Thesis

Deep Equilibrium Models and Diffusion Models in Practice: Methods for improving efficiency

Ashwini Pokle

January 2025
CMU-ML-25-100

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee

Zico Kolter (Chair)
Andrej Risteski
Roger Baker Grosse
Ruslan Salakhutdinov

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

This research was funded by Bosch Center for AI.

Keywords: machine learning, generative models, diffusion models, parallel sampling, deep equilibrium models, efficient differentiation, efficient sampling, fast sampling, implicit gradients, distillation, one-step sampling, flow models, neural ODEs, inverse problems, linear inverse problems, knowledge distillation, steady-state partial differential equation, partial differential equation, fourier neural operator, out-of-distribution generalization, upward generalization, algorithmic generalization

For my parents,
Shrikant & Pratibha Pokle,
for their unwavering love and support,
&
my younger brother, Akhil Pokle,
my lifelong friend.

Abstract

Diffusion models have emerged as a dominant family of generative models for image, video, and audio generation. Deep Equilibrium (DEQ) models are architectures that compute their internal representations by solving for a fixed point in their forward pass. While seemingly distinct, this thesis explores methods that lie at their intersection, as well as broader applications of DEQs in solving partial differential equations (PDEs) and improving out-of-distribution (OOD) generalization on algorithmic tasks.

For diffusion models, we address the challenge of slow sampling. By reformulating the denoising diffusion implicit model (DDIM) as a DEQ, we enable parallel sampling and efficient model inversion through implicit gradients. Additionally, we demonstrate how DEQ-based architectures allow for parameter-efficient distillation of diffusion models, achieving single-pass image generation. We will also look into an application of flows, a family of generative models closely related to diffusion models, in solving linear inverse problems in fewer model evaluations.

In the context of PDEs, we investigate the architectural design space of neural operators for steady state PDEs. Neural operators take as input a PDE in some family, and output its solution. We demonstrate that weight-tied networks and DEQs are effective architectural choices and provide strong inductive bias for neural operators for steady-state PDEs.

Finally, we examine an interesting capability of DEQs to generalize on harder tasks at test-time by leveraging more compute. We find that this ability of DEQs to generalize on harder tasks strongly correlates with the *path independence* of the system—its tendency to converge to the same steady-state behavior regardless of initialization, given enough computation. This motivates use of path independence as a general modeling principle to facilitate scalable test-time usage.

This thesis presents practical methods to enhance the efficiency and utility of diffusion and DEQ models across generative modeling, PDEs, and algorithmic tasks.

Acknowledgement

This thesis marks the culmination of years of learning, challenges, and personal growth. Along this journey, I have been fortunate to receive the encouragement and support of exceptional individuals. Their patience, wisdom, and kindness have not only shaped this work but have also left a lasting impact on me as a person.

I am deeply grateful to my advisor, Zico Kolter, for his invaluable mentorship and for playing a crucial role in shaping both my research and my path as a researcher. Zico introduced me to Deep Equilibrium Models (DEQs) and diffusion models during the second year of my PhD—both of which are now at the core of this thesis. Throughout my PhD, Zico posed challenging yet fascinating questions that pushed me to think critically and refine my approach to research. I joined his group at a time when I was struggling to find an advisor within my department. The COVID-19 pandemic was at its peak, the campus was in lockdown, and our classes were fully remote. Despite these challenges—and despite my background in robotics and reinforcement learning, which initially had little overlap with his primary research interests—Zico welcomed me into his group. His patience and willingness to give me the space and time to navigate a new research area at my own pace has been invaluable to my growth as a researcher. I am truly grateful for his guidance, support, and belief in my potential.

I would also like to extend my sincere gratitude to the rest of my thesis committee—Andrej Risteski, Roger Grosse and Ruslan Salakhutdinov. I am deeply grateful to Andrej Risteski for his mentorship and support throughout my PhD. Over the years, Andrej patiently answered my many questions on generative models such as diffusion models and flows as well as broader topics in machine learning, helping me develop a deeper understanding of the field. I sincerely appreciate the opportunity to attend group meetings, which not only expanded my knowledge but also helped me refine my presentation skills. I would also like to thank Roger Grosse for his guidance during my work on path-independent models. I truly enjoyed collaborating with his students, Anil Cem, Kaiqu Liang, and Johannes Treutlin, and I am grateful for the insightful discussions and shared learning experiences. Finally, I would also like to thank Ruslan Salakhutdinov for agreeing to be on my thesis committee and for his valuable feedback that helped me improve my thesis.

This thesis would not have been possible without my incredible collaborators, whose valuable insights and hard work played a crucial role in many of my research papers. I am sincerely grateful to my coauthors (I apologize if I missed someone) Zhengyang Geng, Tanya Marwah, Anil Cem, Shaojie Bai, Jinjin Tian, Yuchen Li, Kaiqu Liang, Johannes Treutlein, Yuhuai Wu, William Luo, Justin Lin, Vikash Sehwag, Ashwinee Panda, Xinyu Tang, Saeed Mahloujifar, Mung Chiang, Prateek Mittal, Jingyun Yang, Hsiao-Yu Tung, Yunchu Zhang, Gaurav Pathak, and Christopher G Atkeson. I would also like to thank Jianfeng Lu and Zachary Lipton for their insightful feedback on my work on neural operators for steady-state PDEs. I will especially miss many insightful brainstorming sessions and discussions with Zhengyang Geng and Tanya Marwah. I would also like to thank my internship mentors at Meta—Ricky T.Q. Chen, Matthew Muckley, and my internship manager, Brian Karrer—for their valuable insights, technical expertise and feedback during my internship. I am especially grateful for their guidance and for introducing me to flow matching models, which made my work on using flows for solving inverse problems possible. I would also like to thank my mentor during my Bosch internship, Bahar Azari, for valuable discussions.

I would also like to take this moment to thank the many professors, teachers, and mentors who have paved the way for me. I am especially grateful to Silvio Savarese for believing in me and giving me the opportunity to step into the world of research at a time when I had no prior experience. I would also like to express my sincere appreciation to my incredible mentors at Stanford—Marynel Vasquez, Roberto Martín-Martín, and Patrick Goebel—for their guidance and support during my early research journey, when I was still figuring out how to do research. Additionally, I am grateful to my collaborators—Vincent Chow, Hans M. Ewald, Junwei Yang, Zhenkai Wang, Amir Sadeghian, Dorsa Sadigh, Xiaoxue Zang, Juan Carlos Niebles, and Alvaro Soto—whose guidance and support played a crucial role in making my initial research projects possible. They especially played a key role in setting up JackRabbot, the social navigation robot, that I used to work on during my Master's days. I would also like to thank Surekha Bhanot, my undergraduate professor, for her invaluable guidance when I was navigating the decision of whether to pursue graduate school.

I would also like to thank my fellow lab mates from the Locus Lab, many of whom have become good friends, for their camaraderie and support. I would like to thank Victor Akinwande, Brandon Amos, Christina Baek, Anna Bair, Filipe de Avila Belbute-Peres, Jeremy Cohen, Priya Donti, Zhili Feng, Mark Finzi, Zhengyang Geng, Sachin Goyal, Swaminathan Gurumurthy, Kelly He, Yiding Jiang, Eungyeup Kim, Kevin Li, Chun Kai Ling, Pratyush Maini, Gaurav Manek, Vaishnavh Nagarajan, Leslie Rice, and Mel Roderick, Dylan Sam, Yash Savani, Sam Sokota, Mingjie Sun, Avi Schwarzschild, Asher Trockman, Josh Williams, Runtian Zhai, and Andy Zou. Their presence made my PhD journey not only intellectually enriching but also enjoyable and memorable. I am incredibly grateful for the numerous long conversations on random topics I have had with many of them and for their willingness to answer my questions related to their research over Slack.

I also owe a lot to my colleagues and friends in the Machine Learning Department, and at CMU in general, who have been a constant source of support and inspiration. Their guidance and encouragement have helped me navigate the numerous challenges of both my PhD and the job search. I would like to thank Charvi Rastogi, Elan Rosenfeld, Helen Zhou, Aakash Lahoti, Oscar Li, Valerie Chen, Shanda Li, Himangi Mittal, and Niki Hasrati. I would also like to thank my office mates over the years Dhruv Malik, Yewen Fan, Chenghui Zhou, Gati Aher, Namrata Deka, and Xiangchen Song, for fun light-hearted conversations, as well as insightful discussions. I would also like to thank Diane Stidle for keeping MLD running like a well-oiled machine and for her assistance throughout my PhD journey.

A huge shoutout to the incredible women from my PhD batch and my close friends—Bingbin Liu, Arundhati Bannerjee, Yusha Liu, Tanya Marwah, and Stephanie Milani—for being my cheerleaders, unofficial therapists, and partners-in-procrastination throughout this PhD. Your unwavering support (and occasional reality checks) made this journey not just bearable, but genuinely fun. Bingbin and her husband, Tim Hsieh, have been my close friends since my master's days, and I continue to be inspired by her incredible time management skills, discipline, and vast technical knowledge. More recently, our favorite pastime has been bonding over BTS—something I never expected. I am also deeply grateful to Arundhati for being an amazing, patient, and kind roommate for four long years. From late-evening caffeinated conversations to occasional baking expeditions, sharing this journey with her has made it all the more memorable. Yusha made my PhD journey infinitely more fun with her incredible sense of humor. I will miss having thoughtful conversations with her over Indian meals, and, of course, her habit of ensuring that I always have an infinite supply of chili crisp. Tanya has been my sounding

board, an amazing collaborator, and my go-to advisor for all things tech-related. Her insight, support, and friendship have been invaluable. I will also miss Stephanie and her cat Sasha. Most importantly I will miss her mature and thoughtful response to any issue we encounter. I am beyond lucky to have shared this journey with such an incredible group of friends.

To my very dear friends (and I sincerely apologize if I've missed anyone!)—Vidya Madhavan, Shrihari Bhat, Akhila Yerukola, Shalini Pandey, Sukanya Patil, Arun Mallya, Vaishnavh Nagarajan, Ashwini Ramamurthy, Devyani Choudhary, Soham Phade, Astha Sharma, Stephania Ann Mathew, Stefanie Anna Baby, Snehal Bandal, Sumit Binnani–thank you for your unwavering support during crucial moments, for the laughter, the venting sessions, for making sure I was well-fed, and most importantly, for simply being there whenever I needed you. I truly couldn't have done this without you. Most of you have been a part of my life for over a decade (and for those who joined more recently, it certainly feels like a decade!). I wish I could put into words just how much your friendship means to me, but as the saying goes, the better you know someone, the fewer words you need.

I would also like to express my heartfelt gratitude to my aunts and uncles, Usha and Ganesh Rao, and Preeta and Prakash Shet, who have been like second parents to me. Their love, kindness, encouragement, and belief in my abilities since childhood have been a constant source of strength. I would also like to thank Gulab and Prabhakar Shirsat, my aunt and uncle, for their kindness and for ensuring that I have hearty, home-cooked meals before I take my flight back to the US. I would also like to thank my cousins, Prachi Satish, Pavan Shet, Pradnesh Shet, Veena Nayak, and Rajbhushan Nayak for checking in with me regularly throughout my PhD to ensure my mental and emotional well-being.

Pursuing a PhD as an international student comes with an added layer of uncertainty, especially due to the distance from loved ones. One of the hardest realities of this journey is the possibility of losing important people along the way. During the second year of my PhD, I lost two such cherished individuals in my life. I would like to express my sincere gratitude to my late uncle, Anil V. Nayak, who passed away during the COVID-19 pandemic lockdown. He was the one who first sparked my interest in Mathematics during my childhood, shaping the foundation of my academic journey. Some of my fondest memories of him are from our time together in Mumbai, where he would take me to his favorite restaurants and treat me to delicious food. I will miss him everytime I eat Kanchipuram idlis. I would also like to express my gratitude to my mother's late friend, Dayavati (Daksha) Chauriya, who watched me grow up and always showered me with kindness and warmth.

This thesis would not have been possible without the unwavering love and support of my parents, Pratibha and Shrikant Pokle, whose encouragement gave me strength even in moments of self-doubt. I am especially grateful for their patience in listening to me during difficult times, their steadfast belief in my abilities, and their unwavering support for my countless spontaneous decisions since childhood—despite societal expectations. I deeply appreciate the thoughtful care packages filled with snacks and sweets that brought warmth and comfort of home whenever I missed India and my mother's cooking. Their love and presence in my life have been a constant source of motivation, their values and sacrifices have shaped me as a person, and I am truly fortunate to have them by my side. I am also deeply thankful to my younger brother, Akhil Pokle, for being a steadfast friend and source of strength as I navigate this journey.

Contents

Chapter 1

Introduction

Over the past decade, deep learning has led to transformative breakthroughs in multiple domains, pushing the boundaries in various domains such as generative modeling, natural language understanding, and autonomous decision-making. State-of-the-art generative models, such as diffusion models [Song and Ermon, 2019, Song et al., 2020b, Ho et al., 2020] and generative adversarial networks (GANs) [Goodfellow et al., 2014], can synthesize high-fidelity images, videos, and audio with impressive realism. Large-scale language models (LLMs) [Brown et al., 2020b, Hoffmann et al., 2022, OpenAI, 2023, Team et al., 2023, Touvron et al., 2023, Anthropic, 2024, Yang et al., 2024] have significantly advanced natural language processing (NLP), excelling in tasks such as text generation, machine translation, and reasoning. These advancements have enabled the development of systems and tools such as conversational agents, code-generation tools [Guo et al., 2024, Hui et al., 2024], and autonomous decision-making agents [Gao et al., 2024, Yao et al., 2022, Shinn et al., 2023, Liu et al., 2023b]. Additionally, deep learning has accelerated research in scientific applications, such as climate modeling [Doury et al., 2023, Rasp et al., 2018, Scher, 2018], protein structure prediction [Jumper et al., 2021], drug discovery [Pandey et al., 2022, Gupta et al., 2021b], astrophysics [Zhao et al., 2023, Leung and Bovy, 2024], medical imaging [Suganyadevi et al., 2022, Aggarwal et al., 2021], material science [Choudhary et al., 2022, Mishin, 2021], and computational fluid dynamics [Vinuesa and Brunton, 2022, Kochkov et al., 2021]. Furthermore, deep reinforcement learning has achieved superhuman performance in various domains by leveraging neural networks to approximate optimal policies [Granter et al., 2017, Silver et al., 2017, Schrittwieser et al., 2020, Berner et al., 2019]. Similarly, deep learning has dramatically improved robot learning, enabling robots to perceive, reason, and act in real-world environments [Ahn et al., 2022].

Despite these successes, deep learning remains constrained by several fundamental limitations. One of the primary challenges in deep generative models such as diffusion models is efficiency. While diffusion models have set new benchmarks for image [Dhariwal and Nichol, 2021, Karras et al., 2022, 2024, Baldridge et al., 2024] and video generation quality [Brooks et al., 2024, Veo-Team et al., 2024], they suffer from slow sampling efficiency due to their iterative nature. This limitation becomes particularly severe in applications such as video generation, where generating a temporally consistent sequence requires producing a large number of high-resolution frames in real time. As a result, over the past couple of years, there have been widespread efforts to accelerate sampling through methods such as distillation [Salimans and Ho, 2022, Luhman and Luhman, 2021, Nguyen and Tran, 2023, Wang

et al., 2024b, Kohler et al., 2024] and consistency models [Song et al., 2023, Song and Dhariwal, 2023].

In the case of LLMs, despite their impressive performance on many NLP tasks, these models struggle with mathematical reasoning [Mirzadeh et al., 2024, Toshniwal et al., 2024], logical extrapolation [Williams and Huckle, 2024, Wan et al., 2024], and generalization from simpler to more complex problem domains [Ding et al., 2024]. While they exhibit strong performance on in-distribution tasks, they often fail to systematically generalize to out-of-distribution problems, highlighting a fundamental gap in their reasoning and abstraction capabilities. Additionally, LLMs are prone to hallucination [Huang et al., 2024, Li et al., 2024], where they generate factually incorrect or misleading information, posing challenges in high-stakes applications such as medical diagnosis, legal analysis, and scientific discovery.

Furthermore, the application of deep learning in scientific research faces significant data-related challenges. Many scientific domains, such as materials science, genomics, and climate modeling, rely on datasets that are inherently small, noisy, and highly imbalanced. Unlike internet-scale datasets used for training foundation models, scientific datasets often suffer from data scarcity, making it difficult for deep learning models to generalize effectively [Xu et al., 2023a, Dubois et al., 2022, Stephany and Earls, 2024]. This limitation highlights the need for more robust models to improve performance in low-data regimes.

This thesis aims to address a subset of the limitations stated above. We will broadly focus on two models: 1) deep equilibrium models (DEQ) [Bai et al., 2019] which are architectures that compute their internal representations by solving for a fixed point in their forward pass, and 2) diffusion models [Song and Ermon, 2019, Song et al., 2020b, Ho et al., 2020] which are a type of generative model that generate new samples through an iterative denoising process. In recent years, diffusion models have become a dominant approach in generative modeling. However, their high computational cost and slow sampling speed remain significant bottlenecks as described in the previous paragraphs. Simultaneously, DEQs have emerged as an alternative to traditional deep networks, offering adaptive test-time depth and memory efficiency. While diffusion models and DEQs may appear fundamentally different, this thesis explores their intersection and the broader applications of DEQs in generative modeling, solving partial differential equations (PDEs), and improving algorithmic generalization.

This thesis is motivated by three key challenges:

- **Slow sampling in diffusion models**: Traditional sampling methods require sequential evaluations, limiting their practical utility in real-time generation.
- **Architectural design for solving PDEs**: Neural operators for steady-state PDEs lack structured inductive biases, affecting their efficiency and generalization.
- **Test-time generalization in deep learning**: Regular non-weight tied neural networks struggle to generalize on harder problems at test time.

This thesis introduces approaches to address the above challenges and is divided into three parts as described below.

The first part focuses on methods for efficient sampling with diffusion models and flows, as well as their application in solving inverse problems. Chapter 3 introduces an approach for parallel sampling of diffusion models through the lens of DEQs. At the core of this approach is reformulating the widely

used Denoising Diffusion Implicit Model (DDIM) [Song et al., 2020a] as a joint, multivariate fixed point system. This enables use of additional compute to distribute the workload across multiple GPUs during sampling, resulting in speedups. Another advantage of this formulation is that it enables efficient model inversion through use of implicit gradients.

While parallel sampling is a training-free approach for faster sampling of diffusion models, we also introduce a training-based approach for one-step sampling of diffusion models in Chapter 4. Inspired by prior work in distillation [Salimans and Ho, 2022, Luhman and Luhman, 2021], we propose a parameter-efficient distillation technique for single-step image generation. The core idea is to do an offline distillation of diffusion models by training another model to directly predict images from Gaussian noise. These noise/image pairs are obtained from a pre-trained diffusion model [Karras et al., 2022]. We find that the architecture of the distilled model plays a crucial role in the quality of the distillation, and we leverage a DEQ-based model. This model, called Generative Equilibrium Transformer (GET), uses the standard ViT [Dosovitskiy et al., 2021] backbone, and has weight-tied transformer layers. This enables adaptive computation in the forward pass. An advantage of this is that we can improve image quality by using more (lightweight) iterations in the forward pass.

Next, we switch our focus to an application of efficient sampling with flows in solving linear inverse problems. Given noisy measurements generated by a known degradation model, solving an inverse problem involves recovering a clean signal from the given noisy measurements. Many interesting image processing tasks can be cast as an inverse problem such as deblurring, super-resolution, inpainting and JPEG restoration. We will focus on methods that use pretrained diffusion and flow models to solve these problems, and avoid training/fine-tuning a model on image pairs. Existing training-free approaches for solving inverse problems that are based on diffusion models often need hundreds to thousands of steps for a good quality solution. We hypothesize that one of the factors that contributes to this inefficiency is the curvature of the diffusion sampling paths. This motivates us to consider alternate ODE parameterizations that have less curvature.

CNF (hereafter denoted flow model) has the ability to model arbitrary probability paths, and includes diffusion probability paths as a special case. As we will see in Chapter 5, there are certain parametrizations of flow models [Lipman et al., 2022, Liu et al., 2022b, Chen et al., 2018] that have less curvature than the diffusion ODE. We propose a method that leverages pre-trained conditional optimal transport flow models to solve inverse problems in a fewer number of steps, thereby improving its efficiency in terms of number of steps as well as qualitative and quantiative performance in terms of quality of solutions.

In the second part of this thesis, we switch our focus to the problem of solving steady-state partial differential equations (PDEs). PDEs are ubiquitous in scientific domains as they model a wide range of processes in science and engineering. However solving PDEs is challenging because most PDEs do not accept a closed form solution and many existing approaches that use classical numerical methods to solve them can be slow and expensive. Recently, machine learning-based approaches, particularly neural operators, have emerged as a promising alternative. Neural operators learn learn solution to a family of PDEs and are invariant to grid discretization, unlike previous machine learning based approaches such as physics-inspired neural networks (PINN) [Raissi et al., 2017].

Despite their advantages, neural operators do not inherently encode the structural knowledge of the PDEs they are designed to solve. This can limit their generalization and efficiency. In Chapter 6 we

address this limitation by exploring the architectural design space of steady-state PDE solvers, with a particular focus on Deep Equilibrium Models (DEQs). We show that weight-tied architectures, such as DEQs, introduce an inductive bias that aligns naturally with the structure of steady-state PDEs. By leveraging this property, we propose a DEQ-based neural operator that outperforms conventional, non-weight-tied architectures in solving steady-state PDEs.

In the third part of this thesis, we turn our attention to the problem of algorithmic generalization, focusing on scenarios where models are expected to generalize to harder instances of problems they encountered during training. Many real-world tasks, such as mathematical reasoning, games like Sudoku and chess, and puzzles like maze solving, fall into this category. While humans naturally generalize from easy to hard problems by leveraging prior knowledge and increasing cognitive effort by thinking for longer, this remains a fundamental challenge for machine learning models, including large language models.

Recent approaches have attempted to improve generalization to harder problem instances, particularly in language models and algorithmic reasoning tasks. Methods such as chain-of-thought prompting [Wei et al., 2022] encourage models to generate intermediate reasoning steps, while least-to-most prompting enables decomposition of complex problems into simpler subproblems. Other techniques, such as memory-of-thought [Li and Qiu, 2023] and tree-of-thought prompting [Yao et al., 2023], explore persistent memory retention and structured reasoning paths to enhance problem-solving capabilities. Additionally, architectural innovations, including retrieval-augmented models [Tran et al., 2024, Wang et al., 2024a] and memory-augmented networks [Ko et al., 2024, Jin et al., 2024], have demonstrated promising improvements by integrating external memory and long-term reasoning structures. Despite these advances, a significant performance gap persists when it comes to reliably solving harder tasks at test time.

In this work, we explore a different architectural approach—leveraging weight-tying as a mechanism to enhance test-time generalization. One effective strategy for solving increasingly difficult problems is to allow models to allocate more computational resources at test time, refining their solutions through iterative updates. This aligns with how humans tend to think longer and explore more possibilities when faced with a difficult problem. In Chapter 7, we demonstrate that weight-tied architectures, particularly deep equilibrium models, naturally support this form of adaptive test-time computation, enabling models to iteratively refine their outputs and generalize better to harder problems at test-time. We show that a necessary condition to ensure this test-time generalization with weight-tied architectures is *path-independence* of the model—the tendency of model to converge to the same steady-state behaviour regardless of initialization, given enough computation. Through extensive experiments on many types of problems, we demonstrate that deep equilibrium models can successfully exploit additional test-time computation when they learn path independent solutions, making them a compelling architectural choice for improving algorithmic generalization.

While we have tried to address some of the limitations of the modern deep learning, there are many exciting directions to advance the field. Addressing these limitations requires continued advancements in algorithmic efficiency, data-efficient learning paradigms, and more interpretable and reliable AI models. Bridging the gap between current deep learning architectures and truly generalizable AI systems remains an open challenge with significant implications for both fundamental research and real-world applications.

Chapter 2

Background

This thesis explores a diverse set of topics, such as diffusion models, flow matching models, deep equilibrium models, and neural operators. In this chapter, we review relevant literature from prior works. While this is not an exhaustive literature review, we summarize important concepts and topics that are used in this thesis. Revisiting this chapter will be useful for reviewing relevant topics whenever the reader begins a new section.

2.1 Deep Equilibrium Models

Equilibrium models [Liao et al., 2018, Bai et al., 2019, Revay et al., 2020, Winston and Kolter, 2020] compute internal representations by solving for a fixed point in their forward pass. Specifically, consider a deep feedforward network with L layers :

$$z^{[i+1]} = f_\theta^{[i]}\left(z^{[i]}; x\right) \quad \text{for } i = 0, \ldots, L-1 \tag{2.1}$$

where $x \in \mathbb{R}^{n_x}$ is the input injection, $z^{[i]} \in \mathbb{R}^{n_z}$ is the hidden state of i^{th} layer with $z^{[0]} = \mathbf{0}$, and $f_\theta^{[i]} : \mathbb{R}^{n_x \times n_z} \mapsto \mathbb{R}^{n_z}$ is the feature transformation of i^{th} layer, parametrized by θ.

Suppose that the above model is weight-tied, *i.e.*, $f_\theta^{[i]} = f_\theta, \forall i$, and $\lim_{i\to\infty} f_\theta\left(z^{[i]}; x\right)$ exists and its value is $z^\star$. Further, assume that for this $z^\star$, we have $f_\theta\left(z^\star; x\right) = z^\star$. Then, equilibrium models can be interpreted as the infinite-depth limit of the above network such that

$$f_\theta^\infty\left(z^\star; x\right) = z^\star. \tag{2.2}$$

Under certain conditions[1], and for certain classes of f_θ[2], the output $z^\star$ of the above weight-tied network is a fixed point.

A simple way to solve for this fixed point is to use fixed point iterations, *i.e.*, repeatedly apply the update $z^{[t+1]} = f_\theta(z^{[t]}; x)$ some fixed number of times, and backpropagate through the network

[1] The fixed point can be reached if the dynamical system is globally contractive. This is usually not true in practice for most choices of f_θ, and divergence is possible.

[2] Bai et al. [2019] state that f_θ needs to be stable and constrained. In general, by Banach's fixed point theorem, global convergence is guaranteed if f_θ is contractive over its input domain.

to compute gradients. However, this can be computationally expensive. Deep equilibrium (DEQ) models [Bai et al., 2019] explicitly solve for $z^\star$ through iterative root finding methods like Broyden's method [Broyden, 1965], Newton's method, Anderson acceleration [Anderson, 1965]. DEQs use the implicit function theorem to differentiate directly through the fixed point $z^\star$ at equilibrium, thus requiring constant memory to backpropagate through an infinite-depth network. Bai et al. [2019] provide the expression for the implicit gradient used for backpropagation in DEQs as

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial z^\star}\frac{\partial z^\star}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial z^\star}\left(I - \frac{\partial f_\theta(z^\star; x)}{\partial z^\star}\right)^{-1}\frac{\partial f_\theta(z^\star; x)}{\partial \theta}, \tag{2.3}$$

where $\mathcal{L}$ is the loss objective. Thus, the backward gradient of DEQs uses constant memory, even for "infinite" layers of f_θ. This is unlike regular weight-tied networks, where the memory required for backpropagation grows linearly with the depth of the network. There are alternative formulations of equilibrium models [Winston and Kolter, 2020] that guarantee the existence of a unique equilibrium point. However, designing f_θ for these formulations can be challenging, and in this thesis we will use the formulation by Bai et al. [2019].

Challenges of training DEQs. Computing the inverse of the Jacobian is intractable for high-dimensional feature maps. Bai et al. [2019] suggest computing the solution to the following linear fixed point system

$$v^\top = v^\top \frac{\partial f_\theta(z^\star; x)}{\partial z^\star} + \frac{\partial \mathcal{L}}{\partial z^\star}, \tag{2.4}$$

which involves a vector-Jacobian product, and therefore can be computed efficiently by leveraging standard autograd packages. Thus, backward pass of DEQ can also be solved by using root finding methods. However, prior works [Bai et al., 2021] have noted training instability issues that arise due to ill-conditioned Jacobians while training DEQs. Some ways to tackle this training instability issue are using Jacobian regularization [Bai et al., 2021] and fixed-point correction [Bai et al., 2022] during training, and architectural modifications such as use of spectral normalization [Miyato et al., 2018], weight decay, and recurrent dropout [Gal and Ghahramani, 2016] in the layers of DEQ.

Approximate implicit gradients. We can also use inexact implicit gradients that avoid computing the inverse of Jacobian in the backward pass of DEQs, and therefore have more stable and faster training. Fung et al. [2022] propose that the backward pass of the DEQ can be approximated as

$$\frac{\partial \mathcal{L}}{\partial \theta} \approx \frac{\partial \mathcal{L}}{\partial z^\star}\frac{\partial f_\theta(z^\star; x)}{\partial \theta}$$

which replaces the inverse Jacobian term in Eq. (2.3) with an identity. Geng et al. [2021b] propose approximating inverse Jacobian term [Geng et al., 2021b] with a few-step gradient, called phantom gradient, based on the damped unrolling of DEQ at its fixed point. Specifically, it considers the following fixed-point iteration at equilibrium

$$z^{[i+1]} = \lambda f_\theta(z^{[i]}; x) + (1-\lambda)z^{[i]}, \tag{2.5}$$

where λ is an appropriate damping factor. Note that in this case, the gradients for the backward pass can be computed by using standard autograd packages.

2.2 Generative Models

2.2.1 Diffusion Models

Consider a data distribution $p_{\text{data}}(\mathbf{x}_0)$ from which we draw *i.i.d.* samples, then the diffusion process $\{\mathbf{x}_t\}_{t=0}^{T}$ for $t \in [0, T]$ is given by an Itô SDE [Song et al., 2020b]:

$$\mathrm{d}\mathbf{x}_t = \mathbf{f}(\mathbf{x}_t, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}, \tag{2.6}$$

where $\mathbf{w}$ is the standard Wiener process, $\mathbf{f}(\cdot, t) : \mathbb{R}^d \to \mathbb{R}^d$ is the drifting coefficient, $g(\cdot) : \mathbb{R} \to \mathbb{R}$ is the diffusion coefficient, and $\mathbf{x}_0 \sim p_{\text{data}}$ and $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$. Samples can be generated by starting from $\mathbf{x}(T)$ and then solving the reverse-time SDE [Anderson, 1982] given by

$$\mathrm{d}\mathbf{x}_t = \left[\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)\right] \mathrm{d}t + g(t)\mathrm{d}\overline{\mathbf{w}}, \tag{2.7}$$

where $\overline{\mathbf{w}}$ is the standard Wiener process for time $t = T$ to $t = 0$. All diffusion processes have a corresponding deterministic process known as the probability flow ODE (PF-ODE) [Song et al., 2020b] whose trajectories share the same marginal probability densities as the SDE. This ODE can be written as

$$\mathrm{d}\mathbf{x}_t = \left[\mathbf{f}(\mathbf{x}_t, t) - \frac{1}{2} g(t)^2 \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)\right] \mathrm{d}t. \tag{2.8}$$

The quantity $\nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t)$ is also known as the score function. Diffusion models, also known as score matching models, are trained to minimize the following score matching loss

$$\mathcal{L}_{\text{SM}} := \mathbb{E}_{t, p_0(\mathbf{x}_0), p_t(\mathbf{x}_t|\mathbf{x}_0)} \left[\lambda_t ||s_\theta(\mathbf{x}_t) - \nabla_{\mathbf{x}_t} \log p_t(\mathbf{x}_t|\mathbf{x}_0)||_2^2\right], \tag{2.9}$$

where λ_t is an appropriate weighting function. Different choices of the drifting coefficient $\mathbf{f}(\cdot, t)$ and the diffusion coefficient $g(\cdot)$ result in the so-called Variance Preserving (VP) and Variance Exploding SDEs. We note that the drifting coefficient $\mathbf{f}(\mathbf{x}_t, t)$ is of the form $\mathbf{f}(\mathbf{x}_t, t) = \mathbf{f}(t)\mathbf{x}_t$ in these formulations. Inspired by this, and assuming a particular choice of scaling factor as well as diffusion perturbation kernel $p_t(\mathbf{x}_t|\mathbf{x}_0)$, Karras et al. [2022] consider the following equivalent parametrization of PF-ODE

$$\mathrm{d}\mathbf{x}_t = -\dot{\sigma}(t)\sigma(t)\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t, \sigma(t))\mathrm{d}t, \tag{2.10}$$

where $\sigma(t)$ is the noise schedule of diffusion process, and $\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t, \sigma(t))$ represents the score function. Karras et al. [2022] also show that the optimal choice of $\sigma(t)$ in Eq. (2.10) is $\sigma(t) = t$. Thus, PF-ODE can be simplified to $\mathrm{d}\mathbf{x}/\mathrm{d}t = -t\nabla_{\mathbf{x}} \log p(\mathbf{x}_t, \sigma(t)) = (\mathbf{x}_t - D_\theta(\mathbf{x}_t; t))/t$, where $D_\theta(\cdot, t)$ is a denoiser function parametrized with a neural network that minimizes the expected L_2 denoising error for samples drawn from p_{data}. Samples can be efficiently generated from this ODE through numerical methods like Euler's method, Runge-Kutta method, and Heun's second-order solver [Ascher and Petzold, 1998].

2.2.1.1 Formulations of Diffusion Models

We will now describe two formulations of diffusion models that are widely used in the literature.

Variance Preserving (VP) Formulation. Song et al. [2020b] define VP-SDE as

$$\mathrm{d}\mathbf{x}_t = -\frac{1}{2}\beta(t)\mathbf{x}_t\mathrm{d}t + \sqrt{\beta(t)}\mathrm{d}\omega_t. \tag{2.11}$$

which is obtained by setting $\mathbf{f}(\mathbf{x}, t) = -\frac{1}{2}\beta(t)\mathbf{x}_t$, $g(t) = \sqrt{\beta(t)}$ in Eq. (2.6). Here, $\beta(t) = \beta_{\text{min}} + t(\beta_{\text{max}} - \beta_{\text{min}})$. In practice, $\beta_{\text{max}} = 20$ and $\beta_{\text{min}} = 0.1$, and $t \sim [\epsilon, 1]$ where ϵ is set to a small values such as 10^{-5}.

Variance Exploding (VE) Formulation. Song et al. [2020b] define VE-SDE as

$$\mathrm{d}\mathbf{x}_t = \sigma_{\text{min}}\left(\frac{\sigma_{\text{max}}}{\sigma_{\text{min}}}\right)^t\sqrt{2\log\left(\frac{\sigma_{\text{max}}}{\sigma_{\text{min}}}\right)}\mathrm{d}\omega_t. \tag{2.12}$$

which is obtained by setting $\mathbf{f}(\mathbf{x}, t) = 0$, $g(t) = \sigma_{\text{min}}\left(\frac{\sigma_{\text{max}}}{\sigma_{\text{min}}}\right)^t\sqrt{2\log\left(\frac{\sigma_{\text{max}}}{\sigma_{\text{min}}}\right)}$ in Eq. (2.6). In practice, $\sigma_{\text{min}} = 0.01$ and $\sigma_{\text{max}} = 50$ for CIFAR-10.

2.2.1.2 Discrete-time Diffusion Models

So far in this section, we have considered the continuous-time formulation of the diffusion models and derived them from the score matching perspective. We will now consider an equivalent formulation of diffusion models as proposed by Ho et al. [2020].

Given samples from a target distribution $\mathbf{x}_0 \sim q(\mathbf{x}_0)$, the diffusion process [Ho et al., 2020] is a Markov chain that adds Gaussian noises to the data to generate latent states $\mathbf{x}_1, ..., \mathbf{x}_T$ in the same sample space as $\mathbf{x}_0$

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}), \quad \text{where} \quad q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}\left(\sqrt{\frac{\alpha_t}{\alpha_{t-1}}}\mathbf{x}_{t-1}, \left(1 - \frac{\alpha_t}{\alpha_{t-1}}\right)\mathbf{I}\right), \tag{2.13}$$

where $\alpha_{1:T} \in (0, 1]^T$ is a fixed decreasing sequence of hyperparameters. This diffusion process is also known as a forward process and it has the following property:

$$q(\mathbf{x}_t|\mathbf{x}_0) := \int q(\mathbf{x}_{1:t}|\mathbf{x}_0)\mathrm{d}\mathbf{x}_{1:t-1} = \mathcal{N}(\mathbf{x}_t; \sqrt{\alpha_t}\mathbf{x}_0, (1 - \alpha_t)\mathbf{I}) \tag{2.14}$$

This parametrization of diffusion model is also known as Denoising Diffusion Probabilitic Model (DDPM) [Ho et al., 2020]. Given this diffusion process, the generative process is given by the following latent variable model

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}_{0:T})\mathrm{d}\mathbf{x}_{1:T}, \quad \text{where} \quad p_\theta(\mathbf{x}_{0:T}) := p_\theta(\mathbf{x}_T)\prod p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t). \tag{2.15}$$

The parameters θ are learned by using a surrogate variational lower bound [Ho et al., 2020] given by

$$\mathcal{L} = \mathbb{E}_q[-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1) + \sum_t D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) + D_{KL}(q(\mathbf{x}_T|\mathbf{x}_0)||p(\mathbf{x}_T)]. \tag{2.16}$$

The above objective can be further simplified to

$$\mathcal{L} := \sum_{t=1}^{T}\mathbb{E}_{q(\mathbf{x}_0),\epsilon_t\sim\mathcal{N}(0,\mathbf{I})}\left[||\epsilon_\theta^{(t)}(\sqrt{\alpha_t}\mathbf{x}_0 + \sqrt{1-\alpha_t}\epsilon_t) - \epsilon_t||_2^2\right], \tag{2.17}$$

where $\epsilon_\theta^{(t)}(\mathbf{x}_t)$ is an estimator trained to predict the noise given a noisy state $\mathbf{x}_t$. After training, samples can be generated by a reverse Markov chain, *i.e.*, first sampling $\mathbf{x}_T \sim p(\mathbf{x}_T) = \mathcal{N}(0, \mathbf{I})$, and then repeatedly sampling $\mathbf{x}_{t-1}$ till we reach $\mathbf{x}_0$. We note that the diffusion denoising objective Eq. (2.17) and the score matching objective Eq. (2.9) are equivalent (up to a scaling factor) [Karras et al., 2022, Hyvärinen and Dayan, 2005, Vincent, 2011]. Finally, we note that DDPM is discretization of VP-SDE [Song et al., 2020b].

2.2.2 Continuous Normalizing Flows

A Continuous Normalizing Flow (CNF) [Chen et al., 2018] is a time-dependent diffeomorphic map $\phi_t : [0, 1] \times \mathbb{R}^d \to \mathbb{R}^d$ that is defined by the ODE:

$$\frac{\mathrm{d}}{\mathrm{d}t}\phi_t(\mathbf{x}) = \boldsymbol{v}_t(\phi_t(\mathbf{x})); \quad \phi_0(\mathbf{x}) = \mathbf{x} \tag{2.18}$$

where $\mathbf{x} \in \mathbb{R}^d$ and $\boldsymbol{v}_t : [0, 1] \times \mathbb{R}^d \to \mathbb{R}^d$ is a time-dependent vector field that is usually parametrized with a neural network. The generative process of a CNF involves sampling from a simple prior distribution $\mathbf{x}_0 \sim p_0(\mathbf{x}_0)$ (e.g. standard Gaussian distribution) and then solving the initial value problem defined by the ODE in Eq. (2.18) to obtain a sample from the target distribution $\mathbf{x}_1 \sim p_1(\mathbf{x}_1)$. Thus, a CNF reshapes a simple prior distribution p_0 to a more complex distribution p_t, via a push-forward equation based on the instantaneous change of variables formula.

$$p_t = [\phi]_* p_0 \tag{2.19}$$

$$[\phi]_* p_0(\mathbf{x}) = p_0(\phi_t^{-1}(\mathbf{x}))\det\left[\frac{\partial \phi_t^{-1}}{\partial \mathbf{x}}(\mathbf{x})\right] \tag{2.20}$$

CNFs are usually trained by optimizing the maximum likelihood objective. As shown in Chen et al. [2018], the exact likelihood computation can be done via relatively cheap operations despite the Jacobian term. Specifically, from the continuity equation [Villani et al., 2009, Chen et al., 2018], we have

$$\frac{\mathrm{d}}{\mathrm{d}t}\log p_t(\phi_t(\mathbf{x})) + \mathrm{div}(\boldsymbol{v}_t(\phi_t(\mathbf{x}))) = 0. \tag{2.21}$$

Integrating the above over $t \in [0, 1]$ gives us a way to calculate log probability along with the flow trajectory,

$$\log p_1(\phi_1(\mathbf{x})) - \log p_0(\phi_0(\mathbf{x})) = -\int_0^1 \mathrm{div}(\boldsymbol{v}_t(\phi_t(\mathbf{x})))\mathrm{d}t. \tag{2.22}$$

One drawback of the above approach is that it requires restricting the architecture of the neural network to constrain the Jacobian term. FFJORD [Grathwohl et al., 2018] improves upon this by proposing a method that uses Hutchinson's trace estimator to compute log density, and allows CNFs with free-form Jacobians, thereby removing any restrictions on the architecture. Specfically, we can write Eq. (2.22) as

$$\log p_1(\phi_1(\mathbf{x})) - \log p_0(\phi_0(\mathbf{x})) = -\mathbb{E}_z \int_0^1 z^\top D\boldsymbol{v}_t(\phi_t(\mathbf{x}))z\mathrm{d}t. \tag{2.23}$$

where z is sampled from a distribution such that $\mathbb{E}[zz^\top] = \mathbf{I}$. The usual choices are the standard Gaussian distribution and Rademacher distribution. This approach has difficulties for high-dimensional images where the trace estimator is noisy. Flow Matching provides an alternative, scalable approach to training CNFs with arbitrary architectures.

