymbol{r}[n{+}1] \sim \mathcal{N}(\boldsymbol{x}_0, \boldsymbol{C}[n] + \boldsymbol{\Sigma}[n]) = \mathcal{N}(\boldsymbol{x}_0, \sigma^2[n{+}1]\boldsymbol{I}) \tag{C.75}$$

$$\Leftrightarrow \boldsymbol{r}[n{+}1] = \boldsymbol{x}_0 + \sigma[n{+}1]\boldsymbol{\epsilon}[n{+}1], \quad \boldsymbol{\epsilon}[n{+}1] \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}). \tag{C.76}$$

Thus, by induction, if $\boldsymbol{r}[n] = \boldsymbol{x}_0 + \sigma[n]\boldsymbol{\epsilon}[n]$ with $\boldsymbol{\epsilon}[n] \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ holds at $n = 1$, then it holds at all $n > 1$.

Recall that the theorem also assumed that $\nu[n] < \sigma^2[n]$ for all n. Thus, there exists a $\rho > 1$ for which $\sigma^2[n]/\rho > \nu[n]$ for all n, for which we can rewrite (C.67) as

$$\sigma^2[n{+}1] = \sigma^2[n]/\rho \;\forall n. \tag{C.77}$$

Consequently, for any iteration $n \geq 1$ we can write

$$\sigma^2[n] = \sigma^2[1]/\rho^{n-1} = \sigma_{\mathsf{init}}^2/\rho^{n-1}. \tag{C.78}$$

Finally, because the error covariance on $\widehat{\boldsymbol{x}}[n]$ obeys

$$\boldsymbol{C}[n] = \left(\frac{1}{\sigma_{\mathsf{w}}^2}\boldsymbol{A}^\top\boldsymbol{A} + \frac{1}{\nu[n]}\boldsymbol{I}\right)^{-1} < \nu[n]\boldsymbol{I} < \sigma^2[n]\boldsymbol{I} = \frac{\sigma_{\mathsf{init}}^2}{\rho^{n-1}}\boldsymbol{I} \tag{C.79}$$

we see that the error variance in $\widehat{\boldsymbol{x}}[n]$ decreases exponentially with n and thus $\widehat{\boldsymbol{x}}[n]$ converges to the true $\boldsymbol{x}_0$.

Appendix D: Implementation Details

D.1 Conditional Fréchet Inception Distance

With the Gaussian approximation described in Sec. 2.3.1, where $p_{\underline{\mathsf{x}}|\underline{\mathsf{y}}}$ and $p_{\underline{\widehat{\mathsf{x}}}|\underline{\mathsf{y}}}$ are approximated by $\mathcal{N}(\boldsymbol{\mu}_{\underline{\mathsf{x}}|\underline{\mathsf{y}}}, \boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{x}}|\underline{\mathsf{y}}})$ and $\mathcal{N}(\boldsymbol{\mu}_{\underline{\widehat{\mathsf{x}}}|\underline{\mathsf{y}}}, \boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\widehat{\mathsf{x}}}|\underline{\mathsf{y}}})$, respectively, the CWD in (2.24) reduces to

$$\text{CFID} \triangleq \mathbb{E}_{\mathsf{y}} \left\{ \|\boldsymbol{\mu}_{\underline{\mathsf{x}}|\underline{\mathsf{y}}} - \boldsymbol{\mu}_{\underline{\widehat{\mathsf{x}}}|\underline{\mathsf{y}}}\|_2^2 + \operatorname{tr}\left[\boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{x}}|\underline{\mathsf{y}}} + \boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\widehat{\mathsf{x}}}|\underline{\mathsf{y}}} - 2\big(\boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{x}}|\underline{\mathsf{y}}}^{1/2} \boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\widehat{\mathsf{x}}}|\underline{\mathsf{y}}} \boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{x}}|\underline{\mathsf{y}}}^{1/2}\big)^{1/2}\right]\right\}. \tag{D.1}$$

The values in (D.1) are computed using

$$\boldsymbol{\mu}_{\underline{\mathsf{x}}|\underline{\mathsf{y}}} = \boldsymbol{\mu}_{\underline{\mathsf{x}}} + \boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{y}}} \boldsymbol{\Sigma}_{\underline{\mathsf{y}}\underline{\mathsf{y}}}^{-1} (\underline{\boldsymbol{y}} - \boldsymbol{\mu}_{\underline{\mathsf{y}}}) \tag{D.2}$$

$$\boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{x}}|\underline{\mathsf{y}}} = \boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{x}}} - \boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{y}}} \boldsymbol{\Sigma}_{\underline{\mathsf{y}}\underline{\mathsf{y}}}^{-1} \boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{y}}}^{\top} \tag{D.3}$$

$$\boldsymbol{\mu}_{\underline{\widehat{\mathsf{x}}}|\underline{\mathsf{y}}} = \boldsymbol{\mu}_{\underline{\widehat{\mathsf{x}}}} + \boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\mathsf{y}}} \boldsymbol{\Sigma}_{\underline{\mathsf{y}}\underline{\mathsf{y}}}^{-1} (\underline{\boldsymbol{y}} - \boldsymbol{\mu}_{\underline{\mathsf{y}}}) \tag{D.4}$$

$$\boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\widehat{\mathsf{x}}}|\underline{\mathsf{y}}} = \boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\widehat{\mathsf{x}}}} - \boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\mathsf{y}}} \boldsymbol{\Sigma}_{\underline{\mathsf{y}}\underline{\mathsf{y}}}^{-1} \boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\mathsf{y}}}^{\top}. \tag{D.5}$$

Plugging (D.2)-(D.5) into (D.1), the CFID can be written as [80, Lemma 2]:

$$\begin{aligned} \text{CFID} = \|\boldsymbol{\mu}_{\underline{\mathsf{x}}} - \boldsymbol{\mu}_{\underline{\widehat{\mathsf{x}}}}\|_2^2 + \operatorname{tr}\left[(\boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{y}}} - \boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\mathsf{y}}}) \boldsymbol{\Sigma}_{\underline{\mathsf{y}}\underline{\mathsf{y}}}^{-1} (\boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{y}}} - \boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\mathsf{y}}})^{\top}\right] \\ + \operatorname{tr}\left[\boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{x}}|\underline{\mathsf{y}}} + \boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\widehat{\mathsf{x}}}|\underline{\mathsf{y}}} - 2\big(\boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{x}}|\underline{\mathsf{y}}}^{1/2} \boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\widehat{\mathsf{x}}}|\underline{\mathsf{y}}} \boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{x}}|\underline{\mathsf{y}}}^{1/2}\big)^{1/2}\right], \end{aligned} \tag{D.6}$$

where $\boldsymbol{\Sigma}_{\underline{\mathsf{y}}\underline{\mathsf{y}}}^{-1}$ is typically implemented using a pseudo-inverse.

We now detail how the means and covariances in (D.6) are computed. We start with a dataset $\{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^n$ of truth/measurement pairs. For each $\boldsymbol{y}_t$, we generate a set of P posterior samples $\{\widehat{\boldsymbol{x}}_{ti}\}_{i=1}^P$. We merge these samples with P repetitions of $\boldsymbol{x}_t$ and $\boldsymbol{y}_t$ to obtain $\{(\boldsymbol{x}_{ti}, \boldsymbol{y}_{ti}, \widehat{\boldsymbol{x}}_{ti})\}_{i=1}^P$ for $t = 1 \dots n$. These terms are processed by a feature-generating network to yield the feature embeddings $\{(\underline{\boldsymbol{x}}_{ti}, \underline{\boldsymbol{y}}_{ti}, \underline{\widehat{\boldsymbol{x}}}_{ti})\}_{i=1}^P$, which are then packed into matrices $\underline{\boldsymbol{X}}$, $\underline{\boldsymbol{Y}}$, and $\underline{\widehat{\boldsymbol{X}}}$ with Pn rows. We used the VGG-16 feature-generating network [77] for our MRI experiments, since [46] found that it gave results that correlated much better with radiologists' perceptions, while we used the standard Inception-v3 network [91] for our inpainting experiments. The embeddings are then used to compute the sample-mean values

$$\boldsymbol{\mu}_{\underline{\mathsf{x}}} \triangleq \tfrac{1}{Pn}\mathbf{1}^\top \underline{\boldsymbol{X}}, \quad \boldsymbol{\mu}_{\underline{\mathsf{y}}} \triangleq \tfrac{1}{Pn}\mathbf{1}^\top \underline{\boldsymbol{Y}}, \quad \boldsymbol{\mu}_{\underline{\widehat{\mathsf{x}}}} \triangleq \tfrac{1}{Pn}\mathbf{1}^\top \underline{\widehat{\boldsymbol{X}}}. \tag{D.7}$$

We then subtract the sample mean from each row of $\underline{\boldsymbol{X}}$, $\underline{\boldsymbol{Y}}$, and $\underline{\widehat{\boldsymbol{X}}}$ to give the zero-mean embedding matrices $\underline{\boldsymbol{X}}_{\mathsf{zm}} \triangleq \underline{\boldsymbol{X}} - \mathbf{1}\boldsymbol{\mu}_{\underline{\mathsf{x}}}^\top$, $\underline{\boldsymbol{Y}}_{\mathsf{zm}} \triangleq \underline{\boldsymbol{Y}} - \mathbf{1}\boldsymbol{\mu}_{\underline{\mathsf{y}}}^\top$, and $\underline{\widehat{\boldsymbol{X}}}_{\mathsf{zm}} \triangleq \underline{\widehat{\boldsymbol{X}}} - \mathbf{1}\boldsymbol{\mu}_{\underline{\widehat{\mathsf{x}}}}^\top$, which are then used to compute the sample covariance matrices

$$\boldsymbol{\Sigma}_{\underline{\mathsf{xx}}} \triangleq \tfrac{1}{Pn}\underline{\boldsymbol{X}}_{\mathsf{zm}}^\top \underline{\boldsymbol{X}}_{\mathsf{zm}}, \quad \boldsymbol{\Sigma}_{\underline{\mathsf{yy}}} \triangleq \tfrac{1}{Pn}\underline{\boldsymbol{Y}}_{\mathsf{zm}}^\top \underline{\boldsymbol{Y}}_{\mathsf{zm}}, \quad \boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\widehat{\mathsf{x}}}} \triangleq \tfrac{1}{Pn}\underline{\widehat{\boldsymbol{X}}}_{\mathsf{zm}}^\top \underline{\widehat{\boldsymbol{X}}}_{\mathsf{zm}} \tag{D.8a}$$

$$\boldsymbol{\Sigma}_{\underline{\mathsf{x}}\underline{\mathsf{y}}} \triangleq \tfrac{1}{Pn}\underline{\boldsymbol{X}}_{\mathsf{zm}}^\top \underline{\boldsymbol{Y}}_{\mathsf{zm}}, \quad \boldsymbol{\Sigma}_{\underline{\widehat{\mathsf{x}}}\underline{\mathsf{y}}} \triangleq \tfrac{1}{Pn}\underline{\widehat{\boldsymbol{X}}}_{\mathsf{zm}}^\top \underline{\boldsymbol{Y}}_{\mathsf{zm}}. \tag{D.8b}$$

We plug the sample statistics from (D.7)-(D.8) into (D.2)-(D.5), which yields the statistics needed to compute the CFID in (D.6). In [80], the authors use $P = 1$ in all of their experiments. To be consistent with how we evaluated the other metrics, we use $P = 32$ unless otherwise noted.

D.2 Implementation Details for Chapter 2

The code for our model can be found here: `https://github.com/matt-bendel/rcGAN`.

D.2.1 Accelerated MRI

cGAN training. At each training iteration, our cGAN's generator takes in n_{batch} measurement samples $\boldsymbol{y}_t$ and P_{rc} code vectors for every $\boldsymbol{y}_t$, and performs an optimization step on the loss

$$\mathcal{L}_{\mathsf{G}}(\boldsymbol{\theta}) \triangleq \beta_{\mathsf{adv}}\mathcal{L}_{\mathsf{adv}}(\boldsymbol{\theta},\boldsymbol{\phi}) + \mathcal{L}_{1,P_{\mathsf{rc}}}(\boldsymbol{\theta}) - \beta_{\mathsf{SD}}\mathcal{L}_{\mathsf{SD},P_{\mathsf{rc}}}(\boldsymbol{\theta}), \tag{D.9}$$

where by default we use $\beta_{\mathsf{adv}} = 1\text{e-}5$, $n_{\mathsf{batch}} = 36$, $P_{\mathsf{rc}} = 2$, and update β_{SD} via (2.23) using $P_{\mathsf{val}} = 8$. Then, using the $P_{\mathsf{rc}}n_{\mathsf{batch}}$ generator outputs, our cGAN's discriminator performs an optimization step on the loss

$$\mathcal{L}_{\mathsf{D}}(\boldsymbol{\phi}) = -\mathcal{L}_{\mathsf{adv}}(\boldsymbol{\theta},\boldsymbol{\phi}) + \alpha_1\mathcal{L}_{\mathsf{gp}}(\boldsymbol{\phi}) + \alpha_2\mathcal{L}_{\mathsf{drift}}(\boldsymbol{\phi}), \tag{D.10}$$

with gradient penalty $\mathcal{L}_{\text{gp}}$ from [31]. As per [42], $\mathcal{L}_{\text{drift}}$ is a drift penalty, $\alpha_1 = 10$, $\alpha_2 = 0.001$, and one discriminator update was used per generator update. The models were trained for 100 epochs using the Adam optimizer [50] with a learning rate of 1e-3, $\beta_1 = 0$, and $\beta_2 = 0.99$, as in [42]. Running PyTorch on a server with 4 Tesla A100 GPUs, each with 82 GB of memory, the training of an MRI cGAN took approximately 1 day.