2.2.3 Flow Matching Models

Suppose we have samples from an unknown data distribution $\mathbf{x}_1 \sim q(\mathbf{x}_1)$. Let p_t denote a probability path from the prior distribution p_0 to the data distribution p_1 that is approximately equal to q. Flow Matching loss is defined as

$$\mathcal{L}_{FM} = \mathbb{E}_{t,p_t(\mathbf{x})} \|\boldsymbol{v}_t(\mathbf{x};\theta) - \boldsymbol{u}_t(\mathbf{x})\|^2 \tag{2.24}$$

where $\boldsymbol{u}_t(\mathbf{x})$ is a vector field that generates the probability path $p_t(\mathbf{x})$, and θ denotes trainable parameters of the CNF. In practice, we usually do not have any prior knowledge on p_t and u_t, and thus this objective is intractable. Inspired by diffusion models, Lipman et al. [2022] propose Conditional Flow Matching, where both the probability paths and the vector fields are conditioned on the sample $\mathbf{x}_1 \sim q(\mathbf{x}_1)$. The exact objective for Conditional Flow matching is given by

$$\mathcal{L}_{CFM} = \mathbb{E}_{t,q(\mathbf{x}_1),p_t(\mathbf{x}|\mathbf{x}_1)} \|\boldsymbol{v}_t(\mathbf{x};\theta) - \boldsymbol{u}_t(\mathbf{x}|\mathbf{x}_1)\|^2 \tag{2.25}$$

where, $p_t(\mathbf{x}|\mathbf{x}_1)$ denotes a conditional probability path, and $\boldsymbol{u}_t(\mathbf{x}|\mathbf{x}_1)$ denotes the corresponding conditional vector field that generates the conditional probability path. Interestingly, both the loss objectives in Eq. (2.25) and Eq. (2.24) have identical gradients w.r.t. θ. More importantly, past research has proven that $\boldsymbol{u}_t(\mathbf{x}) = \mathbb{E}[\boldsymbol{u}_t(\mathbf{x}|\mathbf{x}_1)|\mathbf{x}_t = \mathbf{x}]$. The optimal solution to the conditional Flow Matching recovers $\boldsymbol{u}_t(\mathbf{x})$ and therefore $\boldsymbol{v}_t(\mathbf{x};\theta)$ generates the desired probability path $p_t(\mathbf{x})$.Thus, we can train a CNF without access to the marginal vector field $\boldsymbol{u}_t(\mathbf{x})$ or probability path $p_t(\mathbf{x})$. Compared to the prior approaches to train flow models, Flow Matching allows simulation-free training with unbiased gradients, and scales easily to high dimensions.

2.3 Neural Operators and Partial Differential Equations

2.3.1 Partial Differential Equations

A partial differential equation (PDE) is a differential equation that contains partial derivatives of an unknown function $u(x_1, \ldots, x_n, t)$ of multiple independent variables (spatial variables $x_1, \ldots, x_n$ and possibly time t), along with its partial derivatives. A general PDE can be written as

$$F\left(x_1, \ldots, x_n, u, \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_n}, \frac{\partial u}{\partial t}, \frac{\partial^2 u}{\partial x_1^2}, \frac{\partial^2 u}{\partial x_1 \partial x_2}, \ldots, t\right) = 0, \tag{2.26}$$

where $x_1, \ldots, x_n$ are independent spatial variables and t is the (optional) time variable. The above is often known as the governing equation of PDE. To obtain a well-posed problem, a PDE must be supplemented with initial conditions and/or boundary conditions. For a PDE involving time t, an initial condition specifies the value of u at time $t = 0$:

$$u(x_1, \ldots, x_n, t) = f(x_1, \ldots, x_n, t) \quad (x_1, \ldots, x_n) \in \Omega. \tag{2.27}$$

Boundary conditions specify constraints on u at the domain boundary $\partial\Omega$. Some examples of boundary condition are:

1. Dirichlet Boundary Condition:

$$u(x_1, \ldots, x_n, t) = g(x_1, \ldots, x_n, t), \quad (x_1, \ldots, x_n) \in \partial\Omega. \tag{2.28}$$

where g is a known function defined on $\partial\Omega$.

2. Neumann Boundary Condition:

$$\frac{\partial u(x_1, \ldots, x_n, t)}{\partial n} = h(x_1, \ldots, x_n, t), \quad (x_1, \ldots, x_n) \in \partial\Omega. \tag{2.29}$$

where n is normal to the boundary $\partial\Omega$ and h is a given scalar function.

3. Periodic Boundary Condition:

$$u(x + L, t) = u(x, t) \tag{2.30}$$

where L is the periodicity.

Partial differential equations (PDEs) are encountered in various physical, biological, and engineering applications. Below are some PDEs categorized based on their nature.

1. Heat equation is used to model heat conduction, diffusion of substances and Brownian motion. A simple 1D heat equation with Dirichlet boundary conditions can be written as

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial t^2}; \quad u(x, 0) = f(x); \quad u(0, t) = 0. \tag{2.31}$$

2. Wave equation is used to model waves such as electromagnetic waves and sound. 1D wave equation can with fixed ends can be written as

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}; \quad u(x, 0) = f(x); \frac{\partial u(x, 0)}{\partial t} = g(x) \quad u(0, t) = 0; u(0, L) = 0. \tag{2.32}$$

3. Laplace equation is used to model steady-state heat conduction, electrostatics, and fluid flow. 2D Laplace equation is given by

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{2.33}$$

We can define appropriate boundary conditions such as Dirichlet and Neumann over the domain.

2.3.2 Solving PDEs with Neural Operators

Classical techniques to solve PDEs can broadly be categorized into analytical methods that provide exact closed-form solutions and numerical methods that provide approximate solutions. The choice of method depends on the type of PDE, boundary/initial conditions, and complexity. In practice, most PDEs do not admit a closed form solution, and are solved using a variety of classical numerical methods such as finite element [LeVeque, 2007], finite volume [Moukalled et al., 2016], and spectral methods [Kopriva, 2009, Boyd, 2001]. These classical methods rely on discretizing the domain into a very fine mesh. Thus, these methods are often very computationally expensive, both as the ambient dimension grows, and as the threshold of desired accuracy increases. In addition, these methods also solve one specific instance of a PDE, and do not generalize across different instances. This motivates use of data-driven approaches to solve PDEs. These methods use machine learning to learn solutions to PDE directly from data. However, obtaining high quality data remains a challenge, as this data is usually obtained by leveraging numerical solvers which can take days to months for high quality data. Another challenge with these approaches is generalization to different instances of PDEs and levels of

grid discretization. As an example, Physics-informed neural networks (PINNs) [Raissi et al., 2017] struggle to learn a solution as dimensions increase, and also do not generalize well across different problem setups. This motivates operator learning for solving PDEs.

Operators are mapping between two infinite dimensional function spaces. In this sense, operators generalize the concept of functions. Some examples of frequently encountered operators are integral and differential operators. We will consider PDEs of form

$$L(a(x), u(x)) = f(x), \qquad \forall x \in \Omega, \tag{2.34}$$

where $u : \Omega \to \mathbb{R}^{d_u}$, $a : \Omega \to \mathbb{R}^{d_a}$ and $f : \Omega \to \mathbb{R}^{d_f}$ are functions defined over the domain Ω, and L is a (possibly non-linear) operator. A natural operator that arises from this PDE is $G^\dagger := L^{-1} f$ to map the parameter to solution $a \mapsto u$. The goal of operator learning for PDEs is to approximate this operator $G^\dagger$ with a neural network. In practice, these are usually implemented as a multi-layer neural network, where each layer is an operator defined over a function space.

There are many instantiations of neural operators depending on the application domain such as Graph Neural Operator (GNO) [Li et al., 2020b], Fourier Neural Operator (FNO) [Li et al., 2020a] and Low Rank Neural Operator (LNO) [Gupta et al., 2021a]. In this thesis, we will explore a specific neural operator called Fourier Neural Operator (FNO). Neural operators are universal approximators [Kovachki et al., 2023] and discretization-invariant *i.e.,* they can be evaluated at any input and output point that belong to the domain Ω.

2.3.2.1 Neural Operator

Li et al. [2020b,a] define a neural operator $G_\theta : \mathbb{R}^{d_u} \to \mathbb{R}^{d_u}$ as an iterative architecture of form

$$G_\theta := \mathcal{Q} \circ \mathcal{L}_L \circ \mathcal{L}_{L-1} \circ \cdots \circ \mathcal{L}_1 \circ \mathcal{P} \tag{2.35}$$

where $\mathcal{P} : L^2(\Omega; \mathbb{R}^{d_u}) \to L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_v})$ and $\mathcal{Q} : L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_v}) \to L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_u})$ are projection operators, and $\mathcal{L}_l : L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_v}) \to L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_v})$ for $l \in [L]$ is the l^{th} operator layer. Li et al. [2020b,a] define this operator layer as

$$\mathcal{L}_l\left(v_l\right) = \sigma\left(W_l v_l + b_l + \mathcal{K}_l(a; \phi) v_l\right)\right). \tag{2.36}$$

Here σ is a non-linear activation function, W_l, b_l are the l^{th} layer weight matrix and bias terms, and $\mathcal{K}_l$ is the l^{th} integral kernel operator. This integral operator is parametrized with a neural network (assume the parameters to be ϕ). Intuitively, integral kernel operator is the function space analog of the weight matrix in a standard feed-forward network since they are infinite-dimensional mapping from one function space to another. Different definitions of kernel integral operators lead to different instantiations of neural operators [Kovachki et al., 2023].

2.3.2.2 Fourier Neural Operator

In this thesis, we follow Li et al. [2020a] and assume that the integral kernel operator is defined as

$$(\mathcal{K}_l(a; \phi) v_l)(x) := \int_D \kappa(x, y, a(x), a(y); \phi) v_t(y) dy, \quad \forall x \in D \tag{2.37}$$

where κ_ϕ is a kernel function. Li et al. [2020a] further remove the dependence on a and impose the condition $\kappa_\phi(x, y) = \kappa_\phi(x - y)$ which indicates that this is a convolution operator. Therefore, applying convolution theorem gives

$$(\mathcal{K}_l(a; \phi)v_l)(x) := \mathcal{F}^{-1}(\mathcal{F}(\kappa_\phi) \cdot \mathcal{F}(v_t))(x); \quad \forall x \in D, \tag{2.38}$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the Fourier transform and the inverse Fourier transform. More specficially, FNO Li et al. [2020a] is defined as follows,

$$\mathcal{K}_l(v_l)(x) = \mathcal{F}^{-1}\left(R_{\phi,l} \cdot (\mathcal{F}v_l)\right)(x) \qquad \forall x \in \Omega, \tag{2.39}$$

with $R_{\phi,l}$ representing the learnable weight-matrix in the Fourier domain. Therefore, the set of trainable parameters is a collection of all the weight matrices and biases, *i.e.*, $\theta := \{W_l, b_l, R_{\phi,l}, \cdots, W_1, b_1, R_{\phi,1}\}$. This can be efficiently computed using FFT. Li et al. [2020a] show that in practice, few Fourier modes can be dropped, which results in quasi-linear complexity.

Part I

Efficient Generative Models: Insights from Deep Equilibrium, Diffusion and Flow Models

Chapter 3

Deep Equilibrium Approaches to Diffusion Models

Diffusion models are extremely effective in generating high-quality images, with generated samples often surpassing the quality of those produced by other generative models such as Generative Adversarial Networks (GANs)[Goodfellow et al., 2014] under several metrics. One distinguishing feature of these models, however, is that they typically require long sampling chains to produce high-fidelity images. This presents a challenge not only from the lenses of sampling time, but also from the inherent difficulty in backpropagating through these chains in order to accomplish tasks such as model inversion, *i.e.,* approximately finding latent states that generate known images. In this chapter, we look at diffusion models through a different perspective, that of a (deep) equilibrium (DEQ) fixed point model [Bai et al., 2019]. Specifically, we extend the recent denoising diffusion implicit model (DDIM) [Song et al., 2020a], and model the entire sampling chain as a joint, multivariate fixed point system. We formally introduce this in detail in Sec. 3.2.

This setup provides an elegant unification of diffusion and equilibrium models, and shows benefits in 1) parallel sampling of images, as it replaces the fully serial typical sampling process with a parallel one; and 2) model inversion (discussed in Sec. 3.3), where we can leverage fast gradients in the DEQ setting to much more quickly find the noise that generates a given image. The approach is also orthogonal and thus complementary to other methods used to reduce the sampling time, or improve model inversion. This chapter is based on Pokle et al. [2022].

3.1 Preliminaries

In this section, we will discuss some concepts of diffusion models that are relevant to this chapter. For a detailed background on deep equilibrium models and diffusion models, we point the readers to Sec. 2.1 and Sec. 2.2.1.

3.1.1 Discrete-time Diffusion Models

Diffusion models [Ho et al., 2020, Sohl-Dickstein et al., 2015, Dhariwal and Nichol, 2021, Song et al., 2020a] or score-based generative models [Song et al., 2020b, Song and Ermon, 2019] progressively perturb images with an increasing amount of Gaussian noise until it leads to a standard Gaussian distribution. The images can be generated by reversing this process by first sampling a noise sample from a standard Gaussian and then sequentially denoising it to generate images. As noted in [Song et al., 2020a, Ho et al., 2020, Dhariwal and Nichol, 2021], the length T of a diffusion process is usually large (*e.g.*, $T = 1000$) as it better approximates the Gaussian conditional distributions in the generative process which is represented with a stochastic differential equation (See Sec. 6.1). However, due to the large value of T, sampling from diffusion models can be visibly slower compared to other deep generative models such as GANs [Goodfellow et al., 2014]. This is an active area of research, and many approaches are being proposed to enable faster sampling. In this thesis, we will look into two primary approaches for faster sampling of diffusion models: parallel sampling of diffusion models in this chapter and diffusion distillation in the next chapter.

Non Markovian forward process. Song et al. [2020a] propose an alternate non-Markovian inference distribution that leads to a "shorter" and deterministic generative process, *i.e.*, denoising diffusion implicit model (DDIM). DDIM shares the same forward process as DDPM and, therefore, can be trained similarly to DDPM, using the variational lower bound shown in Eq. (2.16). However, for sampling, DDIM constructs a nearly non-stochastic scheme that can sample faster from the learned data distribution without introducing additional noises. Specifically, the scheme to generate a sample $\mathbf{x}_{t-1}$ given $\mathbf{x}_t$ is:

$$\mathbf{x}_{t-1} = \sqrt{\alpha_{t-1}}\left(\frac{\mathbf{x}_t - \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_\theta^{(t)}(\mathbf{x}_t)}{\sqrt{\alpha_t}}\right) + \sqrt{1-\alpha_{t-1}-\sigma_t^2}\cdot\boldsymbol{\epsilon}_\theta^{(t)}(\mathbf{x}_t) + \sigma_t\boldsymbol{\epsilon}_t \tag{3.1}$$

where $\boldsymbol{\epsilon}_t \sim \mathbb{N}(\mathbf{0}, \mathbf{I})$ and different values of σ_t define different generative processes. When $\sigma_t = \sqrt{\frac{1-\alpha_{t-1}}{1-\alpha_t}}\sqrt{1-\frac{\alpha_t}{\alpha_{t-1}}}$ for all t, the generative process represents a DDPM. Empirically, this is parameterized as

$$\sigma_t(\eta) = \eta\sqrt{\frac{1-\alpha_{t-1}}{1-\alpha_t}}\sqrt{1-\frac{\alpha_t}{\alpha_{t-1}}}, \tag{3.2}$$

where η is a hyperparameter to control stochasticity. Note that $\eta = 1$ corresponds to a DDPM Sohl-Dickstein et al. [2015], Ho et al. [2020]. Setting $\sigma_t = 0$ for all t gives rise to a DDIM, which results in a deterministic generating process except the initial sampling $\mathbf{x}_T \sim p(\mathbf{x}_T)$.

3.2 Parallelizing the Diffusion Sampling: A DEQ Formulation of Diffusion Models

In this section, we present the main modeling contribution of the paper, a formulation of diffusion processes under the DEQ framework. The specific parameterization of diffusion models that we will focus on is DDIM [Song et al., 2020a]. Although diffusion models may seem to be a natural fit for DEQ modeling (after all, we typically *do not* care about intermediate states in the denoising chain, but only

the final clean image), there are several reasons why setting up the diffusion chain "naively" as a DEQ (*i.e.,* making f_θ be a single sampling step) does not ultimately lead to a functional algorithm. Most fundamentally, the diffusion process is not time-invariant (*i.e.,* not "weight-tied" in the DEQ sense), and the final generated image is practically-speaking independent of the noise used to generate it (*i.e.,* not truly based upon "input injection" either).

Thus, at a high level, our approach to building a DEQ version of the DDIM involves representing *all* the states $\mathbf{x}_{0:T}$ *simultaneously* within the DEQ state. The advantage of this approach is that 1) we can exactly capture the typical diffusion inference chain; and 2) we can create a *more expressive* reverse process where the state $\mathbf{x}_t$ is updated based upon *all* previous states $\mathbf{x}_{t+1:T}$, improving the inference process; 3) we can execute all steps of the inference chain *in parallel* rather than solely in sequence as is typically required in diffusion models; and 4) we can use common DEQ acceleration methods, such as the Anderson solver Anderson [1965] to find the fixed point, which makes the sampling process converge faster. A downside of this formulation is that we need to store all DEQ states simultaneously (*i.e., only* the images, not the intermediate network states).

3.2.1 DEQ-DDIM: A DEQ Formulation for Deterministic Generative Process

The generative process of DDIM is given by which is obtained by setting $\sigma_t = 0$ in Eq. (3.1):

$$\mathbf{x}_{t-1} = \sqrt{\alpha_{t-1}}\left(\frac{\mathbf{x}_t - \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_\theta^{(t)}(\mathbf{x}_t)}{\sqrt{\alpha_t}}\right) + \sqrt{1-\alpha_{t-1}}\cdot\boldsymbol{\epsilon}_\theta^{(t)}(\mathbf{x}_t), \quad t = [1,\dots,T] \tag{3.3}$$

This process also allows us to generate a sample using a subset of latent states $\{\mathbf{x}_{\tau_1},\dots,\mathbf{x}_{\tau_S}\}$, where $\{\tau_1,\dots,\tau_S\} \subseteq T$. While this helps in accelerating the overall generative process, there is a tradeoff between sampling quality and computational efficiency. As noted in Song et al. [2020a], larger T values lead to lower FID scores of the generated images but need more compute time; smaller T are faster to sample from, but the resulting images have worse FID scores.

Reformulating this sampling process as a DEQ addresses multiple concerns raised above. We can define a DEQ, with a sequence of latent states $\mathbf{x}_{1:T}$ as its internal state, that *simultaneously* solves for the equilibrium points at all the timesteps. The global convergence of this process is upper bounded by T steps, by definition. To derive the DEQ formulation of the generative process, first we rearrange the terms in Eq. (3.3):

$$\mathbf{x}_{t-1} = \sqrt{\frac{\alpha_{t-1}}{\alpha_t}}\mathbf{x}_t + \left(\sqrt{1-\alpha_{t-1}} - \sqrt{\frac{\alpha_{t-1}(1-\alpha_t)}{\alpha_t}}\right)\boldsymbol{\epsilon}_\theta^{(t)}(\mathbf{x}_t) \tag{3.4}$$

Let $c_1^{(t)} = \sqrt{1-\alpha_{t-1}} - \sqrt{\frac{\alpha_{t-1}(1-\alpha_t)}{\alpha_t}}$. Then we can write

$$\mathbf{x}_{t-1} = \sqrt{\frac{\alpha_{t-1}}{\alpha_t}}\mathbf{x}_t + c_1^{(t)}\boldsymbol{\epsilon}_\theta^{(t)}(\mathbf{x}_t) \tag{3.5}$$

By induction, we can rewrite the above equation as:

$$\mathbf{x}_{T-k} = \sqrt{\frac{\alpha_{T-k}}{\alpha_T}}\mathbf{x}_T + \sum_{t=T-k}^{T-1}\sqrt{\frac{\alpha_{T-k}}{\alpha_t}}c_1^{(t+1)}\boldsymbol{\epsilon}_\theta^{(t+1)}(\mathbf{x}_{t+1}), \quad k \in [0,..,T] \tag{3.6}$$

This defines a "fully-lower-triangular" inference process, where the update of $\mathbf{x}_t$ depends on the noise prediction network ϵ_θ applied to *all* subsequent states $\mathbf{x}_{t+1:T}$; in contrast to the traditional diffusion process, which updates $\mathbf{x}_t$ based only on $\mathbf{x}_{t+1}$. Specifically, let $h(\cdot)$ represent the function that performs the operations in the equations Eq. (3.6) for a latent $\mathbf{x}_t$ at timestep t, and let $\tilde{h}(\cdot)$ represent the function that performs the same set of operations across all the timesteps simultaneously. We can write the above set of equations as a fixed point system:

$$\begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{x}_{T-2} \\ \vdots \\ \mathbf{x}_0 \end{bmatrix} = \begin{bmatrix} h(\mathbf{x}_T) \\ h(\mathbf{x}_{T-1:T}) \\ \vdots \\ h(\mathbf{x}_{1:T}) \end{bmatrix}$$

or,

$$\mathbf{x}_{0:T-1} = \tilde{h}(\mathbf{x}_{0:T-1}; \mathbf{x}_T) \tag{3.7}$$

The above system of equations represent a DEQ with $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ as input injection. We can simultaneously solve for the roots of this system of equations using black-box solvers such as Anderson acceleration [Anderson, 1965]. Let $g(\mathbf{x}_{0:T-1}; \mathbf{x}_T) = \tilde{h}(\mathbf{x}_{0:T-1}; \mathbf{x}_T) - \mathbf{x}_{0:T-1}$, then we have

$$\mathbf{x}^*_{0:T} = \text{RootSolver}(g(\mathbf{x}_{0:T-1}; \mathbf{x}_T)) \tag{3.8}$$

This DEQ formulation has multiple benefits. Solving for all the equilibria simultaneously leads to a better estimation of the intermediate latent states $\mathbf{x}_t$ in a fewer number of steps (*i.e.*, $\leq t$ steps for $\mathbf{x}_t$). This leads to faster convergence of the sampling process as the final sample $\mathbf{x}_0$, which is dependent on the latent states of all the previous time steps, has a better estimate of these intermediate latent states. Note that by the same reasoning, the intermediate latent states $\mathbf{x}_t$ converge faster too. Thus, we can get images with perceptual quality comparable to DDIM in a significantly fewer number of steps. Of course, we also note that the computational requirements of each individual step has significantly increased, but this is at least largely offset by the fact that the steps can be executed as mini-batched in parallel over each state. Empirically, in fact, we often notice significant *speedup* using this approach on tasks like single image generation.

3.2.2 DEQ-sDDIM: A DEQ Formulation for Stochastic Generative Process

We now extend the formulation for deterministic DDIM to a more general case for $\sigma_t \neq 0$.

Rearranging the terms in Eq. (3.1), we get

$$\mathbf{x}_{t-1} = \sqrt{\frac{\alpha_{t-1}}{\alpha_t}}\mathbf{x}_t + \left(\sqrt{1 - \alpha_{t-1} - \sigma_t^2} - \sqrt{\frac{\alpha_{t-1}(1-\alpha_t)}{\alpha_t}} \right) \epsilon_\theta^{(t)}(\mathbf{x}_t) + \sigma_t \boldsymbol{\epsilon}_t \tag{3.9}$$

Let $c_1^{(t)} = \sqrt{1 - \alpha_{t-1} - \sigma_t^2} - \sqrt{\frac{\alpha_{t-1}(1-\alpha_t)}{\alpha_t}}$. Then we can write

$$\mathbf{x}_{t-1} = \sqrt{\frac{\alpha_{t-1}}{\alpha_t}}\mathbf{x}_t + c_1^{(t)} \epsilon_\theta^{(t)}(\mathbf{x}_t) + \sigma_t \boldsymbol{\epsilon}_t \tag{3.10}$$

By induction, we can rewrite the above equation as:

$$\mathbf{x}_{T-k} = \sqrt{\frac{\alpha_{T-k}}{\alpha_T}}\mathbf{x}_T + \sum_{t=T-k}^{T-1} \sqrt{\frac{\alpha_{T-k}}{\alpha_t}} \left(c_1^{(t+1)} \epsilon_\theta^{(t+1)}(\mathbf{x}_{t+1}) + \sigma_{t+1}\boldsymbol{\epsilon}_{t+1} \right), \quad k \in [0, .., T] \tag{3.11}$$

This again defines a "fully-lower-triangular" inference process, where the update of $\mathbf{x}_t$ depends on the noise prediction network ϵ_θ applied to *all* subsequent states $\mathbf{x}_{t+1:T}$.

Following the notation used in the main paper, let $h(\cdot)$ represent the function that performs the operations in the equations Eq. (3.11) for a latent $\mathbf{x}_t$ at timestep t, let $\tilde{h}(\cdot)$ represent the function that performs the same set of operations across all the timesteps simultaneously, and let $\boldsymbol{\epsilon}_{1:T}$ represent the noise injected into the diffusion process at every timestep. We can write the above set of equations as a fixed point system:

$$\begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{x}_{T-2} \\ \vdots \\ \mathbf{x}_0 \end{bmatrix} = \begin{bmatrix} h(\mathbf{x}_T; \boldsymbol{\epsilon}_T) \\ h(\mathbf{x}_{T-1:T}; \boldsymbol{\epsilon}_{T-1:T}) \\ \vdots \\ h(\mathbf{x}_{1:T}; \boldsymbol{\epsilon}_{1:T}) \end{bmatrix}$$

or,

$$\mathbf{x}_{0:T-1} = \tilde{h}(\mathbf{x}_{0:T-1}; \mathbf{x}_T, \boldsymbol{\epsilon}_{1:T}) \tag{3.12}$$

The above system of equations represent a DEQ with $\mathbf{x}_T$ and $\boldsymbol{\epsilon}_{1:T}$ as an input injection. We refer to this formulation of DEQ for stochastic DDIM with $\eta > 0$ as DEQ-sDDIM.

A major difference between Eq. (3.7) in Sec. 3.2.1 and the above derivation is that DEQ-sDDIM can exploit the noises $\boldsymbol{\epsilon}_{1:T}$ sampled prior to fixed point solving as addition input injections. The insight here is that the noises along the sampling chain are independent of each other, thus allowing us to sample all the noises *simultaneously* and convert a highly stochastic autoregressive sampling process into a deterministic "fully-lower-triangular" DEQ. As usual, we can now use black-box solvers like Anderson acceleration [Anderson, 1965] to solve the roots of this system of equations. Let $g(\mathbf{x}_{0:T-1}; \mathbf{x}_T, \boldsymbol{\epsilon}_{1:T}) = \tilde{h}(\mathbf{x}_{0:T-1}; \mathbf{x}_T, \boldsymbol{\epsilon}_{1:T}) - \mathbf{x}_{0:T-1}$, then we have

$$\mathbf{x}^*_{0:T} = \text{RootSolver}(g(\mathbf{x}_{0:T-1}); \mathbf{x}_T, \boldsymbol{\epsilon}_{1:T}) \tag{3.13}$$

where RootSolver$(\cdot)$ is any black-box fixed point solver.

3.3 Efficient Model Inversion of DDIM with DEQ

One of the primary strengths of DEQs is their constant memory consumption, for both forward pass and backward pass, regardless of their 'effective depth'. This leads to an interesting application of DEQs in inverting DDIMs that fully leverages this advantage along with the other benefits discussed in the previous section.

Algorithm 1 A naive algorithm to invert DDIM

Input: A target image $\mathbf{x}_0 \sim \mathcal{D}$, $\epsilon_\theta(\mathbf{x}_t, t)$ a trained denoising diffusion model, N the total number of epochs

▷ f denotes the sampling process in Eq Eq. (3.3)

Initialize $\hat{\mathbf{x}}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

for epochs from 1 to N **do**

  for $t = T, ..., 1$ **do**

    Sample $\hat{\mathbf{x}}_{t-1} = f(\hat{\mathbf{x}}_t; \epsilon_\theta(\hat{\mathbf{x}}_t, t))$

  end for

  Take a gradient descent step on

$$\nabla_{\hat{\mathbf{x}}_T} \|\hat{\mathbf{x}}_0 - \mathbf{x}_0\|_F^2$$

end for

Output: $\hat{\mathbf{x}}_T$

Algorithm 2 Inverting DDIM with DEQ

Input: A target image $\mathbf{x}_0 \sim \mathcal{D}$, $\epsilon_\theta(\mathbf{x}_t, t)$ a trained denoising diffusion model, N the total number of epochs

▷ g is the function in Eq. Eq. (3.8)

Initialize $\hat{\mathbf{x}}_{0:T} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

for epochs from 1 to N **do**

  ▷ Disable gradient computation

  $\mathbf{x}^*_{0:T} = \text{RootSolver}(g(\mathbf{x}_{0:T-1}); \mathbf{x}_T)$

  ▷ Enable gradient computation

  Compute Loss $\mathcal{L}(\mathbf{x}_0, \mathbf{x}^*_0)$

  Use the 1-step grad to compute $\partial \mathcal{L} / \partial \mathbf{x}_T$

  Take a gradient descent step using above

end for

Output: $\mathbf{x}^*_T$

3.3.1 Problem Setup

Given an arbitrary image $\mathbf{x}_0 \sim \mathcal{D}$, and a denoising diffusion model $\epsilon_\theta(\mathbf{x}_t, t)$ trained on a dataset $\mathcal{D}$, model inversion seeks to determine the latent $\hat{\mathbf{x}}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ that can generate an image $\hat{\mathbf{x}}_0$ identical to the original image $\mathbf{x}_0$ through the generative process for DDIM described in Eq. Eq. (3.3). For an input image $\mathbf{x}_0$, and a generated image $\hat{\mathbf{x}}_0$, this task needs to minimize the squared-Frobenius distance between these images:

$$\mathcal{L}(\mathbf{x}_0, \hat{\mathbf{x}}_0) = \|\mathbf{x}_0 - \hat{\mathbf{x}}_0\|_F^2 \tag{3.14}$$

3.3.2 Inverting DDIM: A Naive Approach

A relatively straightforward way to invert DDIM is to randomly sample $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and update it via gradient descent by first estimating $\mathbf{x}_0$ using the generative process in Eq. (3.3) and backpropagating through this process after computing the loss objective in Eq. (3.14). The overall process has been summarized in Algorithm 1. This process has a large computational overhead. Every training epoch requires sequential sampling for all T time steps. Optimizing through this generative process would require the creation of a large computational graph for storing relevant intermediate variables necessary for the backward pass. Sequential sampling further slows down the entire process.

3.3.3 Efficient Inversion of DDIM with DEQs

Alternatively, we can use the DEQ formulation to develop a much more efficient inversion method. We provide a high-level overview of this approach in Algorithm 2. We can apply implicit function theorem (IFT) to the fixed point, *i.e.*, Eq. (2.3) to compute gradients of the loss $\mathcal{L}(\mathbf{x}_0, \mathbf{x}^*_0)$ in Eq. (3.14)

w.r.t. $(\cdot)$:

$$\frac{\partial \mathcal{L}}{\partial (\cdot)} = -\frac{\partial \mathcal{L}}{\partial \mathbf{x}^*_{0:T}} \left(J^{-1}_{g_\theta}\big|_{\mathbf{x}^*_{0:T}} \right) \frac{\partial \tilde{h}(\mathbf{x}^*_{0:T-1}; \mathbf{x}_T)}{\partial (\cdot)} \tag{3.15}$$

where $(\cdot)$ could be any of the latent states $\mathbf{x}_1, ..., \mathbf{x}_T$, and $J^{-1}_{g_\theta}\big|_{\mathbf{x}^*_{0:T}}$ is the inverse Jacobian of $g(\mathbf{x}_{0:T-1}; \mathbf{x}_T)$ evaluated at $x^*_{0:T}$. Refer to Bai et al. [2019] for a detailed proof. Computing the inverse of Jacobian matrix can become computationally intractable, especially when the latent states $\mathbf{x}_t$ are high-dimensional, as in images. Recent works [Geng et al., 2021a, Fung et al., 2021, Geng et al., 2021b, Bai et al., 2022] suggest that we do not need an exact gradient to train DEQs. We can instead use an approximation to Eq. (3.15), *i.e.*,

$$\frac{\partial \mathcal{L}}{\partial (\cdot)} = -\frac{\partial \mathcal{L}}{\partial \mathbf{x}^*_{0:T}} \mathbf{M} \frac{\partial \tilde{h}(\mathbf{x}^*_{0:T-1}; \mathbf{x}_T)}{\partial (\cdot)} \tag{3.16}$$

where $\mathbf{M}$ is an approximation of $J^{-1}_{g_\theta}\big|_{\mathbf{x}^*_{0:T}}$. For example, [Geng et al., 2021a, Fung et al., 2021, Geng et al., 2021b] show that setting $\mathbf{M} = \mathbf{I}$, *i.e.*, 1-step gradient, works well. In this work, we follow Geng et al. [2021b] to further add a damping factor to the 1-step gradient. The forward pass is given by:

$$\mathbf{x}^*_{0:T} = \text{RootSolver}(g(\mathbf{x}_{0:T-1}); \mathbf{x}_T) \tag{3.17}$$

$$\mathbf{x}^*_{0:T} = \tau \cdot \tilde{h}(\mathbf{x}^*_{0:T-1}; \mathbf{x}^*_T) + (1 - \tau) \cdot \mathbf{x}^*_{0:T} \tag{3.18}$$

The gradients for the backward pass can be computed through standard autograd packages. We provide the PyTorch-style pseudocode of our approach in Algorithm 4. Using inexact gradients for the backward pass has several benefits: 1) It remarkably improves the training stability of DEQs; 2) Our backward pass consists of a single step and is ultra-cheap to compute. It reduces the total training time by a significant amount. It is easy to extend the strategy used in Algorithm 2 and use DEQs to invert DDIMs with stochastic generative process (referred to as DEQ-sDDIM). We provide the key steps of this approach in Algorithm 3.

Algorithm 3 Inverting stochastic DDIM with DEQ (DEQ-sDDIM)

Input: A target image $\mathbf{x}_0 \sim \mathcal{D}$, a trained denoising diffusion model $\epsilon_\theta(\mathbf{x}_t, t)$, the total number of epochs N, diffusion function g defined in Eq. (3.13)

Initialize $\hat{\mathbf{x}}_{0:T} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, $\epsilon_{1:T} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

for epochs from 1 to N **do**

  $\mathbf{x}^*_{0:T-1} = \text{RootSolver}(g(\mathbf{x}_{0:T-1}); \mathbf{x}_T, \epsilon_{1:T})$. ▷ Disable gradient computation

  Compute Loss $\mathcal{L}(\mathbf{x}_0, \mathbf{x}^*_0)$. ▷ Enable gradient computation

  Compute $\partial \mathcal{L} / \partial \mathbf{x}_T$ using 1-step grad.

  Update $\hat{\mathbf{x}}_T$ with gradient descent.

end for

Output: $\mathbf{x}^*_T$

3.4 Details of Experimental Setup

Datasets. We consider four datasets that have images of different resolutions for our experiments: CIFAR10 (32×32) [Krizhevsky, 2009], CelebA (64×64) [Liu et al., 2015], LSUN Bedroom (256×256) and LSUN Outdoor Church (256×256) [Yu et al., 2015].

Algorithm 4 PyTorch-style pseudocode for inversion with DEQ-DDIM

```
# x0: a target image for inversion
# all_xt: all the latents in the diffusion chain or its subsequence; xT is at index 0, x0 is at the
    last index
# func: A function that performs the required operations on the fixed-point system for a single
    timestep
# solver: A fixed-point solver like Anderson acceleration
# optimizer: an optimization algorithm like Adam
# tau: the damping factor τ for phantom gradient
# num_epochs: the max number of epochs

def forward(func, x):
    with torch.no_grad():
        z = solver(func, x)
    z = tau * func(z) + (1 - tau) * z
    return z

def invert(func, all_xt, x0, optimizer, num_epochs):
    for epoch in range(num_epochs):
        optimizer.zero_grad()
        xt_pred = forward(func, all_xt)
        loss = (xt_pred[-1] - x0).norm(p='fro')
        loss.backward()
        optimizer.step()
    return all_xt[0]
```

Model Architecture. We use the standard U-Net [Falk et al., 2019] architecture for $\epsilon_\theta(\mathbf{x}_t, t)$ as used previously in Ho et al. [2020], Song et al. [2020a]. We use pretrained models from Ho et al. [2020] for CIFAR10, LSUN Bedrooms and Outdoor Churches, and from Song et al. [2020a] for CelebA.

DEQ solver details. For all the experiments, we use Anderson acceleration as the default fixed point solver. While training DEQs for model inversion, we use the 1-step gradient Eq. (3.18) to compute the backward pass. The damping factor τ for 1-step gradient is set to 0.1.

General setting. We follow the linear selection procedure to select a subsequence of timesteps $\tau_S \subset T$ for all the datasets except CIFAR10, *i.e.*, we select timesteps such that $\tau_i = \lfloor ci \rfloor$ for some c. For CIFAR10, we select timesteps such that $\tau_i = \lfloor ci^2 \rfloor$ for some c. The constant c is selected so that τ_{-1} is close to T. We use Anderson acceleration [Anderson, 1965] as our fixed-point solver for all the experiments. We set the equilibrium error threshold of solver to 0.001 and set the history length to 5. We allow a maximum of 15 solver forward steps in all the experiments with DEQ-DDIM, and use a maximum of 50 solver forward steps for DEQ-sDDIM. Finally, we use PyTorch's inbuilt `DataParallel` module to handle parallelization. We use upto 4 NVIDIA Quadro RTX 8000 or RTX A6000 GPUs for all our experiments.

Training details for model inversion. We implement and test the code in PyTorch version 1.11.0. We use the Adam [Kingma and Ba, 2014] optimizer with a learning rate of 0.01. We train DEQs for 400 epochs on CIFAR10 and CelebA, and for 500 epochs on LSUN Bedroom, and LSUN Outdoor Church.

The baseline is trained for 1000 epochs on CIFAR10 with $T = 100$, for 3000 epochs on CIFAR10 with $T = 10$, for 2500 epochs on CelebA, and for 2000 epochs on LSUN Bedroom, and LSUN Outdoor Church. At the beginning of inversion procedure with DEQ-DDIM, we sample $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and initialize the latents at all (or the subsequence of) timesteps to this value. For inversion with DEQ-sDDIM we also sample $\boldsymbol{\epsilon}_{1:T} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. We stop the training as soon as the loss falls below 0.5 for CIFAR10, and below 2 for other datasets.