Adler and Öktem's cGAN [4] uses generator loss $\beta_{\mathsf{adv}}\mathcal{L}_{\mathsf{adv}}^{\mathsf{adler}}(\boldsymbol{\theta},\boldsymbol{\phi})$, where $\mathcal{L}_{\mathsf{adv}}^{\mathsf{adler}}(\boldsymbol{\theta},\boldsymbol{\phi})$ was described in (2.5), and discriminator loss $-\mathcal{L}_{\mathsf{adv}}^{\mathsf{adler}}(\boldsymbol{\theta},\boldsymbol{\phi}) + \alpha_1\mathcal{L}_{\mathsf{gp}}(\boldsymbol{\phi}) + \alpha_2\mathcal{L}_{\mathsf{drift}}(\boldsymbol{\phi})$ with the values of α_1 =10, α_2 =0.001, and β_{adv} =1, as in the original paper.

Ohayon et al.'s pscGAN [64] uses generator loss $\beta_{\mathsf{adv}}\mathcal{L}_{\mathsf{adv}}(\boldsymbol{\theta},\boldsymbol{\phi}) + \mathcal{L}_{2,P_{\mathsf{rc}}}(\boldsymbol{\theta})$, where $\mathcal{L}_{2,P_{\mathsf{rc}}}(\boldsymbol{\theta})$ was described in (2.14), and discriminator loss $-\mathcal{L}_{\mathsf{adv}}(\boldsymbol{\theta},\boldsymbol{\phi}) + \alpha_1\mathcal{L}_{\mathsf{gp}}(\boldsymbol{\phi}) + \alpha_2\mathcal{L}_{\mathsf{drift}}(\boldsymbol{\phi})$ with the values α_1 =10, α_2 =0.001, and β_{adv} =1e-5. We modify β_{adv} to re-balance the loss due to an increased magnitude of our discriminator's outputs.

All three cGANs used the same generator and discriminator architectures (detailed below), except that Adler and Öktem's discriminator used extra input channels to facilitate the 3-input loss $\mathcal{L}^{\mathsf{adler}}_{\mathsf{adv}}(\boldsymbol{\theta},\boldsymbol{\phi})$ from (2.5).

cGAN Generator Architecture. For our MRI experiments, we take inspiration from the U-Net architecture from [74], using it as the basis for the cGAN generators. The primary input $\boldsymbol{y}$ is concatenated with the code vector $\boldsymbol{z}$ and fed through the U-Net. The network consists of 4 pooling layers with 128 initial channels. However, instead of pooling, we opt to use convolutions with kernels of size 3×3, "same" padding, and a stride of 2 when downsampling. Likewise, we upsample using transpose convolutions, again with kernels of size 3×3, "same" padding, and a stride of 2. All other convolutions utilize kernels of size 3×3, "same" padding, and a stride of 1.

Within each encoder and decoder layer we include a residual block, the architecture of which can be found in [4]. We use instance-norm for all normalization layers and parametric ReLUs as our activation functions, in which the network learns the optimal "negative slope." Finally, we include 5 residual blocks at the base of the U-Net, in between the encoder and decoder. This is done in an effort to artificially increase the depth of the network and is inspired by [25]. Our generator has 86 734 334 trainable parameters.

cGAN Discriminator Architecture. Our discriminator is a standard CNN with 6 layers and 1 fully-connected layer. In the first 3 layers, we use convolutions

with kernels of size 3×3, "same" padding. We reduce spatial resolution with average pooling, using 2×2 kernels with a stride of 2. We use batch-norm as our normalization layer and leaky ReLUs with a "negative-slope" of 0.2 as our activation functions. The network outputs an estimated Wasserstein score for the whole image.

E2E-VarNet. For the Sriram et al.'s E2E-VarNet [89], we use the same training procedure and hyperparameters outlined in [39] other than replacing the sampling pattern with the GRO undersampling mask. As in [39], we use the SENSE-based coil-combined image as the ground truth instead of the RSS image.

Langevin Approach. For Jalal et al.'s MRI approach [39], we do not modify the original implementation from [38] other than replacing the default sampling pattern with the GRO undersampling mask. We generated 32 samples for 72 different test images using a batch-size of 4, which took roughly 6 days. These samples were generated on a server with 4 NVIDIA V100 GPUs, each with 32 GB of memory. We used 4 samples per batch (and recorded the time to generate 4 samples in Table 2.1) because the code from [38] is written to generate one sample per GPU.

D.2.2 CelebA-HQ Inpainting

Our cGAN. For our generator and discriminator, we use the CoModGAN networks from [111]. Unlike CoModGAN, however, we train our cGAN with $\mathcal{L}_{1,\mathsf{SD},P_{\mathsf{rc}}}$ regularization and we do not use MBSD at the discriminator. We use the same general training and testing procedure described in Sec. 2.3.2, but with $\beta_{\mathsf{adv}} = 5\text{e-}3$, $n_{\mathsf{batch}} = 100$, and 110 epochs of cGAN training. Running PyTorch on a server with 4 Tesla A100 GPUs, each with 82 GB of memory, the training takes approximately 2 days.

CoModGAN. We use the PyTorch implementation of CoModGAN from [103] and train the model to inpaint a 128×128 centered square on 256×256 CelebA-HQ images. The total training time on a server with 4 NVIDIA A100 GPUs, each with 82 GB of memory, is roughly 2 days.

Score-Based SDE. For the inpainting experiment in Sec. 2.3.3, we compare against Song et al.'s more recent SDE technique [87], for which we use the publicly available pretrained weights, the suggested settings for the 256×256 CelebA-HQ dataset, and the code from the official PyTorch implementation [88]. We generate 32 samples for all 1 000 images in our test set, using a batch-size of 20 and generating 32 samples for each batch element concurrently. The total generation time on a server with 4 NVIDIA A100 GPUs, each with 82 GB of memory, is roughly 9 days.

D.3 Implementation Details for Chapter 3

In each experiment, all cGANs were trained using the Adam optimizer with a learning rate of 10^{-3}, $\beta_1 = 0$, and $\beta_2 = 0.99$ as in [42]. The code for our model can be found here: `https://github.com/matt-bendel/pcaGAN`.

D.3.1 Synthetic Gaussian Data

Algorithm 7 captures how we construct the Gaussian priors used in Sec. 3.3.1.

We begin with dimension $d = 100$, generating random mean $\boldsymbol{\mu}_{\mathsf{x}}^{(100)} \in \mathbb{R}^{100}$ and eigenvalues $\{\lambda_k^{(100)}\}_{k=1}^{100}$. To construct the eigenvectors $\{\boldsymbol{v}_k^{(100)}\}_{k=1}^{100}$, we perform a QR decomposition on a 100×100 matrix with i.i.d. $\mathcal{N}(0,1)$ entries and set $\boldsymbol{v}_k^{(100)}$ as the kth column of $\boldsymbol{Q}$. For each remaining $d \in \{90, 80, \ldots, 10\}$, we construct $\boldsymbol{\mu}_{\mathsf{x}}^{(d)}$, $\{\lambda_k^{(d)}\}$, and $\boldsymbol{u}_k^{(d)}$ by truncating the previous quantities to ensure some level of continuity across d.

Algorithm 7 Gaussian prior generation

$\boldsymbol{\mu}_{\mathsf{x}}^{(100)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_{100})$
$\widetilde{\lambda}_k^{(100)} \sim \mathcal{N}(0,1)$ for $k = 1, \ldots, 100$
$\lambda_k^{(100)} = |\widetilde{\lambda}_k^{(100)}|$ for $k = 1, \ldots, 100$
$\boldsymbol{u}_k^{(100)} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_{100})$ for $k = 1, \ldots, 100$
$[\boldsymbol{Q}, \boldsymbol{R}] = \text{QRdecomp}([\boldsymbol{u}_1^{(100)} \boldsymbol{u}_2^{(100)} \ldots \boldsymbol{u}_{100}^{(100)}])$
$\boldsymbol{v}_k^{(100)} = [\boldsymbol{Q}]_{:k}$ for $k = 1, \ldots, 100$

for $d = 90, 80, \ldots, 10$ **do**
    $\boldsymbol{\mu}_{\mathsf{x}}^{(d)} = [\boldsymbol{\mu}_{\mathsf{x}}^{(100)}]_{0:d}$
    $\lambda_k^{(d)} = \lambda_k^{(100)}$ for $k = 1, \ldots, d$
    $\boldsymbol{u}_k^{(d)} = \boldsymbol{v}_{k,0:d}^{(100)}$ for $k = 1, \ldots, d$
    $[\boldsymbol{Q}, \boldsymbol{R}] = \text{QRdecomp}([\boldsymbol{u}_1^{(d)} \boldsymbol{u}_2^{(d)} \ldots \boldsymbol{u}_d^{(d)}])$
    $\boldsymbol{v}_k^{(d)} = [\boldsymbol{Q}]_{:k}$ for $k = 1, \ldots, d$
end for

D.3.2 Synthetic Gaussian Recovery

cGAN training. We choose $\beta_{\mathsf{adv}} = 10^{-5}$, $n_{\mathsf{batch}} = 64$, $P_{\mathsf{rc}} = 2$, and train for 100 epochs for both rcGAN and pcaGAN. Running PyTorch on a server with 4 Tesla A100 GPUs, each with 82 GB of memory, the cGAN training for $d = 100$ takes approximately 8 hours, with training time decreasing with smaller d. For pcaGAN, we choose $K = d$ for each d in this experiment (unless otherwise noted) and $\beta_{\mathsf{pca}} = 10^{-2}$.

cGAN architecture. We exploit the Gaussian nature of the problem, constructing $G_{\boldsymbol{\theta}}$ with two dense layers; one which takes in $\boldsymbol{y}$ as input and one which takes in $\boldsymbol{z}$ as input. The output of each layer is added, yielding $\widehat{\boldsymbol{x}}$. Similarly, $D_{\boldsymbol{\phi}}$ is comprised of a single dense layer which takes in the concatenation of $\boldsymbol{x}$ / $\widehat{\boldsymbol{x}}$ and $\boldsymbol{y}$ and outputs a scalar. We use this architecture for both rcGAN and pcaGAN. Note that there is no listed license for rcGAN.

NPPC. For NPPC, we use the suggested hyperparameters from [61] and opt to train the MMSE network *before* training NPPC. We use the suggested architectures from their Gaussian denoising experiment and train for 100 epochs with $n_{\mathsf{batch}} = 64$. We leverage the authors' implementation in [62], modifying it for this Gaussian problem. There is no listed license for NPPC.

D.3.3 MNIST Denoising

The MNIST dataset is available under the GNU general public license, which we respect through our use.

cGAN training. We choose $\beta_{\mathsf{adv}} = 10^{-5}$, $n_{\mathsf{batch}} = 64$, $P_{\mathsf{rc}} = 2$, and train for 125 epochs for both rcGAN and pcaGAN. Running PyTorch on a server with 4 Tesla A100 GPUs, each with 82 GB of memory, the cGAN training for $d = 100$ takes approximately 8 hours, with training time decreasing with smaller d. For pcaGAN, we train two models, one with $K = 5$ and one with $K = 10$. In both cases, $\beta_{\mathsf{pca}} = 10^{-1}$, $E_{\mathsf{evec}} = 25$, and $E_{\mathsf{eval}} = 50$.

cGAN architecture. For both cGANs, the generator is the standard U-Net which takes in the concatenation of $\boldsymbol{y}$ and code $\boldsymbol{z}$. The network consists of 3 pooling layers and 32 initial channels. The convolutions use a kernel of size 3×3, instance normalization, and leaky ReLU activations with a negative slope of 0.1. For the encoder portion of the U-Net, we use max pooling with a kernel size of 2×2 to reduce spatial resolution by a factor of 2. For the decoder portion of the U-Net, we use nearest-neighbor interpolation to increase spatial resolution by a factor of 2. The discriminator is simply the encoder portion of the U-Net with an additional dense

layer appended to map the U-Net's latent space to a scalar output. Note that there is no listed license for rcGAN.

NPPC. For NPPC, we do not modify the authors' implementation in [62] in any way. We first train the MMSE reconstruction network and then train the NPPC network. There is no listed license for NPPC.

D.3.4 Accelerated MRI

For our MRI experiments, we use the fastMRI dataset which is available under the MIT license, which we respect through our use.

cGAN training. We choose $\beta_{\mathsf{adv}} = 10^{-5}$, $\beta_{\mathsf{pca}} = 10^{-2}$, $n_{\mathsf{batch}} = 2$, $P_{\mathsf{rc}} = 2$, $K = 1$, $E_{\mathsf{evec}} = 25$, and $E_{\mathsf{eval}} = 50$ for pcaGAN. For rcGAN, pscGAN , and Adler's cGAN, we use the hyperparameters and training procedure described in Sec. 2.3.2. All models were trained for 100 epochs using the Adam optimizer [50] with a learning rate of 10^{-3}, $\beta_1 = 0$, and $\beta_2 = 0.99$, as in [42]. Running PyTorch on a server with 4 Tesla A100 GPUs, each with 82 GB of memory, the training of each MRI cGAN took approximately 1 day. Note that there is no listed license for rcGAN, pscGAN, or Adler and Öktem's cGAN.

cGAN architecture. All four cGANs used the same generator and discriminator architectures described in Sec. 2.3.2, except that Adler and Öktem's discriminator used extra input channels to facilitate the 3-input loss.