Evaluation. We compute Fréchet Inception Distance (FID) [Heusel et al., 2017] scores using the code provided by `https://github.com/w86763777/pytorch-gan-metrics` on $50,000$ images. We also use the precomputed statistics for CIFAR10, LSUN Bedrooms and Outdoor Churches provided in this github repository. For CelebA, we compute our own dataset statistics, as the precomputed statistics for images of resolution 64×64 are not included in this repository. While computing the statistics, we preprocess the images of CelebA in exactly the same way as done by Song et al. [2020a].

3.5 Convergence Properties

3.5.1 Convergence of DEQ-DDIM

We verify that DEQ-DDIM converges to a fixed point by plotting the values of $\|\tilde{h}(\mathbf{x}_{0:T}) - \mathbf{x}_{0:T}\|_2$ over Anderson solver steps. As seen in Figure 3.1, DEQ-DDIM converges to a fixed point for generative processes of different lengths. It is easier to reach simultaneous equilibria on smaller sequence lengths than larger sequence lengths. However, this does not affect the quality of images generated. We visualize the latent states of DEQ-DDIM in Figure 3.2. Our experiments demonstrate that DEQ-DDIM is able to generate high-quality images in as few as 15 Anderson solver steps on diffusion chains that were trained on a much larger number of steps T. One might note that DEQ-DDIM converges to a limit cycle for diffusion processes with larger sequence lengths. This is not a limitation as we only want the latent states at the last few timesteps to converge well, which happens in practice as demonstrated in Fig. 3.2. Further, these residuals can be driven down by using more powerful solvers like quasi-Newton methods, *e.g.*, Broyden's method.

3.5.1.1 Effects of choice of initialization on convergence of DEQ-DDIM.

The choice of initialization is critical for fast convergence of DEQs. Bai et al. [2019] initialize the initial estimate of the fixed point of DEQ with zeros. However, in this work, we initialize all latent states with $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$, as it results in a faster convergence as shown in Figure 3.3. Although both initialization schemes eventually converge to high quality images, we observe that initializing with $\mathbf{x}_T$ results in up to $3\times$ faster convergence compared to zero initialization. We observe a significant qualitative difference in the visualization of the intermediate states of the diffusion chain at different solver steps for the two initialization schemes, as observed in Figure 3.4.

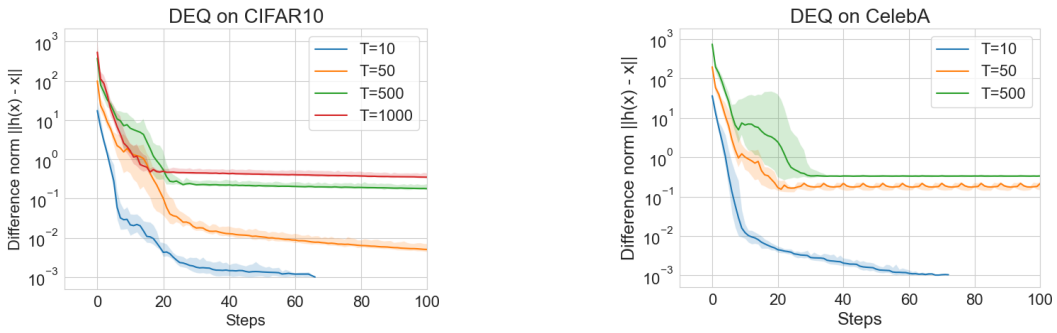


Figure 3.1: DEQ-DDIM finds an equilibrium point. We plot the absolute fixed-point convergence $\|\tilde{h}(\mathbf{x}_{0:T}) - \mathbf{x}_{0:T}\|_2$ during a forward pass of DEQ for CIFAR-10 (left) and CelebA (right) for different number of steps T. The shaded region indicates the maximum and minimum value encountered during any of the 25 runs.

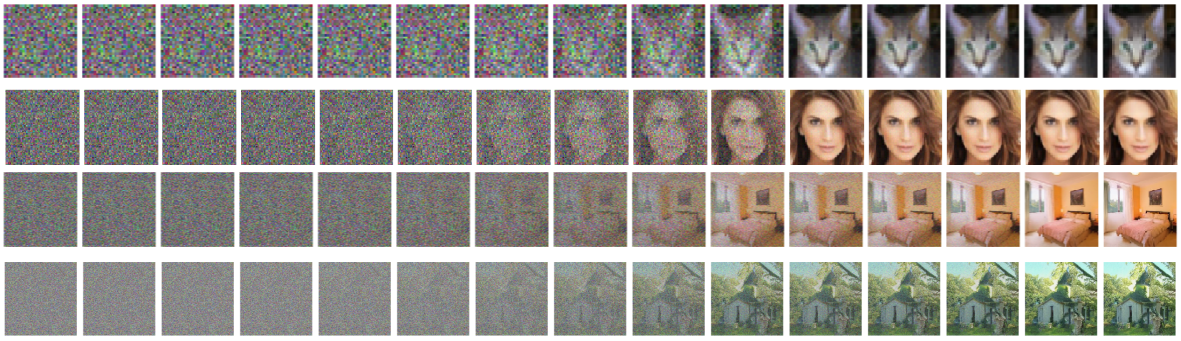


Figure 3.2: Visualization of intermediate latents $\mathbf{x}_t$ of DEQ-DDIM after 15 forward steps with Anderson solver for CIFAR-10 (first row, $T = 500$), CelebA (second row, $T = 500$), LSUN Bedroom (third row, $T = 50$, and LSUN Outdoor Church (fourth row, $T = 50$). For $T = 500$, we visualize every 50^{th} latent, and for $T = 50$, we visualize every 5^{th} latent. In addition, we also visualize $\mathbf{x}_{0:4}$ in the last 5 columns.

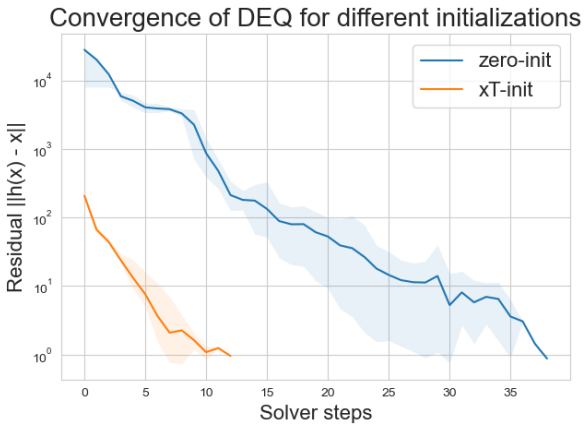


Figure 3.3: Choice of initialization is critical in DEQs: Initializing DEQs with $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$ results in much faster convergence compared to zero initialization. We report the convergence results on CIFAR10 using 5 runs on diffusion chains of length 50.

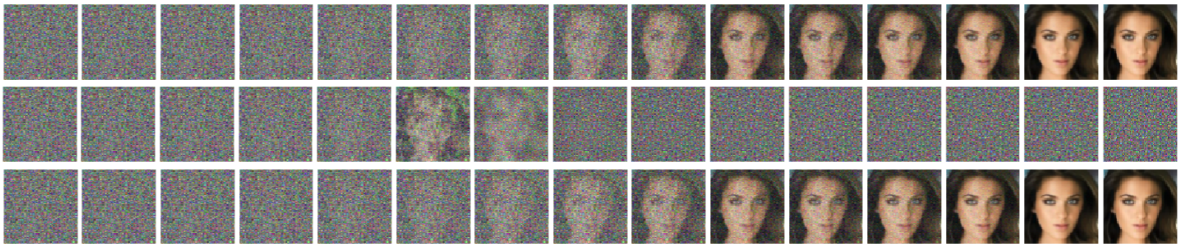


Figure 3.4: Visualization of intermediate latents $\mathbf{x}_t$ for CelebA for different choices of initialization for $T = 50$: (**first row**) Initialization with $\mathbf{x}_T$ after 10 Anderson solver steps (**second row**) Zero initialization after 10 Anderson solver steps (**third row**) Zero initialization after 30 Anderson solver steps. We visualize every 5^{th} latent in $\mathbf{x}_{0:T-1}$ at the given solver step (10 or 30). We also visualize $\mathbf{x}_{0:4}$ in the last 5 columns. Initialization with $\mathbf{x}_T$ produces visually appealing images much faster.

3.5.2 Convergence of DEQ-sDDIM

We verify that our DEQ version for stochastic DDIM converges to a fixed point for different levels of stochasticity, indicated by parameter η, by plotting values of $\|\tilde{h}(\mathbf{x}_{0:T}) - \mathbf{x}_{0:T}\|_2$ over Anderson solver steps in Figure 3.5. As one would expect, we need more solver steps to solve for the fixed point given higher values of η.

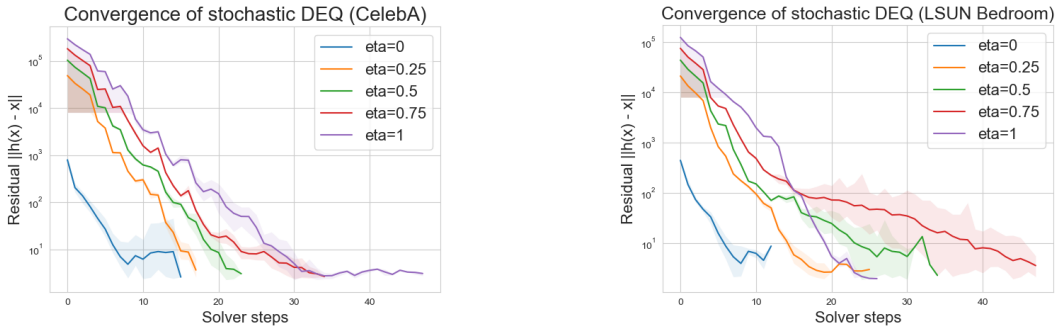


Figure 3.5: DEQ-sDDIM finds an equilibrium point. We plot the absolute fixed-point convergence $\|\tilde{h}(\mathbf{x}_{0:T}) - \mathbf{x}_{0:T}\|_2$ during a forward pass of DEQ for CelebA (left) and LSUN Bedrooms (right) for different number of steps T. The shaded region indicates the maximum and minimum value encountered during any of the 10 runs.

3.6 Results of Parallel Sampling with DEQs

Results for DEQ-DDIM. We verify that DEQ-DDIM can generate images of comparable quality to DDIM by reporting Fréchet Inception Distance (FID) [Heusel et al., 2017] in Table 3.1. For the forward pass of DEQ-DDIM, we run Anderson solver for a maximum of 15 steps for each image. We report FID scores on 50,000 images and average time to generate an image (including GPU time) on 500 images. We note significant gains in wall-clock time in single-shot image generation with DEQ-DDIM on images with smaller resolutions. Specifically, DEQ-DDIM can generate images almost $2\times$ faster

than sequential DDIM sampling on CIFAR-10 (32×32) and CelebA (64×64). We note that these gains vanish on sequences of shorter lengths. This is because the number of fixed point solver iterations needed for convergence becomes comparable to the length of the diffusion chain for small values of T. Thus, lightweight updates performed on short diffusion chains for sequential sampling are faster compared to compute heavy updates in DEQ-DDIM.

Results for DEQ-sDDIM. We report FID scores on DEQ-sDDIM for CIFAR10 in Table 3.2 and CelebA in Table 3.3. We run Anderson solver for a maximum of 50 steps for each image. We observe that while DEQ-sDDIM is slower than DDIM, it always generates images with comparable or better FID scores. For higher levels of stochasticity *i.e.,* for larger values of η, DEQ-sDDIM needs more Anderson solver iterations to converge to a fixed point, increasing image generation wall clock time. Finally, we also find that on full batch inference with larger batches, sequential sampling could outperform DEQ-DDIM, as the latter would have larger memory requirements in this case, *i.e.,* processing smaller batches of size B might be faster than processing larger batches of size BT.

Dataset	T	DDPM		DDIM		DEQ-DDIM	
		FID	Time	FID	Time	FID	Time
CIFAR10	1000	**3.17**	24.45s	4.07	20.16s	3.79	**2.91s**
CelebA	500	5.32	14.95s	3.66	10.31s	**2.92**	**5.12s**
LSUN Bedroom	25	184.05	1.72s	8.76	**1.19s**	**8.73**	3.82s
LSUN Church	25	122.18	1.77s	**13.44**	**1.68s**	13.55	3.99s

Table 3.1: FID scores and time for single image generation for DDPM, DDIM and DEQ-DDIM.

η	T	FID Scores		Time (in seconds)	
		DDIM	DEQ-sDDIM	DDIM	DEQ-sDDIM
0.2	20	7.19	**6.99**	**0.33**	0.51
0.5	20	8.35	**8.22**	**0.35**	0.51
1	20	18.37	**17.72**	**0.34**	0.93
0.2	50	4.69	**4.44**	**0.88**	0.88
0.5	50	5.26	**4.99**	**0.83**	1.00
1	50	8.02	**7.85**	**0.83**	1.58

Table 3.2: FID scores for single image generation for DDIM and DEQ-sDDIM on CIFAR10. Note that DDPM Ho et al. [2020] with a larger variance achieves FID scores of 133.37* and 32.72* respectively for $T = 20$ and $T = 50$, where $*$ indicates numbers reported from Song et al. [2020a]

.

3.7 Results of Model Inversion with DEQs

Results for DEQ-DDIM. We report the minimum values of squared Frobenius norm between the recovered and target images averaged from 100 different runs in Table 3.4. We report results for DEQ

η	T	FID Scores		Time (in seconds)	
		DDIM	DEQ-sDDIM	DDIM	DEQ-sDDIM
0.2	20	13.85	**13.52**	**0.42**	1.53
0.5	20	15.67	**15.27**	**0.42**	2.17
1	20	25.85	**25.31**	**0.42**	2.35
0.2	50	9.33	**8.66**	**1.05**	4.17
0.5	50	10.75	**9.73**	**1.05**	5.18
1	50	18.22	**15.57**	**1.05**	8.59

Table 3.3: FID scores for single image generation using stochastic DDIM and DEQ-sDDIM on CelebA. Note that DDPM with a larger variance achieves FID score of 183.83* and 71.71* on $T = 20$ and $T = 50$, respectively, where $*$ indicates the numbers from Song et al. [2020a]

with $\eta = 0$ (*i.e.,* DEQ-DDIM) in this table, and additional results for $\eta > 0$ (*i.e.,* DEQ-sDDIM) are reported in Figure 3.12. DEQ outperforms the baseline method on all the datasets by a significant margin. We also plot the training loss curves of DEQ-DDIM and the baseline in Figure 3.6. We observe that DEQ-DDIM converges faster and has much lower loss values than the baseline method induced by DDIM. We also visualize the images generated with the recovered latent states for DEQ-DDIM in Figure 3.7. It is worth noting that images generated with DEQs capture more vivid details of the original images, like textures of foliage, crevices, and other finer details than the baseline. We include additional results of model inversion with DEQ-sDDIM on different datasets in Sec. 3.9.

Dataset	T	Baseline		DEQ-DDIM	
		Min loss ↓	Avg Time (mins) ↓	Min loss ↓	Avg Time (mins) ↓
CIFAR10	100	15.74 ± 8.7	49.07 ± 1.76	$\mathbf{0.76 \pm 0.35}$	12.99 ± 0.97
CIFAR10	10	2.59 ± 3.67	14.36 ± 0.26	$\mathbf{0.68 \pm 0.32}$	2.54 ± 0.41
CelebA	20	14.13 ± 5.04	30.09 ± 0.57	$\mathbf{1.03 \pm 0.37}$	28.09 ± 1.76
Bedroom	10	1114.49 ± 795.86	26.41 ± 0.17	$\mathbf{36.37 \pm 22.86}$	33.7 ± 1.05
Church	10	1674.68 ± 1432.54	29.7 ± 0.75	$\mathbf{47.94 \pm 24.78}$	33.54 ± 3.02

Table 3.4: Comparison of minimum loss and average time required to generate an image. All the results have been reported on 100 images.

Results for DEQ-sDDIM. We report the minimum values of squared Frobenius norm between the recovered and target images averaged from 25 different runs in Figure 3.12. We use the same hyperparameters as the ones used for training DEQ models for DDIM in these experiments. DEQ-sDDIM is able to achieve low values of the reconstruction loss even for large values of η like 1 as noted in Figure 3.12. We also plot training loss curves for different values of η in Figure 3.11 on CIFAR10. We note that it indeed takes longer time to invert DEQ-sDDIM for higher values of η. This is primarily because the fixed point solver needs more iterations to converge. However, despite that we obtain impressive model inversion results on CIFAR10 and CelebA. We visualize images generated with the recovered latent states in Figure 3.8, Figure 3.9 and Figure 3.10.

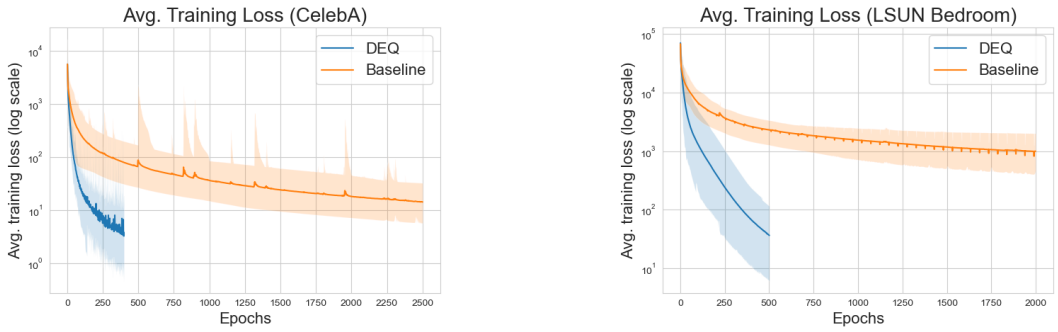


Figure 3.6: Training loss for CelebA and LSUN Bedroom over epochs. DEQ-DDIM converges in fewer epochs, and achieves lower values of loss compared to the baseline. The shaded region indicates the maximum and minimum value of loss encountered during any of the 100 runs.

Figure 3.7: Model inversion on CIFAR10, CelebA, LSUN Bedrooms and Churches, respectively. Each triplet has the original image (left), DDIM's inversion (middle), and DEQ-DDIM's inversion (right).

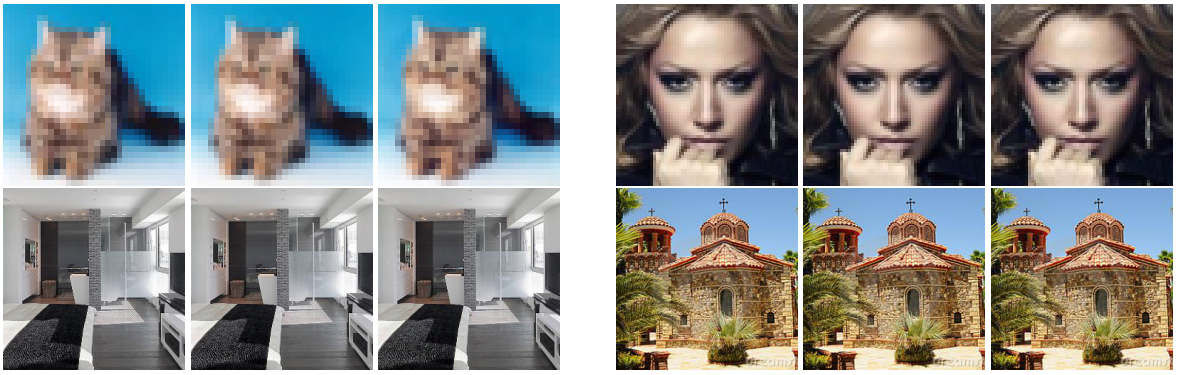


Figure 3.8: Model inversion of DEQ-sDDIM on CIFAR10, CelebA, LSUN Bedrooms and Churches, respectively. Each triplet displays the original image (left), and images obtained through inversion with DEQ-sDDIM for $\eta = 0.5$ (middle), and $\eta = 1$ (right).

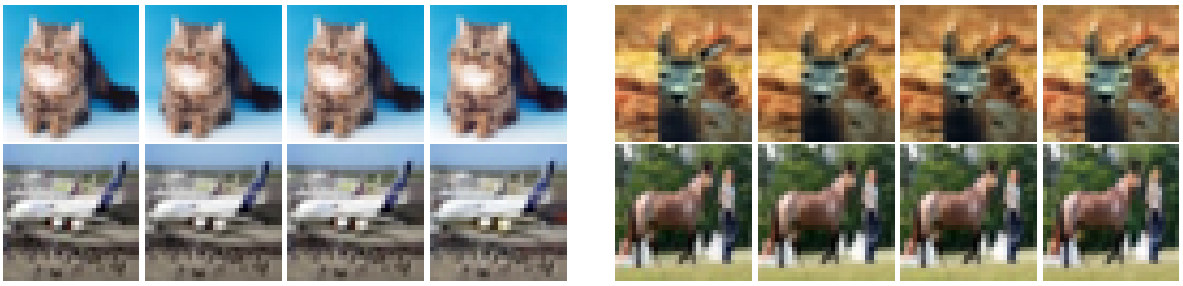


Figure 3.9: Results of model inversion of DEQ-sDDIM on CIFAR10: For every set of four images from **left to right** we display the original image, and images obtained through inversion for $\eta = 0$, $\eta = 0.5$, and $\eta = 1$.

Figure 3.10: Results of model inversion of DEQ-sDDIM on CelebA: For every set of four images from **left to right** we display the original image, and images obtained through inversion for $\eta = 0$, $\eta = 0.5$, and $\eta = 1$.

Figure 3.11: Training loss of inversion for DEQ-sDDIM on CIFAR10 averaged over 25 different runs

Dataset	T	η	Min loss	Epochs
CIFAR10	10	0	0.76 ± 0.35	400
CIFAR10	10	0.5	0.49 ± 0.001	600
CIFAR10	10	1	6.64 ± 3.45	1000
CelebA	20	0	0.41 ± 0.07	1500
CelebA	20	0.5	1.57 ± 1.63	1500
CelebA	20	1	5.03 ± 1.57	1500

Figure 3.12: Minimum loss for inversion with DEQ-sDDIM

3.8 Ablation Studies for Model Inversion with DEQ-DDIM

3.8.1 Effect of length of sampling chain

3.8.1.1 Effect of length of subsequence τ_S during training

We study the effect of the length of the diffusion chain on the convergence rate of optimization for model inversion in Fig. 3.13. We note that for sequential sampling, loss decreases slightly faster for the

smaller diffusion chain ($\tau_S = 10$ and 20) than the longer one ($\tau_S = 100$) for the baseline. However, for DEQ-DDIM, the length of diffusion chain doesn't seem to have an effect on the rate of convergence as the loss curves for $\tau_S = 100$ and $\tau_S = 10$ and 20 are nearly identical.

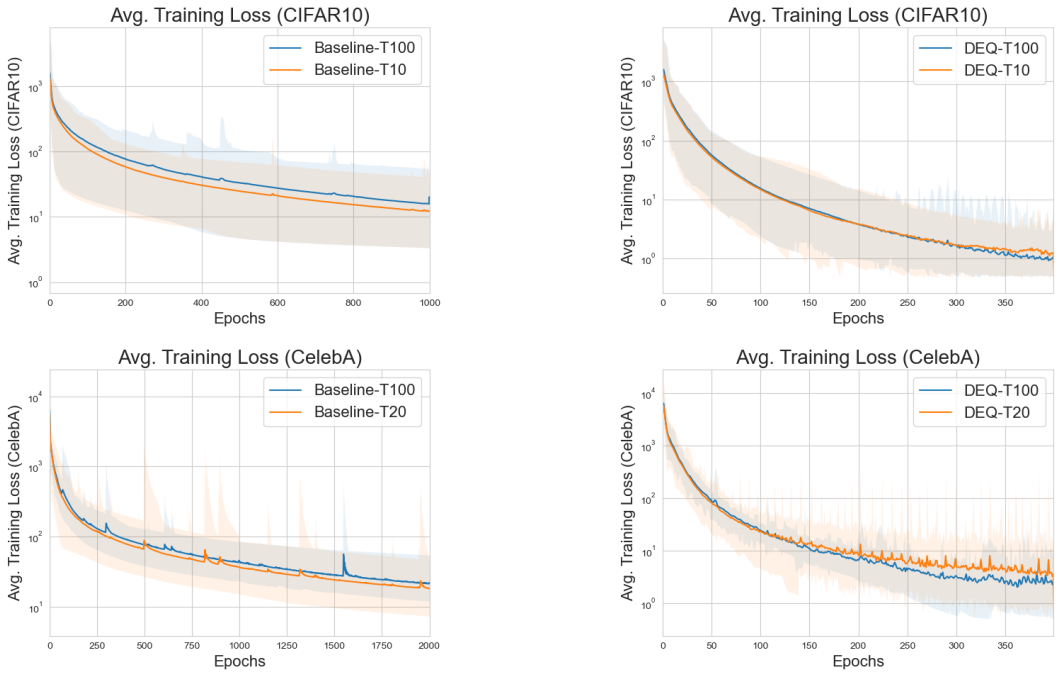


Figure 3.13: Effect of length of diffusion chain on optimization process for model inversion with DEQ-DDIM. We display training loss curves for (left) baseline and (right) DEQ for CIFAR10 (top row) and CelebA (bottom row). It is slightly easier to optimize smaller diffusion chain for the Baseline. The error bar indicates the maximum and minimum value of loss encountered during any of the 100 runs for CIFAR10, and 25 runs for CelebA.

3.8.1.2 Effect of length of subsequence τ_S during sampling

All the images in Fig. 3.7 are sampled with a subsequence of timesteps $\tau_S \subset T$, *i.e.*, the number of latents in the diffusion chain used for training and the number of timesteps used for sampling an image from the recovered $\hat{x}_T$ were equal. We investigate if sampling with $\tau_S = T = 1000$ results in images with a better perceptual quality for the baseline. We display the recovered images for LSUN Bedrooms in Fig. 3.14. The length of diffusion chain during training time is $\tau_S = 10$. We note that using more sampling steps does not result in inverted images that are closer to the original image. In some cases, samples generated with more sampling steps have some additional artifacts that are not present in the original image.

3.8.2 Comparison of exact vs inexact gradients for backward pass of DEQ-DDIM

The choice of gradient calculation for the backward pass of DEQ affects both the training stability and convergence of DEQ-DDIM. Here, we compare the performance of the exact gradients and inexact gradients. Computing the inverse of Jacobian in Eq. (3.15) is difficult because the Jacobian can be

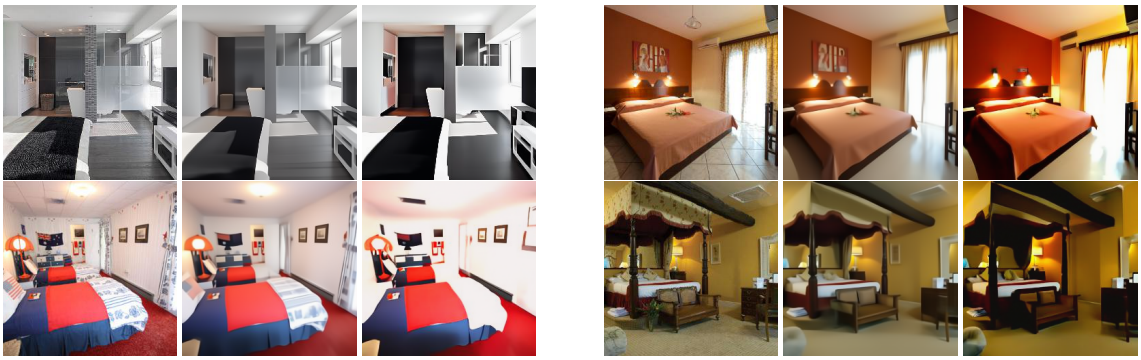


Figure 3.14: Model inversion on LSUN Bedrooms (Baseline): Using more sampling steps does not generate images that are closer to the ground truth image. Each triplet has the original image (left), images sampled with $T = 10$ (middle) and $T = 1000$ (right). The length of diffusion chain at training time was 10.

prohibitively large. We follow Bai et al. [2019] to compute exact gradients using the following linear system

$$\left(J_{g_\theta}^{-1}\big|_{\mathbf{x}^*_{0:T}}\right) \boldsymbol{v}^\top + \left(\frac{\partial \mathcal{L}}{\partial \mathbf{x}^*_{0:T}}\right)^\top = 0 \tag{3.19}$$

We use Broyden's method [Broyden, 1965] to efficiently solve for $\boldsymbol{v}^\top$ in this linear system. We compare it againt inexact gradients *i.e.,* Jacobian free gradient used in Algorithm 2. We observe that training DEQ-DDIM with exact gradients becomes increasingly unstable as the training proceeds, especially for larger learning rates like 0.005. However, we can converge faster with larger learning rates like 0.01 with inexact gradients.

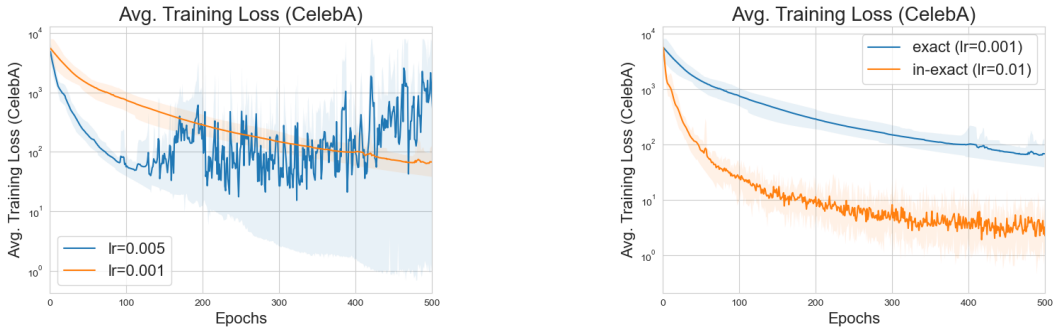


Figure 3.15: Training DEQ-DDIM becomes increasingly unstable with exact gradients for larger learning rates like 0.005 (left, blue curve). However, it is possible to train DEQ-DDIM with exact gradients for smaller learning rate like 0.001 (left, orange curve). However, inexact gradients (right, orange curve) converge faster than exact gradients (right, blue curve) on larger learning rates like 0.01. We report results on 10 runs for CelebA dataset with diffusion chains of length 20.

3.9 Additional Qualitative Results for Model Inversion

We provide additional qualitative results of model inversion with DDIM for CIFAR-10 in Figure 3.16, CelebA in Figure 3.17, LSUN Churches in Figure 3.19, and LSUN Bedrooms in Figure 3.18.

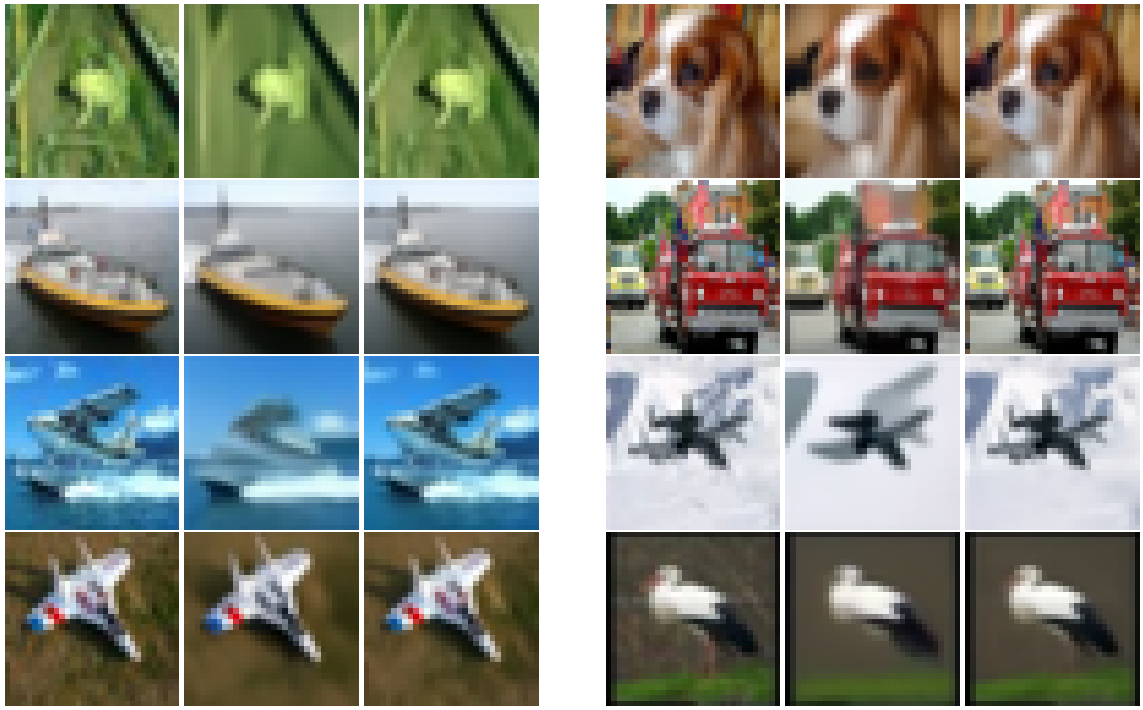


Figure 3.16: Model inversion on CIFAR10. Each triplet has the original image (left), DDIM's inversion (middle), and DEQ's inversion (right). The number of sampling steps for all the images is $T = 100$.

3.10 Discussion

In this chapter, we propose an approach to elegantly unify diffusion models and deep equilibrium (DEQ) models. We model the entire sampling chain of the denoising diffusion implicit model (DDIM) as a joint, multivariate (deep) equilibrium model. This setup replaces the traditional sequential sampling process with a parallel one, enabling us to enjoy the speedup obtained from multiple GPUs. In addition, we can leverage inexact gradients to optimize the entire sampling chain quickly, which results in significant gains in model inversion. We demonstrate the advantages of this approach on 1) single-shot image generation, where we were able to obtain FID scores on par with or slightly better than those of DDIM; and 2) model inversion, where we achieved much faster convergence. We also propose an easy way to extend DEQ formulation for deterministic DDIM to its stochastic variants.

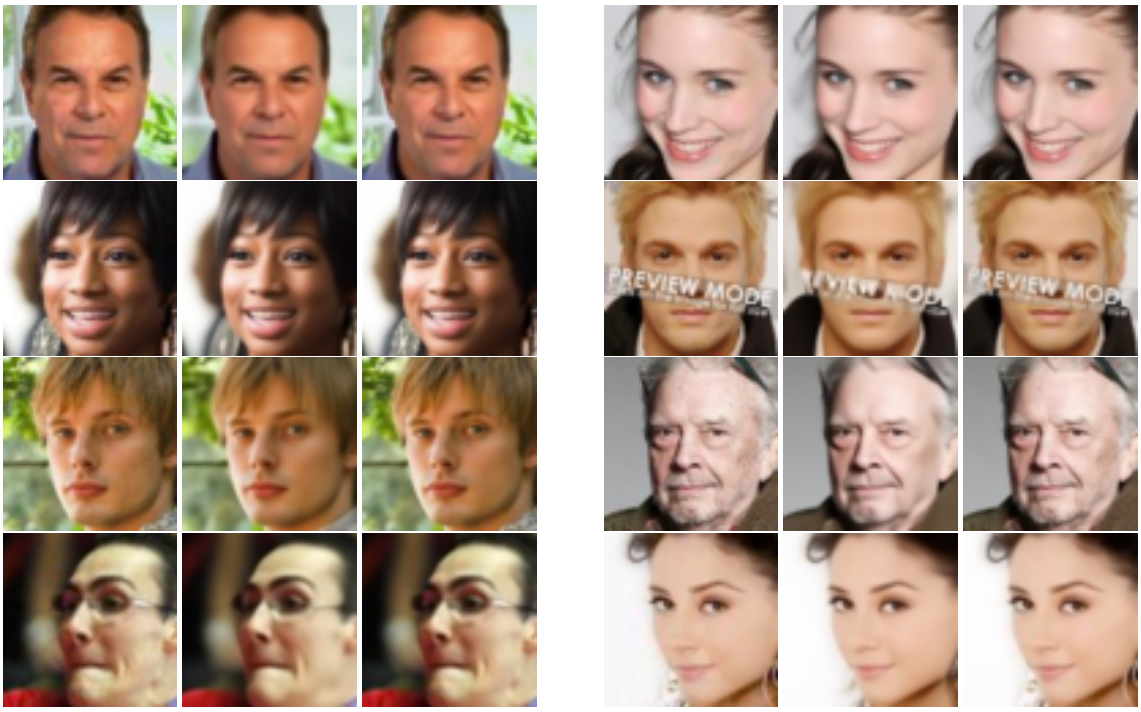


Figure 3.17: Model inversion on CelebA. Finer details like hair, background, and texture of skin are better captured by DEQ. Each triplet has the original image (left), DDIM's inversion (middle), and DEQ's inversion (right). The number of sampling steps for all the generated images is $T = 10$.

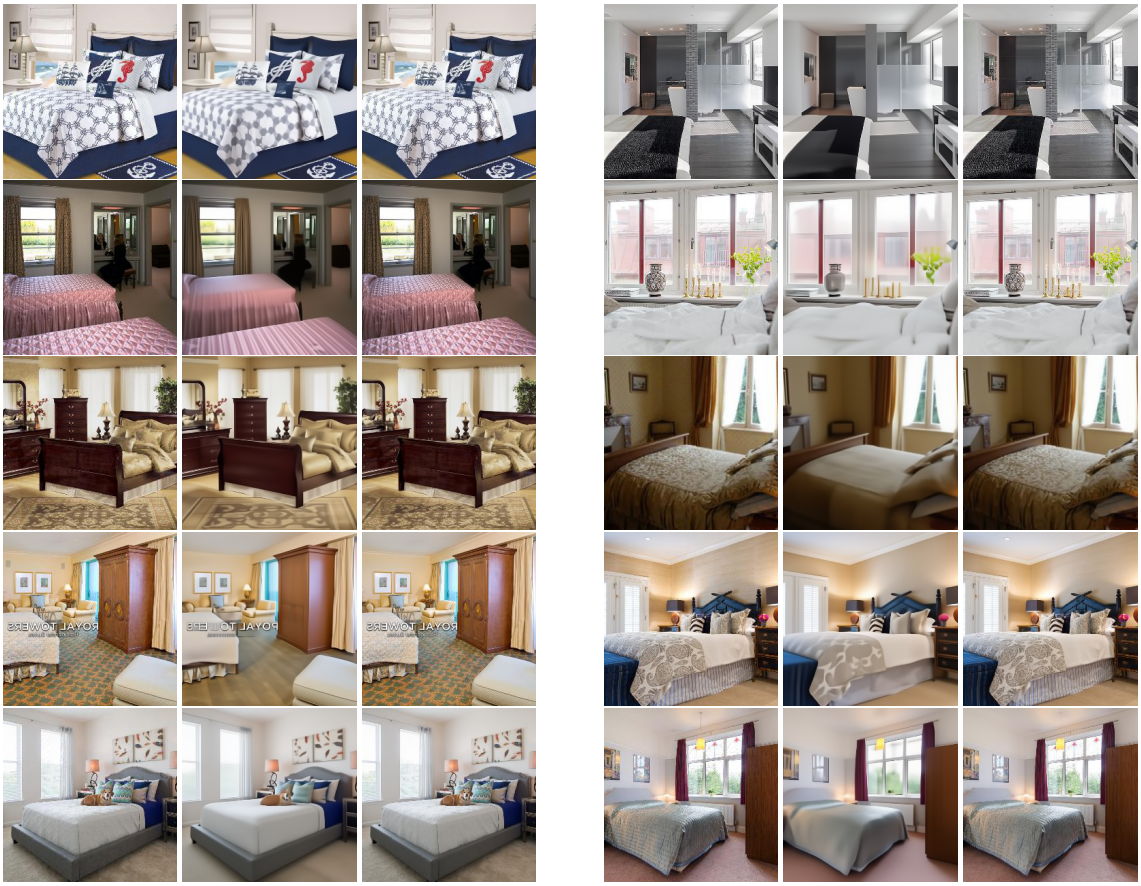


Figure 3.18: Model inversion on LSUN Bedrooms. Each triplet has the original image (left), DDIM's inversion (middle), and DEQ-DDIM's inversion (right). The number of sampling steps for all the generated images is $T = 10$.