E2E-VarNet. For the Sriram et al.'s E2E-VarNet [89], we use the same training procedure and hyperparameters outlined in [39] with modification to the GRO sampling pattern. As in [39], we use the SENSE-based coil-combined image as the ground truth

instead of the RSS image. The E2E-VarNet is available under the MIT license, which we respect.

Langevin approach. For Jalal et al.'s MRI approach [39], we do not modify the authors' implementation from [38] other than replacing the default sampling pattern with the GRO sampling mask. We borrow both generated samples and results from Chapter 2. The authors' code is available under the MIT license, which we respect.

D.3.5 FFHQ Inpainting

For our inpainting experiment, we use the FFHQ dataset which is available under the Creative Commons BY-NC-SA 4.0 license, which we respect through our use.

cGAN training. We choose $\beta_{\mathsf{adv}} = 5 \times 10^{-3}$, $\beta_{\mathsf{pca}} = 10^{-3}$, $n_{\mathsf{batch}} = 5$, $P_{\mathsf{rc}} = 2$, $K = 2$, $E_{\mathsf{evec}} = 25$, and $E_{\mathsf{eval}} = 50$ for pcaGAN. Running PyTorch on a server with 4 Tesla A100 GPUs, each with 82 GB of memory, the training of our cGAN took approximately 1.5 days.

cGAN architecture. As in Chapter 2, we use the CoModGAN networks from [111] which extend the StyleGAN2 [45] network. The StyleGAN2 architecture is available under the NVIDIA Source Code License, which we respect.

rcGAN. We follow the training procedure outlined in Chapter 2, only modifying the inpainting mask to be random. The total training time on a server with 4 NVIDIA A100 GPUs, each with 82 GB of memory, is roughly 1 day. There is no listed license for rcGAN.

pscGAN. We use the same training procedure outlined in Chapter 2, modifying the inpainting masks to be random and using the $\mathcal{L}_{2,P}$ objective described briefly in Sec. 4.1 with $P = 8$. The total training time on a server with 4 NVIDIA A100

GPUs, each with 82 GB of memory, is roughly 1.5 days. There is no listed license for pscGAN.

CoModGAN. We use the PyTorch implementation of CoModGAN from [103] and train the model. The total training time on a server with 4 NVIDIA A100 GPUs, each with 82 GB of memory, is roughly 1 day. There is no listed license for CoModGAN, beyond the NVIDIA Source Code License.

Diffusion Methods

For all three diffusion methods, we use the pretrained weights from [17].

DPS. We use the suggested settings for the 256×256 FFHQ dataset and the code from the official PyTorch implementation [16]. We found the LPIPS-minimizing step-size ζ via grid search over a 1000 image validation set. We generate 1 sample for all 20 000 images in our test set, using a batch-size of 1 and 1000 NFEs. The total generation time on a server with 4 NVIDIA A100 GPUs, each with 82 GB of memory, is roughly 9 days. There is no listed license for DPS.

DDNM. We use the code from the official PyTorch implementation [95]. We generate 1 sample for all 20 000 images in our test set, using a batch-size of 1 and 100 NFEs. The total generation time on a server with 4 NVIDIA A100 GPUs, each with 82 GB of memory, is roughly 1.5 days. There is no listed license for DDNM.

DDRM. We use the code from the official PyTorch implementation [49]. We generate 1 sample for all 20 000 images in our test set, using a batch-size of 1 and 20 NFEs. The total generation time on a server with 4 NVIDIA A100 GPUs, each with 82 GB of memory, is roughly 5.5 hours. There is no listed license for DDRM.

D.4 Implementation Details for Chapter 4

D.4.1 Speeding up CG

In this section, we describe a small modification to FIRE that can help to speed up the CG step. When CG is used to solve (4.10), its convergence speed is determined by the condition number of $\boldsymbol{A}^\top\boldsymbol{A} + (\sigma_{\mathsf{w}}^2/\nu)\boldsymbol{I}$ [56]. Thus CG can converge slowly when $\sigma_{\mathsf{w}}^2/\nu$ is small, which can happen in early DDfire iterations. To speed up CG, we propose to solve (4.10) using $\widehat{\sigma}_{\mathsf{w}}$ in place of σ_{w}, for some $\widehat{\sigma}_{\mathsf{w}} > \sigma_{\mathsf{w}}$. Since the condition number of $\boldsymbol{A}^\top\boldsymbol{A} + (\widehat{\sigma}_{\mathsf{w}}^2/\nu)\boldsymbol{I}$ is at most $\nu s_{\mathsf{max}}^2/\widehat{\sigma}_{\mathsf{w}}^2 + 1$, we can guarantee a conditional number of at most 10 001 by setting

$$\widehat{\sigma}_{\mathsf{w}}^2 = \nu s_{\mathsf{max}}^2 \max\{10^{-4}, \sigma_{\mathsf{w}}^2/(\nu s_{\mathsf{max}}^2)\}. \tag{D.11}$$

Although using $\widehat{\sigma}_{\mathsf{w}} > \sigma_{\mathsf{w}}$ in (4.10) will degrade the MSE of $\widehat{\boldsymbol{x}}$, the degradation is partially offset by the fact that less noise will be added when renoising $\boldsymbol{r}$. In any case, the modified (4.10) can be written as

$$\widehat{\boldsymbol{x}} = (\boldsymbol{A}^\top\boldsymbol{A}/\widehat{\sigma}_{\mathsf{w}}^2 + \boldsymbol{I}/\nu)^{-1}(\boldsymbol{A}^\top\boldsymbol{y}/\widehat{\sigma}_{\mathsf{w}}^2 + \overline{\boldsymbol{x}}/\nu) \tag{D.12}$$

$$= (\boldsymbol{A}^\top\boldsymbol{A}/\widehat{\sigma}_{\mathsf{w}}^2 + \boldsymbol{I}/\nu)^{-1}(\boldsymbol{A}^\top[\boldsymbol{A}\boldsymbol{x}_0 + \sigma_{\mathsf{w}}\boldsymbol{w}]/\widehat{\sigma}_{\mathsf{w}}^2 + [\boldsymbol{x}_0 - \sqrt{\nu}\boldsymbol{e}]/\nu) \tag{D.13}$$

$$= \boldsymbol{x}_0 + (\boldsymbol{A}^\top\boldsymbol{A}/\widehat{\sigma}_{\mathsf{w}}^2 + \boldsymbol{I}/\nu)^{-1}(\boldsymbol{A}^\top\boldsymbol{w}\sigma_{\mathsf{w}}/\widehat{\sigma}_{\mathsf{w}}^2 - \boldsymbol{e}/\sqrt{\nu}), \tag{D.14}$$

in which case $\widehat{\boldsymbol{x}} \sim \mathcal{N}(\boldsymbol{x}_0, \boldsymbol{C})$ with covariance

$$\boldsymbol{C} = (\boldsymbol{A}^\top\boldsymbol{A}/\widehat{\sigma}_{\mathsf{w}}^2 + \boldsymbol{I}/\nu)^{-1}\big(\boldsymbol{A}^\top\boldsymbol{A}\sigma_{\mathsf{w}}^2/\widehat{\sigma}_{\mathsf{w}}^4 + \boldsymbol{I}/\nu\big)(\boldsymbol{A}^\top\boldsymbol{A}/\widehat{\sigma}_{\mathsf{w}}^2 + \boldsymbol{I}/\nu)^{-1} \tag{D.15}$$

$$= \big(\boldsymbol{V}\boldsymbol{S}^\top\boldsymbol{S}\boldsymbol{V}^\top/\widehat{\sigma}_{\mathsf{w}}^2 + \boldsymbol{I}/\nu\big)^{-1}\big(\boldsymbol{V}\boldsymbol{S}^\top\boldsymbol{S}\boldsymbol{V}^\top\sigma_{\mathsf{w}}^2/\widehat{\sigma}_{\mathsf{w}}^4 + \boldsymbol{I}/\nu\big)\big(\boldsymbol{V}\boldsymbol{S}^\top\boldsymbol{S}\boldsymbol{V}^\top/\widehat{\sigma}_{\mathsf{w}}^2 + \boldsymbol{I}/\nu\big)^{-1} \tag{D.16}$$

$$= \boldsymbol{V}\,\mathrm{Diag}(\boldsymbol{\gamma})\boldsymbol{V}^\top \text{ for } \gamma_i = \frac{s_i^2\sigma_{\mathsf{w}}^2/\widehat{\sigma}_{\mathsf{w}}^4 + 1/\nu}{[s_i^2/\widehat{\sigma}_{\mathsf{w}}^2 + 1/\nu]^2}. \tag{D.17}$$

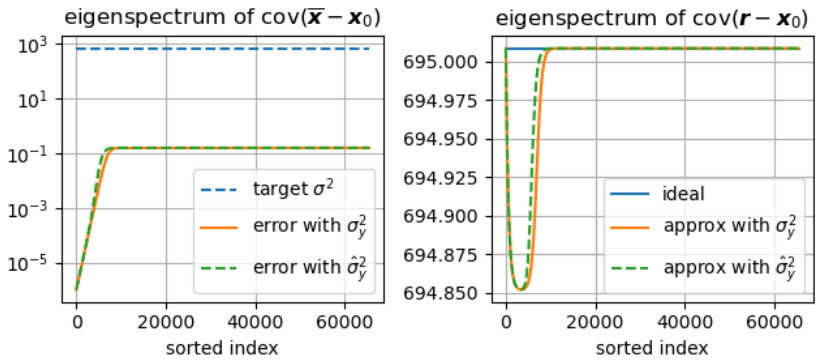


Figure D.1: For FFHQ Gaussian deblurring, the left plot shows the eigenspectrum of the error covariance $\mathrm{Cov}\{\overline{\boldsymbol{x}} - \boldsymbol{x}_0\}$ with either $\widehat{\sigma}_{\mathsf{w}}^2$ from (D.11) (if CG speedup) or $\widehat{\sigma}_{\mathsf{w}}^2 = \sigma_{\mathsf{w}}^2$ (if no CG speedup), as well as the eigenspectrum of the target error covariance $\sigma^2 \boldsymbol{I}$ to aim for when renoising. The right plot shows the eigenvalues of the renoised error covariance $\mathrm{Cov}\{\boldsymbol{r} - \boldsymbol{x}_0\}$ for the ideal case when $\boldsymbol{\Sigma}$ is used (possible with SVD) and the case when $\widehat{\boldsymbol{\Sigma}}$ from (4.16) is used (if no SVD), with either $\widehat{\sigma}_{\mathsf{w}}^2$ or σ_{w}^2. Here we used $\sigma_{\mathsf{w}}^2 = 10^{-6}$, $\nu = 0.16$ (corresponding to the first FIRE iteration of the first DDIM step), and $\rho = 35.7$ (corresponding to the example in Fig. 4.2).

The desired renoising variance then becomes

$$\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I} - \boldsymbol{C} = \boldsymbol{V}\,\mathrm{Diag}(\boldsymbol{\lambda})\boldsymbol{V}^\top \text{ for } \lambda_i = \sigma^2 - \frac{s_i^2 \sigma_{\mathsf{w}}^2 / \widehat{\sigma}_{\mathsf{w}}^4 + 1/\nu}{[s_i^2/\widehat{\sigma}_{\mathsf{w}}^2 + 1/\nu]^2} \tag{D.18}$$

and we can generate the colored noise $\boldsymbol{c}$ via (4.15) if the SVD is practical. If not, we approximate $\boldsymbol{\Sigma}$ by

$$\widehat{\boldsymbol{\Sigma}} = (\sigma^2 - \nu)\boldsymbol{I} + \xi \boldsymbol{A}^\top \boldsymbol{A} \;\text{ with }\; \xi = \frac{1}{s_{\mathsf{max}}^2}\left(\nu - \frac{s_{\mathsf{max}}^2 \sigma_{\mathsf{w}}^2 / \widehat{\sigma}_{\mathsf{w}}^4 + 1/\nu}{[s_{\mathsf{max}}^2/\widehat{\sigma}_{\mathsf{w}}^2 + 1/\nu]^2}\right) \tag{D.19}$$

and generate the colored noise $\boldsymbol{c}$ via (4.18). It is straightforward to show that $\xi \geq 0$ whenever $\widehat{\sigma}_{\mathsf{w}} \geq \sigma_{\mathsf{w}}$, in which case $\sigma^2 \geq \nu$ guarantees that $\widehat{\boldsymbol{\Sigma}}$ is a valid covariance matrix. Figure D.1 shows the close agreement between the ideal and approximate $\widehat{\boldsymbol{\Sigma}}$-renoised error spectra both when $\widehat{\sigma}_{\mathsf{w}} = \sigma_{\mathsf{w}}$ and when $\widehat{\sigma}_{\mathsf{w}} > \sigma_{\mathsf{w}}$.

D.4.2 Inverse Problems

For the linear inverse problems, the measurements were generated as

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_0 + \sigma_{\mathsf{w}}\boldsymbol{w}, \quad \boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}) \tag{D.20}$$

with appropriate $\boldsymbol{A}$. For box inpainting, Gaussian deblurring, and super-resolution we used the $\boldsymbol{A}$ and $\boldsymbol{A}^\top$ implementations from [49]. For motion deblurring, we implemented our own $\boldsymbol{A}$ and $\boldsymbol{A}^\top$ with reflect padding. All methods used these operators implementations except DiffPIR, which used the authors' implementations. Motion-blur kernels were generated using [12].

D.4.3 Evaluation Protocol

For the linear inverse problems, we run each method once for each measurement $\boldsymbol{y}$ in the 1000-sample test set and compute average PSNR, average LPIPS, and FID from the resulting recoveries.

D.4.4 Unconditional Diffusion Models

For the FFHQ experiments, all methods used the pretrained model from [17]. For the ImageNet experiments, all methods used the pretrained model from [26]. In both cases, $T = 1000$.