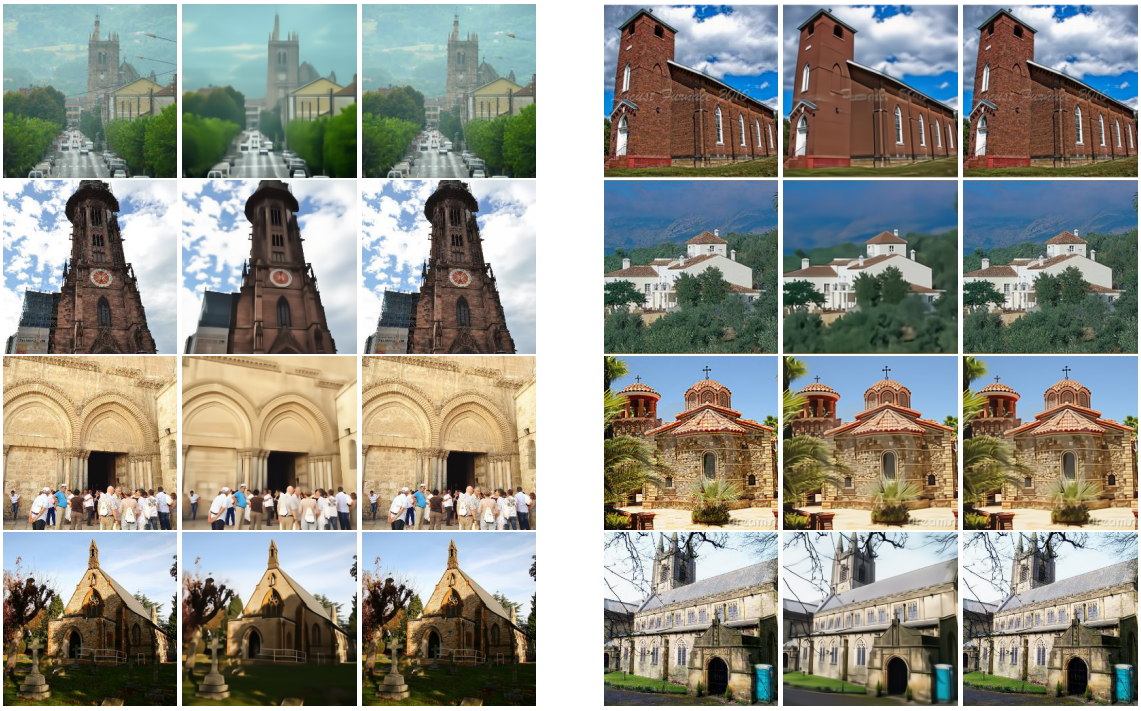


Figure 3.19: Model inversion on LSUN Outdoor Churches. Each triplet has the original image (left), DDIM's inversion (middle), and DEQ-DDIM's inversion (right). The number of sampling steps for all the generated images is $T = 10$.

Figure 3.20: Visualization of model inversion on CIFAR10. The first column is the original image and the last column is the final generated image after 1000 steps for the baseline, and 500 steps for DEQ-DDIM. The 6 columns in between contain images sampled from $\mathbf{x}_T$ after n training updates where n is 0 (initialization), 1, 50, 100, 150, and 200 for DEQ-DDIM (first row) and baseline (second row).

Figure 3.21: Visualization of model inversion on LSUN Bedrooms. The first column is the original image and the last column is the final generated image after 2000 steps for the baseline, and 500 steps for DEQ-DDIM. The 6 columns in between contain images sampled from $\mathbf{x}_T$ after n training updates where n is 0 (initialization), 1, 50, 100, 150, and 200 for DEQ-DDIM (first row) and baseline (second row).

Figure 3.22: Visualization of model inversion on LSUN Churches. The first column is the original image and the last column is the final generated image after 2000 steps for the baseline, and 500 steps for DEQ-DDIM. The 6 columns in between contain images sampled from $\mathbf{x}_T$ after n training updates where n is 0 (initialization), 1, 50, 100, 150, and 200 for DEQ-DDIM (first row) and baseline (second row).

Chapter 4

One-Step Diffusion Distillation with Deep Equilibrium Models

In the previous chapter, we leveraged Deep Equilibrium models (DEQs) for parallel sampling of diffusion models, as well as efficient differentiation through the diffusion sampling chain. Although parallel sampling indeed helps with faster sampling, this method is completely training-free. In this chapter, we will instead look at another approach for faster sampling of diffusion models, namely distillation, which requires training another model but achieves much faster sampling speeds compared to training-free fast samplers. Inspired by prior work in distillation of deep neural networks [Hinton et al., 2015] as well as in the distillation of diffusion models [Luhman and Luhman, 2021], we propose a method for offline distillation of diffusion models that can directly predict an image from the initial Gaussian noise. Of particular importance to our approach is to leverage a new DEQ model as the distilled architecture: the Generative Equilibrium Transformer (GET) which we introduce in Sec. 4.2. Our method achieves superior performance compared to prior approaches for one-step image generation such as Salimans and Ho [2022], Luhman and Luhman [2021], Meng et al. [2022] on comparable training budgets. This chapter is based on Geng et al. [2023].

4.1 Preliminaries

In this section, we provide an overview of continuous-time diffusion models and some techniques to distill these models. We refer the reader to Sec. 2.1 and Sec. 2.2.1for a detailed overview of DEQs.

4.1.1 Distillation Techniques for Diffusion Models

Distillation methods draw inspiration from knowledge distillation [Hinton et al., 2015] and are perhaps the most widespread training-based approach to accelerate the diffusion sampling process. In diffusion distillation, a pretrained diffusion model (DM), which requires hundreds to thousands of model evaluations to generate samples, acts as a teacher. A student model is trained to match the teacher model's sample quality, enabling it to generate high-quality samples in a few steps.

There are two main lines of work in this area. The first category involves trajectory matching, where the student learns to match points on the teacher's sampling trajectory. Methods in this category include offline distillation [Luhman and Luhman, 2021, Zheng et al., 2022], which requires an offline synthetic dataset generated by sampling from a pre-trained DM to distill a teacher model into a few-step student model; progressive distillation [Salimans and Ho, 2022, Meng et al., 2022], and TRACT [Berthelot et al., 2023], which require multiple training passes or offline datasets to achieve the same goal; and BOOT [Gu et al., 2023], Consistency Distillation (CD)[Song et al., 2023], and Imagine-Flash [Kohler et al., 2024], which minimize the difference between student predictions at carefully selected points on the sampling trajectory.

The second category minimizes the probabilistic divergence between the data and model distributions, *i.e.,* distribution matching [Poole et al., 2022, Wang et al., 2024b, Luo et al., 2024, Yin et al., 2023, Zhou et al., 2024b]. These methods [Luo et al., 2024, Yin et al., 2023, Nguyen and Tran, 2023, Sauer et al., 2023, Kohler et al., 2024, Xu et al., 2023b, Lin et al., 2024, Zhou et al., 2024a] use score distillation or adversarial loss to distill an expensive teacher model into an efficient student model. However, they can be challenging to train in a stable manner because of the alternating updating schemes from either adversarial or score distillation. Some of these methods, such as DreamFusion [Poole et al., 2022] and ProlificDreamer [Wang et al., 2024b] are used for 3D object generation.

This work falls into the first category, where we first create an offline dataset of noise-image pairs from a pre-trained teacher diffusion model and then train a student model on these pairs to predict image directly from noise.

4.2 Generative Equilibrium Transformer (GET): An Architectural Overview

We introduce the Generative Equilibrium Transformer (GET), a Deep Equilibrium (DEQ) vision transformer designed to distill diffusion models into generative models that are capable of rapidly sampling images using only a single model evaluation. Our approach builds upon the key components and best practices of the classic transformer [Vaswani et al., 2017a], the Vision transformer (ViT) [Dosovitskiy et al., 2021], and the Diffusion transformer (DiT) [Peebles and Xie, 2022]. We will now describe each component of the GET in detail.

GET. Generative Equilibrium Transformer (GET) directly maps Gaussian noises $\mathbf{e}$ and optional class labels $\mathbf{y}$ to images $\tilde{\mathbf{x}}$. The major components of GET include an injection transformer (InjectionT, Eq. (4.2)) and an equilibrium transformer (EquilibriumT, Eq. (4.3)). The InjectionT transforms tokenized noise embedding $\mathbf{h}$ to an intermediate representation $\mathbf{n}$ that serves as the input injection for the equilibrium transformer. The EquilibriumT, which is the equilibrium layer, solves for the fixed point $\mathbf{z}_\star$ by taking in the noise injection $\mathbf{n}$ and an optional class embedding $\mathbf{c}$. Finally, this fixed point $\mathbf{z}_\star$ is decoded and rearranged to generate an image sample $\tilde{\mathbf{x}}$ (Eq. (4.4)). Figure 4.1 provides an overview of the GET architecture. Note that because we are directly distilling the entire generative process, there is

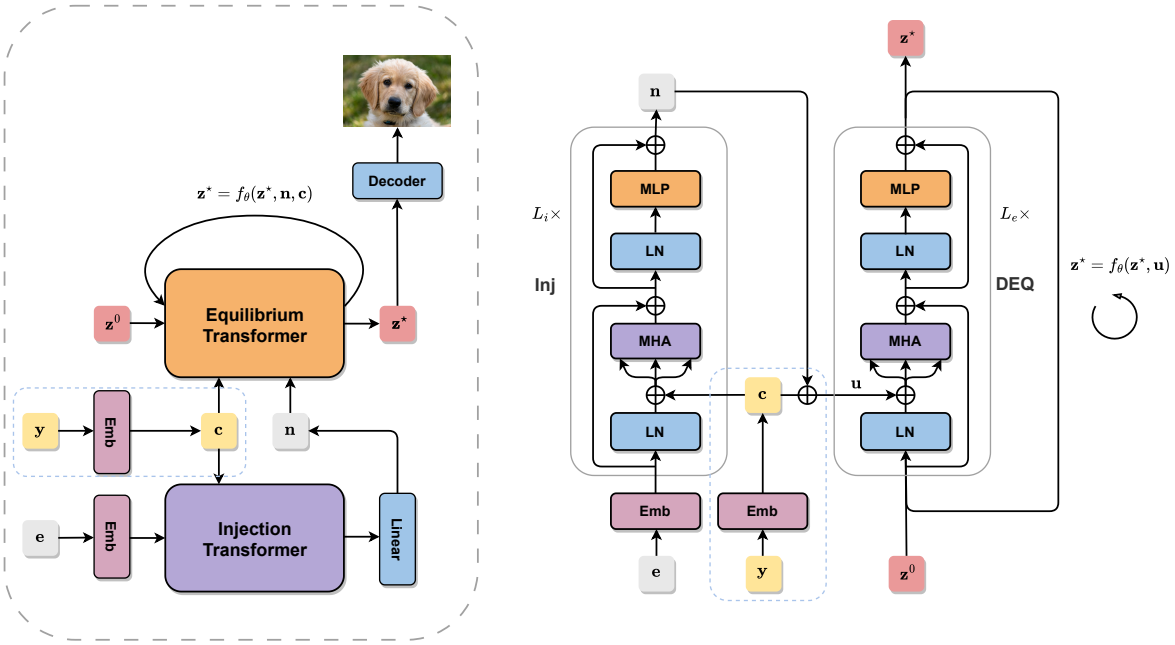


Figure 4.1: **Generative Equilibrium Transformer (GET).** (*Left*) GET consists of two major components: Injection transformer and Equilibrium transformer. The Injection transformer transforms noise embeddings into an input injection for the Equilibrium transformer. The Equilibrium transformer is the equilibrium layer that takes in noise input injection and an optional class embedding and solves for the fixed point. (*Right*) Details of transformer blocks in the Injection transformer (**Inj**) and Equilibrium transformer (**DEQ**), respectively. Blue dotted boxes denote optional class label inputs.

no need for a time embedding t as is common in standard diffusion models.

$$\mathbf{h}, \mathbf{c} = \text{Emb}(\mathbf{e}),\ \text{Emb}(\mathbf{y});\ \text{if } \mathbf{y} \notin \emptyset \tag{4.1}$$

$$\mathbf{n} = \text{InjectionT}(\mathbf{h}, \mathbf{c}) \tag{4.2}$$

$$\mathbf{z}_\star = \text{EquilibriumT}(\mathbf{z}_\star, \mathbf{n}, \mathbf{c}) \tag{4.3}$$

$$\tilde{\mathbf{x}} = \text{Decoder}(\mathbf{z}_\star) \tag{4.4}$$

Noise Embedding. GET first converts an input noise $\mathbf{e} \in \mathbb{R}^{H \times W \times C}$ into a sequence of 2D patches $\mathbf{p} \in \mathbb{R}^{N \times (P^2 \cdot C)}$, where C is the number of channels, P is the size of patch, H and W denotes height and width of the original image, and $N = HW/P^2$ is the resulting number of patches. Let $D = P^2 \cdot C$ denote the width of the network. We follow ViT to use a linear layer to project the N patches to D dimensional embedding. We add standard sinusoidal position encoding [Vaswani et al., 2017a] to produce the noise embedding $\mathbf{h}$. Position encoding plays a crucial role in capturing the spatial structure of patches by encoding their relative positional information.

InjectionT & EquilibriumT. Both InjectionT and EquilibriumT are composed of a sequence of Transformer blocks. InjectionT is called only once to produce the noise injection $\mathbf{n}$, while EquilibriumT defines the function $\mathbf{f}$ of the implicit layer $\mathbf{z}^\star = \mathbf{f}(\mathbf{z}^\star, \mathbf{n}, \mathbf{c})$ that is called multiple times—creating a weight-tied computational graph—until convergence. A linear layer is added at the end of InjectionT to compute the noise injection $\mathbf{n}_l \in \mathbb{R}^{N \times 3D}$, $l \in [L_e]$, for each of the L_e GET blocks in EquilibriumT. For convenience, we overload the notation $\mathbf{n}_l$ and $\mathbf{n}$, in the following paragraphs.

Transformer Block. GET utilizes a near-identical block design for the noise injection (InjectionT) and the equilibrium layer (EquilibriumT), differing only at the injection interface. Specifically, the transformer block is built upon the standard Pre-LN transformer block [Xiong et al., 2020, Dosovitskiy et al., 2021, Peebles and Xie, 2022], as shown below:

$$\mathbf{z} = \mathbf{z} + \text{Attention}(\text{LN}(\mathbf{z}), \mathbf{u})$$
$$\mathbf{z} = \mathbf{z} + \text{FFN}(\text{LN}(\mathbf{z}))$$

Here, $\mathbf{z} \in \mathbb{R}^{N \times D}$ represents the latent token, $\mathbf{u} \in \mathbb{R}^{N \times 3D}$ is the input injection, LN, FFN, and Attention stand for Layer Normalization [Ba et al., 2016a], a 2-layer Feed-Forward Network with a hidden dimension of size $D \times E$, and an attention [Vaswani et al., 2017a] layer with an injection interface, respectively.

For blocks in the injection transformer, $\mathbf{u}$ is equal to the class embedding token $\mathbf{c} \in \mathbb{R}^{1 \times 3D}$ for conditional image generation, *i.e.*, , $\mathbf{u} = \mathbf{c}$ for conditional models, and $\mathbf{u} = \mathbf{0}$ otherwise. In contrast, for blocks in the equilibrium transformer, $\mathbf{u}$ is the broadcast sum of noise injection $\mathbf{n} \in \mathbb{R}^{N \times 3D}$ and class embedding token $\mathbf{c} \in \mathbb{R}^{1 \times 3D}$, i.e., $\mathbf{u} = \mathbf{n} + \mathbf{c}$ for conditional models and $\mathbf{u} = \mathbf{n}$ otherwise.

We modify the standard transformer attention layer to incorporate an additive injection interface before the query $\mathbf{q} \in \mathbb{R}^{N \times D}$, key $\mathbf{k} \in \mathbb{R}^{N \times D}$, and value $\mathbf{v} \in \mathbb{R}^{N \times D}$,

$$\mathbf{q}, \mathbf{k}, \mathbf{v} = \mathbf{z}\mathbf{W}_i + \mathbf{u}$$
$$\mathbf{z} = \text{MHA}(\mathbf{q}, \mathbf{k}, \mathbf{v})$$
$$\mathbf{z} = \mathbf{z}\mathbf{W}_o$$

where $\mathbf{W}_i \in \mathbb{R}^{D\times 3D}$, $\mathbf{W}_o \in \mathbb{R}^{D\times D}$. The injection interface enables interactions between the latent tokens and the input injection in the multi-head dot-product attention (MHA) operation,

$$\mathbf{q}\mathbf{k}^\top = (\mathbf{z}\mathbf{W}_q + \mathbf{u}_q)(\mathbf{z}\mathbf{W}_k + \mathbf{u}_k)^\top = \mathbf{z}\mathbf{W}_q\mathbf{W}_k^\top\mathbf{z}^\top + \mathbf{z}\mathbf{W}_q\mathbf{u}_k^\top + \mathbf{u}_q\mathbf{W}_k^\top\mathbf{z}^\top + \mathbf{u}_q^\top\mathbf{u}_k, \tag{4.5}$$

where $\mathbf{W}_q, \mathbf{W}_k \in \mathbb{R}^{D\times D}$ are slices from $\mathbf{W}_i$, and $\mathbf{u}_q, \mathbf{u}_k \in \mathbb{R}^{N\times D}$ are slices from $\mathbf{u}$. This scheme adds no more computational cost compared to the standard MHA operation, yet it achieves a similar effect as cross-attention and offers good stability during training.

Image Decoder. The output of the GET-DEQ is first normalized with Layer Normalization Ba et al. [2016a]. The normalized output is then passed through another linear layer to generate patches $\bar{\mathbf{p}} \in \mathbb{R}^{N\times D}$. The resulting patches $\bar{\mathbf{p}}$ are rearranged back to the resolution of the input noise $\mathbf{e}$ to produce the image sample $\tilde{\mathbf{x}} \in \mathbb{R}^{H\times W\times C}$. Thus, the decoder maps the features back to the image space.

4.3 Details of Experimental Setup and Methodology

4.3.1 Data Collection

For unconditional image generation on CIFAR-10 [Krizhevsky, 2009], we generate 1M noise/image pairs from the pretrained unconditional EDM [Karras et al., 2022]. This dataset is denoted as `EDM-Uncond-1M`. As in EDM, we sample 1M images using Heun's second-order deterministic solver [Ascher and Petzold, 1998]. Generating a batch of images takes 18 steps or 35 NFEs (Number of Function Evaluations). Overall, this dataset takes up around 29 GB of disk space. The entire process of data generation takes about *4 hours* on 4 NVIDIA A6000 GPUs using PyTorch [Paszke et al., 2019] Distributed Data Parallel (DDP) and a batch size of 128 per GPU. In addition to unconditional image generation, we sample 1M noise-label/image pairs from the conditional VP-EDM [Karras et al., 2022] using the same settings. This dataset is denoted as `EDM-Cond-1M`. Both the datasets will be released for future studies.

4.3.2 Details of Offline Distillation

We distill a pretrained EDM [Karras et al., 2022] into ViTs and GETs by training on a dataset $\mathcal{D}$ with noise/image pairs sampled from the teacher diffusion model using a reconstruction loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathbf{e},\mathbf{x}\sim\mathcal{D}}\|\mathbf{x} - G_\theta(\mathbf{e})\|_1$$

where $\mathbf{x}$ is the desired ground truth image, $G_\theta(\cdot)$ is unconditional ViT/GET with parameters θ, and $\mathbf{e}$ is the initial Gaussian noise. To train a class-conditional GET, we also use class labels $\mathbf{y}$ in addition to noise/image pairs:

$$\mathcal{L}(\theta) = \mathbb{E}_{\mathbf{e},\mathbf{y},\mathbf{x}\sim\mathcal{D}}\|\mathbf{x} - G^c_\theta(\mathbf{e},\mathbf{y})\|_1$$

where $G^c_\theta(\cdot)$ is class-conditional ViT/GET with parameters θ. As is the standard practice, we also maintain an exponential moving average (EMA) of weights of the model, which in turn is used at inference time for sampling.

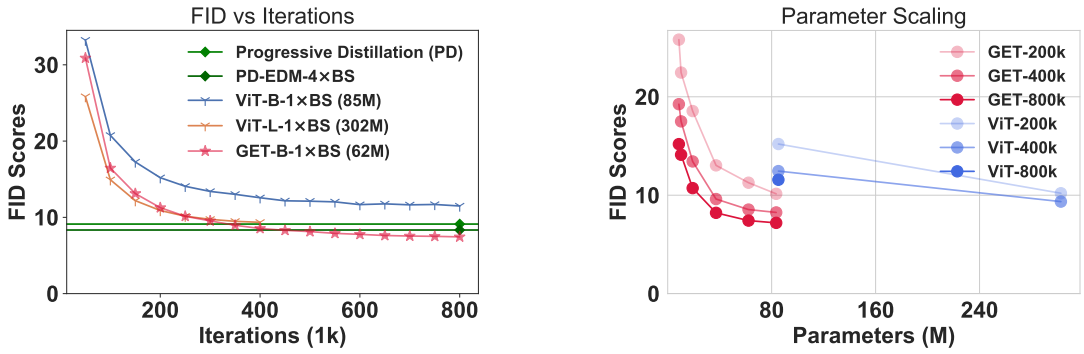


Figure 4.2: **Data and Parameter Efficiency of GET**:(a) (*Left*) GET outperforms PD and a 5× larger ViT in fewer iterations, yielding better FID scores. Additionally, longer training times lead to improved FID scores. (b) (*Right*) Smaller GETs can achieve better FID scores than larger ViTs, demonstrating DEQ's parameter efficiency. Each curve in this plot connects models of different sizes within the same model family at identical training iterations, as indicated by the numbers after the model names in the legend.

4.3.3 Training Details and Evaluation Metrics

We use AdamW [Loshchilov and Hutter, 2017] optimizer with a learning rate of $1e-4$, a batch size of 128 (denoted as $1\times$BS), and 800k training iterations, which are identical to Progressive Distillation (PD) [Salimans and Ho, 2022]. For conditional models, we adopt a batch size of 256 ($2\times$BS). No warm-up, weight decay, or learning rate decay is applied. We convert input noise to patches of size 2×2. We use 6 steps of fixed point iterations in the forward pass of GET-DEQ and differentiate through it. For the $\mathcal{O}(1)$ memory mode, we utilize gradient checkpoint [Chen et al., 2016] for DEQ's computational graph. We measure image sample quality for all our experiments via Frechet inception distance (FID) [Heusel et al., 2017] of 50k samples. We also report Inception Score (IS) [Salimans et al., 2016] computed on 50k images. We include other relevant metrics such as FLOPs, training speed, memory, sampling speed, and the Number of Function Evaluations (NFEs), wherever necessary.

4.4 Experimental Results

4.4.1 Data and Parameter Efficiency of GET

Models trained with offline distillation require high data efficiency to make optimal use of limited training data sampled from pretrained diffusion models. DEQs have a natural regularization mechanism due to weight-tying, which allows us to efficiently fit significantly compact data-efficient models even in limited data settings. In Figure 4.2(a), we observe that even with a fixed and limited offline data budget of 1M samples, GET achieves parity with online distilled EDM [Karras et al., 2022, Salimans and Ho, 2022, Song et al., 2023] while using only half the number of training iterations. For comparison, PD, TRACT, and CM use a much larger data budget of 96M, 256M, and 409.6M samples, respectively. Moreover, GET is able to match the FID score of a $5\times$ large ViT, suggesting substantial parameter efficiency.

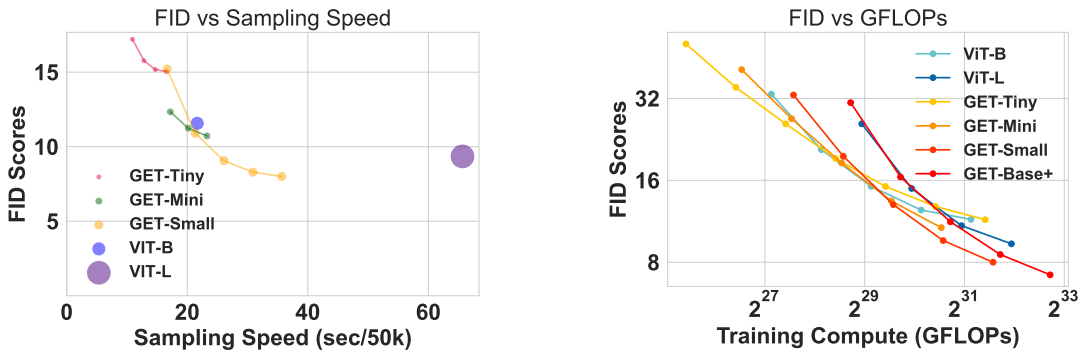


Figure 4.3: (a) (*Left*) **Sampling speed of GET**: GET can sample faster than large ViTs, while achieving better FID scores. The size of each individual circle is proportional to the model size. For GETs, we vary the number of iterations in the Equilibrium transformer (2 to 6 iterations). The trends indicate that GETs can improve their FID scores by using more compute. (b) (*Right*) **Compute efficiency of GET**: Larger GET models use training compute more efficiently. For a given GET, the training budget is calculated from training iterations. Refer to Table 4.2 for the exact size of GET models.

4.4.2 Sampling speed of GET

Figure 4.3(a) illustrates the sampling speed of both ViT and GET. A smaller GET (37.2M) can achieve faster sampling than a larger ViT (302.6M) while achieving lower FID scores. GET can also improve its FID score by increasing its test-time iterations in the Equilibrium transformer at the cost of speed. Note that despite this trade-off, GET still outperforms larger ViT in terms of both sampling speed and sample quality.

4.4.3 Results of One-step Image Generation with GET

We provide results for unconditional and class-conditional image generation on CIFAR-10 in Table 4.1 and Table 4.3, respectively. GET outperforms a much more complex distillation procedure—PD with classifier-free guidance—in class-conditional image generation. GET also outperforms PD and KD in terms of FID score for unconditional image generation. This effectiveness is intriguing, given that our approach for offline distillation is relatively simpler when compared to other state-of-the-art distillation techniques. We have outlined key differences in the experimental setup between our approach and other distillation techniques in Table 4.4.

We also visualize random CIFAR-10 [Krizhevsky, 2009] samples generated by GET for both unconditional and class-conditional cases in Figure 4.4. GET can learn rich semantics and world knowledge from the dataset, as depicted in the images. For instance, GET has learned the symmetric layout of dog faces solely using reconstruction loss in the pixel space, as shown in Figure 4.4(b).

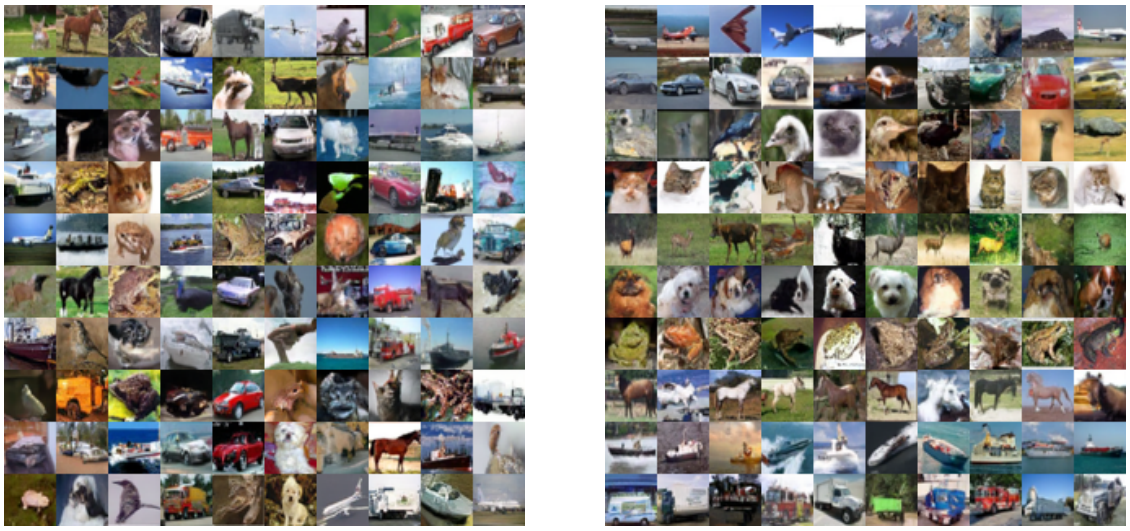


Figure 4.4: Uncurated CIFAR-10 image samples generated by *(Left)* (a) unconditional GET and *(Right)* (b) class-conditional GET. Each row corresponds to a class in CIFAR-10.

4.4.4 Scaling laws of Generative Equilibrium Transformer

Why Scaling laws for Implicit Models? As a prospective study, we preliminarily investigated the scaling properties of Deep Equilibrium models using GET. The scaling law is an attractive property, as it enables us to predict models' performance at extremely large compute based on the performance of tiny models. This predictive capability allows us to select the most efficient model given the constraints of the available training budget [Brown et al., 2020b, Hoffmann et al., 2022, OpenAI, 2023]. Although the scaling law for explicit networks has been extensively studied, its counterpart for implicit models remains largely unexplored. Implicit models are different from explicit models, as they utilize more computation through weight-tying under similar parameters and model designs. Therefore, it is natural to question whether their scaling laws align with those of their explicit counterparts.

Scaling Model Size We conduct extensive experiments to understand the trends in sample quality as we scaled the size of the GET model. Table 4.2 provides a summary of our findings on single-step unconditional image generation. We find that even small GET models with 10-20M parameters can generate images with sample quality on par with NAS-derived AutoGAN [Gong et al., 2019]. In general, sample quality improves with the increase in model size.

Scaling Training Compute Our experimental results support the findings of Peebles and Xie [2022] for explicit models (DiT) and extend them to implicit models. Specifically, for both implicit and explicit models, larger models are better at exploiting training FLOPs. Figure 4.3 shows that larger models eventually outperform smaller models when training compute increases. For implicit models, there also exists a "sweet spot" in terms of model size under a fixed training budget, *e.g.*, GET-Small outperforms both smaller and larger GETs at 2^{31} training GFLOPs. Furthermore, because of the internal dynamics of implicit models, they can match a much larger explicit model in terms of performance while using fewer parameters. This underscores the potential of implicit models as candidates for

compute-optimal models [Hoffmann et al., 2022] with substantially better parameter efficiency. For example, at 2^{31} training GFLOPs, Figure 4.3(b) suggests that we should choose GET-Small (31.2M) among implicit models for the best performance, which is much more parameter efficient and faster in sampling than the explicit model with the best performance, ViT-L (302M), in this training budget.

Table 4.1: Generative performance on unconditional CIFAR-10.

Method	NFE ↓	FID ↓	IS ↑
Diffusion Models			
DDPM [Ho et al., 2020]	1000	3.17	9.46
Score SDE [Song et al., 2020b]	2000	2.2	9.89
DDIM [Song et al., 2020a]	10	13.36	-
LSGM [Vahdat et al., 2021]	147	2.10	
FastDPM [Kong and Ping, 2021]	10	9.9	
DPM-solver [Lu et al., 2022a]	10	4.7	-
DEIS [Zhang and Chen, 2023]	10	4.17	-
EDM [Karras et al., 2022]	35	2.04	9.84
GANs			
StyleGAN2 [Karras et al., 2020b]	1	8.32	9.18
StyleGAN2 + ADA Karras et al. [2020a]	1	5.33	10.22
StyleGAN-XL [Sauer et al., 2022]	1	1.85	-
Diffusion Distillation			
KD [Luhman and Luhman, 2021]	1	9.36	8.36
PD [Salimans and Ho, 2022]	1	9.12	-
DFNO [Zheng et al., 2022]	1	4.12	-
TRACT-EDM [Berthelot et al., 2023]	1	4.17	-
PD-EDM [Salimans and Ho, 2022, Song et al., 2023]	1	8.34	8.69
CD-EDM (LPIPS) Song et al. [2023]	1	3.55	9.48
Consistency Models			
CT [Song et al., 2023]	1	8.70	8.49
CT [Song et al., 2023]	2	5.83	8.85
Ours			
GET-Base	1	7.42	9.16
GET-Mini (LPIPS)	1	7.20	9.08
GET-Base	1	6.91	9.16

Table 4.2: Generative performance of GETs on unconditional CIFAR-10.

Models	Params	NFE ↓	FID ↓	IS ↑
GET-Tiny	8.6M	1	15.19	8.37
GET-Mini	19.2M	1	10.72	8.69
GET-Small	37.2M	1	8.00	9.03
GET-Base	62.2M	1	7.42	9.16
GET-Base+	83.5M	1	7.19	9.09
More Training				
GET-Tiny-4×Iters	8.6M	1	11.47	8.64
GET-Base-2×BS	62.2M	1	6.91	9.16

4.4.5 Ablation Study

4.4.5.1 Benchmarking GET against ViT

Table 4.5 summarizes key metrics for unconditional image generation for ViT and GET. Our experiments indicate that a smaller GET (19.2M) can generate higher-quality images faster than a much larger ViT (85.2M) while utilizing less training memory and fewer FLOPs. GET also demonstrates substantial parameter efficiency over ViTs as shown in Figure 4.2(b) where smaller GETs achieve better FID scores than larger ViTs.

4.4.5.2 Comparison of Number of Function Evaluations of Teacher Model

Offline distillation requires significantly fewer number of function evaluations (NFEs) for the teacher network compared to other online distillation methods. In the experimental setup used in this paper, GET requires 35M overall NFEs for the teacher model, as we train on 1M data samples and use 35 NFEs to generate each data sample with EDM. In contrast, progressive distillation requires 179M NFEs to get 1-step distilled student model. Using the hyperparameters reported in Salimans and Ho [2022], PD with the DDIM model requires 13 passes of distillation. The initial 12 passes use 50K iterations, and the last pass uses 100K iterations. Each step of PD uses 2 teacher model NFEs. Thus, the overall number of NFEs from teacher models can be evaluated as 2×128 (batch size) $\times$ (12 passes $\times$50K + 100K) = 179M samples. The number of NFEs of the teacher model increases to 1.433B if we assume that each of 8 TPUs use a batch size of 128. Consistency distillation [Song et al., 2023] requires 409.6 M teacher model NFEs (512 batch size $\times$ 800K iterations = 409.6M). In addition, perceptual loss requires *double* NFEs as the teacher model.

4.4.5.3 Design choices for Class Conditioning

As both GET and ViT share the same class injection interface, we perform an ablation study on ViT. We consider two types of input injection schemes for class labels: 1) additive injection scheme 2) injection

Table 4.3: Generative performance on class-conditional CIFAR-10. w indicates the level of classifier guidance.

Method	NFE ↓	FID ↓	IS ↑
GANs			
BigGAN [Brock et al., 2018]	1	14.73	9.22
StyleGAN2-ADA [Karras et al., 2020a]	1	2.42	10.14
Diffusion Models			
DDIM [Meng et al., 2022]	2048	2.73	9.66
EDM [Karras et al., 2022]	35	1.79	-
NCSN++-G [Chao et al., 2022]		2000	2.25
EDM-G++ [Kim et al., 2022]	35	1.64	-
Diffusion Distillation			
Guided Distillation ($w = 0$) [Meng et al., 2022]	1	8.34	8.63
Guided Distillation ($w = 0.3$) [Meng et al., 2022]	1	7.34	8.90
Guided Distillation ($w = 1$) [Meng et al., 2022]	1	8.62	9.21
Guided Distillation ($w = 2$) [Meng et al., 2022]	1	13.23	9.23
Ours			
GET-Base	1	6.25	9.40

with adaptive layer normalization (AdaLN-Zero) as used in DiT [Peebles and Xie, 2022]. These results are summarized in Table 4.6. Despite using almost the same parameters as unconditional ViT-B, the class-conditional ViT-B using additive injection interface has an FID of 12.43 at 200k, while the ViT-B w/ AdaLN-Zero class embedding [Peebles and Xie, 2022] set up an FID of 17.19 at 200k iterations. Another surprising observation is that ViT-B w/ AdaLN-Zero class embedding performs worse than unconditional ViT in terms of FID score. Therefore, it seems that adaptive layer normalization might not be useful when used only with class embedding.

4.5 Discussion

In this chapter, we proposed a simple yet effective approach to distill diffusion models into generative models that are capable of sampling with just a single model evaluation. Our method involves training Generative Equilibrium Transformer (GET), a DEQ-based architecture derived from Vision Transformer (ViT), directly on noise/image pairs generated from a pretrained diffusion model. This eliminates the need for diffusion sampling trajectory information, as well as temporal embeddings that are needed in traditional diffusion models. GET demonstrates superior performance over more complex online distillation techniques such as progressive distillation [Salimans and Ho, 2022, Meng et al., 2022] in both class-conditional and unconditional settings. In addition, a small GET can generate higher quality images than a 5× larger ViT, sampling faster while using less training memory and fewer compute FLOPs, demonstrating its effectiveness.

Table 4.4: Comparison of relevant training and hyperparameter settings for common distillation techniques. GET requires neither multiple training phases nor any trajectory information. We only count the number of models involved in the forward pass and exclude EMA in #Models.
† indicates offline distillation techniques.
▲For CD, we count the VGG network used in the perceptual loss [Zhang et al., 2018].
BS indicates batch size.

Model	FID ↓	IS ↑	BS	Training Phases	#Models	Trajectory	Teacher
KD [Luhman and Luhman, 2021]†	9.36	-	4×	1	1	✗	DDIM
PD [Salimans and Ho, 2022]	9.12	-	1×	$\log_2(T)$	2	✓	DDIM
DFNO [Zheng et al., 2022]†	4.12	-	2×	1	1	✓	DDIM
TRACT [Berthelot et al., 2023]	14.40	-	2×	1	1	✓	DDIM
TRACT [Berthelot et al., 2023]	4.17	-	2×	2	1	✓	EDM
PD-EDM [Salimans and Ho, 2022, Song et al., 2023]	8.34	8.69	4×	$\log_2(T)$	2	✓	EDM
CD▲ [Song et al., 2023]	3.55	9.48	4×	1	3	✓	EDM
Ours†	7.42	9.16	1×	1	1	✗	EDM
Ours†	6.91	9.16	2×	1	1	✗	EDM
Guided Distillation [Meng et al., 2022]	7.34	8.90	4×	$\log_2(T)+1$	3	✓	DDIM
Ours†	6.25	9.40	2×	1	1	✗	EDM

Table 4.5: Benchmarking GET against ViT on unconditional image generation on CIFAR-10. For the first time, implicit models for generative tasks *strictly* surpass explicit models in all metrics. Results are benchmarked on 4 A6000 GPUs using a batch size of 128, 800k iterations, and PyTorch [Paszke et al., 2019] distributed training protocol. Training Mem stands for training memory consumed per GPU. $\mathcal{O}(1)$ symbolizes the $\mathcal{O}(1)$ training memory mode, which differs only in training memory and speed.

Model	FID↓	IS↑	Params↓	FLOPs↓	Training Mem↓	Training Speed↑
ViT-Base	11.49	8.61	85.2M	23.0G	10.1GB	4.83 iter/sec
GET-Mini	10.72	8.69	19.2M	15.2G	9.2GB	5.79 iter/sec
GET-Mini-$\mathcal{O}(1)$	-	-	-	-	5.0GB	4.53 iter/sec

4.6 Addition Appendices

4.6.1 Model Configuration

We set the EMA momentum to 0.9999 for all the models.

The configuration of different GET architectures are listed in Table 4.7. Here, L_i and L_e denote the number of transformer blocks in the Injection transformer and Equilibrium transformer, respectively. D denotes the width of the network. E corresponds to the expanding factor of the FFN layer in the Equilibrium transformer, which results in the hidden dimension of $E \times D$. For the injection transformer, we always adopt an expanding factor of 4.

We have listed relevant model configuration details of ViT in Table 4.8. The model configurations are adopted from DiT [Peebles and Xie, 2022], whose effectiveness was tested for learning diffusion models. In this table, L denotes the number of transformer blocks in ViT. D stands for the width of the

Table 4.6: Ablation on design choices for class conditioning.

Model	FID↓	IS↑	Params↓
ViT-Uncond	15.20	8.27	85.2M
ViT-AdaLN-Zero	17.19	8.38	128.9M
ViT-Inj-Interface	12.43	8.69	85.2M

Table 4.7: Details of configuration for GET architectures.

Model	Params	L_i	L_e	D	E
GET-Tiny	8.9M	6	3	256	6
GET-Mini	19.2M	6	3	384	6
GET-Small	37.2M	6	3	512	6
GET-Base	62.2M	1	3	768	12
GET-Base+	83.5M	6	3	768	8

network. We always adopt an expanding factor of 4 following the common practice [Vaswani et al., 2017a, Dosovitskiy et al., 2021, Peebles and Xie, 2022].

Table 4.8: Details of configuration for ViT architectures.

Model	Params	L	D
ViT-B	85.2M	12	768
ViT-L	302.6M	24	1024

4.6.2 Additional Related work

4.6.2.1 Fast Samplers for Diffusion Models

Fast samplers are usually training-free, unlike distillation-based approaches, and use advanced solvers to simulate the diffusion stochastic differential equation (SDE) or ordinary differential equation (ODE) to reduce the number of sampling steps. These methods reduce the discretization error during sampling by analytically solving a part of SDE or ODE [Lu et al., 2022b,c, Xue et al., 2024], by using exponential integrators and higher-order polynomials for a better approximation of the solution [Zhang and Chen, 2022], using higher-order numerical methods [Karras et al., 2022], using a better approximation of noise levels during sampling [Kong and Ping, 2021], correcting predictions at each step of sampling [Zhao et al., 2024] and ensuring that the solution of the ODE lies on a desired manifold [Liu et al., 2022a]. Another orthogonal strategy is the parallel sampling process such as the one covered in Chapter 1, where we solve for the fixed points of the entire sampling trajectory. A drawback of these fast samplers is that the quality of the samples is drastically reduced as the number of sampling steps falls below a threshold such as 10 steps.

Chapter 5

Training-free Linear Image Inverses via Flows

Solving inverse problems without any training involves using a pretrained generative model and making appropriate modifications to the generation process to avoid fine-tuning of the generative model. While recent methods have explored the use of diffusion models, they still require the manual tuning of many hyperparameters for different inverse problems. In this chapter, we propose a training-free method for solving linear inverse problems in non-blind setting by using pretrained unconditional flow models, leveraging the simplicity and efficiency of Flow Matching models [Lipman et al., 2022, Liu et al., 2022b, Albergo et al., 2023], using theoretically-justified weighting schemes, and thereby significantly reducing the amount of manual tuning. In particular, we draw inspiration from two main sources: adopting prior gradient correction methods to the flow regime, and a solver scheme based on conditional Optimal Transport paths. As pretrained diffusion models are widely accessible, we also show how to practically adapt diffusion models for our method. Empirically, our approach requires no problem-specific tuning across an extensive suite of noisy linear inverse problems on high-dimensional datasets, ImageNet-64/128 and AFHQ-256, and we observe that our flow-based method for solving inverse problems improves upon closely-related diffusion-based methods in most settings. This chapter is based on Pokle et al. [2023].

5.1 Introduction

Solving an inverse problem involves recovering a clean signal from noisy measurements generated by a known degradation model. Many interesting image processing tasks can be cast as an inverse problem. Some instances of these problems are super-resolution, inpainting, deblurring, colorization, denoising etc. As we have seen in the previous chapters, diffusion models or score-based generative models [Sohl-Dickstein et al., 2015, Song and Ermon, 2019, Song et al., 2020b, Ho et al., 2020] have emerged as a leading family of generative models for solving inverse problems for images [Saharia et al., 2022b,a, Wang et al., 2022, Chung et al., 2022a, Song et al., 2022, Mardani et al., 2023]. However, sampling with diffusion models is known to be slow, and the quality of generated images is affected

by the curvature of SDE/ODE solution trajectories [Karras et al., 2022]. While Karras et al. [2022] observed ODE sampling for image generation could produce better results, sampling via SDE is still common for solving inverse problems, whereas ODE sampling has been rarely considered, perhaps due to the use of diffusion probability paths.

Continuous Normalizing Flow (CNF) [Chen et al., 2018] trained with Flow Matching [Lipman et al., 2022, Liu et al., 2022b, Albergo et al., 2023] has been recently proposed as a powerful alternative to diffusion models. CNF (hereafter denoted flow model) has the ability to model arbitrary probability paths, and includes diffusion probability paths as a special case. Of particular interest to us are Gaussian probability paths that correspond to optimal transport (OT) displacement [McCann, 1997]. Recent works [Lipman et al., 2022, Albergo et al., 2023, Liu et al., 2022b, Shaul et al., 2023] have shown that these conditional OT probability paths are straighter than diffusion paths, which results in faster training and sampling with these models. Due to these properties, conditional OT flow models are an appealing alternative to diffusion models for solving inverse problems.

In this work, we introduce a training-free method to utilize pretrained unconditional flow models for solving linear inverse problems. Our approach adds a gradient-based adaptation term to the unconditional vector field that takes into account knowledge from the degradation model and converts it to a conditional vector field. Specifically, we introduce an algorithm that extends ΠGDM [Song et al., 2022] gradient adaptation to ODE sampling with an affine Gaussian probability path. Given the wide availability of pretrained diffusion models trained with diffusion probability paths, we also present a way to convert these models to other affine Gaussian probability paths. Empirically, we observe images restored via a conditional OT path exhibit perceptual quality better than that achieved by the model's original diffusion path, as well as recently proposed diffusion approaches such as ΠGDM [Song et al., 2022] and RED-Diff [Mardani et al., 2023], particularly in noisy settings. To summarize, our key contributions are:

- We present a training-free approach to solve linear inverse problems in non-blind setting that can be applied to any continuous-time diffusion or flow model under affine Gaussian probability paths that extends ΠGDM gradient adaptation to this more generic setting.
- We explain the subtleties in converting models between different affine Gaussian probability paths. Specifically, we enable the use of pre-trained continuous-time diffusion models with conditional OT probability paths by an adjusted initialization procedure.
- We demonstrate that images restored via our ODE algorithm using conditional OT probability paths have perceptual quality that is largely on par with, or better than that achieved by diffusion probability paths, and other recent methods like ΠGDM and RED-Diff, without the need for problem-specific hyperparameter tuning.

5.2 Preliminaries

We introduce relevant background knowledge and notation from conditional diffusion and flow modeling, as well as training-free inference with diffusion models.

Notation. Both diffusion and flow models consider two distinct processes indexed by time between $[0, 1]$ that convert data to noise and noise to data. Here, we follow the temporal notation used in prior work [Lipman et al., 2022] where the distribution at $t = 1$ is the data distribution and $t = 0$ is a standard Gaussian distribution. Note that this is opposite of the prevalent notation used in diffusion model literature [Song and Ermon, 2019, Ho et al., 2020, Song et al., 2020a,b].

We let $\boldsymbol{x}_t$ denote a real-valued vector at time t, without regard to which process (*i.e.,* diffusion or flow) it was drawn from. The probability density for the data to noise process is denoted q and the parameterized probability density for the noise to data process is denoted p_θ. Expectations with respect to q are denoted via $\mathbb{E}_q$, where the relevant random variables are noisy $\boldsymbol{x}_t$, clean data $\boldsymbol{x}_1$, and conditioning $\boldsymbol{y}$ with density $q(\boldsymbol{x}_t, \boldsymbol{x}_1, \boldsymbol{y}) = q(\boldsymbol{x}_t|\boldsymbol{x}_1, \boldsymbol{y})q(\boldsymbol{x}_1, \boldsymbol{y})$. Here $q(\boldsymbol{x}_1, \boldsymbol{y})$ is unknown and $q(\boldsymbol{x}_t|\boldsymbol{x}_1, \boldsymbol{y})$ will be a modeling choice. Conditional diffusion or flow models aim to produce samples $\boldsymbol{x}_1 \sim q(\boldsymbol{x}_1|\boldsymbol{y})$. We generally keep function arguments of t implicit (i.e. $f(\boldsymbol{x}_t, t)$ is informally written as $f(\boldsymbol{x}_t)$.)

5.2.1 Conditional diffusion models.

Suppose we have samples $\boldsymbol{x}_1$ (*e.g.,* an image) and conditioning $\boldsymbol{y}$ (*e.g.,* a distorted image) drawn from a data distribution $q(\boldsymbol{x}_1, \boldsymbol{y})$. Conditional diffusion models use latent variables $\boldsymbol{x}_{0:1} = \{\boldsymbol{x}_t | t \in [0, 1)\}$ to model the joint distribution $p_\theta(\boldsymbol{x}_{0:1}, \boldsymbol{x}_1|\boldsymbol{y})$ for the noise to data process p_θ.[1] The data to noise process q approximates the posterior $q(\boldsymbol{x}_{0:1}|\boldsymbol{x}_1, \boldsymbol{y})$ and is defined as a Markov chain[2] that adds Gaussian noise to data, which in continuous time satisfies a stochastic differential equation (SDE)[Song and Ermon, 2019, Ho et al., 2020, Song et al., 2020b]. The parameters of p_θ are learned via minimizing a regression loss derived from the variational bound on negative log-likelihood with respect to $\widehat{\boldsymbol{x}_1}$:

$$L_{\text{diffusion}}(\widehat{\boldsymbol{x}_1}) = \int_0^1 w(t)\mathbb{E}_{\boldsymbol{x}_t \sim q(\boldsymbol{x}_t|\boldsymbol{x}_1, \boldsymbol{y}), \boldsymbol{x}_1, \boldsymbol{y} \sim q(\boldsymbol{x}_1, \boldsymbol{y})} \left[\|\widehat{\boldsymbol{x}_1}(\boldsymbol{x}_t, \boldsymbol{y}) - \boldsymbol{x}_1\|^2 \right] dt, \tag{5.1}$$

where $w(t)$ are positive weights [Kingma et al., 2021, Song et al., 2021a, Kingma and Gao, 2023], and $\widehat{\boldsymbol{x}_1}$ is a deterministic parametrized denoiser. The optimal solution for $\widehat{\boldsymbol{x}_1}$ with the squared L_2 error in Eq. (5.1) is $\mathbb{E}_{\boldsymbol{x}_1 \sim q(\boldsymbol{x}_1|\boldsymbol{x}_t, \boldsymbol{y})}[\boldsymbol{x}_1|\boldsymbol{x}_t, \boldsymbol{y}]$. For brevity as well as ease of readability, henceforth we will typically write the expectations with respect to q such as the one that appears in Eq. (5.1) in short as $\mathbb{E}_q$. Many equivalent parameterizations exist for the loss in Eq. (5.1) and have known conversions to denoising. Sampling using p_θ proceeds by starting from $\boldsymbol{x}_0 \sim p_\theta(\boldsymbol{x}_0|\boldsymbol{y})$ and integrating the SDE using $\widehat{\boldsymbol{x}_1}$ to $t = 1$. If $p_\theta(\boldsymbol{x}_0|\boldsymbol{y}) = q(\boldsymbol{x}_0|\boldsymbol{y})$, the SDE is integrated exactly, and $\widehat{\boldsymbol{x}_1}(\boldsymbol{x}_t, \boldsymbol{y}) = \mathbb{E}_q[\boldsymbol{x}_1|\boldsymbol{x}_t, \boldsymbol{y}]$, the resulting $\boldsymbol{x}_1 \sim q(\boldsymbol{x}_1|\boldsymbol{y})$ as desired. Please see Sec. 3.1.1 for an overview of discrete-time diffusion models, and Sec. 2.2.1 for an overview of continuous-time diffusion models.

5.2.2 Conditional flow models

Alternatively, continuous normalizing flow models [Chen et al., 2018] define the data generation process through an ODE. This leads to simpler formulations and does not introduce extra noise during intermediate steps of sample generation. Recently, simulation-free training algorithms have been

[1] In discrete time, we can write this joint distribution as $p_\theta(\boldsymbol{x}_{0:1}|\boldsymbol{y}) = p_\theta(\boldsymbol{x}_0|\boldsymbol{y}) \prod_t p_\theta(\boldsymbol{x}_{t+\Delta}|\boldsymbol{x}_t, \boldsymbol{y})$ where Δ denotes an appropriate step size of time discretization in $[0, 1]$. Please see Sec. 3.1.1 for a detailed overview.

[2] In discrete time, the forward process is $q(\boldsymbol{x}_{0:1}|\boldsymbol{x}_1, \boldsymbol{y}) = \prod_t q(x_{t-\Delta}|\boldsymbol{x}_t, \boldsymbol{y})$. Please see Sec. 3.1.1 for a detailed overview.

designed specifically for such models [Lipman et al., 2022, Liu et al., 2022b, Albergo et al., 2023], an example being the Conditional Flow Matching loss [Lipman et al., 2022],

$$L_{\text{cfm}}(\widehat{\boldsymbol{v}}) = \int_0^1 \mathbb{E}_{\boldsymbol{x}_t \sim q(\boldsymbol{x}_t|\boldsymbol{x}_1,\boldsymbol{y}),\boldsymbol{x}_1,\boldsymbol{y}\sim q(\boldsymbol{x}_1,\boldsymbol{y})} \left[\|\widehat{\boldsymbol{v}}(\boldsymbol{x}_t,\boldsymbol{y}) - \boldsymbol{v}(\boldsymbol{x}_t,\boldsymbol{y},\boldsymbol{x}_1)\|^2 \right] dt, \tag{5.2}$$

where $\widehat{\boldsymbol{v}}(\boldsymbol{x}_t,\boldsymbol{y})$ denotes a parameterized vector field defining the ODE

$$\frac{d\boldsymbol{x}_t}{dt} = \widehat{\boldsymbol{v}}(\boldsymbol{x}_t,\boldsymbol{y}), \tag{5.3}$$

and data-conditional vector field $\boldsymbol{v}(\boldsymbol{x}_t,\boldsymbol{y},\boldsymbol{x}_1)$ is determined by modeling choice $q(\boldsymbol{x}_t|\boldsymbol{x}_1,\boldsymbol{y})$ If trained perfectly, the marginal distributions of $\boldsymbol{x}_t$ from ODE integration, denoted $p_\theta(\boldsymbol{x}_t|\boldsymbol{y})$, will match the marginal distributions of $q(\boldsymbol{x}_t|\boldsymbol{y})$. Hence sampling from $q(\boldsymbol{x}_1|\boldsymbol{y})$ as desired can be achieved by sampling initial value $\boldsymbol{x}_{t'} \sim q(\boldsymbol{x}_{t'}|\boldsymbol{y})$ and integrating the ODE from t' to 1. Typically, one samples from $t' = 0$ since $q(\boldsymbol{x}_0|\boldsymbol{y})$ is a tractable distribution.

5.2.3 Gaussian probability paths

The time-dependent densities $q(\boldsymbol{x}_t|\boldsymbol{x}_1,\boldsymbol{y})$ are referred to as conditional probability paths. We focus on the class of affine Gaussian probability paths of the form

$$q(\boldsymbol{x}_t|\boldsymbol{x}_1,\boldsymbol{y}) = q(\boldsymbol{x}_t|\boldsymbol{x}_1) = \mathcal{N}(\alpha_t\boldsymbol{x}_1, \sigma_t^2\boldsymbol{I}), \tag{5.4}$$

where non-negative α_t and σ_t are monotonically increasing and decreasing respectively. This class includes the probability paths for conditional diffusion as well as the conditional Optimal Transport (OT) path [Lipman et al., 2022], where $\alpha_t = t$ and $\sigma_t = 1 - t$. The conditional OT path used by flow models has been demonstrated to have good empirical properties, including faster inference and better sampling in practice, and has theoretical support in high-dimensions [Shaul et al., 2023]. As emphasized in Lin et al. [2023], a desirable property for probability paths, obeyed by conditional OT but not commonly used diffusion paths, is to ensure $q(\boldsymbol{x}_0|\boldsymbol{y})$ is known (i.e. $\mathcal{N}(0,\boldsymbol{I})$), as otherwise one cannot exactly sample $\boldsymbol{x}_0$ which can add substantial error. When using these affine Gaussian probability paths with a conditional flow model, one sets [Lipman et al., 2022]

$$\boldsymbol{v}(\boldsymbol{x}_t,\boldsymbol{y},\boldsymbol{x}_1) = \frac{d\alpha_t}{dt}\boldsymbol{x}_1 + \frac{d\sigma_t}{dt}\left(\frac{\boldsymbol{x}_t - \alpha_t\boldsymbol{x}_1}{\sigma_t}\right). \tag{5.5}$$

Converting between flow and diffusion models In our framing for affine Gaussian probability paths, a model is identified as flow or diffusion by whether an ODE or SDE is used for sampling respectively. For this class of paths though, we can convert directly between flow and diffusion models. To see this, note that the optimal $\boldsymbol{v}(\boldsymbol{x}_t,\boldsymbol{y})$ for the Conditional Flow Matching loss in Eq. (5.2) is $\mathbb{E}_q[\boldsymbol{v}(\boldsymbol{x}_t,\boldsymbol{y},\boldsymbol{x}_1)|\boldsymbol{x}_t,\boldsymbol{y}]$, which for affine Gaussian probability paths using Eq. 5.5 is

$$\boldsymbol{v}(\boldsymbol{x}_t,\boldsymbol{y}) = \frac{d\alpha_t}{dt}\mathbb{E}_q[\boldsymbol{x}_1|\boldsymbol{x}_t,\boldsymbol{y}] + \frac{d\sigma_t}{dt}\left(\frac{\boldsymbol{x}_t - \alpha_t\mathbb{E}_q[\boldsymbol{x}_1|\boldsymbol{x}_t,\boldsymbol{y}]}{\sigma_t}\right). \tag{5.6}$$

This equivalence has been noted by Karras et al. [2022], leveraging a more complex conversion from SDE to probability flow ODE from Song et al. [2020b]. Rearranging Eq. (5.6), a diffusion model's denoiser $\widehat{\boldsymbol{x}_1}(\boldsymbol{x}_t,\boldsymbol{y})$ trained using affine Gaussian probability path q can be interchanged with a flow model's $\widehat{\boldsymbol{v}}(\boldsymbol{x}_t,\boldsymbol{y})$ with the same path via

$$\widehat{\boldsymbol{v}} = \left(\alpha_t\frac{d\ln(\alpha_t/\sigma_t)}{dt}\right)\widehat{\boldsymbol{x}_1} + \frac{d\ln\sigma_t}{dt}\boldsymbol{x}_t. \tag{5.7}$$

5.2.4 Training-free conditional inference using unconditional diffusion.

Given pretrained *unconditional* diffusion models that are trained to approximate $\mathbb{E}_q[\boldsymbol{x}_1|\boldsymbol{x}_t]$, training-free approaches for conditional inference aim to approximate $\mathbb{E}_q[\boldsymbol{x}_1|\boldsymbol{x}_t, \boldsymbol{y}]$ without any fine-tuning of the unconditional model. Under affine Gaussian probability paths, the two terms are related by Tweedie's identity [Robbins, 1992] which expresses $\mathbb{E}_q[\boldsymbol{x}_1|\boldsymbol{x}_t, \boldsymbol{y}] = (\boldsymbol{x}_t + \sigma_t^2 \nabla_{\boldsymbol{x}_t} \ln q(\boldsymbol{x}_t|\boldsymbol{y}))/\alpha_t$. Applying this identity (twice for both $\mathbb{E}_q[\boldsymbol{x}_1|\boldsymbol{x}_t, \boldsymbol{y}]$ and $\mathbb{E}_q[\boldsymbol{x}_1|\boldsymbol{x}_t]$) and simplifying gives

$$\mathbb{E}_q[\boldsymbol{x}_1|\boldsymbol{x}_t, \boldsymbol{y}] = \mathbb{E}_q[\boldsymbol{x}_1|\boldsymbol{x}_t] + \frac{\sigma_t^2}{\alpha_t} \nabla_{\boldsymbol{x}_t} \ln q(\boldsymbol{y}|\boldsymbol{x}_t). \tag{5.8}$$

Following Eq. 5.8, past approaches (*e.g.*, Chung et al. [2022a], Song et al. [2022]) have used the pretrained model for the first term and approximated the second intractable term to produce an approximate $\widehat{\boldsymbol{x}_1}(\boldsymbol{x}_t, \boldsymbol{y})$. Diffusion posterior sampling (DPS) [Chung et al., 2022a] proposed to approximate $q(\boldsymbol{y}|\boldsymbol{x}_t)$ via $q(\boldsymbol{y}|\boldsymbol{x}_1 = \widehat{\boldsymbol{x}_1}(\boldsymbol{x}_t))$. Later, Pseudo-inverse Guided Diffusion Models (ΠGDM) [Song et al., 2022] improved upon DPS for linear noisy observations where $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x} + \sigma_y \epsilon$, where $\boldsymbol{A}$ is some measurement matrix and $\epsilon \sim \mathcal{N}(0, \boldsymbol{I})$, by approximating $q(\boldsymbol{y}|\boldsymbol{x}_t)$ as $\mathcal{N}(\boldsymbol{A}\widehat{\boldsymbol{x}_1}(\boldsymbol{x}_t), \sigma_{\boldsymbol{y}}^2 \boldsymbol{I} + r_t^2 \boldsymbol{A}\boldsymbol{A}^T)$, derived via first approximating $q(\boldsymbol{x}_1|\boldsymbol{x}_t)$ as $\mathcal{N}(\widehat{\boldsymbol{x}_1}(\boldsymbol{x}_t), r_t^2 \boldsymbol{I})$, where r_t is an appropriate time-dependent standard deviation. ΠGDM also suggested adaptive weighting, replacing σ_t^2/α_t with another function of time to account for the approximation. While these past approaches have used Eq. (5.8) for diffusion probability paths, this equation is valid for any affine Gaussian probability path.

5.3 Solving Linear Inverse Problems without Conditional Training via Flows

5.3.1 Problem setup: Linear Inverse Problem

In the standard setup of a linear inverse problem, we observe measurements $\boldsymbol{y} \in \mathbb{R}^n$ such that

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_1 + \epsilon \tag{5.9}$$

where $\boldsymbol{x}_1 \in \mathbb{R}^m$ is drawn from an unknown data distribution $q(\boldsymbol{x}_1)$, $\boldsymbol{A} \in \mathbb{R}^{n \times m}$ is a known measurement matrix, and $\epsilon \sim \mathcal{N}(0, \sigma_y^2 \boldsymbol{I})$ is unknown *i.i.d.* Gaussian noise with known standard deviation σ_y. Given a pretrained flow model with $\widehat{\boldsymbol{v}}(\boldsymbol{x}_t)$ that can sample from $q(\boldsymbol{x}_1)$, and measurements $\boldsymbol{y}$, our goal is to produce clean samples from the posterior $q(\boldsymbol{x}_1|\boldsymbol{y}) \propto q(\boldsymbol{y}|\boldsymbol{x}_1)q(\boldsymbol{x}_1)$ without training a problem-specific conditional flow model defined by $\widehat{\boldsymbol{v}}(\boldsymbol{x}_t, \boldsymbol{y})$. In this section, we motivate and propose our approach to solving this problem using flows.

5.3.2 Adapting the vector field of unconditional flow models for conditional sampling

To solve linear inverse problems without any training via flow models, we derive an expression similar to Eq. 5.8 that relates conditional vector fields under Gaussian probability paths to unconditional vector fields.

Let q be a Gaussian probability path described by Eq. 5.4. Assume we observe $\boldsymbol{y} \sim q(\boldsymbol{y}|\boldsymbol{x}_1)$ for arbitrary $q(\boldsymbol{y}|\boldsymbol{x}_1)$ and $\boldsymbol{v}(\boldsymbol{x}_t)$ is a vector field enabling sampling $\boldsymbol{x}_t \sim q(\boldsymbol{x}_t)$. Note that Eq. (5.6) without $\boldsymbol{y}$ also holds for optimal unconditional $\boldsymbol{v}(\boldsymbol{x}_t)$. Inserting Eq. (5.8) into Eq. (5.6) and taking the difference $(\boldsymbol{v}(\boldsymbol{x}_t, \boldsymbol{y}) - \boldsymbol{v}(\boldsymbol{x}_t))$ yields

$$\boldsymbol{v}(\boldsymbol{x}_t, \boldsymbol{y}) = \boldsymbol{v}(\boldsymbol{x}_t) + \sigma_t^2 \frac{d \ln(\alpha_t/\sigma_t)}{dt} \nabla_{\boldsymbol{x}_t} \ln q(\boldsymbol{y}|\boldsymbol{x}_t). \tag{5.10}$$

We use Eq. (5.10) in our training-free algorithm for solving linear inverse using flows by incorporating ΠGDM's adaptation. In particular, given $\widehat{\boldsymbol{v}}(\boldsymbol{x}_t)$ (or $\widehat{\boldsymbol{x}_1}(\boldsymbol{x}_t)$), our approximation will be

$$\widehat{\boldsymbol{v}}(\boldsymbol{x}_t, \boldsymbol{y}) = \widehat{\boldsymbol{v}}(\boldsymbol{x}_t) + \sigma_t^2 \frac{d \ln(\alpha_t/\sigma_t)}{dt} \gamma_t \nabla_{\boldsymbol{x}_t} \ln q^{app}(\boldsymbol{y}|\boldsymbol{x}_t), \tag{5.11}$$

where $q^{app}(\boldsymbol{y}|\boldsymbol{x}_t)$ denotes an approximation for $q(\boldsymbol{y}|\boldsymbol{x}_t)$ and γ_t denotes time-dependent weights.

We refer to $\gamma_t = 1$ as unadaptive and other choices as adaptive weights. In general, we view adaptive weights $\gamma_t \neq 1$ as an adjustment for error in $q^{app}(\boldsymbol{y}|\boldsymbol{x}_t)$.

Approximating $q(\boldsymbol{y}|\boldsymbol{x}_t)$. The update for adapting the unconditional vector field in Eq. (5.10) requires $q(\boldsymbol{y}|\boldsymbol{x}_t)$ which is intractable to compute as it involves marginalization over $\boldsymbol{x}_1$

$$q(\boldsymbol{y}|\boldsymbol{x}_t) = \int_{\boldsymbol{x}_1} q(\boldsymbol{y}|\boldsymbol{x}_1) q(\boldsymbol{x}_1|\boldsymbol{x}_t) d\boldsymbol{x}_1. \tag{5.12}$$

In this equation, the first term $q(\boldsymbol{y}|\boldsymbol{x}_1)$ is tractable as it is equal to $\mathcal{N}(\boldsymbol{A}\boldsymbol{x}_1, \sigma_y^2 \boldsymbol{I})$. However, it is computationally expensive to estimate the second term $q(\boldsymbol{x}_1|\boldsymbol{x}_t)$ with a flow model or a diffusion model. We therefore use an approximation for $q(\boldsymbol{y}|\boldsymbol{x}_t)$ and refer to this approximation as $q^{app}(\boldsymbol{y}|\boldsymbol{x}_t)$. Following ΠGDM, we set

$$q(\boldsymbol{x}_1|\boldsymbol{x}_t) \approx \mathcal{N}(\widehat{\boldsymbol{x}_1}(\boldsymbol{x}_t), r_t^2 I). \tag{5.13}$$

where r_t is an appropriately chosen time-dependent standard deviation. We can now compute $q^{app}(\boldsymbol{y}|\boldsymbol{x}_t)$ in closed form as $q^{app}(\boldsymbol{y}|\boldsymbol{x}_t) \approx \mathcal{N}(\boldsymbol{A}\widehat{\boldsymbol{x}_1}(\boldsymbol{x}_t), \sigma_y^2 I + r_t^2 \boldsymbol{A}\boldsymbol{A}^\top)$ which gives the following approximation for $\nabla_{\boldsymbol{x}_t} \ln q^{app}(\boldsymbol{y}|\boldsymbol{x}_t)$:

$$\nabla_{\boldsymbol{x}_t} \ln q^{app}(\boldsymbol{y}|\boldsymbol{x}_t) \approx (\boldsymbol{y} - \boldsymbol{A}\widehat{\boldsymbol{x}_1})^\top (r_t^2 \boldsymbol{A}\boldsymbol{A}^\top + \sigma_y^2 \boldsymbol{I})^{-1} \boldsymbol{A} \frac{\partial \widehat{\boldsymbol{x}_1}}{\partial \boldsymbol{x}_t}. \tag{5.14}$$

Note that this is a vector-Jacobian product and can computed efficiently with packages for automatic differentiation. With this we generalize ΠGDM to any Gaussian probability path described by Eq. 5.4 by using an alternate r_t^2. We choose r_t^2 following the Π GDM derivation which assumes that $q(\boldsymbol{x}_1)$ is $\mathcal{N}(0, \boldsymbol{I})$ to derive r_t^2. We have $q(\boldsymbol{x}_t|\boldsymbol{x}_1) = \mathcal{N}(\alpha_t \boldsymbol{x}_1, \sigma_t^2 I)$. Thus by Bayes' rule, we can write the posterior as

$$q(\boldsymbol{x}_1|\boldsymbol{x}_t) \propto q(\boldsymbol{x}_1) q(\boldsymbol{x}_t|\boldsymbol{x}_1) = \mathcal{N}\left(\frac{\alpha_t \boldsymbol{x}_t}{\alpha_t^2 + \sigma_t^2}, \frac{\sigma_t^2}{\alpha_t^2 + \sigma_t^2} \boldsymbol{I}\right). \tag{5.15}$$

With this, we approximate r_t^2 as

$$r_t^2 = \frac{\sigma_t^2}{\sigma_t^2 + \alpha_t^2}. \tag{5.16}$$

When $\alpha_t = 1$, we recover ΠGDM's r_t^2 as expected under their Variance-Exploding path specification.

5.3.3 Converting between affine Gaussian probability paths

To complete our derivation, we demonstrate how one can train with path q' and perform sampling with alternative path q. This conversion is crucial to enable sampling with any probability path, including particularly the conditional OT probability path, without training given an existing pre-trained model. Such conversions have been noted previously by Karras et al. [2022] using an SDE perspective. Our derivation exposes important subtleties when converting between affine Gaussian probability paths.

Consider two affine Gaussian probability paths, with joint densities q and q', defined by Eq. 5.4 with α_t, σ_t and α'_t, σ'_t respectively. Define $t'(t)$ as the unique solution to $\alpha_t/\sigma_t = \alpha'_{t'}/\sigma'_{t'}$ when it exists for given t. The solution for $t'(t)$ is unique due to the monotonicity requirements of both α and σ. By definition, the joint densities q and q' share the same distribution on the data $\boldsymbol{x}_1$, $q(\boldsymbol{x}_1|\boldsymbol{y}) = q'(\boldsymbol{x}_1|\boldsymbol{y})$. Then for affine Gaussian probability paths, $q(\boldsymbol{X}_t = \boldsymbol{x}_t|\boldsymbol{x}_1, \boldsymbol{y}) = q'(\boldsymbol{X}'_{t'(t)} = \alpha'_{t'(t)}\boldsymbol{x}_t/\alpha_t|\boldsymbol{x}_1, \boldsymbol{y})$ when $t'(t)$ exists. Since the joint densities are identical, the conditional distributions over $\boldsymbol{x}_1$ used by the optimal denoiser and vector fields are also identical at these values.

So $\widehat{\boldsymbol{x}_1}$ trained under q' can be used for sampling under q via evaluating at $\widehat{\boldsymbol{x}_1}(\alpha'_{t'(t)}\boldsymbol{x}_t/\alpha_t, t'(t), \boldsymbol{y})$ (with explicit time for clarity) whenever $t'(t)$ exists, with identical argument changes for vector fields. In particular, if sampling uses the conditional OT probability path, we have

$$t'(t) = \text{SNR}_{q'}^{-1}(\text{SNR}_q(t))) = \text{SNR}_{q'}^{-1}\left(\frac{t}{1-t}\right). \tag{5.17}$$

where signal-to-noise ratio $\text{SNR}(t) = \alpha_t/\sigma_t$. The main avenue for nonexistence for $t'(t)$ is if the model under q' is trained using a minimum SNR above zero, which induces a minimum t for which $t'(t)$ exists. When a minimum t exists, we can only perform sampling with q starting from $\boldsymbol{x}_t \sim q(\boldsymbol{x}_t|\boldsymbol{y})$. Approximating this sample is entirely analogous to approximating $\boldsymbol{x}_0 \sim q'(\boldsymbol{x}_0|\boldsymbol{y})$. This error already exists for q' because $q'(\boldsymbol{x}_0|\boldsymbol{y})$ is not $\mathcal{N}(0, \boldsymbol{I})$ unless q' is trained to zero SNR. An initialization problem cannot be avoided if q' has a limited SNR range by switching paths to q. This problem is relevant when converting pre-trained diffusion models as typical diffusion paths have a nonzero minimum SNR.

5.3.4 Our algorithm for solving linear inverse problems with Cond-OT flows

Starting flow sampling at time $t > 0$ Initializing conditional diffusion model sampling at $t > 0$ has been proposed by Chung et al. [2022c]. For flows, we similarly want $\boldsymbol{x}_t \sim q(\boldsymbol{x}_t|\boldsymbol{y})$ at initialization time t. In our experiments, we examine (approximately) initializing at different times $t > 0$ using

$$\boldsymbol{x}_t = \alpha_t \boldsymbol{y} + \sigma_t \epsilon \tag{5.18}$$

for $\epsilon \sim \mathcal{N}(0, \boldsymbol{I})$ when $\boldsymbol{y}$ is the correct shape. For superresolution, we used nearest-neighbor interpolation on $\boldsymbol{y}$ instead. We also consider using $\boldsymbol{A}^\dagger y$ as an ablation in Sec. 5.9 (where $\boldsymbol{A}^\dagger$ is the pseudo-inverse of $\boldsymbol{A}$ [Song et al., 2022]). We may be forced to use this initialization for flow sampling due to converting a diffusion model not trained to zero SNR. However as shown in [Chung et al., 2022c] for diffusion, this initialization can improve results more generally. Conceptually, if the resulting $\boldsymbol{x}_t$ is closer to $\boldsymbol{x}_t \sim q(\boldsymbol{x}_t|\boldsymbol{y})$ than achieved via starting from an earlier time t' and integrating, then this initialization can result in less overall error.

Algorithm Summary Putting this altogether, our proposed approach using flow sampling and conditional OT probability paths is succinctly summarized in Algorithm 5, derived via inserting $\alpha_t = t$ and $\sigma_t = 1 - t$. This algorithm uses the unconditional vector field adaptation proposed in Eq. (5.11) and uses this vector field adaptation $\hat{\boldsymbol{v}}_{\text{adapted}}$ to integrate the ODE from some initial time t_0 to 1 to get the final corrected image $\boldsymbol{x}_1$. We can integrate the ODE by using any standard numerical methods such as Euler method, Runge-Kutta method etc. As an example, an intermediate update of Euler method from time t to $t + \Delta t$ during ODE integration is given by $\boldsymbol{x}_{t+\Delta t} = \boldsymbol{x}_t + \hat{\boldsymbol{v}}_{\text{adapted}} \Delta t$. Unlike ΠGDM, we propose unadaptive weights $\gamma_t = 1$. By default, we set initialization time $t_0 = 0.2$. The algorithm therefore has no additional hyperparameters to tune over traditional diffusion or flow sampling. In Appendix 5.3.5, we detail our algorithm for other Gaussian probability paths, and the equivalent formulation when a pretrained vector field is available instead.

Algorithm 5 Solving linear inverse problems via flows using conditional OT probability path

Input: Pretrained denoiser $\widehat{\boldsymbol{x}_1}(\boldsymbol{x}_t)$ converted to conditional OT probability path, noisy measurement $\boldsymbol{y}$, measurement matrix $\boldsymbol{A}$, initial time t_0, and std σ_y

Initialize $\boldsymbol{x}_{t_0} = t_0 \mathbf{y} + (1 - t_0)\epsilon$, where $\epsilon \sim \mathcal{N}(0, \boldsymbol{I})$ ▷ Initialize $\boldsymbol{x}_t$, Eq. (5.18)

$\boldsymbol{x}_t = \boldsymbol{x}_{t_0}$

for each time step t of ODE integration **do** ▷ Integrate ODE from $t = t_0$ to 1.

  $r_t^2 = \frac{(1-t)^2}{t^2 + (1-t)^2}$ ▷ Value of r_t^2 from Eq. (5.16)

  $\widehat{\boldsymbol{v}} = \frac{\widehat{\boldsymbol{x}_1}(\boldsymbol{x}_t) - \boldsymbol{x}_t}{1-t}$ ▷ Convert $\widehat{\boldsymbol{x}_1}$ to vector field, Eq. (5.7)

  $\boldsymbol{g} = (\boldsymbol{y} - \boldsymbol{A}\widehat{\boldsymbol{x}_1})^\top (r_t^2 \boldsymbol{A}\boldsymbol{A}^\top + \sigma_y^2 \boldsymbol{I})^{-1} \boldsymbol{A} \frac{\partial \widehat{\boldsymbol{x}_1}}{\partial \boldsymbol{x}_t}$ ▷ $\nabla_{\boldsymbol{x}_t} \ln q^{app}(y|\boldsymbol{x}_t)$

  $\widehat{\boldsymbol{v}}_{\text{adapted}} = \widehat{\boldsymbol{v}} + \frac{1-t}{t} \boldsymbol{g}$ ▷ Adapt the unconditional vector field $\widehat{\boldsymbol{v}}$, Eq. (5.11)

  $\boldsymbol{x}_{t+\Delta t} = \text{ODESolverStep}(\boldsymbol{x}_t, \widehat{\boldsymbol{v}}_{\text{adapted}})$ ▷ One step of ODE solver to update $\boldsymbol{x}_t$

end for

return $\boldsymbol{x}_1$ ▷ This is the solution of ODE integration.

5.3.5 Our algorithm for any affine gaussian probability path

Algorithm 5 in the previous section is specific to conditional OT probability paths. Here we provide Algorithm 6 for any Gaussian probability path specified by Eq. 5.4. Algorithm 5 and Algorithm 6 are written assuming a denoiser $\widehat{\boldsymbol{x}_1}(\boldsymbol{x}_t)$ is provided from a pretrained diffusion model. For completeness, we also include equivalent Algorithm 7 that assumes $\widehat{\boldsymbol{v}}(\boldsymbol{x}_t)$ is provided from a pretrained flow model. In all cases, the vector field or denoiser is evaluated only once per iteration.

Our VP-ODE sampling results correspond to α_t and σ_t given from the Variance-Preserving path, which can be found in [Lipman et al., 2022].

5.4 Experimental Details

5.4.1 Datasets

We verify the effectiveness of our proposed approach on three datasets: face-blurred ImageNet 64×64 and 128×128 [Deng et al., 2009, Russakovsky et al., 2015, Yang et al., 2022], and AnimalFacesHQ

Algorithm 6 A training-free approach to solve inverse problems via flows with a pretrained denoiser

Input: Pretrained denoiser $\widehat{\boldsymbol{x}_1}(\boldsymbol{x}_t)$ converted to Gaussian probability path with α_t and σ_t, noisy measurement $\boldsymbol{y}$, measurement matrix $\boldsymbol{A}$, initial time t_0, adaptive weights γ_t, and std σ_y

Initialize $\boldsymbol{x}_{t_0} = \alpha_{t_0}\mathbf{y} + \sigma_{t_0}\epsilon$, where $\epsilon \sim \mathcal{N}(0, \boldsymbol{I})$ ▷ Initialize $\boldsymbol{x}_t$, Eq. (5.18)

$\boldsymbol{x}_t = \boldsymbol{x}_{t_0}$

for each time step t of ODE integration **do** ▷ Integrate ODE from $t = t_0$ to 1.