D.4.5 Recovery Methods

DDfire. Our Python/Pytorch codebase is a modification of the DPS codebase from [16] and is available at `https://github.com/matt-bendel/DDfire`. For all but one row of the ablation study in Table 4.1 and the dashed line in Fig. 4.4, we ran DDfire without an SVD and thus with the approximate renoising in (4.18).

Table D.1: Hyperparameter values used for DDfire.

		Inpaint (box)		Deblur (Gaussian)		Deblur (Motion)		4× Super-resolution	
Dataset	σ_{w}	K	δ	K	δ	K	δ	K	δ
FFHQ	0.05	100	0.50	650	0.60	500	0.20	650	0.60
ImageNet	0.05	100	0.50	500	0.20	500	0.20	650	0.60

For the linear inverse problems, unless noted otherwise, we ran DDfire for 1000 NFEs using $\eta_{\mathsf{ddim}} = 1.5$, and we did not use stochastic denoising (i.e., $\widehat{\nu}_\phi(\sigma) = 0\ \forall\sigma$ in Alg. 2), as suggested by our ablation study. We tuned the (K, δ) hyperparameters to minimize LPIPS on a 100-sample validation set, yielding the parameters in Table D.1. For the runtime results in Fig. 4.4, we used $\eta_{\mathsf{ddim}} = 1.0$ for $N_{\mathsf{tot}} \in \{50, 100, 200, 500\}$ and $\eta_{\mathsf{ddim}} = 0$ for $N_{\mathsf{tot}} = 20$, and we used $K = \text{NFE}/2$ and $\delta = 0.2$ for all cases. For the ν-estimation step in Alg. 2, we used $\|\boldsymbol{A}\|_F^2 \approx \frac{1}{L}\sum_{l=1}^{L}\|\boldsymbol{A}\boldsymbol{w}_l\|^2$ with i.i.d. $\boldsymbol{w}_l \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ and $L = 25$.

DDRM. We ran DDRM for 20 NFEs using the authors' implementation from [49] with minor changes to work with our codebase.

DiffPIR. We ran DiffPIR for 20 NFEs using the authors' implementation from [113] without modification. Hyperparameters were set according to the reported values in [114].

ΠGDM. We ran ΠGDM for 100 NFEs. Since the authors do not provide a ΠGDM implementation for noisy inverse problems in [63], we coded ΠGDM ourselves in Python/PyTorch. With problems for which an SVD is available, we computed

$(\boldsymbol{A}\boldsymbol{A}^\top + \zeta_k \boldsymbol{I})^{-1}$ using the efficient SVD implementation of $\boldsymbol{A}$ from the DDRM codebase [49], and otherwise we used CG.

DDS. We ran DDS for 100 NFEs. We leveraged the authors' implementation in [19] to reimplement DDS in our codebase. We tuned the DDS regularization parameter γ_{dds} via grid search and used $\eta_{\mathsf{ddim}} = 0.8$ and 50 CG iterations.

DPS. For the linear inverse problems, we ran DPS for 1000 NFEs using the authors' implementation from [16] without modification, using the suggested tuning from [17, Sec. D.1].

RED-diff. We ran RED-diff for 1000 NFEs using the authors' implementation from [63], with minor changes to work with our codebase. We tuned the RED-diff learning rate, λ, and data fidelity weight v_t to minimize LPIPS with a 100-image validation set.

DAPS. We ran DAPS for 1000 NFEs using the authors' implementation from [104], with minor changes to work in our codebase. The tuning parameters were set as in [105].

D.4.6 Compute

All experiments were run on a single NVIDIA A100 GPU with 80GB of memory. The runtime for each method on the GPU varies, as shown in Figure 4.4.

D.4.7 DDfire Hyperparameter Tuning Curves

Figures D.2–D.5 show PSNR and LPIPS over the parameter grids $K \in \{10, 20, 50, 100, 200, 500, 1000\}$ and $\delta \in \{0.05, 0.1, 0.2, 0.5, 0.75\}$ for box inpainting, Gaussian deblurring, motion deblurring, and 4x super resolution, respectively, using 50 ImageNet validation images. While there are some noticeable trends, DDfire is relatively sensitive to hyperparemeter selection, particularly in cases where the degradation is not global (e.g., in inpainting).

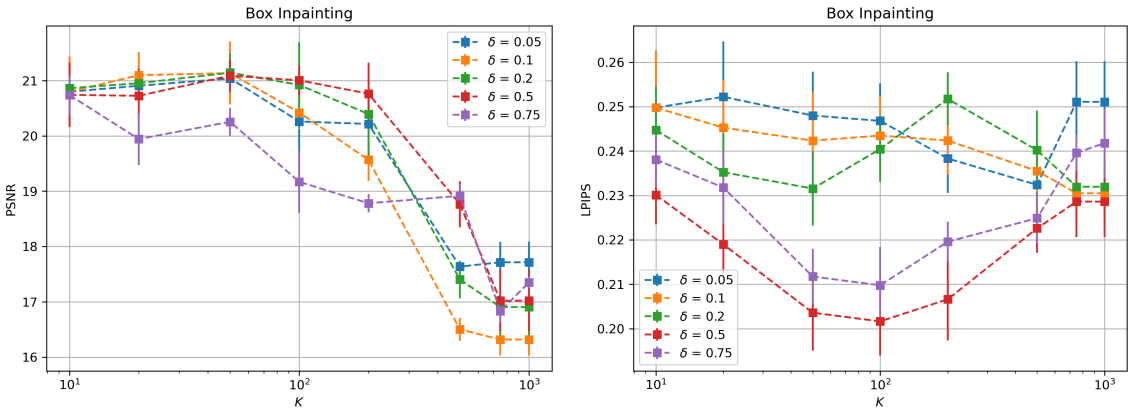


Figure D.2: PSNR and LPIPS tuning results for box inpainting with 50 ImageNet validation images.

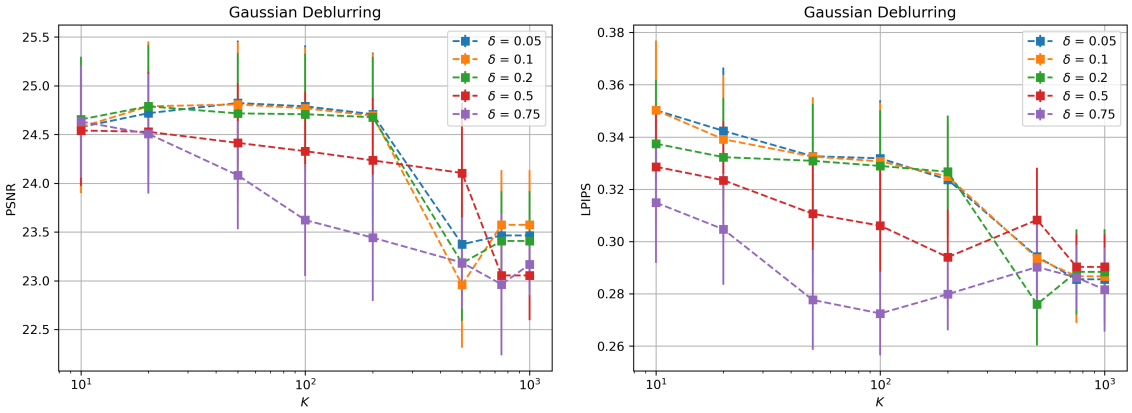


Figure D.3: PSNR and LPIPS tuning results for gaussian deblurring with 50 ImageNet validation images.

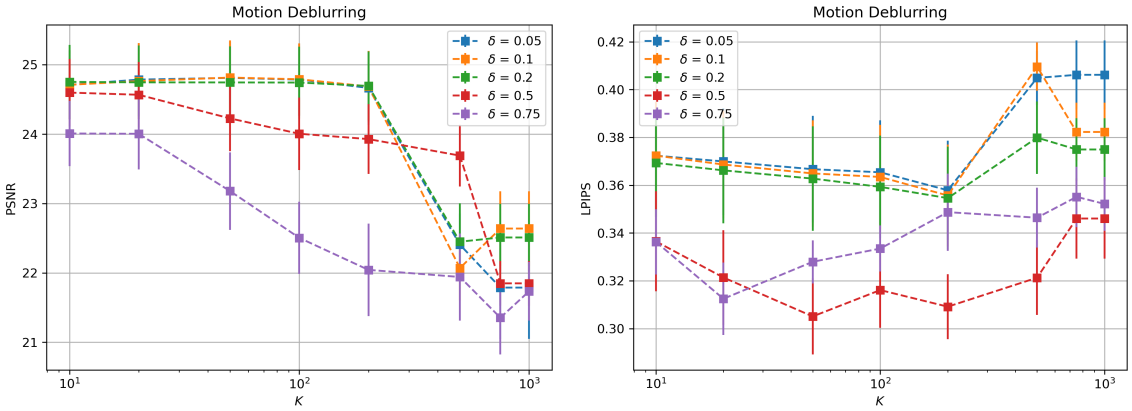


Figure D.4: PSNR and LPIPS tuning results for motion deblurring with 50 ImageNet validation images.

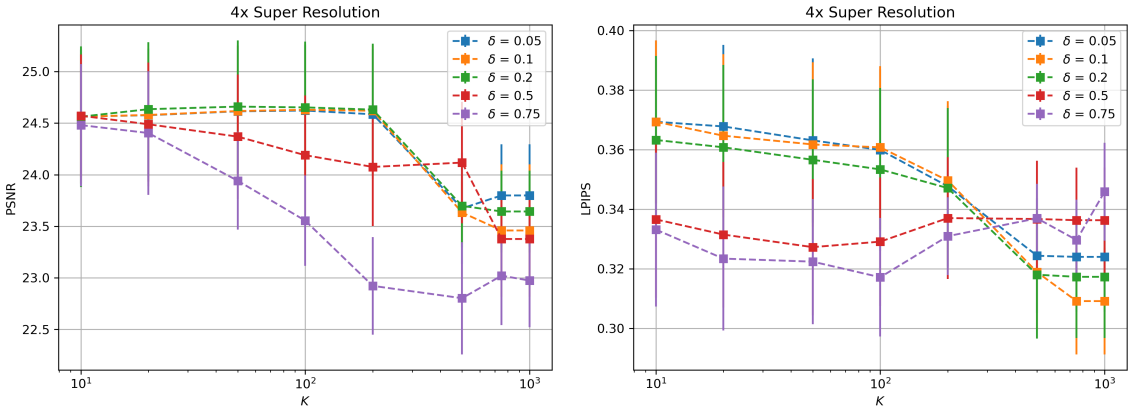


Figure D.5: PSNR and LPIPS tuning results for 4x super resolution with 50 ImageNet validation images.

Appendix E: Additional Reconstruction Plots

Here, we present some additional reconstruction plots for experiments in Chapters 2 and 3.

E.1 Additional Reconstructions for Chapter 2

E.1.1 MRI at Acceleration $R = 4$

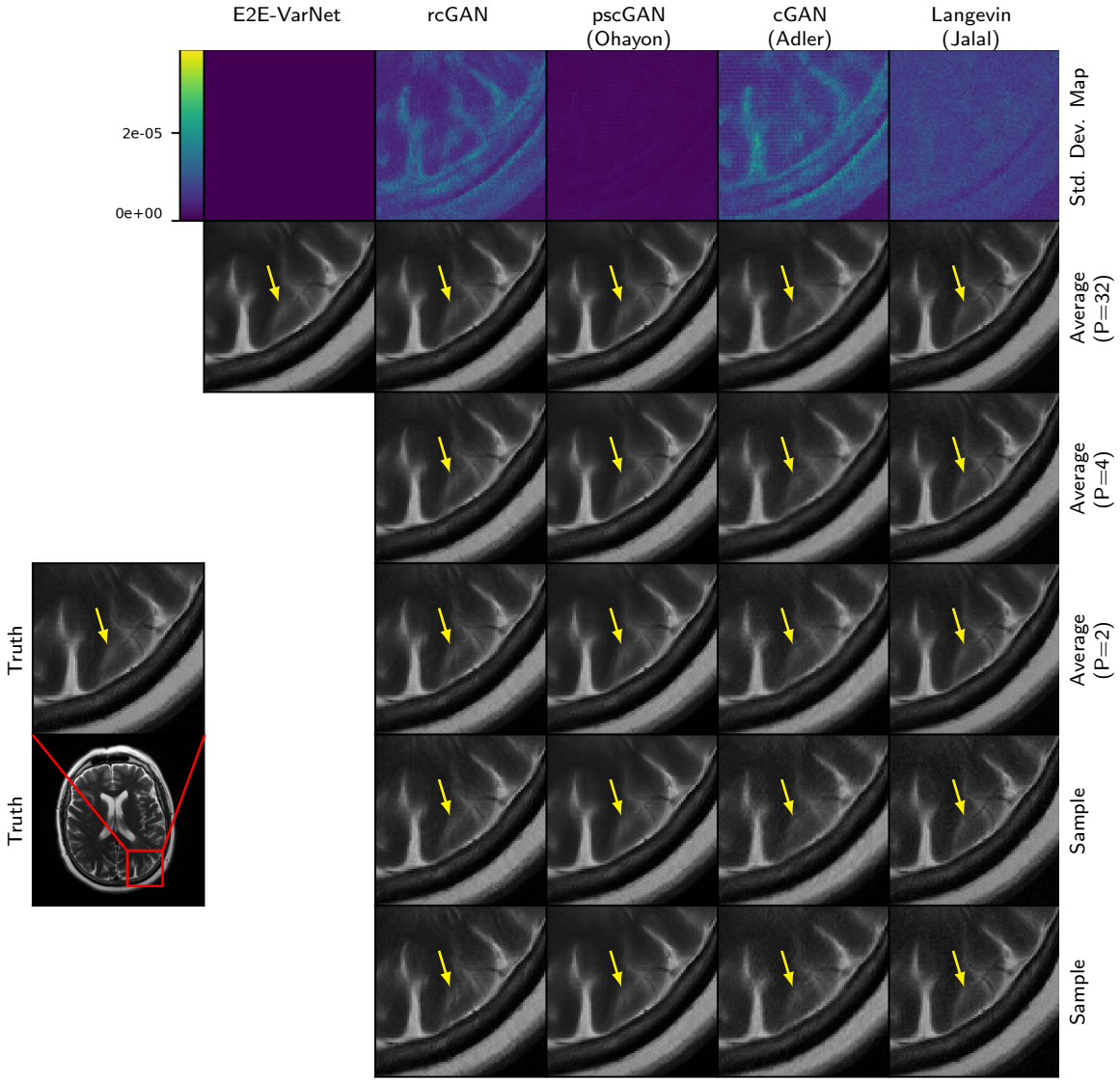


Figure E.1: Example $R = 4$ MRI reconstruction. Row one: pixel-wise SD with $P = 32$, Row two: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 32$, Row three: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 4$, Row four: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 2$, Rows five and six: posterior samples. The arrows indicate regions of meaningful variation across posterior samples.