  $r_t^2 = \frac{\sigma_t^2}{\sigma_t^2 + \alpha_t^2}$ ▷ Value of r_t^2 from Eq. (5.16)

  $\widehat{\boldsymbol{v}} = \left(\alpha_t \frac{d \ln(\alpha_t/\sigma_t)}{dt}\right) \widehat{\boldsymbol{x}_1} + \frac{d \ln \sigma_t}{dt} \boldsymbol{x}_t$ ▷ Convert $\widehat{\boldsymbol{x}_1}$ to vector field, Eq. (5.7)

  $\boldsymbol{g} = (\boldsymbol{y} - \boldsymbol{A}\widehat{\boldsymbol{x}_1})^\top (r_t^2 \boldsymbol{A}\boldsymbol{A}^\top + \sigma_y^2 \boldsymbol{I})^{-1} \boldsymbol{A} \frac{\partial \widehat{\boldsymbol{x}_1}}{\partial \boldsymbol{x}_t}$ ▷ $\nabla_{\boldsymbol{x}_t} \ln q^{app}(y|\boldsymbol{x}_t)$

  $\widehat{\boldsymbol{v}}_{\text{adapted}} = \widehat{\boldsymbol{v}} + \sigma_t^2 \frac{d \ln(\alpha_t/\sigma_t)}{dt} \gamma_t \boldsymbol{g}$ ▷ Adapt unconditional vector field $\widehat{\boldsymbol{v}}$, Eq. (5.11)

  $\boldsymbol{x}_{t+\Delta t} = \text{ODESolverStep}(\boldsymbol{x}_t, \widehat{\boldsymbol{v}}_{\text{adapted}})$ ▷ One step of ODE solver to update $\boldsymbol{x}_t$

end for

return $\boldsymbol{x}_1$ ▷ This is the solution of ODE integration.

Algorithm 7 A training-free approach to solve inverse problems via flows with a pretrained vector field

Input: Pretrained vector field $\widehat{\boldsymbol{v}}(\boldsymbol{x}_t)$ converted to Gaussian probability path with α_t and σ_t, noisy measurement $\boldsymbol{y}$, measurement matrix $\boldsymbol{A}$, initial time t_0, adaptive weights γ_t, and std σ_y

Initialize $\boldsymbol{x}_{t_0} = \alpha_{t_0}\mathbf{y} + \sigma_{t_0}\epsilon$, where $\epsilon \sim \mathcal{N}(0, \boldsymbol{I})$ ▷ Initialize $\boldsymbol{x}_t$, Eq. (5.18)

$\boldsymbol{x}_t = \boldsymbol{x}_{t_0}$

for each time step t of ODE integration **do** ▷ Integrate ODE from $t = t_0$ to 1.

  $\widehat{\boldsymbol{v}} = \widehat{\boldsymbol{v}}(\boldsymbol{x}_t)$ ▷ $\boldsymbol{x}_t$ is value of $\boldsymbol{x}_t$ at time t during ODE integration

  $r_t^2 = \frac{\sigma_t^2}{\sigma_t^2 + \alpha_t^2}$ ▷ Value of r_t^2 from Eq. (5.16)

  $\widehat{\boldsymbol{x}_1} = \left(\alpha_t \frac{d \ln(\alpha_t/\sigma_t)}{dt}\right)^{-1} \left(\widehat{\boldsymbol{v}} - \frac{d \ln \sigma_t}{dt} \boldsymbol{x}_t\right)$ ▷ Convert vector field to $\widehat{\boldsymbol{x}_1}$, Eq. (5.7)

  $\boldsymbol{g} = (\boldsymbol{y} - \boldsymbol{A}\widehat{\boldsymbol{x}_1})^\top (r_t^2 \boldsymbol{A}\boldsymbol{A}^\top + \sigma_y^2 \boldsymbol{I})^{-1} \boldsymbol{A} \frac{\partial \widehat{\boldsymbol{x}_1}}{\partial \boldsymbol{x}_t}$ ▷ $\nabla_{\boldsymbol{x}_t} \ln q^{app}(y|\boldsymbol{x}_t)$

  $\widehat{\boldsymbol{v}}_{\text{adapted}} = \widehat{\boldsymbol{v}} + \sigma_t^2 \frac{d \ln(\alpha_t/\sigma_t)}{dt} \gamma_t \boldsymbol{g}$ ▷ Adapt unconditional vector field $\widehat{\boldsymbol{v}}$, Eq. (5.11)

  $\boldsymbol{x}_{t+\Delta t} = \text{ODESolverStep}(\boldsymbol{x}_t, \widehat{\boldsymbol{v}}_{\text{adapted}})$ ▷ One step of ODE solver to update $\boldsymbol{x}_t$

end for

return $\boldsymbol{x}_1$ ▷ This is the solution of ODE integration.

(AFHQ) 256×256 [Choi et al., 2020]. We report our results on 10K randomly sampled images from validation split of ImageNet, and 1500 images from test split of AFHQ.

5.4.2 Tasks

We report results on the following linear inverse problems: inpainting (center-crop), Gaussian deblurring, super-resolution, and denoising. The exact details of the measurement operators are: 1) For inpainting, we use centered mask of size 20×20 for ImageNet-64, 40×40 for ImageNet-128, and 80×80 for AFHQ. In addition, for images of size 256×256, we also use free-form masks simulating brush strokes similar to the ones used in Saharia et al. [2022a], Song et al. [2022]. 2) For super-resolution, we apply bicubic interpolation to downsample images by $4\times$ for datasets that have images with resolution 256×256 and downsample images by $2\times$ otherwise. 3) For Gaussian deblurring, we apply Gaussian blur kernel of size 61×61 with standard deviation of 1 for ImageNet-64 and ImageNet-128, and 61×61 with standard deviation of 3 for AFHQ. 4) For denoising, we add *i.i.d.* Gaussian noise with $\sigma_y = 0.05$ to the images. For tasks besides denoising, we consider *i.i.d.* Gaussian noise with $\sigma_y = 0$ and 0.05 to the images. Images $\boldsymbol{x}_1$ are normalized to range $[-1, 1]$.

5.4.3 Implementation Details

We trained our own continuous-time conditional VP-SDE model, and conditional Optimal Transport (conditional OT) flow model from scratch on the above datasets following the hyperparameters and training procedure outlined in Song et al. [2020b] and Lipman et al. [2022]. These models are conditioned on class labels, not noisy images. All derivations hold with class label c since $q(\boldsymbol{y}|c, \boldsymbol{x}_1) = q(\boldsymbol{y}|\boldsymbol{x}_1)$ (i.e. the noisy image is independent of class label given the image). We use the open-source implementation of the Euler method provided in torchdiffeq library [Chen, 2018] to solve the ODE in our experiments. Our choice of Euler is intentionally simple, as we focus on flow sampling with the conditional OT path, and not on the choice of ODE solver.

5.4.4 Metrics

We follow prior works [Chung et al., 2022a, Kawar et al., 2022] and report Fréchet Inception Distance (FID) [Heusel et al., 2017], Learned Perceptual Image Patch Similarity (LPIPS) [Zhang et al., 2018], peak signal-to-noise ratio (PSNR), and structural similarity index (SSIM). We use open-source implementations of these metrics in the TorchMetrics library [Detlefsen et al., 2022].

5.4.5 Methods and Baselines

We use our two pretrained model checkpoints— a conditional OT flow model and continuous VP-SDE diffusion model, and perform flow sampling with both conditional OT and Variance-Preserving (VP) paths, labeling our methods as OT-ODE and VP-ODE respectively. We compare our OT-ODE and VP-ODE methods against ΠGDM [Song et al., 2022] and RED-Diff [Mardani et al., 2023] as relevant baselines. We selected these baselines because they achieve state-of-the-art performance in solving

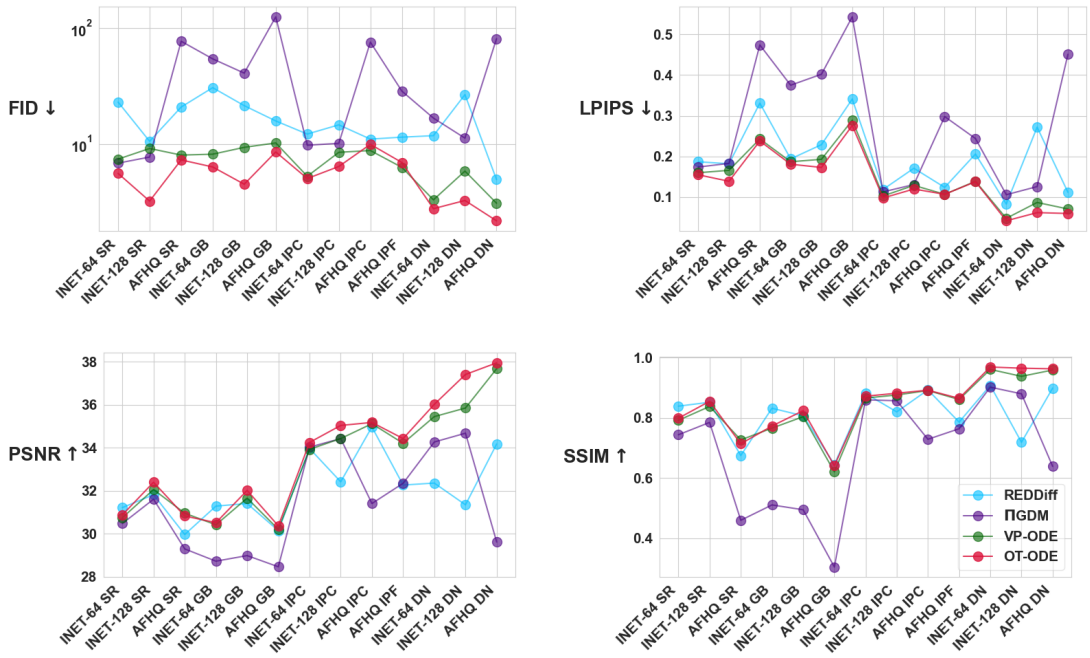


Figure 5.1: Quantitative evaluation of pretrained VP-SDE model for solving linear inverse problems on super-resolution (SR), gaussian deblurring (GB), inpainting with centered (IPC) and freeform mask (IPF), and denoising (DN) with $\sigma_y = 0.05$. We present results on face-blurred ImageNet-64 (INET-64), face-blurred ImageNet-128 (INET-128), and AFHQ.

linear inverse problems using diffusion models. The code for both baseline methods is available on github, and we make minimal changes while reimplementing these methods in our codebase. A fair comparison between methods requires considering the number of function evaluations (NFEs) used during sampling.

We utilize at most 100 NFEs for our OT-ODE and VP-ODE sampling, and utilize 100 for ΠGDM as recommended in Song et al. [2022]. We allow RED-Diff 1000 NFEs since it does not require gradients of $\widehat{\boldsymbol{x}_1}$. For OT-ODE following Algorithm 5, we use $\gamma_t = 1$ and initial $t = 0.2$ for all datasets and tasks. For VP-ODE following Algorithm 6 in the Appendix, we use $\gamma_t = \sqrt{\frac{\alpha_t}{\alpha_t^2+\sigma_t^2}}$ and initial $t = 0.4$ for all datasets and tasks. Ablations of these mildly tuned hyperparameters are shown in Sec. 5.9. We extensively tuned hyperparameters for RED-Diff and ΠGDM as described in Sec. 5.12, including different hyperparameters per dataset and task.

5.5 Experimental Results for Variance Preserving SDE

We report quantitative results for the VP-SDE model, across all datasets and linear measurements, in Figure 5.1 for $\sigma_y = 0.05$, and in Figure 5.2 for $\sigma_y = 0$. Additionally, we report results for the conditional OT flow model in Figure 5.7 and Figure 5.6 for $\sigma_y = 0.05$ and $\sigma_y = 0$, respectively. Exact numerical values for all the metrics across all datasets and tasks can also be found in Sec. 5.7. Qualitative images have been selected for demonstration purposes.

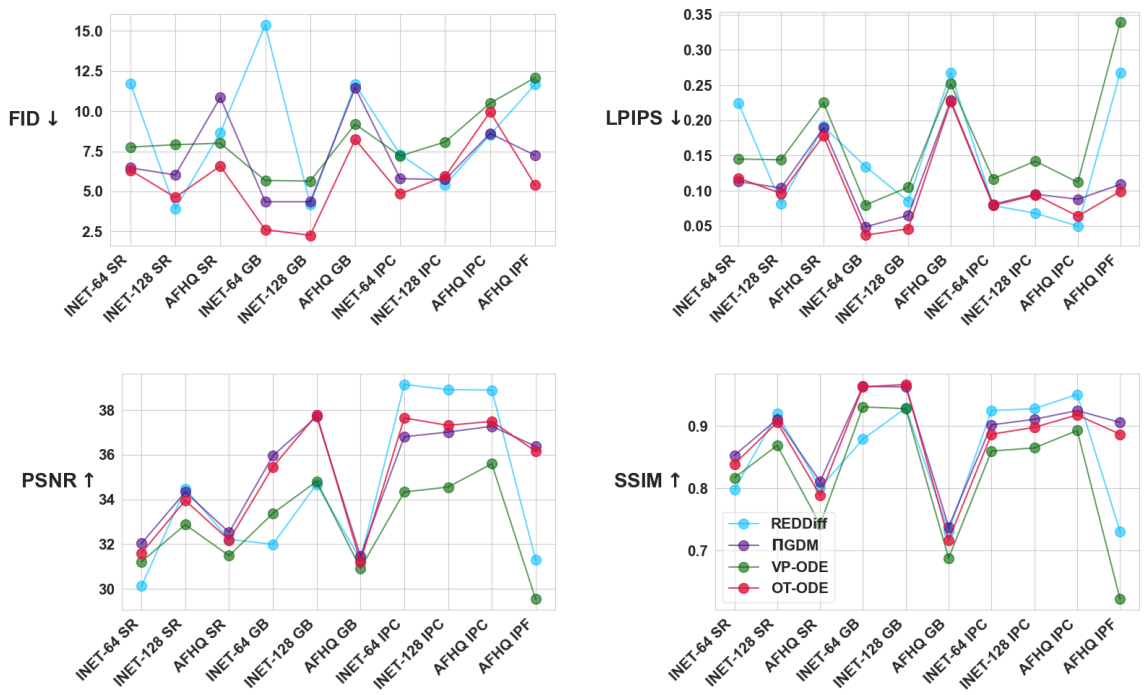


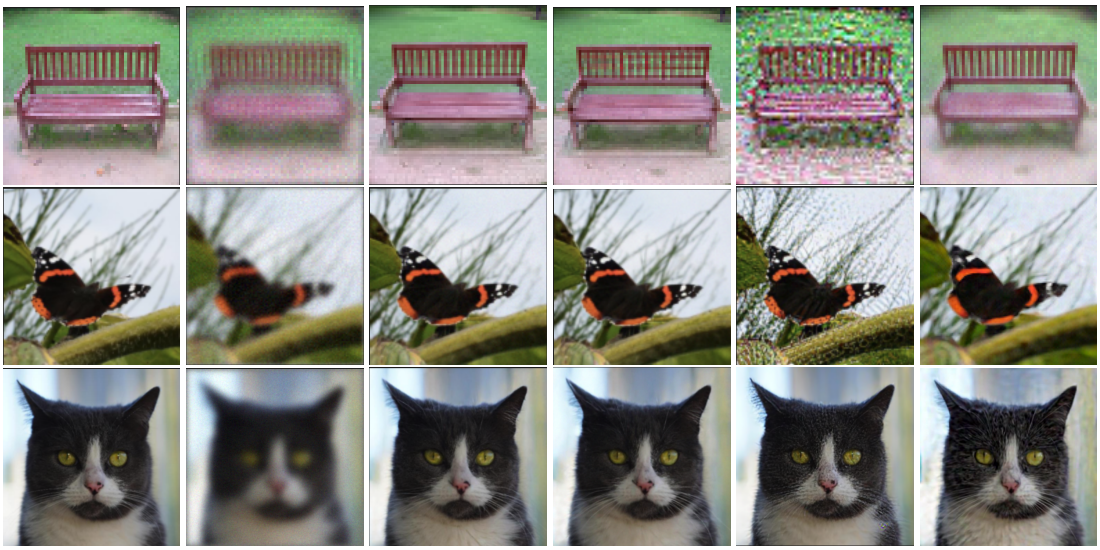
Figure 5.2: Quantitative evaluation of pretrained VP-SDE model for linear inverse problems on super-resolution (SR), gaussian deblurring (GB), image inpainting - centered mask (IPC) and inpainting - free-form (IPF) with $\sigma_y = 0$. We show results on face-blurred ImageNet-64 (INET-64), face-blurred ImageNet-128 (INET-128), and AFHQ-256 (AFHQ).

5.5.1 Gaussian Deblurring

We report qualitative noisy results for the VP-SDE model in Figure 5.3 and for the conditional OT flow (cond-OT) model in Figure 5.11. We observe that OT-ODE and VP-ODE outperforms ΠGDM and RED-Diff, both qualitatively and quantitatively, across all datasets for $\sigma_y = 0.05$. As shown in these figures, ΠGDM tends to sharpen the images, which sometimes results in unnatural textures in the images. Further, we also observe some unnatural textures and background noise with RED-Diff for $\sigma_y = 0.05$. For $\sigma_y = 0$, OT-ODE has better FID and LPIPS, but ΠGDM shows improved PSNR and SSIM. Figure 5.18 and Figure 5.17 show qualitative examples for $\sigma_y = 0$.

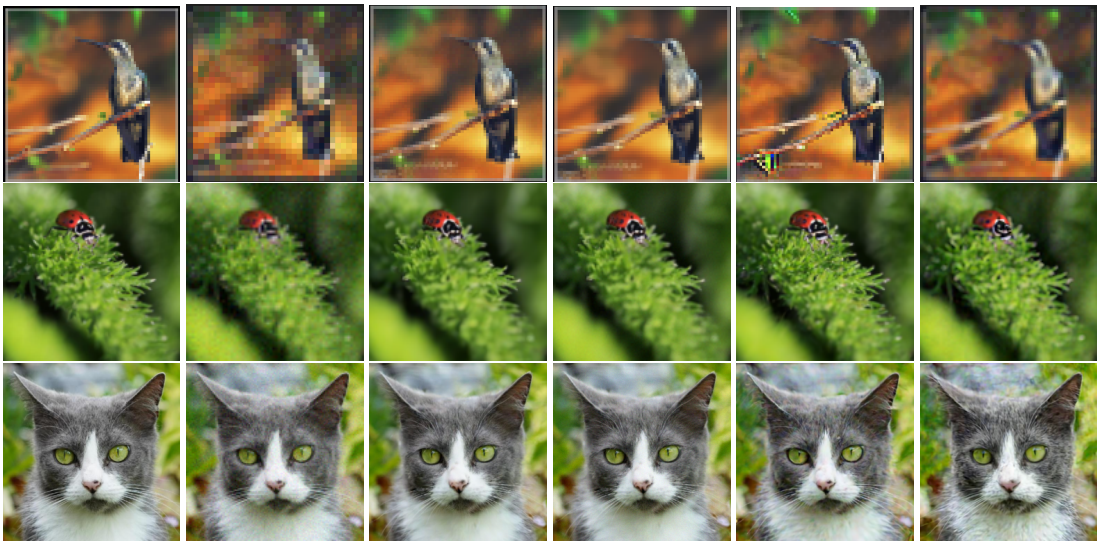
5.5.2 Super-resolution

We report qualitative noisy results for the VP-SDE model in Figure 5.4 and for the cond-OT model in Figure 5.12. OT-ODE consistently achieves better FID, LPIPS and PSNR metrics compared to other methods for $\sigma_y = 0.05$ (See Figure 5.1 and 5.7). Similar to Gaussian deblurring, ΠGDM tends to produce sharper edges. This is certainly desirable to achieve good super-resolution, but sometimes this results in unnatural textures in the images (See Figure 5.4). RED-Diff for $\sigma_y = 0.05$ gives slightly blurry images. In our experiments, we observe RED-Diff is sensitive to the values of σ_y, and we get good quality results for smaller values of σ_y, but the performance deteriorates with increase in value of σ_y.



(a) Reference (b) Distorted (c) OT-ODE (d) VP-ODE (e) ΠGDM (f) REDDiff

Figure 5.3: Results for Gaussian deblurring with VP-SDE model and $\sigma_y = 0.05$ for (**first row**) ImageNet-64, (**second row**) ImageNet-128, and (**third row**) AFHQ.



(a) Reference (b) Distorted (c) OT-ODE (d) VP-ODE (e) ΠGDM (f) REDDiff

Figure 5.4: Results for super-resolution with VP-SDE model and $\sigma_y = 0.05$ for (**first row**) ImageNet-64 2×, (**second row**) ImageNet-128 2×, and (**third row**) AFHQ 4×.

For $\sigma_y = 0$, as shown in Figure 5.19 and Figure 5.20, all the methods achieve comparable performance and the method declared best varies per metric and dataset.

(a) Reference (b) Distorted (c) OT-ODE (d) VP-ODE (e) ΠGDM (f) REDDiff

Figure 5.5: Results for inpainting (centered mask) with VP-SDE model and $\sigma_y = 0.05$ for (**first row**) ImageNet-64, (**second row**) ImageNet-128, and (**third row**) AFHQ.

5.5.3 Inpainting

For centered mask inpainting, OT-ODE outperforms ΠGDM and RED-Diff in terms of LPIPS, PSNR and SSIM across all datasets at $\sigma_y = 0.05$. Regarding FID, OT-ODE performs comparably to or better than VP-ODE (See Figure 5.1 and 5.7). Similar observations hold true for inpainting with freeform mask on AFHQ. We present qualitative noisy results for the VP-SDE model in Figure 5.5 and the cond-OT model in Figure 5.13. As evident in these images, OT-ODE can result in more semantically meaningful inpainting (for instance, the shape of bird's neck, and shape of hot-dog bread in Figure 5.5). In contrast, the inpainted regions generated by RED-Diff tend to be blurry and less semantically meaningful. However, we note that OT-ODE (and VP-ODE) inpainting occasionally produces artifacts in the inpainted region as the resolution of image increases. We show examples of such negative inpainting results in Sec. 5.10.8.

Empirically, we observe that performance of RED-Diff and ΠGDM improves as σ_y decreases. For $\sigma_y = 0$, RED-Diff achieves higher PSNR and SSIM, but performs worse than OT-ODE in terms of FID and LPIPS (Refer to Figure 5.2). OT-ODE's tendency to produce inpainting artifacts for higher resolution images remains for $\sigma_y = 0$, and can occur for the same images as $\sigma_y = 0.05$. These artifacts can significantly degrade the pixel-based metrics PSNR and SSIM more than the perceptual metrics such as FID and LPIPS. We further note that noiseless inpainting for OT-ODE can be improved by incorporating null-space decomposition [Wang et al., 2022]. We describe this adjustment in Sec. 5.11.

5.6 Empirical Results for Conditional OT Flow Model

Sec. 5.5 includes Figure 5.1 with $\sigma_y = 0.05$ and Figure 5.2 with $\sigma_y = 0$ produced with the denoiser from the continuous-time VP-SDE diffusion model showing plots of various metrics across all datasets and

tasks. In this section, we provide the same for our pre-trained conditional OT flow matching model using Algorithm 7 in Figure 5.7, and for the noiseless case in Figure 5.6. To save compute, for the flow model we only include our ΠGDM baseline as RED-Diff required extensive hyperparameter tuning. The qualitative results using the flow model instead of diffusion model checkpoint are identical.

5.7 Detailed Empirical Results for all Tasks and Datasets

This section includes tables that contain the numerical values of the metrics in all data sets and tasks. The tables are hierarchically organized by noise, dataset, and task in consistent ordering. Noisy results with $\sigma_y = 0.05$ are in Table 5.1 to 5.6 and noiseless results with $\sigma_y = 0$ are in Table 5.5 to 5.10.

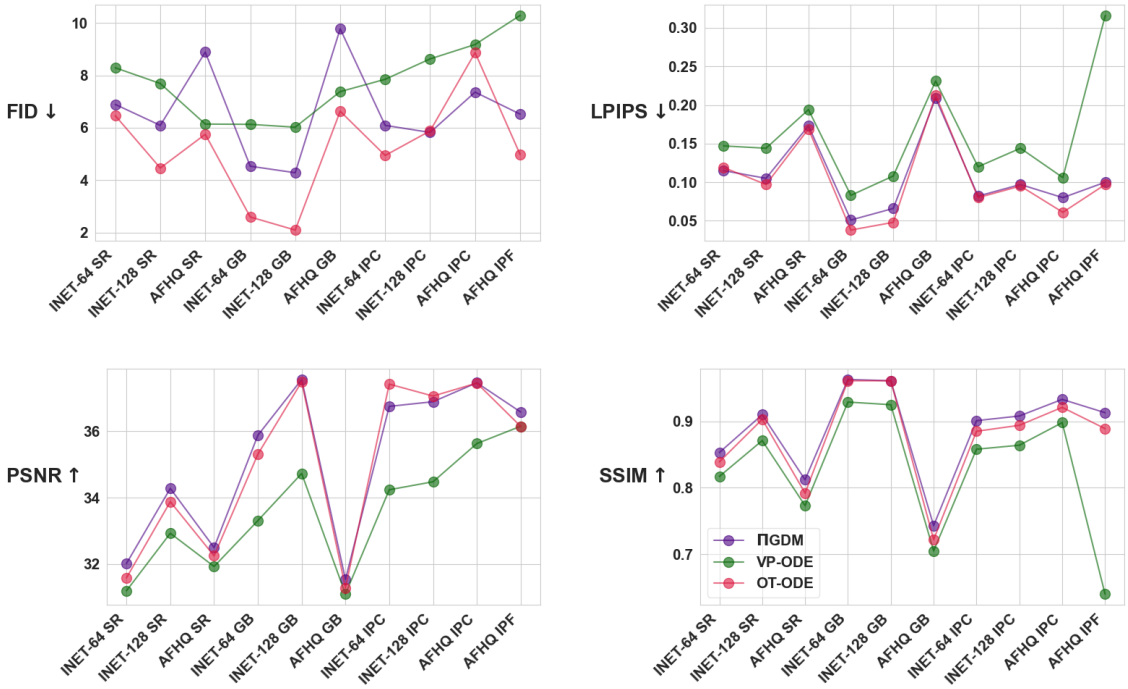


Figure 5.6: Quantitative evaluation of pretrained conditional OT model for linear inverse problems on super-resolution (SR), gaussian deblurring (GB), image inpainting - centered mask (IPC) and inpainting - freeform (IPF) with $\sigma_y = 0$. We show results on face-blurred ImageNet-64 (INET-64), face-blurred ImageNet-128 (INET-128), and AFHQ-256 (AFHQ).

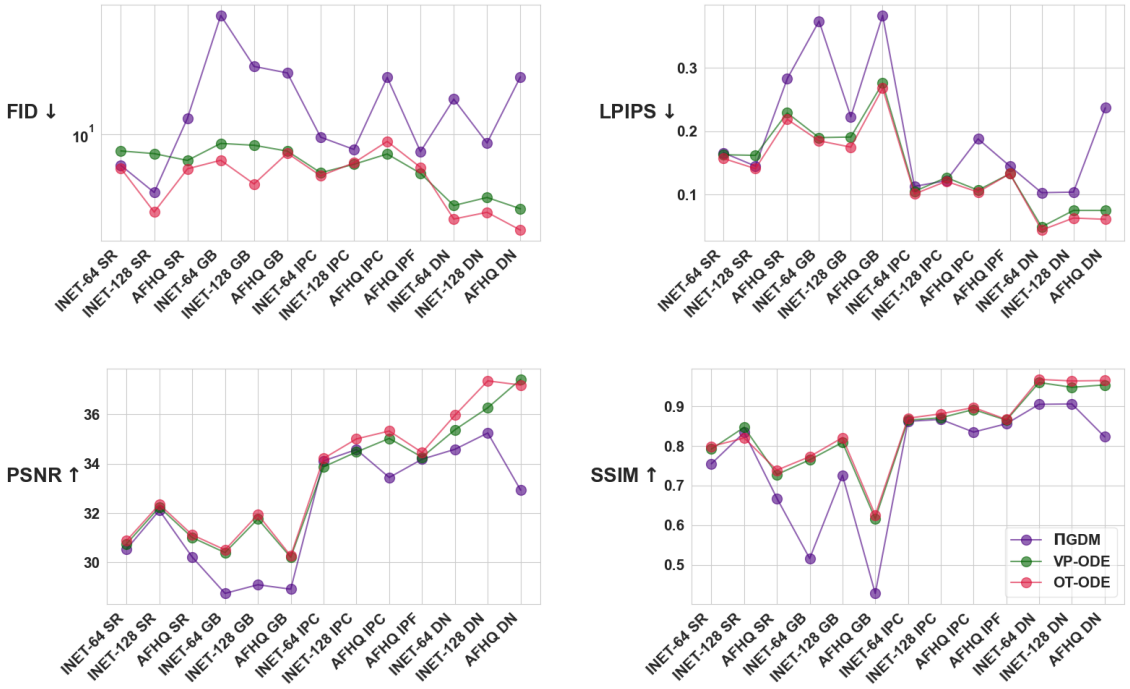


Figure 5.7: Quantitative evaluation of pretrained conditional OT model for linear inverse problems on super-resolution (SR), gaussian deblurring (GB), image inpainting - centered mask (IPC) and denoising (DN) with $\sigma_y = 0.05$. We show results on face-blurred ImageNet-64 (INET-64), face-blurred ImageNet-128 (INET-128), and AFHQ-256 (AFHQ).

Table 5.1: Quantitative evaluation of linear inverse problems on face-blurred ImageNet-64 $\times$ 64

Model	Inference	NFEs ↓	SR 2×, $\sigma_y = 0.05$				Gaussian deblur, $\sigma_y = 0.05$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
OT	OT-ODE	80	**6.07**	**0.157**	**30.88**	**0.799**	**6.83**	**0.185**	**30.51**	**0.773**
OT	VP-ODE	80	7.82	0.163	30.75	0.792	8.72	0.190	30.40	0.765
OT	ΠGDM	100	6.52	0.168	30.54	0.753	55.19	0.374	28.74	0.516
VP-SDE	OT-ODE	80	**5.57**	**0.155**	30.88	0.799	**6.33**	**0.181**	**30.52**	0.773
VP-SDE	VP-ODE	80	7.40	0.160	30.75	0.792	8.16	0.187	30.42	0.766
VP-SDE	ΠGDM	100	6.84	0.174	30.48	0.743	54.77	0.376	28.74	0.511
VP-SDE	RED-Diff	1000	23.02	0.187	**31.22**	**0.839**	51.20	0.236	30.19	**0.776**

Table 5.2: Quantitative evaluation of linear inverse problems on face-blurred ImageNet-64 $\times$ 64

Model	Inference	NFEs ↓	Inpainting-*Center*, $\sigma_y = 0.05$				Denoising, $\sigma_y = 0.05$			
			FID ↓	LPIPS ↓	SSIM ↑	PSNR ↑	FID ↓	LPIPS ↓	SSIM ↑	PSNR ↑
OT	OT-ODE	80	**5.45**	**0.101**	**34.21**	**0.870**	**2.91**	**0.044**	**35.96**	**0.968**
OT	VP-ODE	80	5.70	0.105	33.87	0.865	3.54	0.049	35.37	0.960
OT	ΠGDM	100	9.25	0.111	34.13	0.863	16.59	0.102	34.60	0.906
VP-SDE	OT-ODE	80	**5.03**	**0.098**	**34.25**	**0.872**	**2.76**	**0.042**	**36.02**	**0.969**
VP-SDE	VP-ODE	80	5.26	0.103	33.93	0.866	3.29	0.048	35.45	0.961
VP-SDE	ΠGDM	100	9.75	0.113	34.03	0.860	17.19	0.107	34.25	0.901
VP-SDE	RED-Diff	1000	12.18	0.119	33.97	0.881	6.02	0.041	35.64	0.964

Table 5.3: Quantitative evaluation of linear inverse problems on face-blurred ImageNet-128 $\times$ 128

Model	Inference	NFEs ↓	SR 2×, $\sigma_y = 0.05$				Gaussian deblur, $\sigma_y = 0.05$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
OT	OT-ODE	70	**3.22**	**0.141**	**32.35**	0.820	**4.84**	**0.175**	**31.94**	**0.821**
OT	VP-ODE	70	7.52	0.162	32.24	**0.847**	8.49	0.191	31.76	0.809
OT	ΠGDM	100	4.38	0.148	32.07	0.831	30.30	0.328	29.96	0.606
VP-SDE	OT-ODE	70	**3.21**	**0.139**	**32.40**	**0.855**	**4.49**	**0.173**	**32.02**	**0.824**
VP-SDE	VP-ODE	70	9.14	0.166	32.06	0.838	9.35	0.193	31.66	0.804
VP-SDE	ΠGDM	100	7.55	0.183	31.61	0.785	55.61	0.463	28.57	0.414
VP-SDE	RED-Diff	1000	10.54	0.182	31.82	0.852	21.43	0.229	31.41	0.807

Table 5.4: Quantitative evaluation of linear inverse problems on face-blurred ImageNet-128 $\times$ 128

Model	Inference	NFEs ↓	Inpainting-*Center*, $\sigma_y = 0.05$				Denoising, $\sigma_y = 0.05$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
OT	OT-ODE	70	6.58	**0.121**	**35.00**	**0.881**	**3.21**	**0.063**	**37.35**	**0.964**
OT	VP-ODE	70	**6.44**	0.127	34.47	0.871	3.98	0.075	36.26	0.948
OT	ΠGDM	100	7.99	0.122	34.57	0.867	9.60	0.107	35.11	0.903
VP-SDE	OT-ODE	70	**6.39**	**0.120**	**35.04**	**0.882**	**3.25**	**0.062**	**37.41**	**0.965**
VP-SDE	VP-ODE	70	8.47	0.129	34.43	0.876	5.83	0.087	35.85	0.938
VP-SDE	ΠGDM	100	9.75	0.130	34.45	0.858	10.69	0.124	34.72	0.882
VP-SDE	RED-Diff	1000	14.63	0.171	32.42	0.820	9.19	0.105	33.52	0.895

Table 5.5: Quantitative evaluation of linear inverse problems on AFHQ-256 $\times$ 256

Model	Inference	NFEs $\downarrow$	SR 4$\times$, $\sigma_y = 0.05$				Gaussian deblur, $\sigma_y = 0.05$			
			FID $\downarrow$	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$	FID $\downarrow$	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$
OT	OT-ODE	100	**6.03**	**0.219**	**31.12**	**0.739**	**7.57**	**0.268**	**30.27**	**0.626**
OT	VP-ODE	100	6.81	0.229	31.01	0.728	7.80	0.276	30.21	0.616
OT	ΠGDM	100	12.69	0.285	30.18	0.665	24.60	0.383	28.93	0.429
VP-SDE	OT-ODE	100	**7.28**	**0.238**	30.83	0.714	**8.53**	**0.276**	**30.37**	0.641
VP-SDE	VP-ODE	100	8.02	0.243	**30.96**	**0.727**	10.21	0.289	30.21	0.621
VP-SDE	ΠGDM	100	77.49	0.469	29.34	0.469	116.42	0.535	28.49	0.313
VP-SDE	RED-Diff	1000	20.84	0.331	29.97	0.675	15.81	0.341	30.15	**0.645**

Table 5.6: Quantitative evaluation of linear inverse problems on AFHQ-256 $\times$ 256

Model	Inference	NFEs $\downarrow$	Inpainting-*Center*, $\sigma_y = 0.05$				Denoising, $\sigma_y = 0.05$			
			FID $\downarrow$	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$	FID $\downarrow$	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$
OT	OT-ODE	100	8.98	**0.104**	**35.32**	**0.897**	**2.48**	**0.061**	37.18	**0.965**
OT	VP-ODE	100	**7.48**	0.107	35.02	0.892	3.38	0.075	**37.41**	0.954
OT	ΠGDM	100	19.09	0.153	34.20	0.855	22.87	0.237	32.93	0.823
VP-SDE	OT-ODE	100	9.93	**0.107**	**35.18**	**0.892**	**2.17**	**0.060**	**37.95**	**0.963**
VP-SDE	VP-ODE	100	**8.78**	**0.107**	35.12	0.891	3.08	0.071	37.68	0.959
VP-SDE	ΠGDM	100	57.46	0.239	32.40	0.773	81.15	0.451	29.62	0.639
VP-SDE	RED-Diff	1000	11.02	0.124	34.97	0.893	4.93	0.112	34.18	0.899

Table 5.7: Quantitative evaluation of linear inverse problems on face-blurred ImageNet-64 $\times$ 64

Model	Inference	NFEs $\downarrow$	SR 2$\times$, $\sigma_y = 0$				Gaussian deblur, $\sigma_y = 0$			
			FID $\downarrow$	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$	FID $\downarrow$	LPIPS $\downarrow$	PSNR $\uparrow$	SSIM $\uparrow$
OT	OT-ODE	80	**6.46**	0.119	31.59	0.839	**2.59**	**0.038**	35.31	0.961
OT	VP-ODE	80	8.29	0.147	31.20	0.817	6.13	0.083	33.31	0.929
OT	ΠGDM	100	6.89	**0.115**	**32.02**	**0.853**	4.53	0.051	**35.88**	**0.963**
VP-SDE	OT-ODE	80	**6.32**	0.118	31.60	0.839	**2.61**	**0.037**	35.45	0.963
VP-SDE	VP-ODE	80	7.76	0.145	31.21	0.817	5.68	0.080	33.37	0.931
VP-SDE	ΠGDM	100	6.47	**0.113**	**32.03**	**0.853**	4.35	0.049	**35.95**	**0.964**
VP-SDE	RED-Diff	1000	11.74	0.224	30.12	0.798	15.39	0.134	31.99	0.879

Table 5.8: Quantitative evaluation of linear inverse problems on face-blurred ImageNet-64 $\times$ 64

Model	Inference	NFEs ↓	Inpainting-*Center*, $\sigma_y = 0$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
OT	OT-ODE	80	**4.94**	**0.080**	**37.42**	0.885
OT	VP-ODE	80	7.85	0.120	34.24	0.858
OT	ΠGDM	100	6.09	0.082	36.75	**0.901**
VP-SDE	OT-ODE	80	**4.85**	**0.079**	37.64	0.887
VP-SDE	VP-ODE	80	7.21	0.117	34.33	0.860
VP-SDE	ΠGDM	100	5.79	0.081	36.81	0.902
VP-SDE	RED-Diff	1000	7.29	**0.079**	**39.14**	**0.925**

Table 5.9: Quantitative evaluation of linear inverse problems on face-blurred ImageNet-128 $\times$ 128

Model	Inference	NFEs ↓	Inpainting-*Center*, $\sigma_y = 0$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
OT	OT-ODE	70	5.88	**0.095**	**37.06**	0.894
OT	VP-ODE	70	8.63	0.144	34.48	0.864
OT	ΠGDM	100	**5.82**	0.097	36.89	**0.908**
VP-SDE	OT-ODE	70	5.93	0.094	37.31	0.898
VP-SDE	VP-ODE	70	8.08	0.142	34.55	0.865
VP-SDE	ΠGDM	100	5.74	0.095	37.01	0.911
VP-SDE	RED-Diff	1000	**5.40**	**0.068**	**38.91**	**0.928**

Table 5.10: Quantitative evaluation of linear inverse problems on face-blurred ImageNet-128 $\times$ 128

Model	Inference	NFEs ↓	SR 2×, $\sigma_y = 0$				Gaussian deblur, $\sigma_y = 0$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
OT	OT-ODE	70	**4.46**	**0.097**	33.88	0.903	**2.09**	**0.048**	37.49	**0.961**
OT	VP-ODE	70	7.69	0.144	32.93	0.871	6.02	0.108	34.73	0.925
OT	ΠGDM	100	6.09	0.105	**34.28**	**0.910**	4.28	0.066	**37.56**	**0.961**
VP-SDE	OT-ODE	70	4.62	0.096	33.95	0.906	**2.26**	**0.046**	**37.79**	**0.967**
VP-SDE	VP-ODE	70	7.91	0.144	32.87	0.869	5.64	0.105	34.81	0.928
VP-SDE	ΠGDM	100	6.02	0.104	34.33	0.911	4.35	0.065	37.70	0.963
VP-SDE	RED-Diff	1000	**3.90**	**0.082**	**34.47**	**0.92**	4.19	0.085	34.68	0.929

Table 5.11: Quantitative evaluation of linear inverse problems on AFHQ-256 $\times$ 256

Model	Inference	NFEs ↓	SR 4×, $\sigma_y = 0$				Gaussian deblur, $\sigma_y = 0$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
OT	OT-ODE	100	**5.75**	**0.169**	32.25	0.792	**6.63**	**0.213**	31.29	0.722
OT	VP-ODE	100	6.14	0.194	31.93	0.773	7.38	0.231	31.10	0.705
OT	ΠGDM	100	8.89	0.173	**32.57**	**0.812**	9.78	0.209	**31.54**	**0.743**
VP-SDE	OT-ODE	100	**6.58**	**0.178**	32.18	0.789	**8.24**	**0.226**	31.21	0.717
VP-SDE	VP-ODE	100	8.00	0.225	31.48	0.742	9.19	0.252	30.91	0.688
VP-SDE	ΠGDM	100	10.85	0.189	**32.52**	**0.811**	11.46	0.228	**31.47**	**0.738**
VP-SDE	RED-Diff	1000	8.65	0.191	32.21	0.801	11.67	0.268	31.30	0.731

Table 5.12: Quantitative evaluation of linear inverse problems on AFHQ-256 $\times$ 256

Model	Inference	NFEs ↓	Inpainting-*Center*, $\sigma_y = 0$				Inpainting-*Free-form*, $\sigma_y = 0$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
OT	OT-ODE	100	8.87	**0.061**	**37.45**	0.921	**4.98**	**0.097**	36.15	0.889
OT	VP-ODE	100	9.18	0.106	35.63	0.898	6.92	0.135	34.72	0.869
OT	ΠGDM	100	**7.36**	0.080	**37.45**	**0.933**	6.52	0.100	**36.58**	**0.913**
VP-SDE	OT-ODE	100	9.95	0.064	37.49	0.918	**5.39**	0.099	36.15	0.887
VP-SDE	VP-ODE	100	10.50	0.112	35.59	0.893	7.36	0.139	34.65	0.865
VP-SDE	ΠGDM	100	8.61	0.088	37.27	0.925	7.25	0.109	36.37	**0.906**
VP-SDE	RED-Diff	1000	**8.53**	**0.050**	**38.89**	**0.951**	7.27	**0.090**	**36.88**	0.892

5.8 Related Work

5.8.1 Solving Inverse Problems with Diffusion Models.

The challenge of solving noisy linear inverse problems without any training has been tackled in many ways, often with other solution concepts than posterior sampling [Elad et al., 2023]. Utilizing a diffusion model has a host of recent research that we build upon. Our state-of-the-art baselines ΠGDM [Song et al., 2022] and RED-Diff [Mardani et al., 2023] correspond to lines of research in gradient-based adaptations and variational inference.