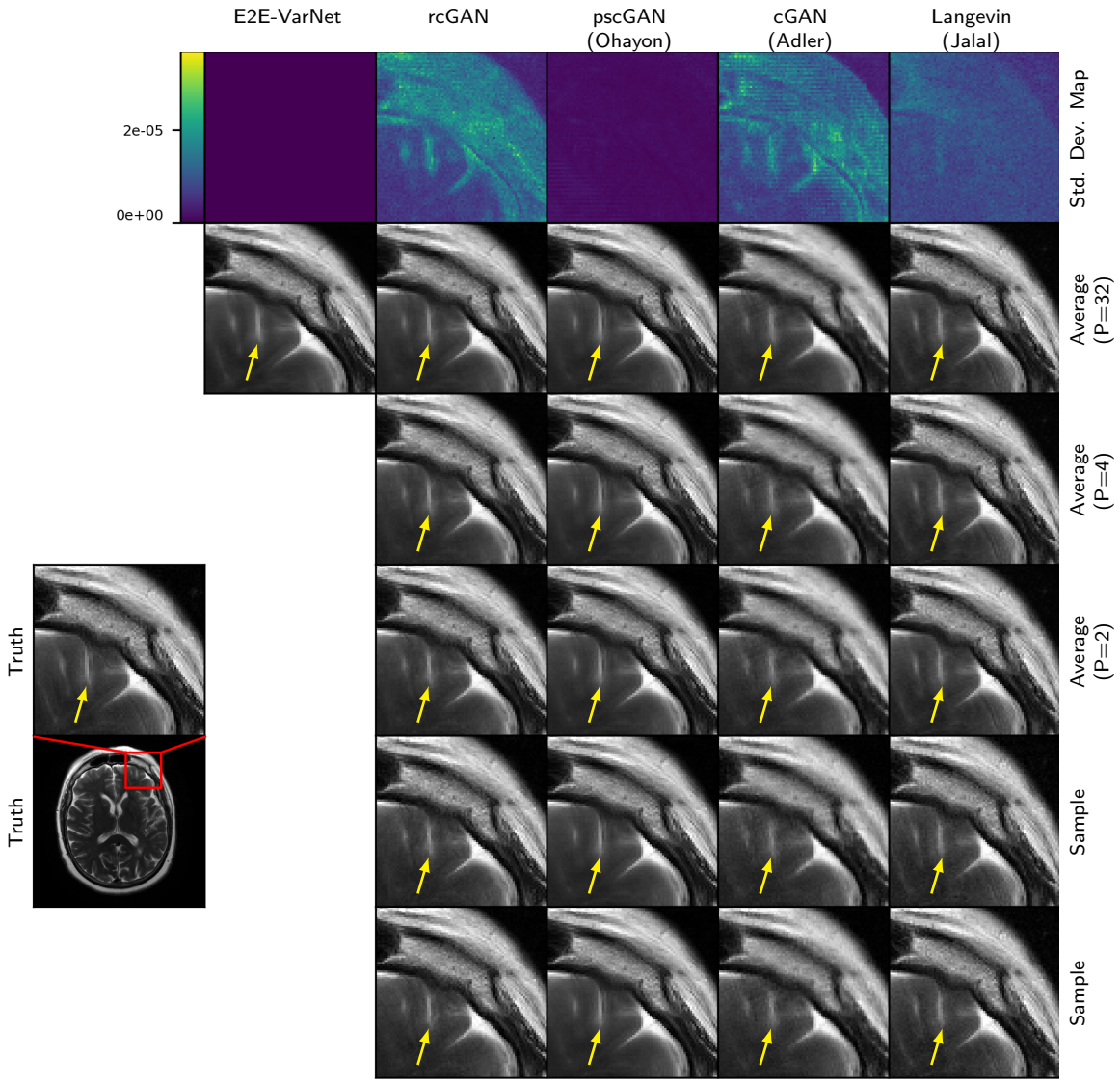


Figure E.2: Example $R = 4$ MRI reconstruction. Row one: pixel-wise SD with $P = 32$, Row two: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 32$, Row three: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 4$, Row four: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 2$, Rows five and six: posterior samples. The arrows indicate regions of meaningful variation across posterior samples.

E.1.2 MRI at Acceleration $R = 8$

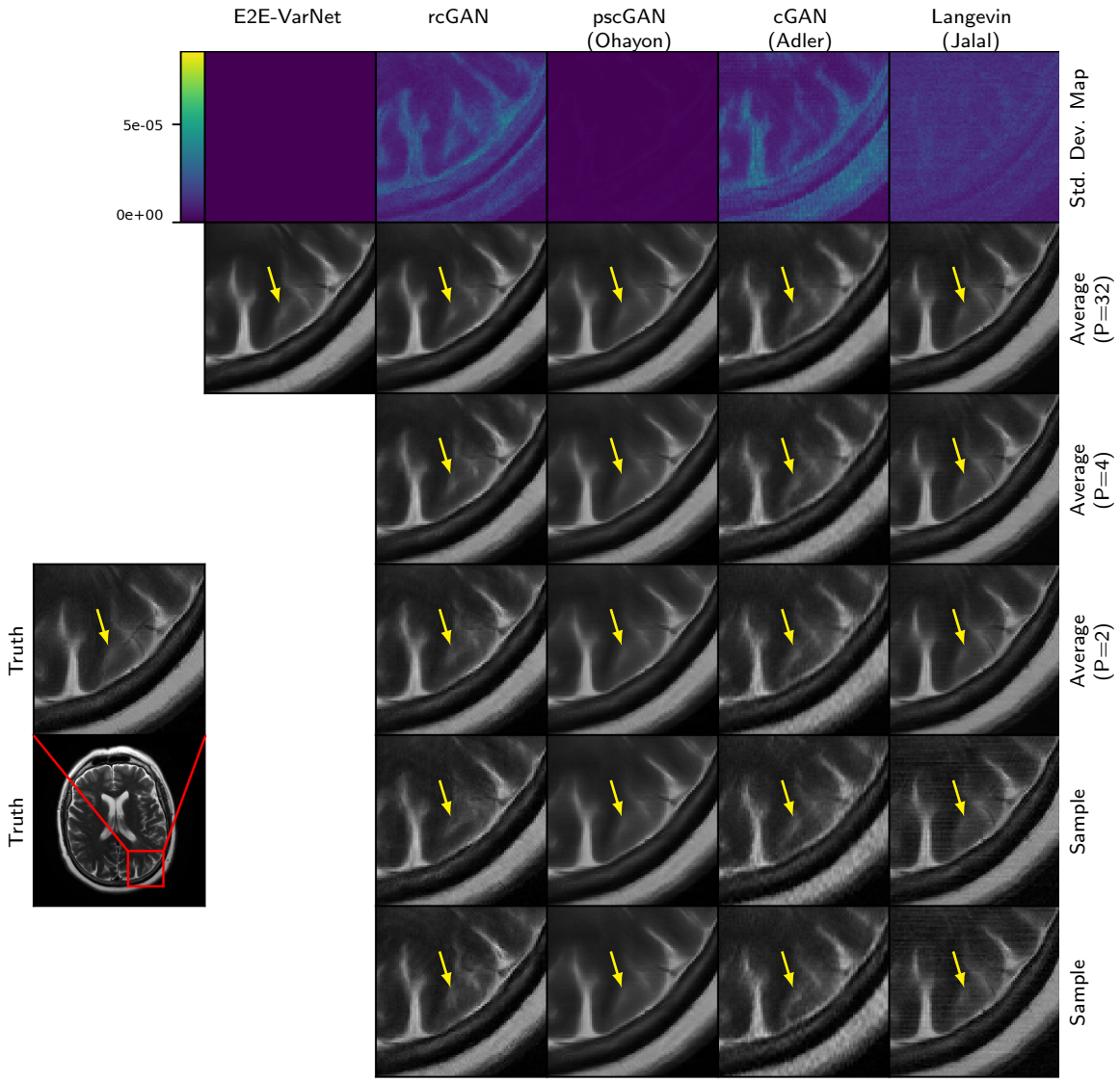


Figure E.3: Example $R = 8$ MRI reconstruction. Row one: pixel-wise SD with $P = 32$, Row two: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 32$, Row three: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 4$, Row four: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 2$, Rows five and six: posterior samples. The arrows indicate regions of meaningful variation across posterior samples.

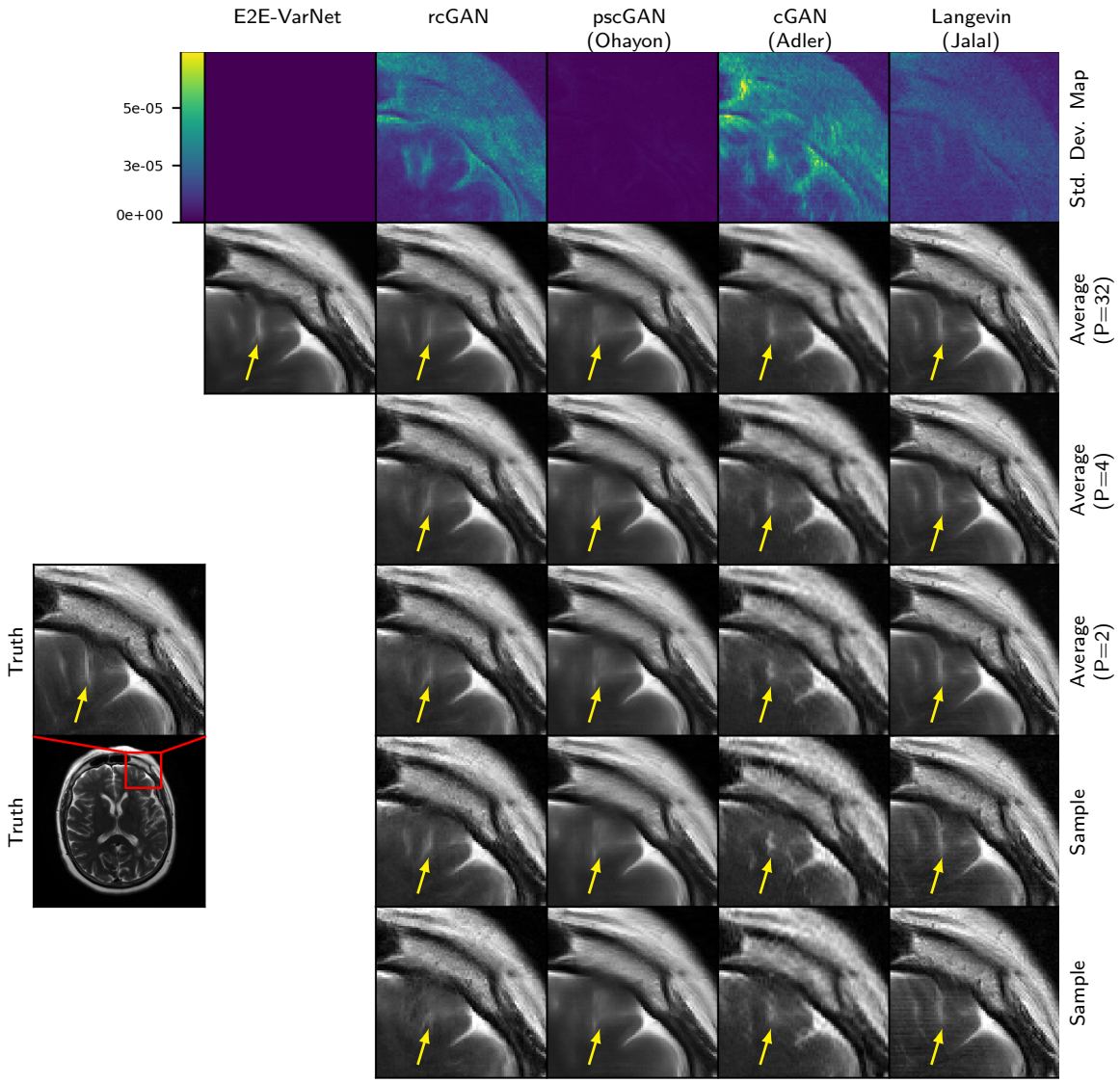


Figure E.4: Example $R = 8$ MRI reconstruction. Row one: pixel-wise SD with $P = 32$, Row two: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 32$, Row three: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 4$, Row four: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 2$, Rows five and six: posterior samples. The arrows indicate regions of meaningful variation across posterior samples.

E.1.3 Inpainting

Figure E.5: Example of inpainting a 128×128 square on a 256×256 resolution CelebA-HQ image.

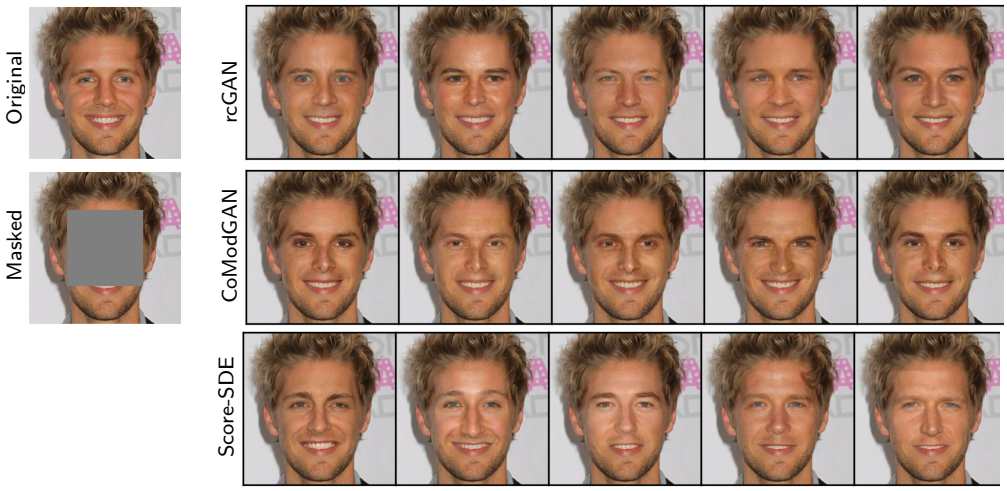


Figure E.6: Example of inpainting a 128×128 square on a 256×256 resolution CelebA-HQ image.

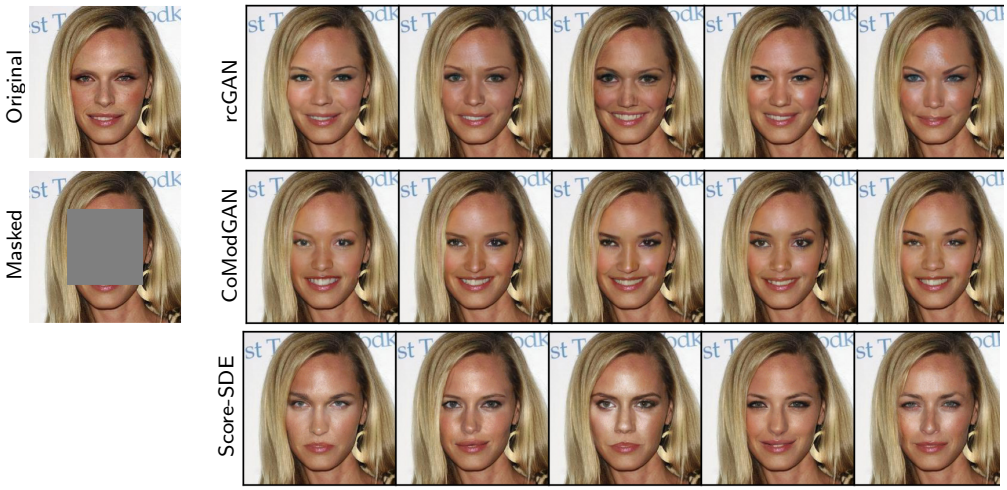


Figure E.7: Example of inpainting a 128×128 square on a 256×256 resolution CelebA-HQ image.

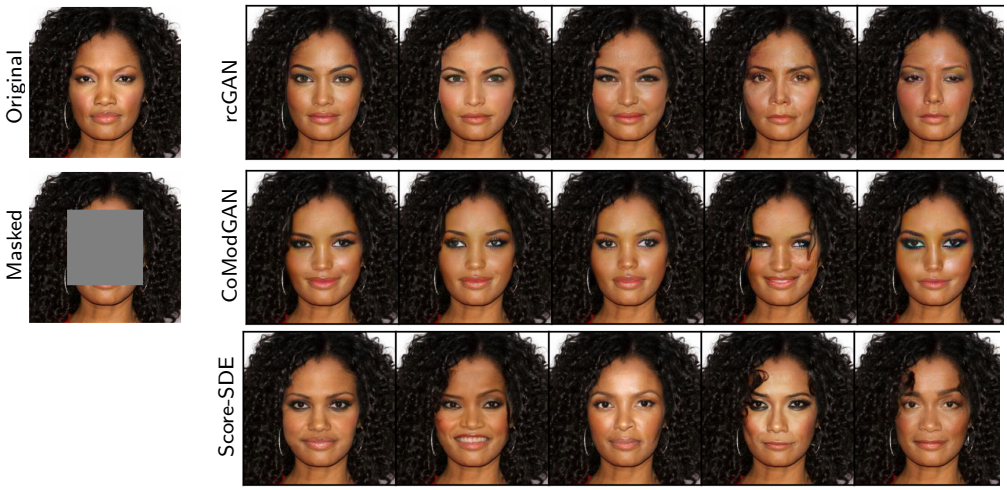


Figure E.8: Example of inpainting a 128×128 square on a 256×256 resolution CelebA-HQ image.

E.2 Additional Reconstructions for Chapter 3

E.2.1 MNIST Denoising

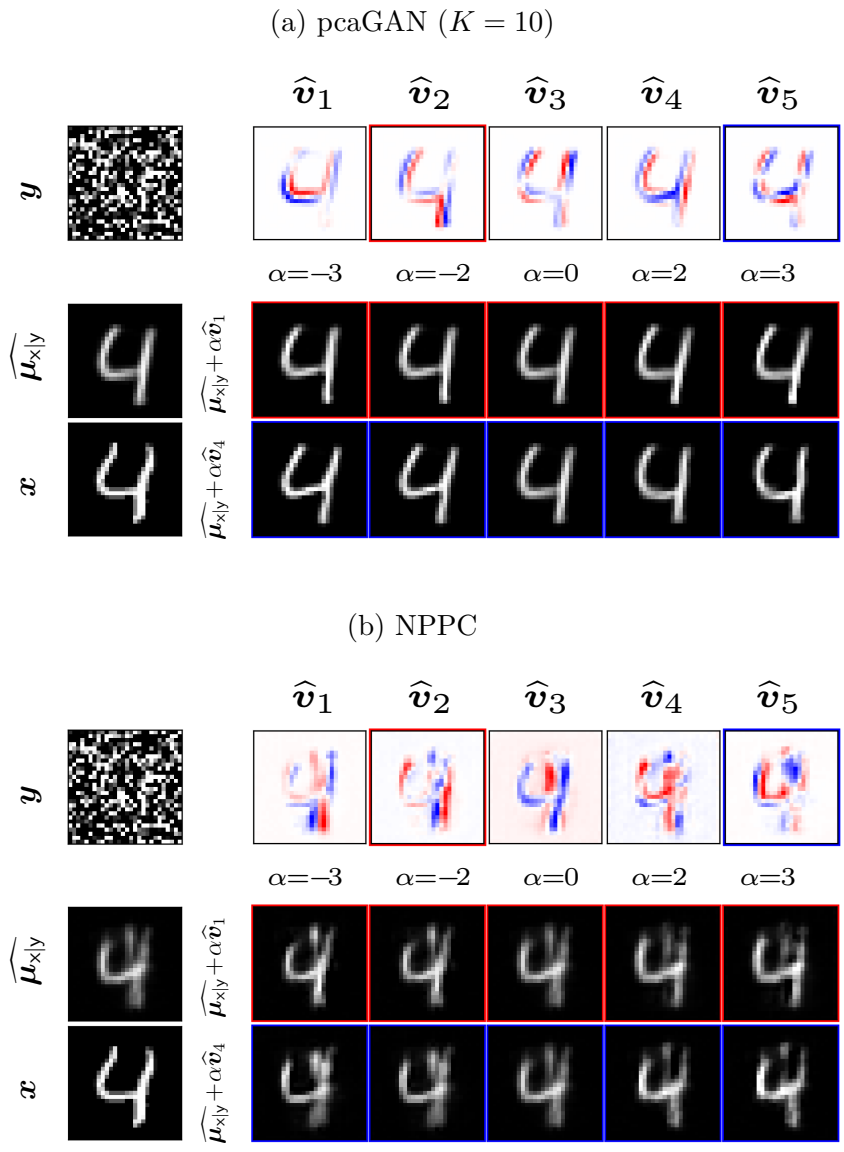
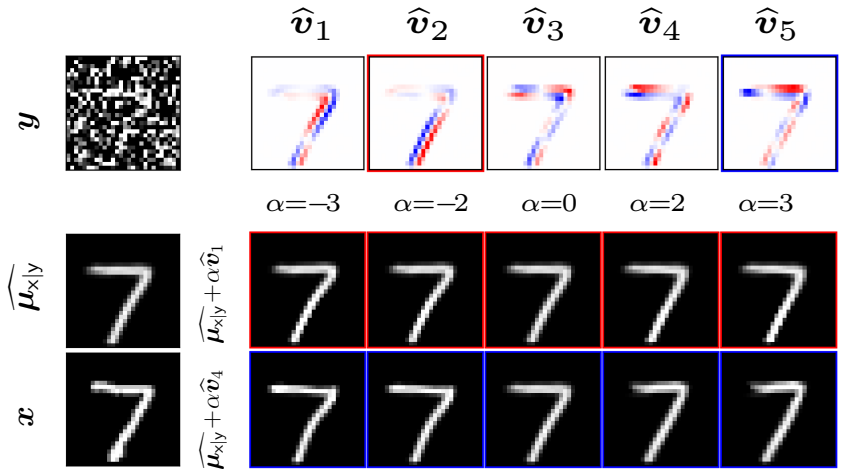


Figure E.9: For (a) pcaGAN and (b) NPPC, this figure shows the true image $\boldsymbol{x}$, noisy measurements $\boldsymbol{y}$, the conditional mean $\widehat{\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}}$, principal eigenvectors $\{\widehat{\boldsymbol{v}}_k\}$, and two perturbations of $\widehat{\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}}$.

(a) pcaGAN ($K = 10$)



(b) NPPC

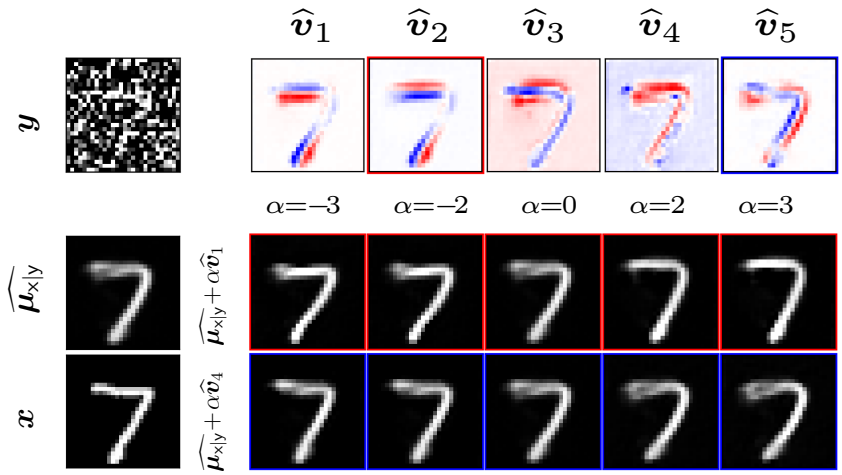
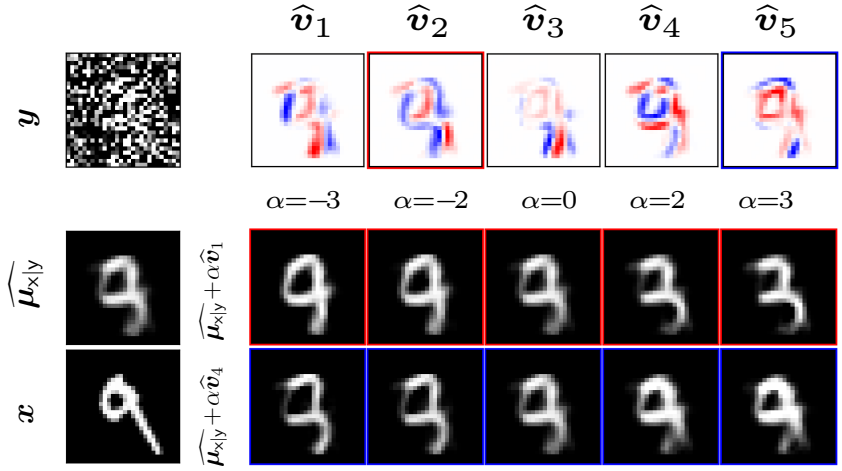


Figure E.10: For (a) pcaGAN and (b) NPPC, this figure shows the true image $\boldsymbol{x}$, noisy measurements $\boldsymbol{y}$, the conditional mean $\widehat{\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}}$, principal eigenvectors $\{\widehat{\boldsymbol{v}}_k\}$, and two perturbations of $\widehat{\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}}$.