Earlier gradient-based adaptations that approximate $\nabla_{\boldsymbol{x}_t} \ln q(\boldsymbol{y}|\boldsymbol{x}_t)$ in various ways include Diffusion Posterior Sampling (DPS) [Chung et al., 2022a], Manifold Constrained Gradient [Chung et al., 2022b], and an annealed approximation [Jalal et al., 2021]. ΠGDM out-performs earlier methods combining adaptive weights and Gaussian posterior approximation with discrete-time denoising diffusion implicit model (DDIM) sampling [Song et al., 2020a]. Here we adapt ΠGDM to all Gaussian probability paths and to flow sampling. Our results show adaptive weights are unnecessary for strongly performing conditional OT flow sampling. Denoising Diffusion Null Models (DDNM) [Wang et al., 2022] proposed an alternative approximation of $\mathbb{E}_q[\boldsymbol{x}_1|\boldsymbol{x}_t, \boldsymbol{y}]$ using a null-space decomposition specific to linear inverse problems, which has been explored in combination with our method in Appendix 5.11.

RED-Diff [Mardani et al., 2023] approximates intractable $q(\boldsymbol{x}_1|\boldsymbol{y})$ directly using variational inference, solving for parameters via optimization. RED-Diff was reported to have mode-seeking behavior confirmed by our results where RED-Diff performed better for noiseless inference. Another earlier variational inference method is Denoising Diffusion Restoration Models (DDRM) [Kawar et al., 2022]. DDRM showed SVD can be memory-efficient for image applications, and we adapt their SVD implementations for super-resolution and blur. DDRM incorporates noiseless method ILVR [Choi et al., 2021], and leverages a measurement-dependent forward process (i.e. $q(\boldsymbol{x}_t|\boldsymbol{x}_1, \boldsymbol{y}) \neq q(\boldsymbol{x}_t|\boldsymbol{x}_1)$) like earlier SNIPS [Kawar et al., 2021]. SNIPS collapses in special cases to variants proposed in Song and Ermon [2019], Song et al. [2020b], Kadkhodaie and Simoncelli [2020] for linear inverse problems.

5.8.2 Classical Approaches for Solving Inverse Problems.

Inverse problems are ubiquitous in various domains like image processing [Krishnan and Fergus, 2009, Rick Chang et al., 2017, Gilton et al., 2019, Bertalmio et al., 2000, Yang et al., 2010], medical imaging [Ribes and Schmitt, 2008, Jin et al., 2017, Liang et al., 2020, Song et al., 2021b], and remote sensing [Krasnopolsky, 2009, Krasnopolsky and Schiller, 2003, Dong et al., 2018]. Many approaches have been developed over years to solve inverse problems. Variational methods [Agrawal et al., 2022] formulate the inverse problem as an optimization task with a regularization term [Benning and Burger, 2018] for certain desirable properties in the solution. Some of the well-known frameworks in this category include plug-and-play prior (P^3) [Venkatakrishnan et al., 2013, Chan et al., 2016, Kamilov et al., 2017, Meinhardt et al., 2017, Zhang et al., 2017, Vidal et al., 2020], deep image prior (DIP) [Ulyanov et al., 2018, Van Veen et al., 2018], and regularization by denoising (RED) [Romano et al., 2017, Cohen et al., 2021]. Subsequent works, inspired by these prior works, have extended these frameworks to include flow models [Whang et al., 2021a,b], optimal transport [Vidal et al., 2020], and

more recently, diffusion models [Mardani et al., 2023, Graikos et al., 2022, Liu et al., 2023c] as prior. Optimization-based inversion methods were also extended to include GANs [Bora et al., 2017, Shah and Hegde, 2018, Raj et al., 2019, Daras et al., 2021, Pan et al., 2021]. Optimization-based approaches for solving inverse problems, despite their widespread popularity, have certain drawbacks. These methods are often computationally expensive as they involve optimizing an objective, which might require many steps to converge to a solution. Further, designing the optimization objective itself can be challenging. In addition, these methods are sensitive to the choice of hyperparameters like regularization parameter, as noted in our experiments with RED-Diff [Mardani et al., 2023].

With emergence of diffusion models, another family of gradient-based approaches for inverse problems have emerged. These approaches do not explicitly optimize an objective, *i.e.,* they are training-free, but they use gradients to guide the sampling process with diffusion model as prior. These approaches usually involve iterative denoising through a SDE and gradient-based correction that is applied at each step of the process. Some of the approaches in this category include Diffusion Posterior Sampling (DPS) [Chung et al., 2022a], Manifold Constraint Gradient (MCG) [Chung et al., 2022b], ΠGDM [Song et al., 2022], and shortcut sampling for diffusion (SSD) [Liu et al., 2023a]. Our proposed approach for solving linear inverse problems with flow models also falls under this category. Computing gradients at each step of denoising can be expensive. There are many gradient-free iterative methods for inversion that utilize diffusion models as generative prior. Some prominent approaches in this category are denoising diffusion restoration models (DDRM) [Kawar et al., 2022] and denoising diffusion null-space model (DDNM) [Wang et al., 2022]. We have covered important distinctions between these approaches briefly in Sec. 5.8 of the main paper.

The aforementioned approaches for solving inverse problems with diffusion models use pre-trained diffusion models and are not specific to a particular measurement operator. There are works such as [Saharia et al., 2021, 2022a] that train a conditional diffusion model to solve a specific inverse problem. This approach for solving inverse problems is more computationally expensive as it involves training a model from scratch. Further, the resulting model is specific to the measurement operator used in the training data and cannot be reused to solve inverse problems with a different measurement operator.In addition to the above, there is also a line of research that considers the more general setting of blind inverse problem where the method to solve an inverse problem is agnostic to the measurement operator. Some works that have advanced this line of research are Chung et al. [2023], Gan et al. [2024], Laroche et al. [2024]. Finally, we note that there are previously proposed methods such as Whang et al. [2021a,b], Hong et al. [2023] which solve inverse problems using CNFs. As noted in these prior works, using CNFs for solving inverse problems presents computational challenges as well as challenges due to restricted architecture. In this work, we consider CNFs that are trained with flow matching (or similarly converted diffusion models) which are more computationally more efficient and do not suffer from drawbacks observed due to restricted architectures.

5.9 Ablation Study

5.9.1 Choice of initialization

We initialize the flow at time $t > 0$ as $\boldsymbol{x}_t = \alpha_t \boldsymbol{y} + \sigma_t \epsilon$ (y-init) where $\epsilon \sim \mathcal{N}(0, \boldsymbol{I})$. Another choice of initialization is to use $\boldsymbol{x}_t = \alpha_t \boldsymbol{A}^\dagger \boldsymbol{y} + \sigma_t \epsilon$. However, empirically we find that this initialization performs worse that y-init on cond-OT model with OT-ODE sampling. We summarize the results of our ablation study in Table 5.13. We find that on Gaussian deblurring, initialization with $\boldsymbol{A}^\dagger \boldsymbol{y}$ does worse than y-init, while the performance of both the initializations is comparable for super-resolution. In all our experiments, we use y-init, due to its better performance on Gaussian deblurring.

Table 5.13: Quantitative evaluation of choice of initialization for conditional OT flow model with OT-ODE sampling on AFHQ dataset. We find that y-init outperforms $\boldsymbol{A}^\dagger \boldsymbol{y}$ on Gaussian deblurring.

Initialization	Start time	NFEs ↓	Gaussian deblur, $\sigma_y = 0.05$				SR 4×, $\sigma_y = 0.05$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
y init	0.2	100	**7.57**	**0.268**	**30.28**	**0.626**	**6.03**	**0.219**	**31.12**	**0.739**
$\boldsymbol{A}^\dagger \boldsymbol{y}$	0.1	100	41.22	0.449	28.79	0.392	12.93	0.292	30.46	0.664
$\boldsymbol{A}^\dagger \boldsymbol{y}$	0.2	100	56.42	0.554	28.11	0.249	6.09	0.219	31.12	0.739

5.9.2 Ablation over γ_t for VP-ODE sampling

We compare the performance of $\gamma_t = 1$ against $\gamma_t = \sqrt{\frac{\alpha_t}{\alpha_t^2 + \sigma_t^2}}$. We show results of VP-ODE sampling with VP-SDE model in Table 5.14 and Table 5.15. As seen our choice of γ_t outperform $\gamma_t = 1$ across all the metrics on face-blurred ImageNet-128.

Table 5.14: Quantitative evaluation of value of γ_t in VP-ODE sampling with VP-SDE model on face-blurred ImageNet-128 dataset.

γ_t	Start time	NFEs ↓	SR 2×, $\sigma_y = 0.05$				Gaussian deblur, $\sigma_y = 0.05$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
1	0.4	60	32.66	0.371	29.06	0.530	29.31	0.346	29.12	0.554
$\sqrt{\frac{\alpha_t}{\alpha_t^2 + \sigma_t^2}}$	0.4	60	**9.14**	**0.167**	**32.06**	**0.838**	**10.14**	**0.196**	**31.59**	**0.800**

Table 5.15: Quantitative evaluation of value of γ_t in VP-ODE sampling with VP-SDE model on face-blurred ImageNet-128 dataset.

γ_t	Start time	NFEs ↓	Inpainting-*Center*, $\sigma_y = 0.05$				Denoising, $\sigma_y = 0.05$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
1	0.3	70	53.03	0.285	31.55	0.737	28.37	0.238	31.63	0.786
$\sqrt{\frac{\alpha_t}{\alpha_t^2 + \sigma_t^2}}$	0.3	70	**8.47**	**0.129**	**34.43**	**0.876**	**5.83**	**0.087**	**35.85**	**0.938**

5.9.3 Variation of performance with NFEs

We analyze the variation in performance of OT-ODE, VP-ODE and ΠGDM for solving linear inverse problems as NFEs are varied. The results have been summarized in Figure 5.8. We observe that OT-ODE consistently outperforms VP-ODE and ΠGDM across all measurements in terms of FID and LPIPS metrics, even for NFEs as small as 20. We also note that the choice of starting time matters to achieve good performance with OT-ODE. For instance, starting at $t = 0.4$ outperforms $t = 0.2$ when NFEs are small, but eventually as NFEs is increased, $t = 0.2$ performs better. We also note that ΠGDM achieves higher values of PSNR and SSIM at smaller NFEs for super-resolution but has inferior FID and LPIPS compared to OT-ODE.

5.9.4 Choice of starting time

We plot the variation in performance of OT-ODE and VP-ODE sampling with change in start times for conditional OT model and VP-SDE model on AFHQ dataset in Figure 5.9 and Figure 5.10, respectively. We note that in general, OT-ODE sampling achieves optimal performance across all measurements and all metrics at $t = 0.2$ while VP-ODE sampling achieves optimal performance between start times of $t = 0.3$ and 0.4. In this work, for all the experiments, we use $t = 0.2$ for OT-ODE sampling and $t = 0.4$ for VP-ODE sampling.

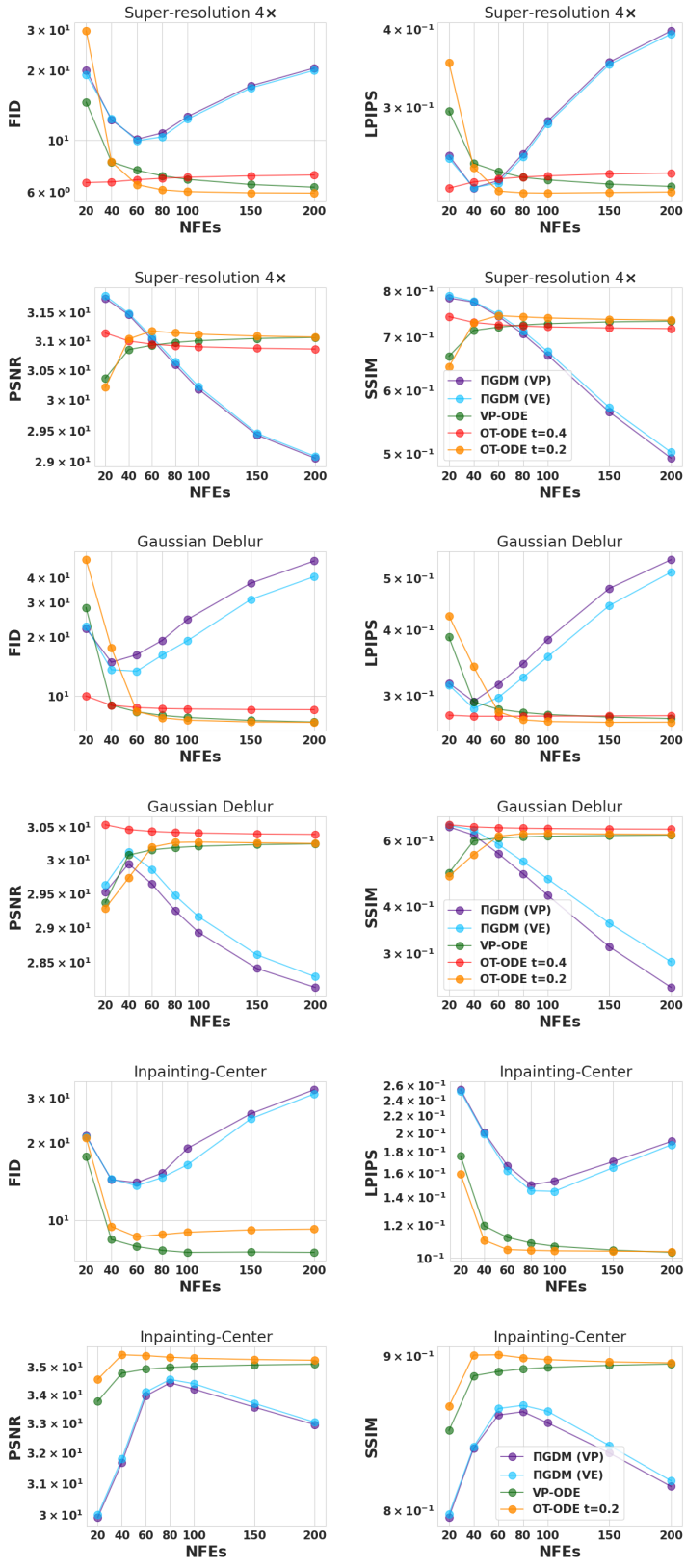


Figure 5.8: Performance of different procedures for solving linear inverse problems with variation in NFEs on AFHQ dataset. We use pretrained conditional OT model and set $\sigma_y = 0.05$. The legends VP and VE indicate the choice of r_t^2 used in ΠGDM (See Sec. 5.12.1). Time $t = 0.2$ and 0.4 indicates the starting time of sampling with OT-ODE.

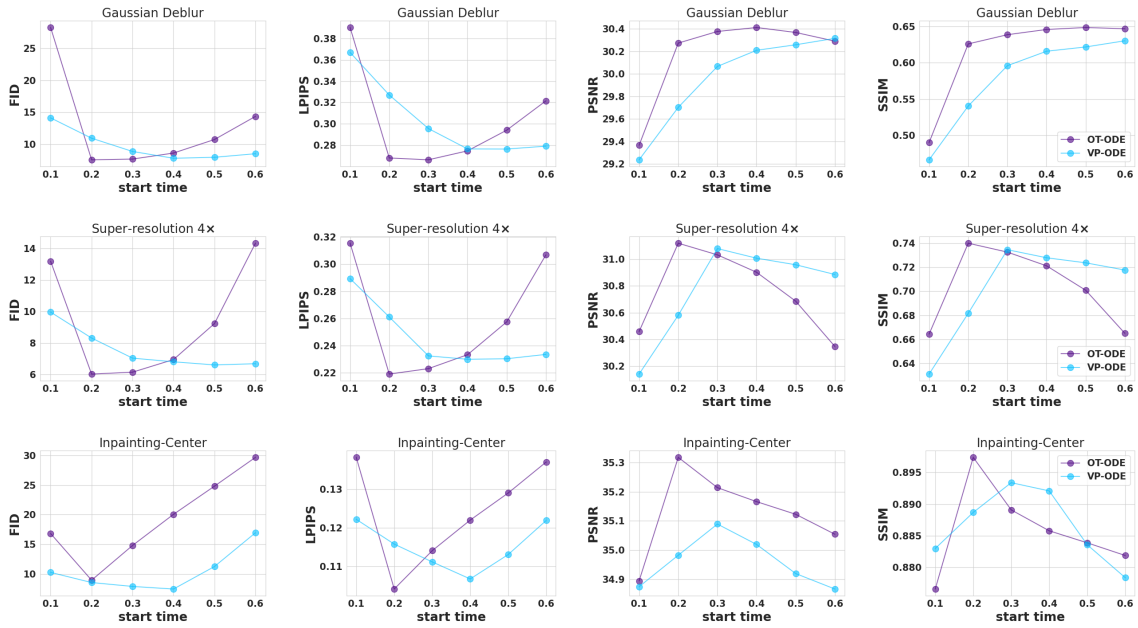


Figure 5.9: Performance of OT-ODE and VP-ODE in solving linear inverse problems with varying start times on AFHQ dataset. We use pretrained cond-OT model and set $\sigma_y = 0.05$.

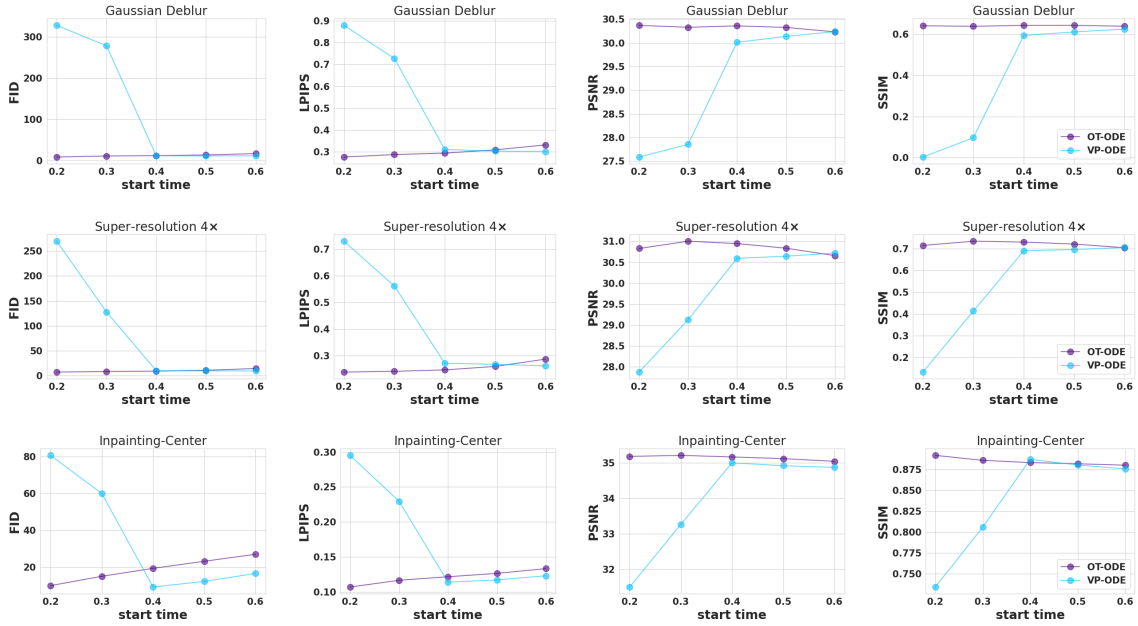
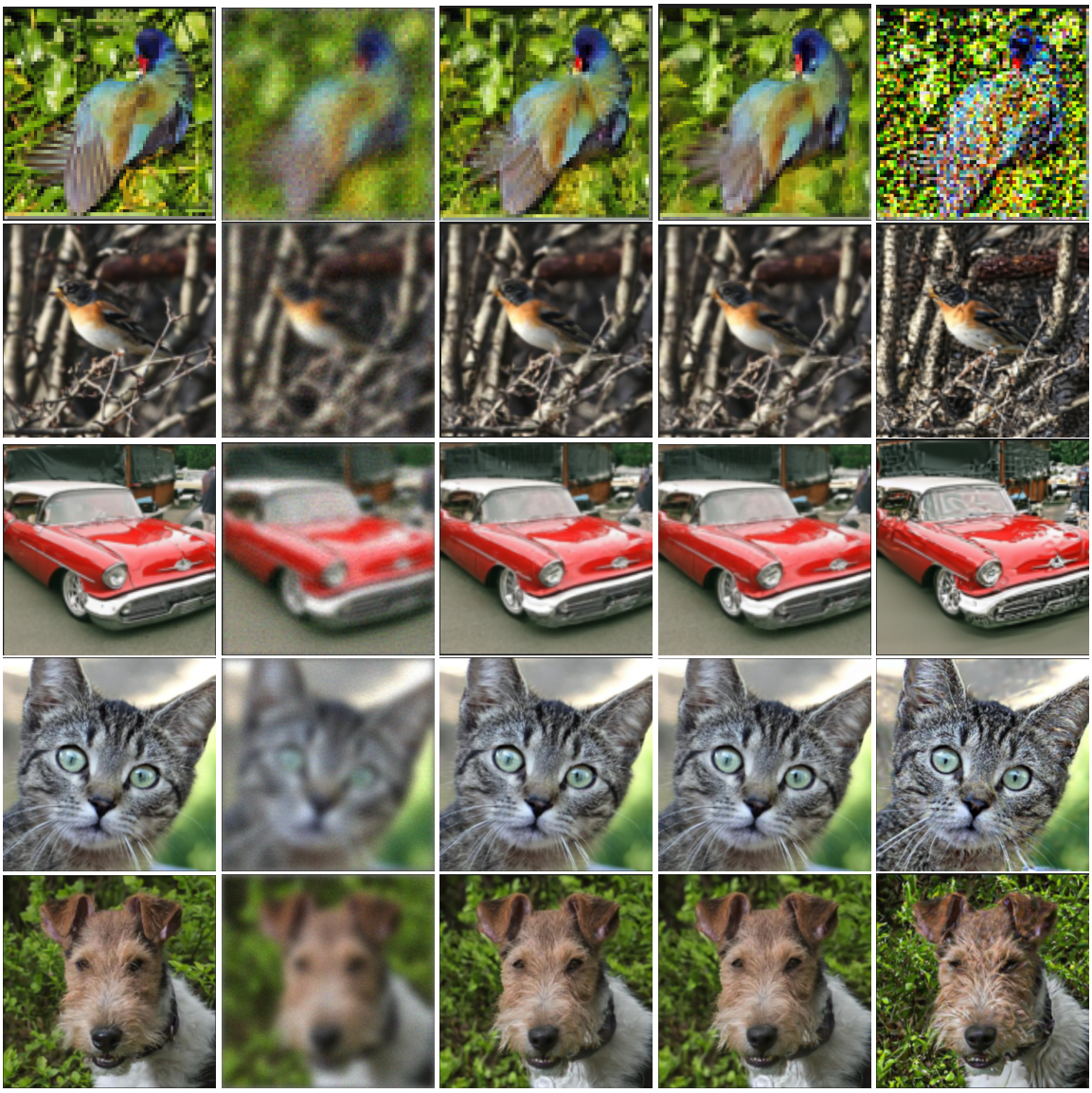


Figure 5.10: Performance of OT-ODE and VP-ODE in solving linear inverse problems with varying start times on AFHQ dataset. We use pretrained VP-SDE model and set $\sigma_y = 0.05$.

5.10 Additional Qualitative Results

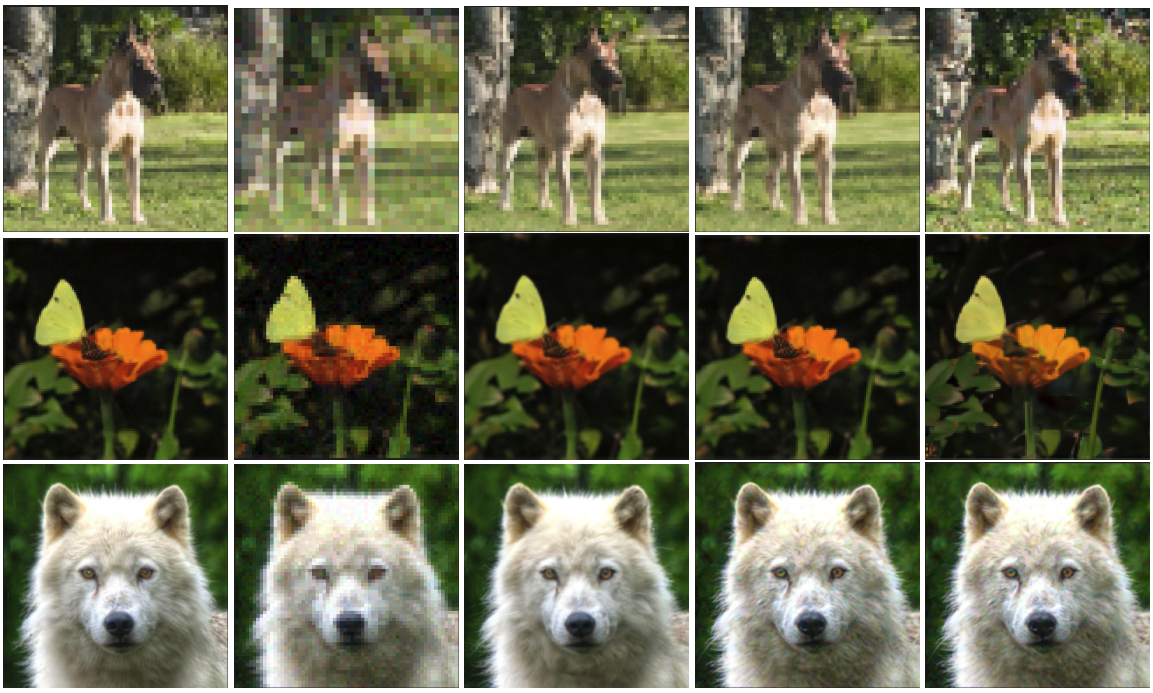
5.10.1 Qualitative Results for Gaussian Deblur



(a) Reference (b) Distorted (c) OT-ODE (d) VP-ODE (e) ΠGDM

Figure 5.11: Gaussian-deblur with conditional OT model and $\sigma_y = 0.05$ for (**first row**) face-blurred ImageNet-64, (**second and third row**) face-blurred ImageNet-128, and (**fourth and fifth row**) AFHQ.

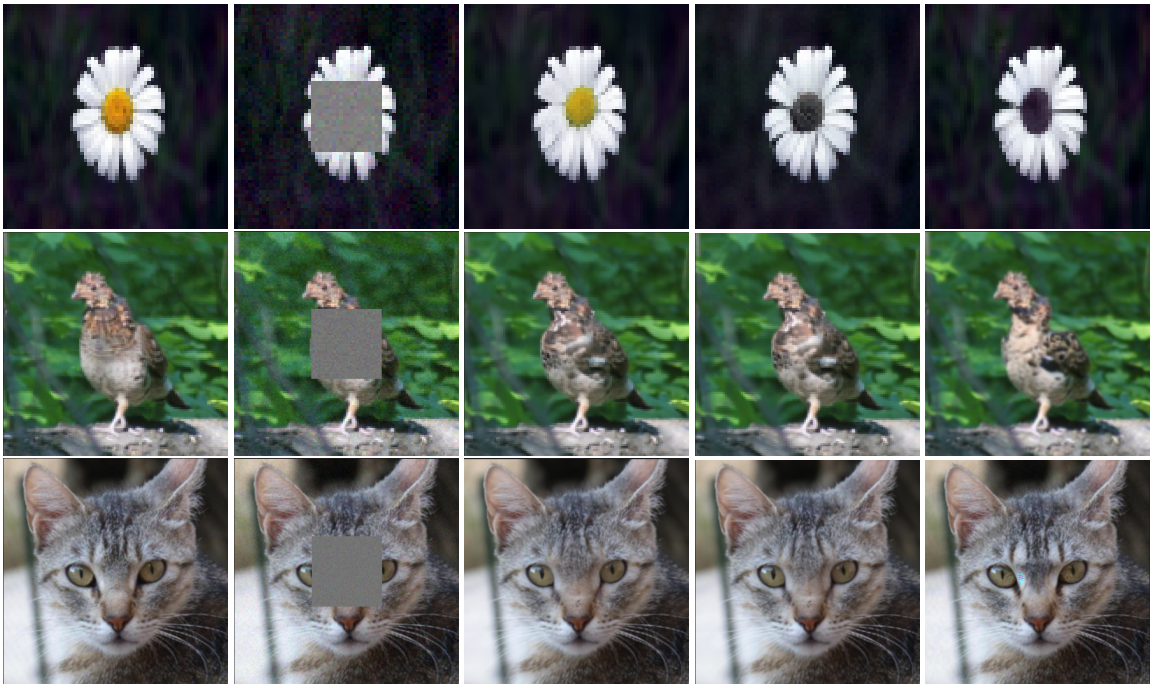
5.10.2 Qualitative Results for Super-resolution



(a) Reference (b) Distorted (c) OT-ODE (d) VP-ODE (e) ΠGDM

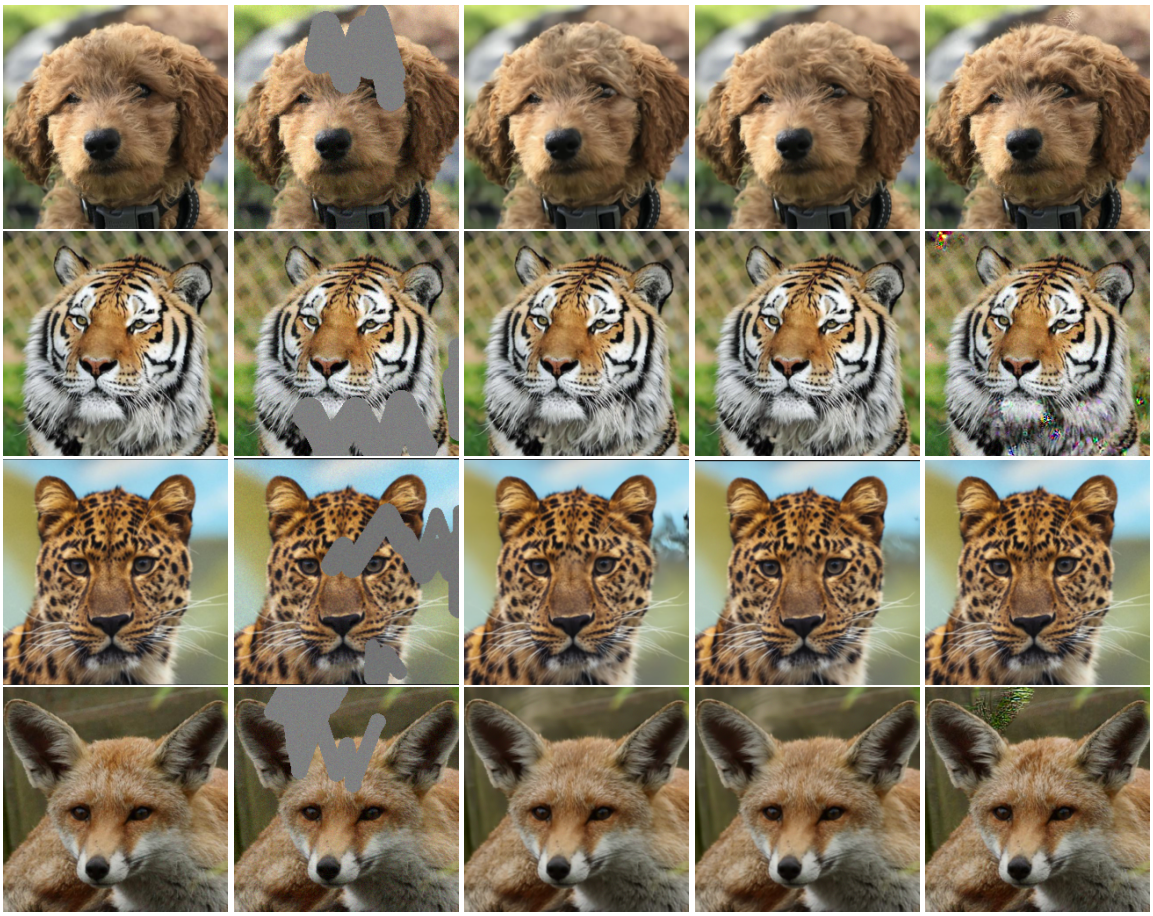
Figure 5.12: Super-resolution with conditional OT model and $\sigma_y = 0.05$ for (**first row**) face-blurred ImageNet-64 2×, (**second row**) face-blurred ImageNet-128 2×, and (**third row**) AFHQ 4×.

5.10.3 Qualitative Results for Inpainting



(a) Reference (b) Distorted (c) OT-ODE (d) VP-ODE (e) ΠGDM

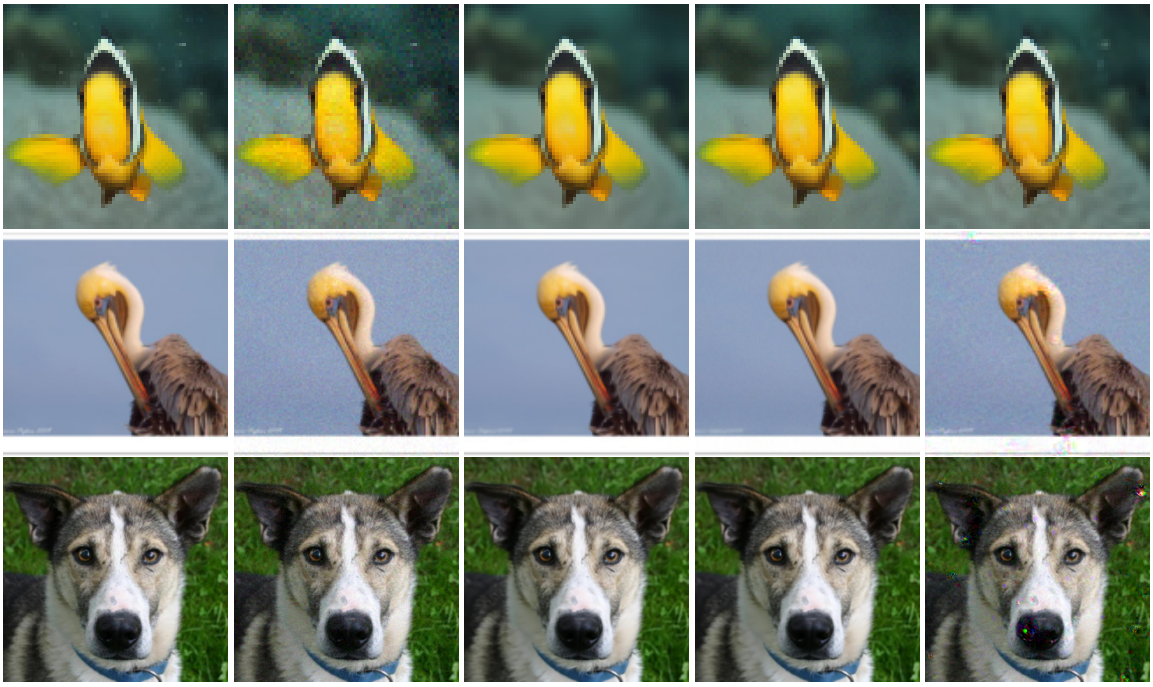
Figure 5.13: Inpainting (Center mask) with conditional OT model and $\sigma_y = 0.05$ for (**first row**) face-blurred ImageNet-64, (**second row**) face-blurred ImageNet-128, and (**third row**) AFHQ.



(a) Reference (b) Distorted (c) OT-ODE (d) VP-ODE (e) ΠGDM

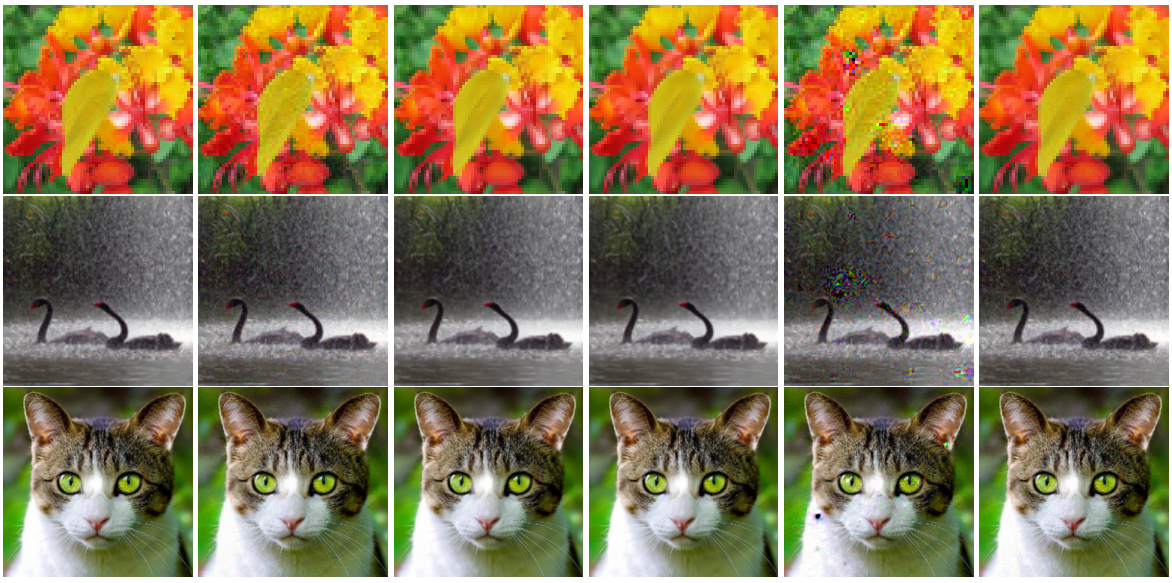
Figure 5.14: Inpainting (Free-form mask) with conditional OT model and $\sigma_y = 0.05$ for AFHQ.