(a) pcaGAN ($K = 10$)



(b) NPPC

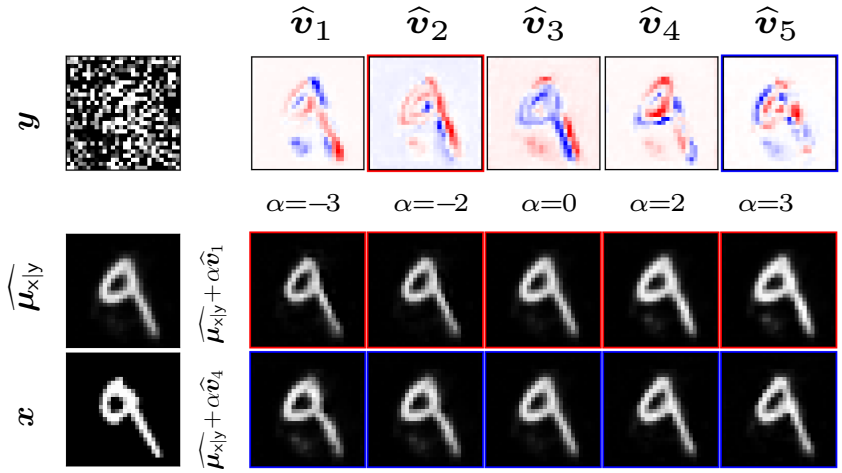


Figure E.11: For (a) pcaGAN and (b) NPPC, this figure shows the true image $\boldsymbol{x}$, noisy measurements $\boldsymbol{y}$, the conditional mean $\widehat{\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}}$, principal eigenvectors $\{\widehat{\boldsymbol{v}}_k\}$, and two perturbations of $\widehat{\boldsymbol{\mu}_{\mathsf{x}|\mathsf{y}}}$.

E.2.2 MRI at Acceleration $R = 4$

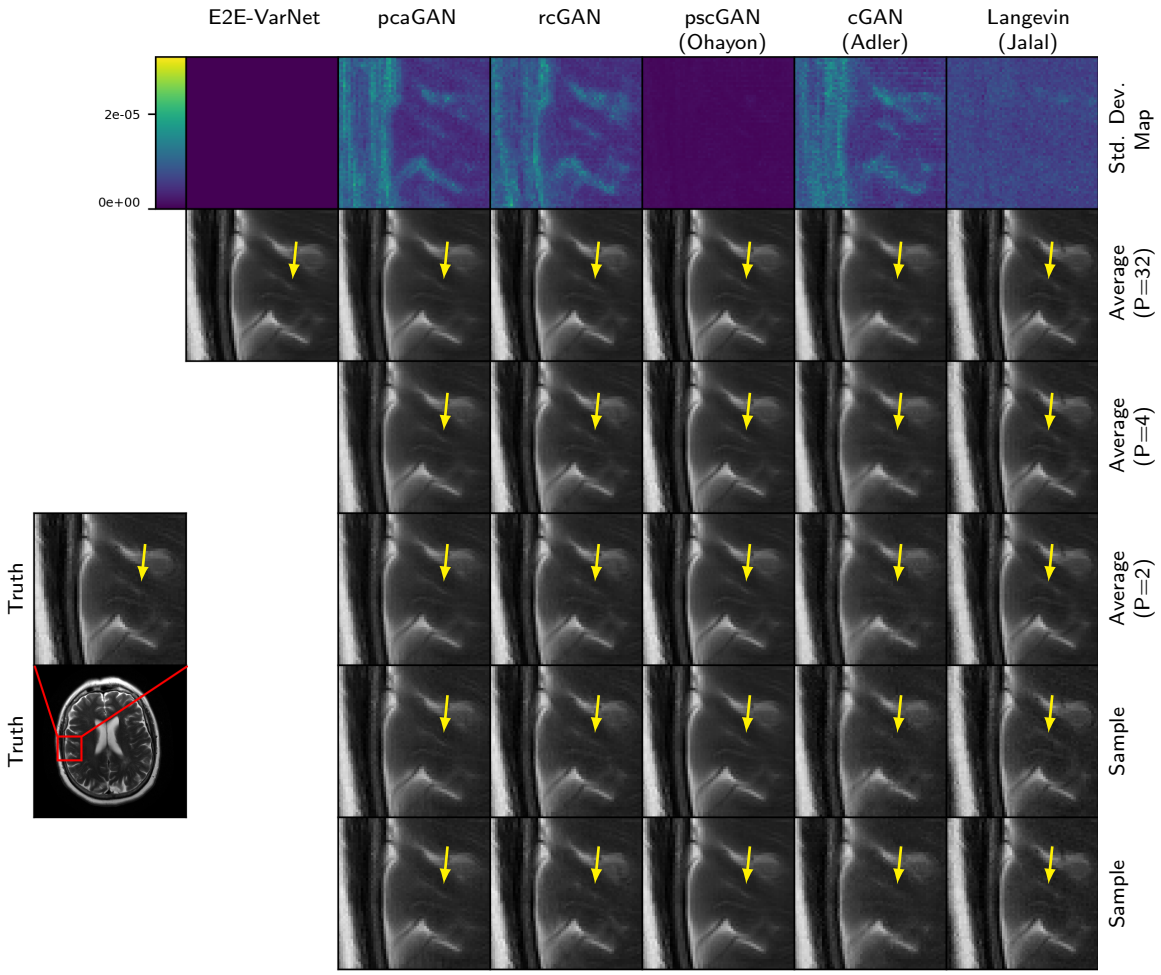


Figure E.12: Example $R = 4$ MRI reconstruction. Row one: pixel-wise SD with $P = 32$, Row two: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 32$, Row three: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 4$, Row four: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 2$, Rows five and six: posterior samples. The arrows indicate regions of meaningful variation across posterior samples.

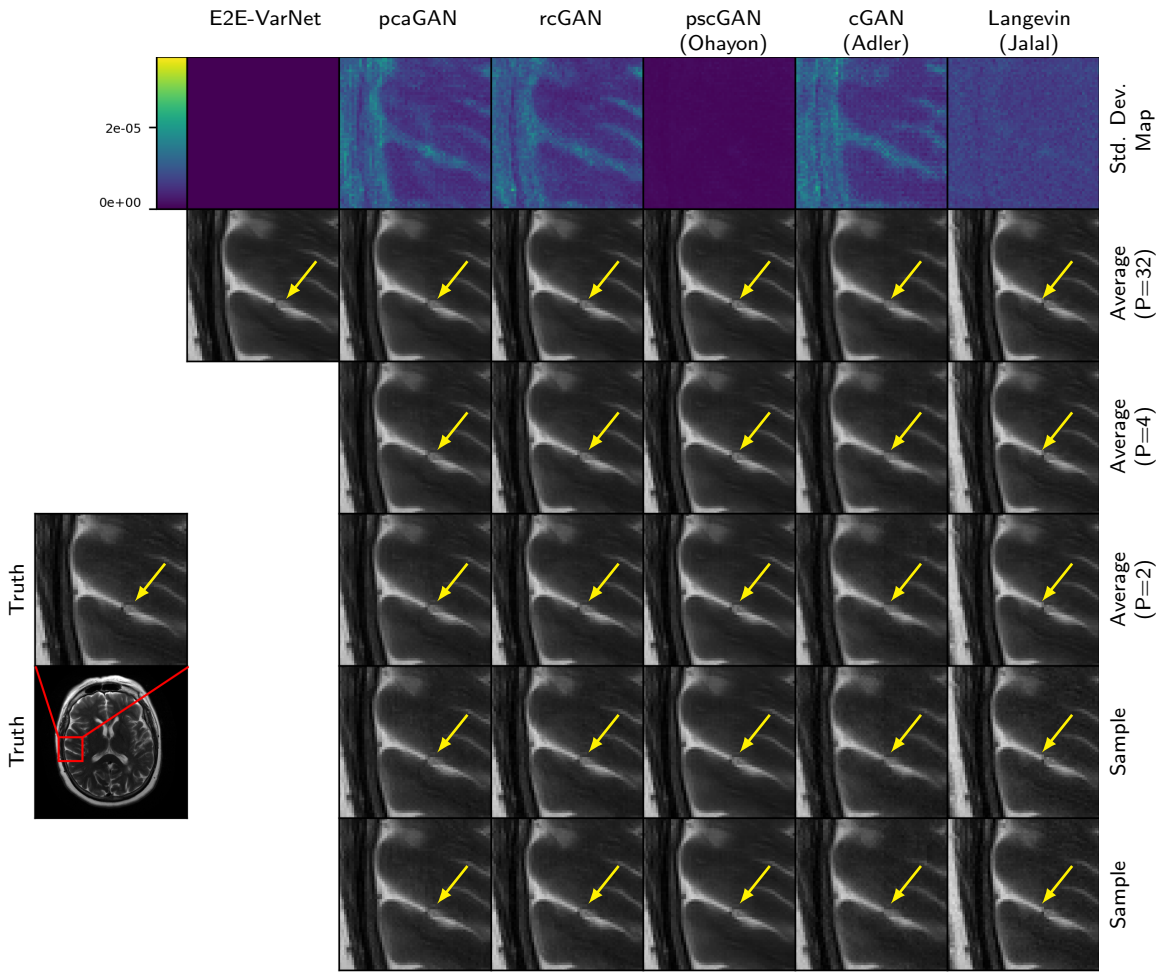


Figure E.13: Example $R = 4$ MRI reconstruction. Row one: pixel-wise SD with $P = 32$, Row two: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 32$, Row three: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 4$, Row four: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 2$, Rows five and six: posterior samples. The arrows indicate regions of meaningful variation across posterior samples.

E.2.3 MRI at Acceleration $R = 8$

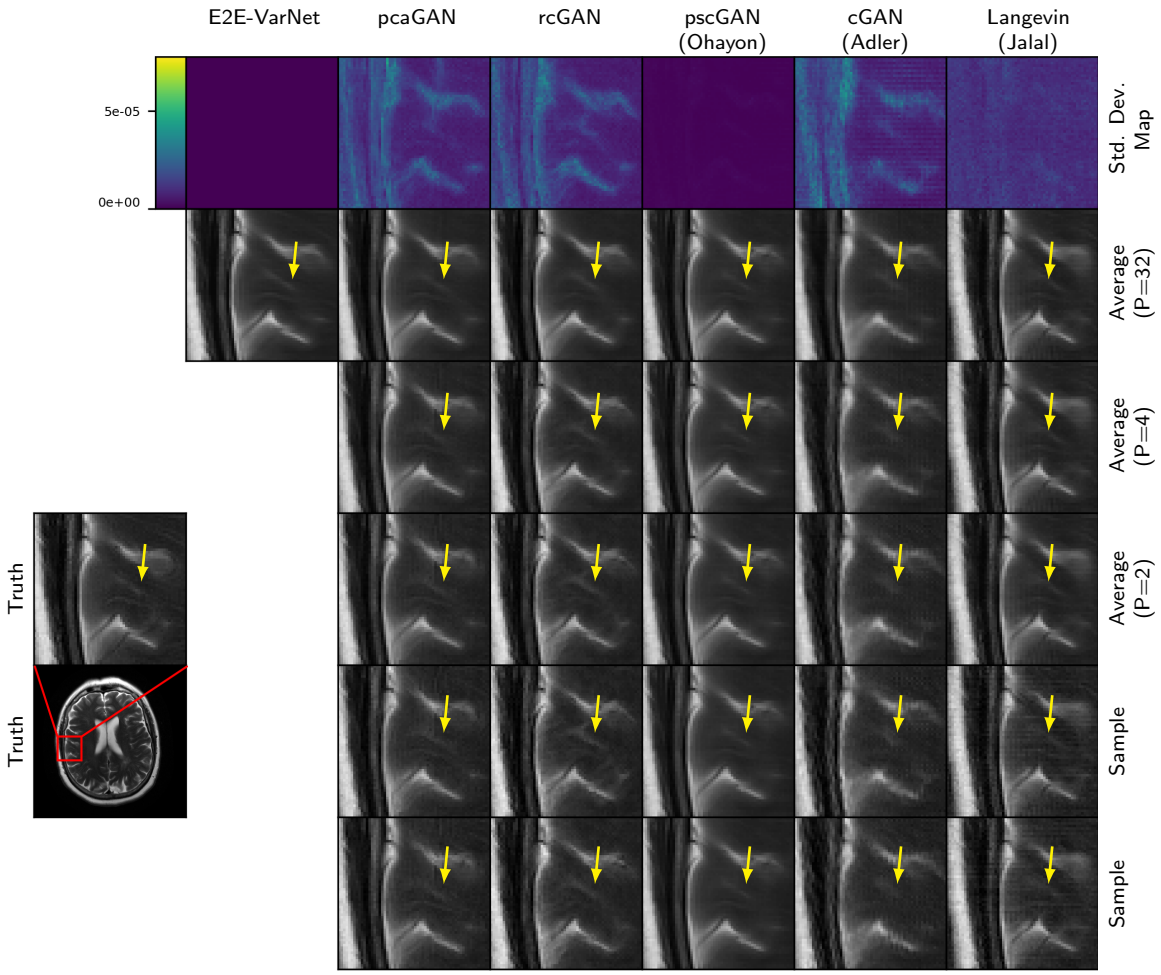


Figure E.14: Example $R = 8$ MRI reconstruction. Row one: pixel-wise SD with $P = 32$, Row two: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 32$, Row three: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 4$, Row four: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 2$, Rows five and six: posterior samples. The arrows indicate regions of meaningful variation across posterior samples.

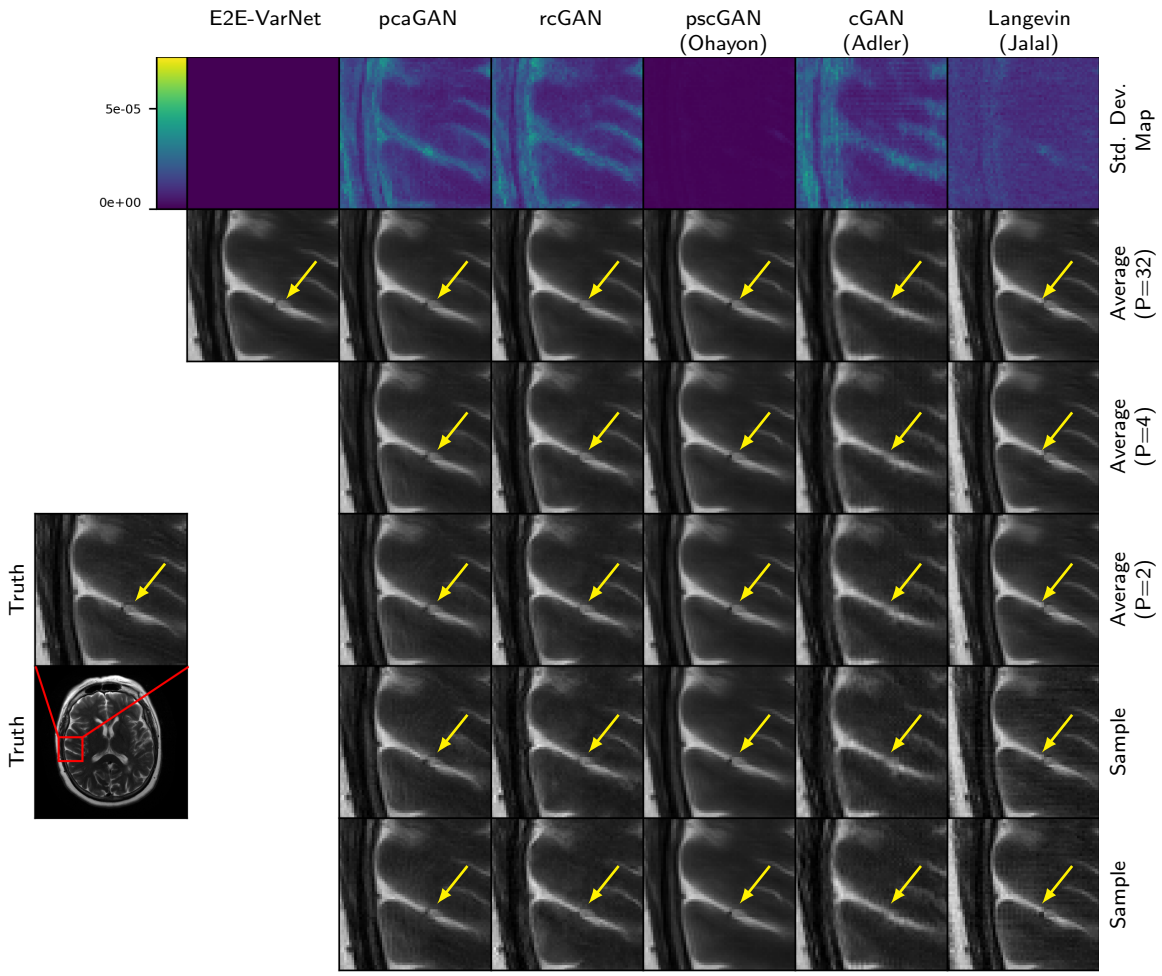


Figure E.15: Example $R = 8$ MRI reconstruction. Row one: pixel-wise SD with $P = 32$, Row two: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 32$, Row three: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 4$, Row four: $\widehat{\boldsymbol{x}}_{(P)}$ with $P = 2$, Rows five and six: posterior samples. The arrows indicate regions of meaningful variation across posterior samples.

E.2.4 Inpainting

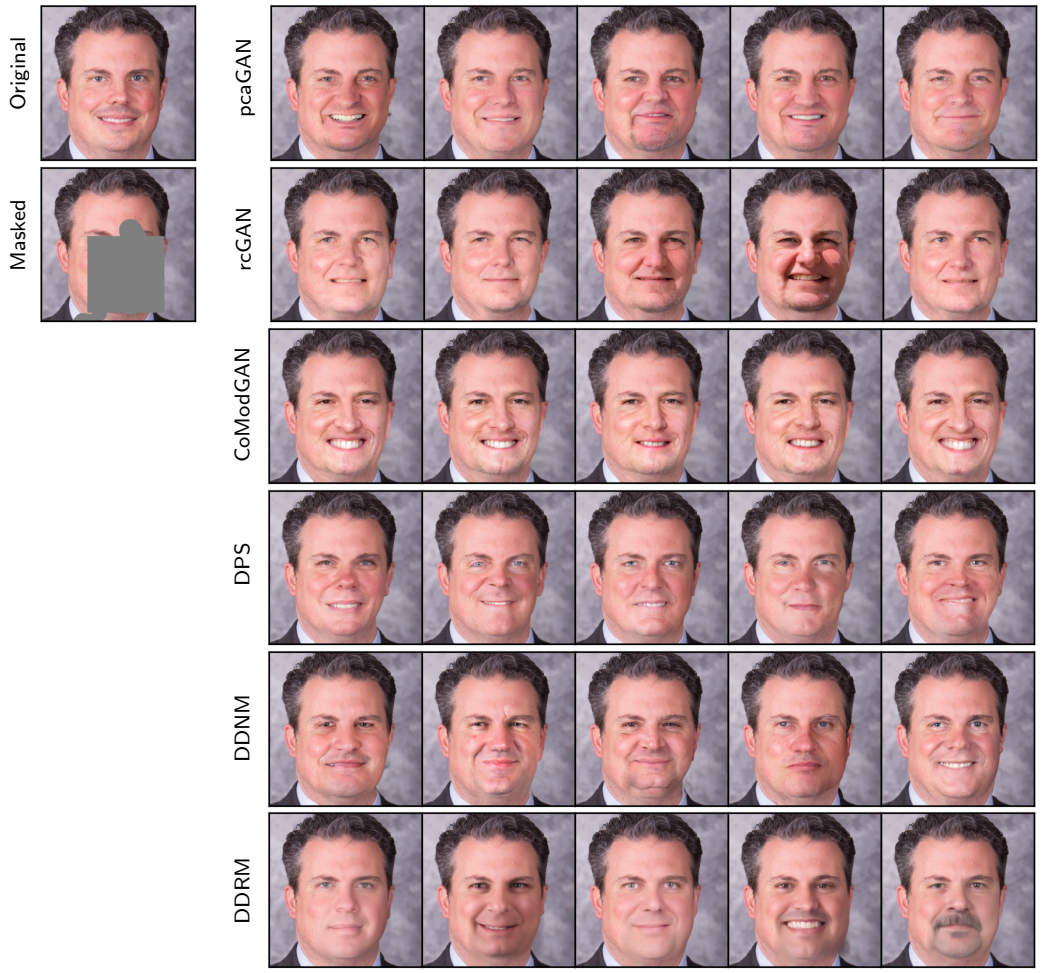


Figure E.16: Example of inpainting a randomly generated mask on a 256×256 FFHQ face image.

Figure E.17: Example of inpainting a randomly generated mask on a 256×256 FFHQ face image.

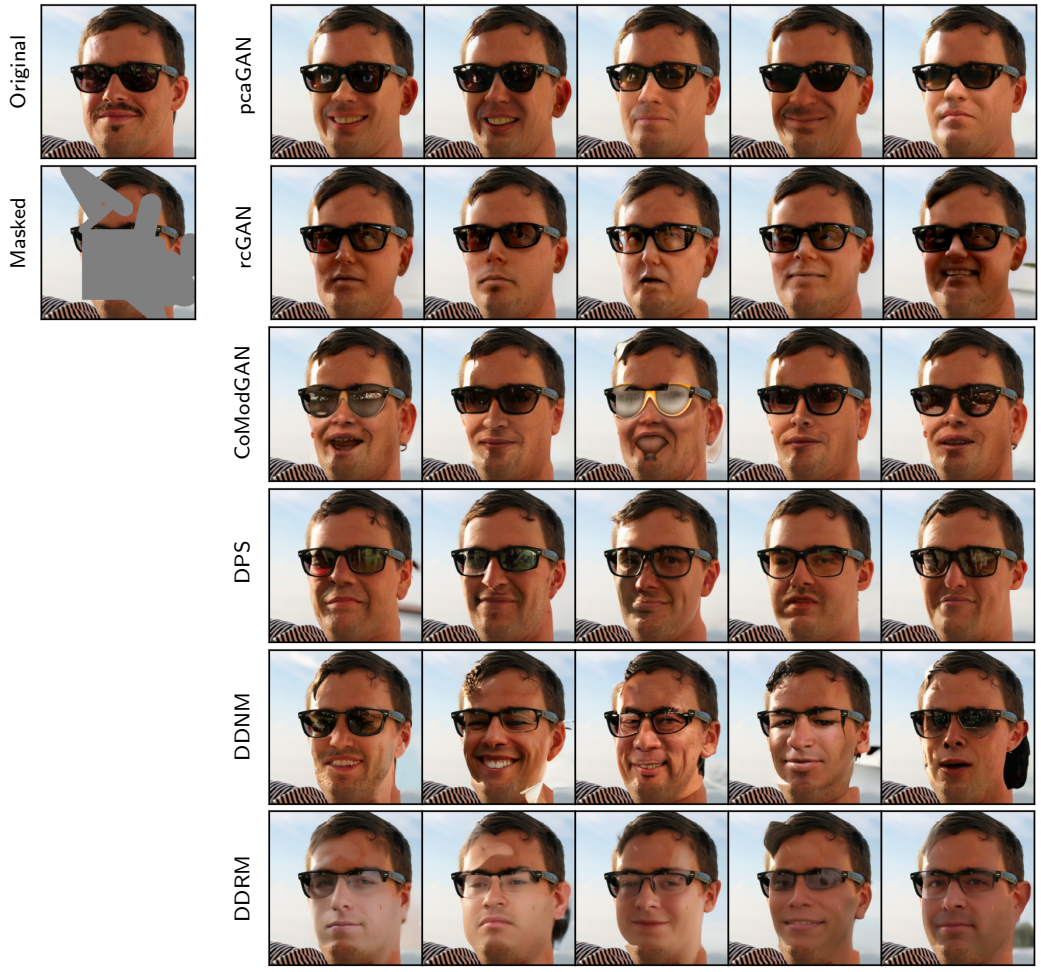


Figure E.18: Example of inpainting a randomly generated mask on a 256×256 FFHQ face image.

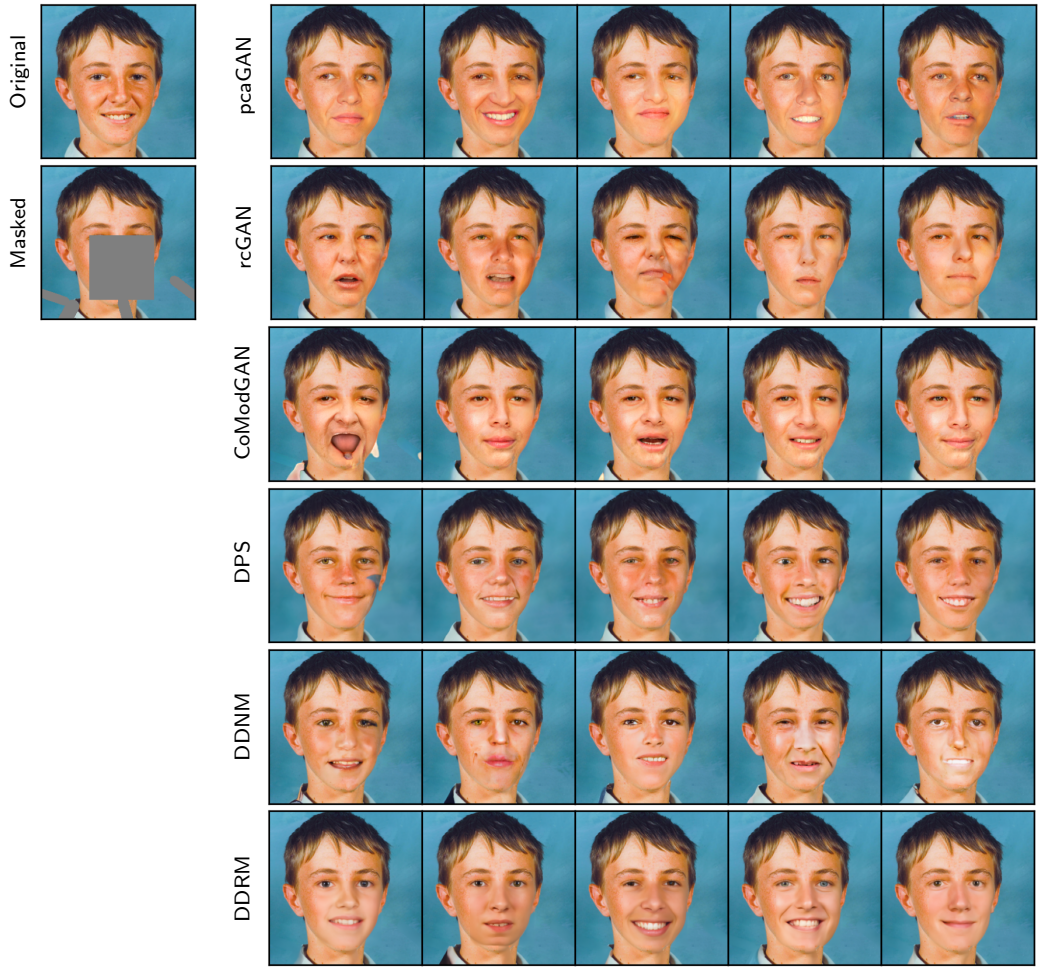


Figure E.19: Example of inpainting a randomly generated mask on a 256×256 FFHQ face image.