5.10.4 Qualitative Results for Denoising



(a) Reference (b) Distorted (c) OT-ODE (d) VP-ODE (e) ΠGDM

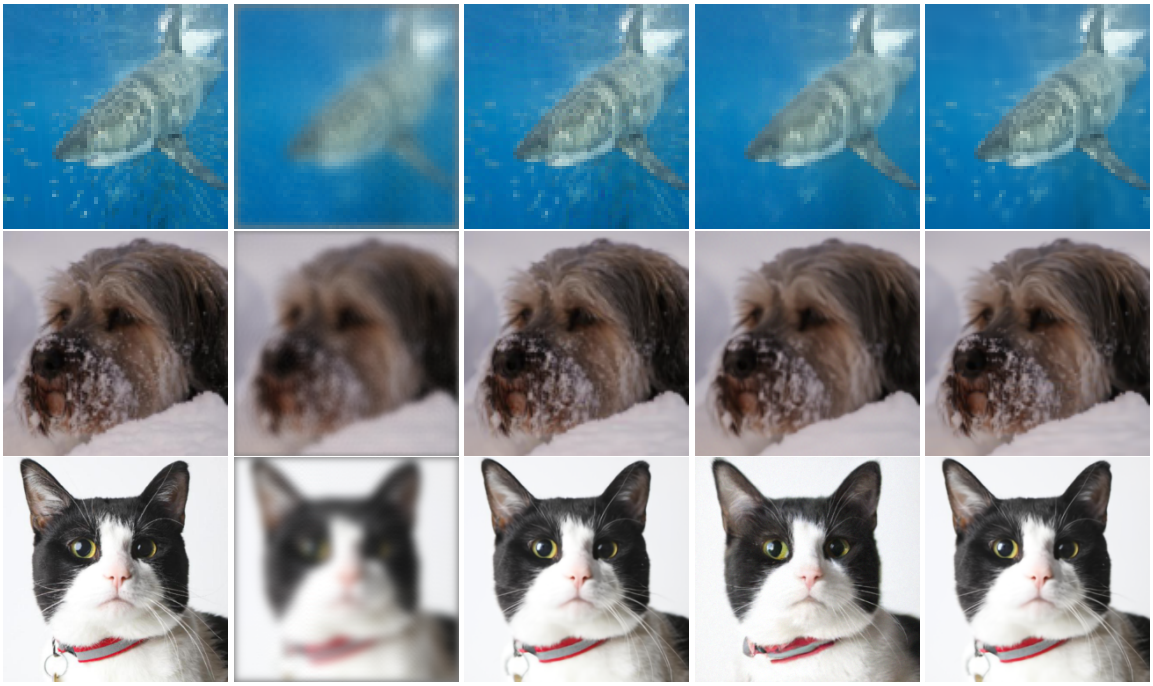
Figure 5.15: Denoising with conditional OT model and $\sigma_y = 0.05$ for (**first row**) face-blurred ImageNet-64, (**second row**) face-blurred ImageNet-128, and (**third row**) AFHQ.



(a) Reference (b) Distorted (c) OT-ODE (d) VP-ODE (e) ΠGDM (f) RED-Diff

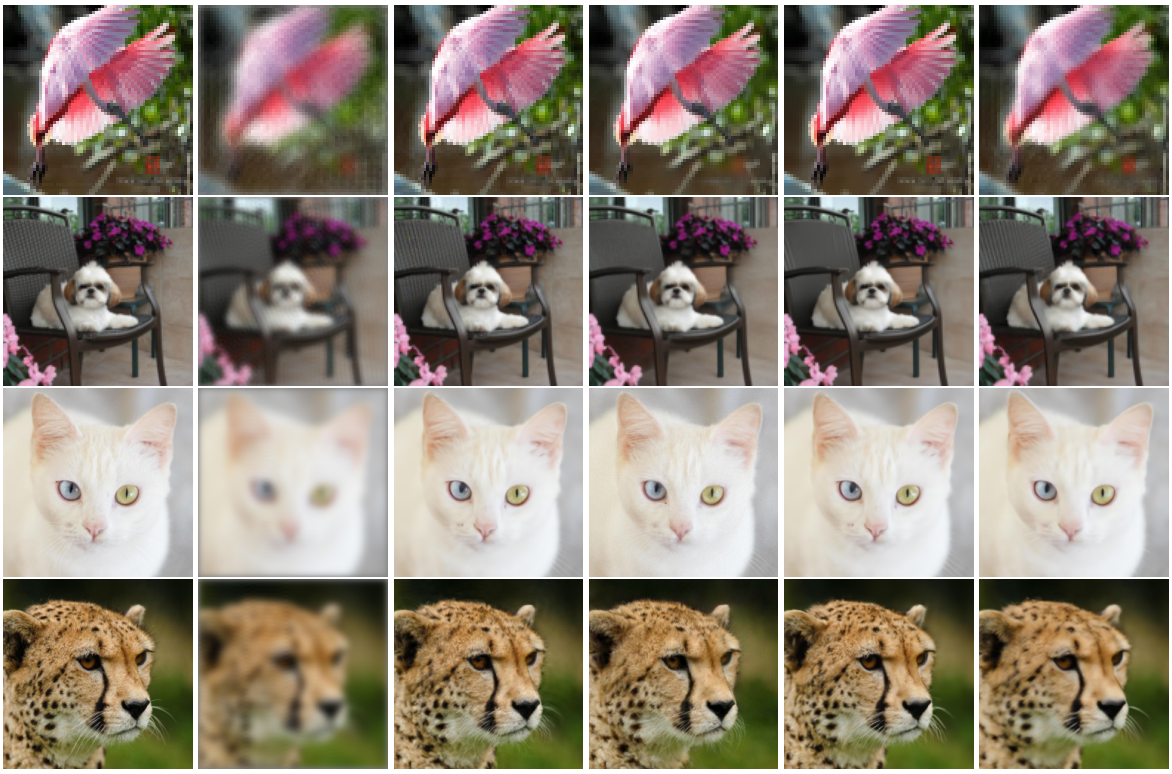
Figure 5.16: Denoising with pretrained VP-SDE model and $\sigma_y = 0.05$ for (**first row**) face-blurred ImageNet-64, (**second row**) face-blurred ImageNet-128, and (**third row**) AFHQ.

5.10.5 Qualitative Results for Noiseless Gaussian Deblur



(a) Reference (b) Distorted (c) OT-ODE (d) VP-ODE (e) ΠGDM

Figure 5.17: Gaussian deblurring with conditional OT model and $\sigma_y = 0$ for (**first row**) face-blurred ImageNet-64, (**second row**) face-blurred ImageNet-128 and (**third row**) AFHQ.



(a) Reference (b) Distorted (c) OT-ODE (d) VP-ODE (e) ΠGDM (f) RED-Diff

Figure 5.18: Gaussian deblurring with VP-SDE model and $\sigma_y = 0$ for (**first row**) face-blurred ImageNet-64, (**second row**) face-blurred ImageNet-128 and (**third and fourth row**) AFHQ.

5.10.6 Qualitative Results for Noiseless Super-resolution

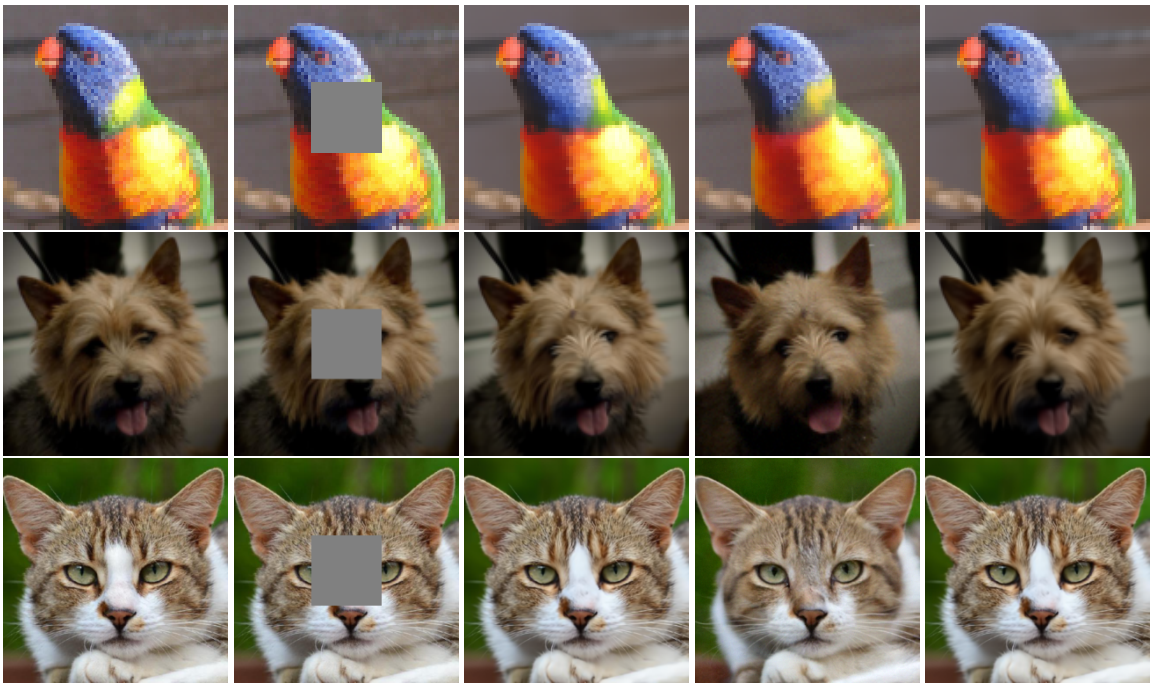
(a) Reference (b) Distorted (c) OT-ODE (d) VP-ODE (e) ΠGDM

Figure 5.19: Super-resolution with conditional OT model and $\sigma_y = 0$ for (**first row**) face-blurred ImageNet-64, (**second row**) face-blurred ImageNet-128 and (**third row**) AFHQ.

(a) Reference (b) Distorted (c) OT-ODE (d) VP-ODE (e) ΠGDM (f) RED-Diff

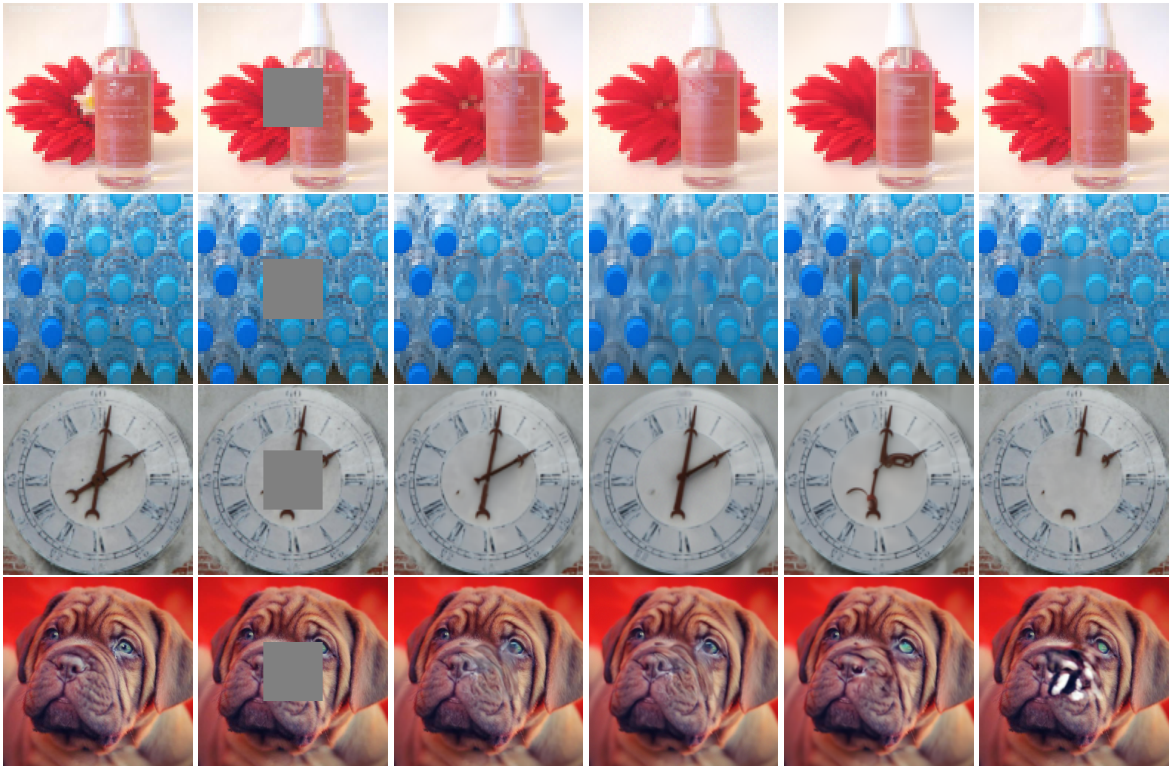
Figure 5.20: Super-resolution with VP-SDE model and $\sigma_y = 0$ for (**first row**) face-blurred ImageNet-64, (**second row**) face-blurred ImageNet-128 and (**third row**) AFHQ.

5.10.7 Qualitative Results for Noiseless Inpainting



(a) Reference (b) Distorted (c) OT-ODE (d) VP-ODE (e) ΠGDM

Figure 5.21: Inpainting (centered mask) with conditional OT model and $\sigma_y = 0$ for (**first row**) face-blurred ImageNet-64, (**second row**) face-blurred ImageNet-128 and (**third row**) AFHQ.



(a) Reference (b) Distorted (c) OT-ODE (d) VP-ODE (e) ΠGDM (f) RED-Diff

Figure 5.22: Inpainting (centered mask) with VP-SDE model and $\sigma_y = 0$ for (**first and second row**) face-blurred ImageNet-64, (**third row**) face-blurred ImageNet-128 and (**fourth row**) AFHQ.

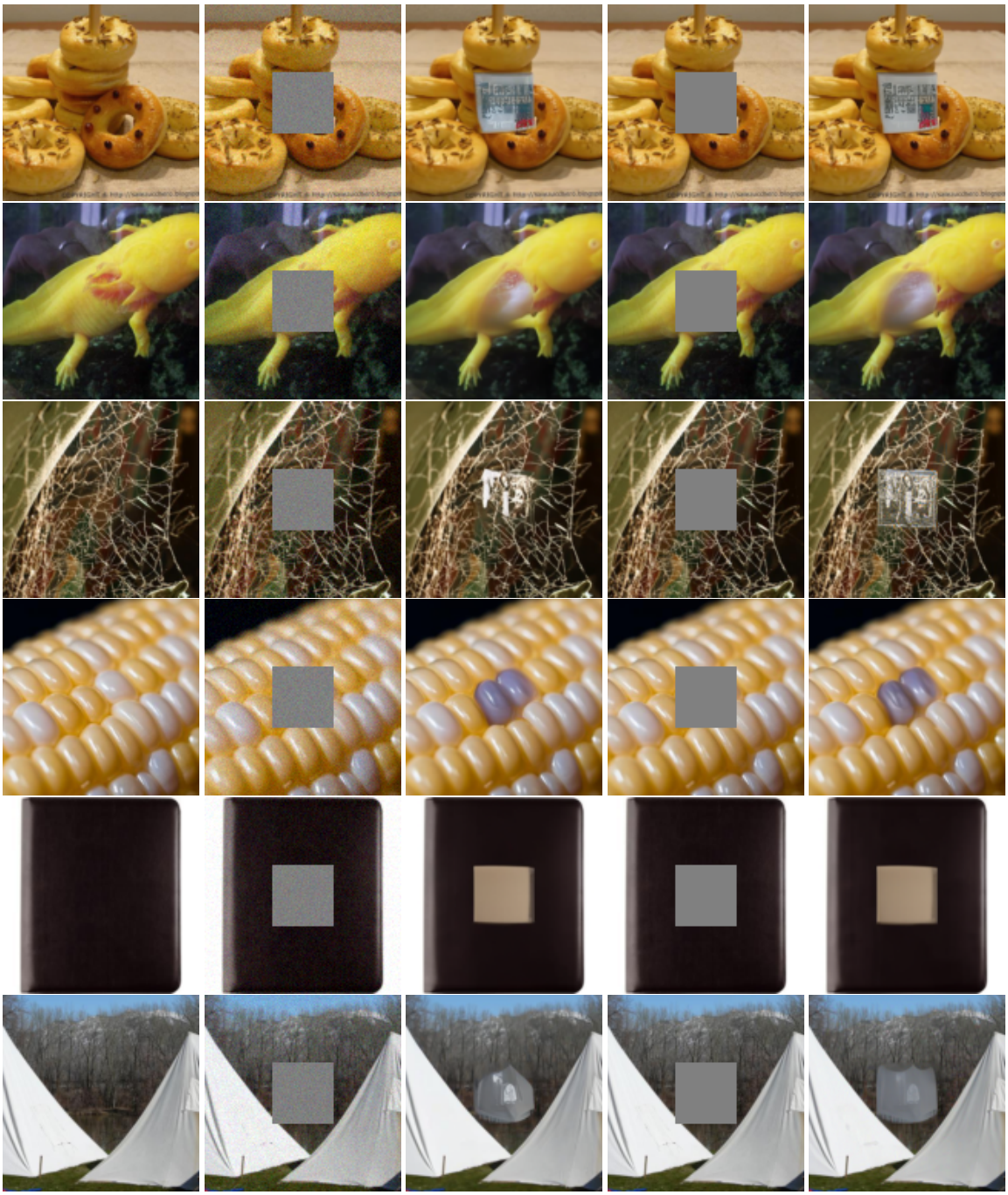
(a) Reference (b) Distorted (c) OT-ODE (d) VP-ODE (e) ΠGDM (f) RED-Diff

Figure 5.23: Inpainting (freeform mask) with VP-SDE model and $\sigma_y = 0$ for AFHQ.

5.10.8 Negative results from Inpainting

(a) Reference (b) Masked-noisy (c) Corrected-noisy (d) Masked $\sigma_y = 0$ (e) Corrected $\sigma_y = 0$

Figure 5.24: Negative results for inpainting with OT-ODE on AFHQ. We can observe artifacts in high-resolution images where the masked region is not inpainted correctly and there are patches in the inpainted region that are semantically incorrect. The observed artifacts are present in both the noiseless (e) and noisy (c) columns.



(a) Reference (b) Masked-noisy (c) Corrected-noisy (d) Masked $\sigma_y = 0$ (e) Corrected $\sigma_y = 0$

Figure 5.25: Negative results for inpainting with OT-ODE on face-blurred ImageNet-128. We can observe artifacts in high-resolution images where the masked region is not inpainted correctly and there are patches in the inpainted region that are semantically incorrect. The observed artifacts are present in both the noiseless (e) and noisy (c) columns.

5.11 Noiseless null and range space decomposition

When $\sigma_y^2 = 0$, we can produce a vector field approximation with even lower Conditional Flow Matching loss by applying a null-space and range-space decomposition motivated by DDNM [Wang et al., 2022].

In particular, when $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}_1$, we have that $\boldsymbol{A}^\dagger \boldsymbol{y} = \boldsymbol{A}^\dagger \boldsymbol{A}\boldsymbol{x}_1$ (where $\boldsymbol{A}^\dagger$ is the pseudo-inverse of $\boldsymbol{A}$) and so

$$\mathbb{E}_q[\boldsymbol{x}_1|\boldsymbol{x}_t, \boldsymbol{y}] = \mathbb{E}_q[\boldsymbol{A}^\dagger \boldsymbol{A}\boldsymbol{x}_1 + (\boldsymbol{I} - \boldsymbol{A}^\dagger \boldsymbol{A})\boldsymbol{x}_1|\boldsymbol{x}_t, \boldsymbol{y}] = \boldsymbol{A}^\dagger \boldsymbol{y} + (\boldsymbol{I} - \boldsymbol{A}^\dagger \boldsymbol{A})\mathbb{E}_q[\boldsymbol{x}_1|\boldsymbol{x}_t, \boldsymbol{y}]. \tag{5.19}$$

So when $\sigma_{\boldsymbol{y}}^2 = 0$, it is only necessary to approximate the second term, as the first term is known through $\boldsymbol{y}$. The regression loss is minimized for the first term automatically and $\widehat{\boldsymbol{x}_1}(\boldsymbol{x}_t, \boldsymbol{y})$ is only responsible for predicting the second term.

In our experiments, we find that null space decomposition helps in inpainting but not other measurements. We summarize the results in Table 5.16 to 5.21 and show qualitative results for inpainting in Figure 5.26 to 5.28.

Table 5.16: Comparison of performance OT-ODE sampling and OT-ODE sampling with null and range space decomposition (NRSD) on face-blurred ImageNet-64 $\times$ 64. For inpainting, OT-ODE sampling with null and range space decomposition outperforms simple OT-ODE sampling.

Model	Inference	NFEs ↓	Inpainting-*Center*, $\sigma_y = 0$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
OT	OT-ODE	80	4.94	0.080	37.42	0.885
OT	OT-ODE-NRSD	80	**3.84**	**0.072**	**38.23**	**0.888**
OT	VP-ODE	80	7.85	0.120	34.24	0.858
VP-SDE	OT-ODE	80	4.85	0.079	37.64	0.887
VP-SDE	OT-ODE-NRSD	80	**3.77**	**0.072**	**38.24**	**0.888**
VP-SDE	VP-ODE	80	7.21	0.117	34.33	0.860

Table 5.17: Comparison of performance OT-ODE sampling and OT-ODE sampling with null and range space decomposition (NRSD) on face-blurred ImageNet-64 $\times$ 64. For tasks like super-resolution and Gaussian deblurring, OT-ODE sampling without null and range space decomposition outperforms other methods.

Model	Inference	NFEs ↓	SR 2×, $\sigma_y = 0$				Gaussian deblur, $\sigma_y = 0$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
OT	OT-ODE	80	**6.46**	**0.119**	**31.59**	**0.839**	**2.59**	**0.038**	**35.31**	**0.961**
OT	OT-ODE-NRSD	80	7.37	0.134	31.05	0.799	3.05	0.044	35.19	0.956
OT	VP-ODE	80	8.29	0.147	31.20	0.817	6.13	0.083	33.31	0.929
VP-SDE	OT-ODE	80	**6.32**	**0.118**	**31.60**	**0.839**	**2.61**	**0.037**	**35.45**	**0.963**
VP-SDE	OT-ODE-NRSD	80	7.13	0.133	31.06	0.798	2.99	0.044	35.24	0.956
VP-SDE	VP-ODE	80	7.76	0.145	31.21	0.817	5.68	0.080	33.37	0.931

Table 5.18: Comparison of performance OT-ODE sampling and OT-ODE sampling with null and range space decomposition (NRSD) on face-blurred ImageNet-128 $\times$ 128.

Model	Inference	NFEs ↓	SR 2×, $\sigma_y = 0$				Gaussian deblur, $\sigma_y = 0$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
OT	OT-ODE	70	4.46	**0.097**	**33.88**	**0.903**	2.09	0.048	37.49	0.961
OT	OT-ODE-NRSD	70	**3.62**	0.099	33.24	0.876	**1.42**	**0.036**	**38.35**	**0.969**
OT	VP-ODE	70	7.69	0.144	32.93	0.871	6.02	0.108	34.73	0.925
VP-SDE	OT-ODE	70	4.62	**0.096**	**33.95**	**0.906**	2.26	0.046	37.79	0.967
VP-SDE	OT-ODE-NRSD	70	**3.44**	0.098	33.28	0.877	**1.36**	**0.035**	**38.44**	**0.969**
VP-SDE	VP-ODE	70	7.91	0.144	32.87	0.869	5.64	0.105	34.81	0.928

Table 5.19: Comparison of performance OT-ODE sampling and OT-ODE sampling with null and range space decomposition (NRSD) on face-blurred ImageNet-128 $\times$ 128

Model	Inference	NFEs ↓	Inpainting-*Center*, $\sigma_y = 0$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
OT	OT-ODE	70	5.88	0.095	37.06	0.894
OT	OT-ODE-NRSD	70	**3.95**	**0.074**	**38.27**	**0.906**
OT	VP-ODE	70	8.63	0.144	34.48	0.864
VP-SDE	OT-ODE	70	5.93	0.094	37.31	0.898
VP-SDE	OT-ODE-NRSD	70	**3.84**	**0.073**	**38.27**	**0.906**
VP-SDE	VP-ODE	70	8.08	0.142	34.55	0.865

Table 5.20: Comparison of performance OT-ODE sampling and OT-ODE sampling with null and range space decomposition (NRSD) on AFHQ-256 $\times$ 256

Model	Inference	NFEs ↓	SR 4×, $\sigma_y = 0$				Gaussian deblur, $\sigma_y = 0$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
OT	OT-ODE	100	5.75	**0.169**	**32.25**	**0.792**	**6.63**	**0.213**	**31.29**	**0.722**
OT	OT-ODE-NRSD	100	**5.73**	0.179	31.69	0.753	7.32	0.237	30.72	0.665
OT	VP-ODE	100	6.14	0.194	31.93	0.773	7.38	0.231	31.10	0.705
VP-SDE	OT-ODE	100	**6.58**	**0.178**	**32.18**	**0.789**	**8.24**	**0.226**	**31.21**	**0.717**
VP-SDE	OT-ODE-NRSD	100	6.99	0.195	31.65	0.752	10.19	0.255	30.66	0.662
VP-SDE	VP-ODE	100	8.00	0.225	31.48	0.742	9.19	0.252	30.91	0.688

5.12 Baselines

5.12.1 ΠGDM

5.12.1.0.1 Implementation details.

We closely follow the official code available on github while implementing ΠGDM. For noisy case, we closely follow the Algorithm 1 in the appendix of Song et al. [2022]. We use adaptive weighted guidance for both noiseless and noisy cases as in the original work. We always use uniform spacing

Table 5.21: Comparison of performance OT-ODE sampling and OT-ODE sampling with null and range space decomposition (NRSD) on AFHQ-256 × 256

Model	Inference	NFEs ↓	Inpainting-*Center*, $\sigma_y = 0$				Inpainting-*Free-form*, $\sigma_y = 0$			
			FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
OT	OT-ODE	100	8.87	0.061	37.45	**0.921**	4.98	0.097	36.15	0.889
OT	OT-ODE-NRSD	100	**7.95**	**0.046**	**38.01**	**0.921**	**4.12**	**0.083**	**36.62**	**0.890**
OT	VP-ODE	100	9.18	0.106	35.63	0.898	6.92	0.135	34.72	0.869
VP-SDE	OT-ODE	100	**9.95**	0.064	37.49	**0.918**	5.39	0.099	36.15	**0.887**
VP-SDE	OT-ODE-NRSD	100	10.96	**0.052**	**37.95**	0.916	**4.87**	**0.089**	**36.52**	0.884
VP-SDE	VP-ODE	100	10.50	0.112	35.59	0.893	7.36	0.139	34.65	0.865

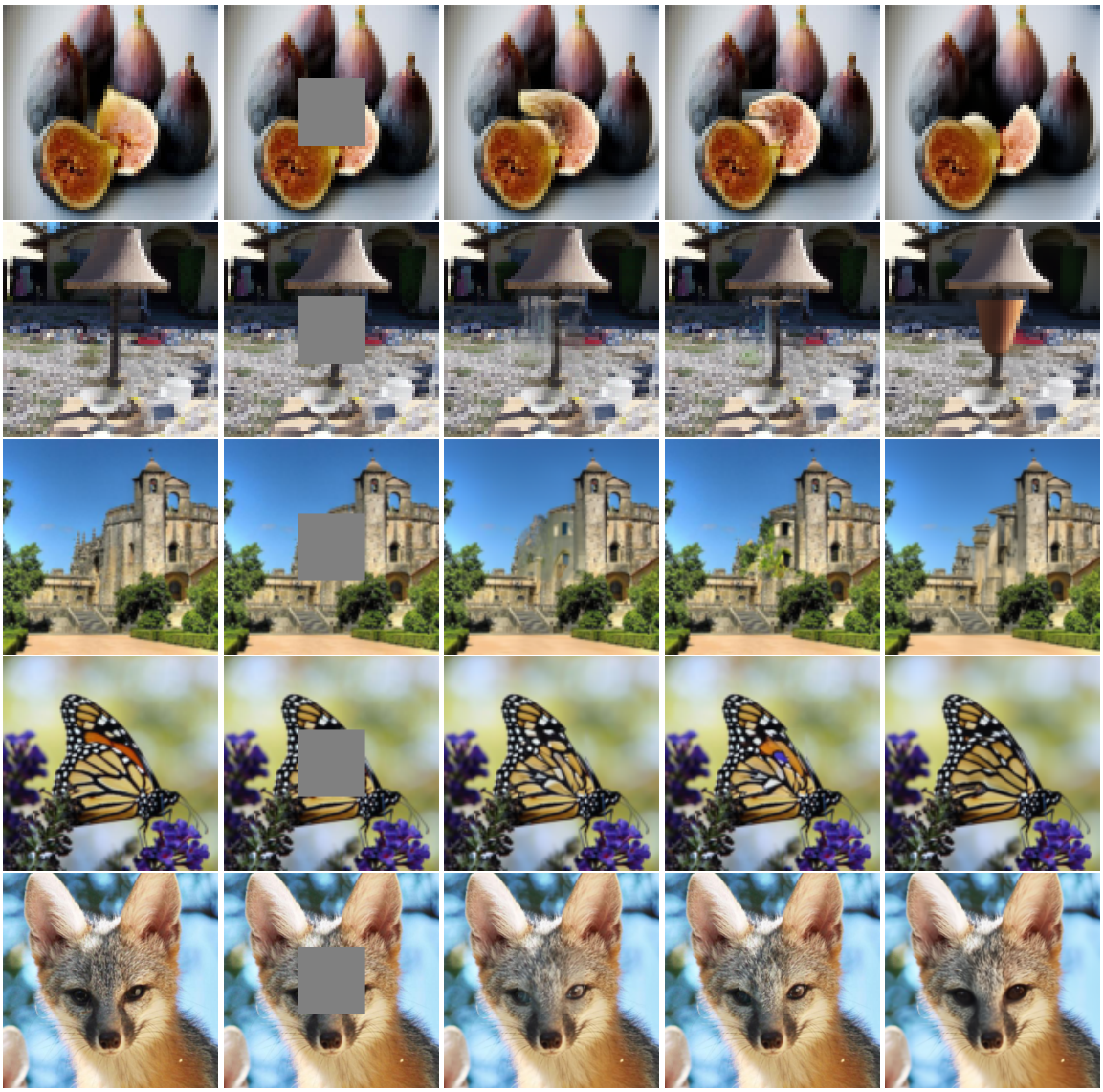
while iterating the timestep over 100 steps. We use ascending time from 0 to 1. Note that the original paper uses descending time from T to 0. According to the notational convention used in this paper, this is equivalent to ascending time from 0 to 1. For the choice of r_t^2, we consider the values derived from both variance exploding formulation and variance preserving formulation.

5.12.1.0.2 Value of r_t^2.

ΠGDM sets the value of $r_t^2 = \frac{\sigma_{1-t}^2}{1+\sigma_{1-t}^2}$ for VE-SDE, where $q(\boldsymbol{x}_t|\boldsymbol{x}_1) = \mathcal{N}(\boldsymbol{x}_1, \sigma_{1-t}^2\boldsymbol{I})$. We can follow the same procedure as outlined in Song et al. [2022], and solve for r_t^2 in closed form for VP-SDE. We know for that VP-SDE, $q(\boldsymbol{x}_t|\boldsymbol{x}_1) = \mathcal{N}(\alpha_{1-t}\boldsymbol{x}_1, (1-\alpha_{1-t}^2)\boldsymbol{I})$, where $\alpha_t = e^{-\frac{1}{2}T(t)}$, $T(t) = \int_0^t \beta(s)ds$, and $\beta(s)$ is the noise scale function. Using (5.16) for VP-SDE gives $r_t^2 = 1 - \alpha_{1-t}^2$. We can also obtain an alternate r_t^2 by plugging in value of σ_t^2 for VP-SDE into the expression of r_t^2 derived for VE-SDE, which evaluates to $r_t^2 = \frac{1-\alpha_{1-t}^2}{2-\alpha_{1-t}^2}$. Empirically, we find that r_t^2 for VE-SDE marginally outperforms VP-SDE. We report performance of ΠGDM with both choices of r_t^2 in Table 5.22 to 5.24.

Table 5.22: Relative performance of ΠGDM on face-blurred ImageNet-64 with VE and VP derived r_t^2 with $\sigma_y = 0.05$

Measurement	Model	VP				VE			
		FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
SR 2×	OT	6.52	0.168	30.54	0.753	5.91	0.160	30.60	0.762
Gaussian deblur	OT	55.19	0.374	28.74	0.516	39.36	0.326	29.00	0.572
Inpainting-*Center*	OT	9.25	0.111	34.13	0.863	8.70	0.109	34.17	0.864
Denoising	OT	16.59	0.102	34.60	0.906	16.44	0.101	34.64	0.907
SR 2×	VP-SDE	6.84	0.174	30.48	0.743	6.11	0.166	30.54	0.753
Gaussian deblur	VP-SDE	54.77	0.376	28.74	0.511	39.14	0.329	28.99	0.567
Inpainting-*Center*	VP-SDE	9.75	0.113	34.03	0.860	9.36	0.112	34.06	0.862
Denoising	VP-SDE	17.19	0.107	34.25	0.901	15.54	0.102	34.41	0.906

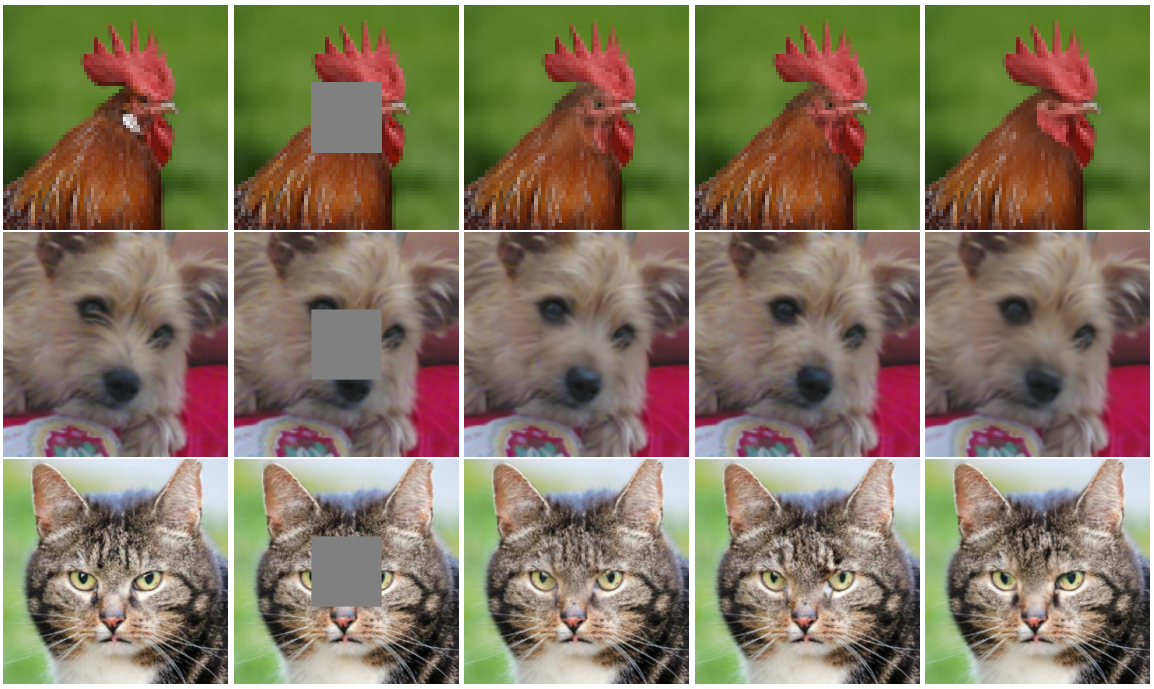


(a) Reference (b) Distorted (c) OT-ODE (d) OT-ODE-NRSD (e) ΠGDM

Figure 5.26: Comparison of inpainting (center mask) via OT-ODE sampling with and without null and range space decomposition (NRSD). We use conditional OT model and $\sigma_y = 0$ for (**first and second row**) face-blurred ImageNet-64, (**third row**) face-blurred ImageNet-128, and (**fourth row**) AFHQ.

5.12.1.0.3 Choice of starting time.

For OT-ODE sampling and VP-ODE sampling, we observe that starting at time $t > 0$ improves the performance. We therefore perform an ablation study on ΠGDM baseline, and vary the start time to verify whether starting at $t > 0$ helps to improve the performance. We plot the metrics for three different measurements in Figure 5.29. We observe that starting later at time $t > 0$ consistently leads to worse performance compared to starting at time $t = 0$. Therefore, for all our experiments with ΠGDM, we always start at time $t = 0$.



(a) Reference (b) Distorted (c) OT-ODE (d) OT-ODE-NRSD (e) ΠGDM

Figure 5.27: Comparison of inpainting (center mask) via OT-ODE sampling with and without null and range space decomposition (NRSD) for (**first row**) face-blurred ImageNet-64, (**second row**) face-blurred ImageNet-128, and (**third row**) AFHQ. We use VP-SDE model and $\sigma_y = 0$.

Table 5.23: Relative performance of ΠGDM on face-blurred ImageNet-128 with VE and VP derived r_t^2 with $\sigma_y = 0.05$

Measurement	Model	VP				VE			
		FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
SR 2×	OT	4.38	0.148	32.07	0.831	4.26	0.145	32.12	0.834
Gaussian deblur	OT	30.30	0.328	29.96	0.606	22.42	0.296	30.17	0.642
Inpainting-*Center*	OT	7.99	0.122	34.57	0.867	7.64	0.120	34.61	0.869
Denoising	OT	9.60	0.107	35.11	0.903	9.30	0.104	35.21	0.906
SR 2×	VP-SDE	7.55	0.183	31.61	0.785	6.14	0.168	31.79	0.803
Gaussian deblur	VP-SDE	55.61	0.463	28.57	0.414	41.69	0.404	28.98	0.493
Inpainting-*Center*	VP-SDE	9.75	0.130	34.45	0.858	9.46	0.129	34.49	0.859
Denoising	VP-SDE	10.69	0.124	34.72	0.882	10.11	0.119	34.92	0.886

5.12.2 RED-Diff

5.12.2.0.1 Implementation details.

We use VP-SDE model for all experiments with RED-Diff. We closely follow the official code available on github while implementing RED-Diff. Similar to [Mardani et al., 2023], we always use uniform spacing while iterating the timestep over 1000 steps. We use ascending time from 0 to 1. Note that the

(a) Reference (b) Distorted (c) OT-ODE (d) OT-ODE-NRSD (e) ΠGDM

Figure 5.28: Comparison of inpainting (free-form mask) via OT-ODE sampling with and without null and range space decomposition (NRSD) for AFHQ. We use conditional OT model and $\sigma_y = 0$.

Table 5.24: Relative performance of ΠGDM on AFHQ with VE and VP derived r_t^2 with $\sigma_y = 0.05$

Measurement	Model	VP				VE			
		FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
SR 4×	OT	12.69	0.285	30.18	0.665	12.31	0.282	30.23	0.672
Gaussian deblur	OT	24.60	0.383	28.93	0.429	19.66	0.355	29.16	0.475
Inpainting-*Center*	OT	19.09	0.153	34.20	0.855	16.51	0.145	34.40	0.863
Denoising	OT	11.20	0.159	34.49	0.876	10.92	0.153	34.78	0.883
SR 4×	VP-SDE	77.49	0.469	29.34	0.469	54.12	0.413	29.73	0.549
Gaussian deblur	VP-SDE	116.42	0.535	28.49	0.313	95.09	0.493	28.74	0.368
Inpainting-*Center*	VP-SDE	57.46	0.239	32.40	0.773	56.86	0.238	32.42	0.775
Denoising	VP-SDE	81.15	0.451	29.62	0.639	35.33	0.278	31.72	0.776

original paper uses descending time from T to 0. According to the notational convention used in this paper, this is equivalent to ascending time from 0 to 1. We use Adam optimizer and use the momentum pair $(0.9, 0.99)$ similar to the original work. Further, we use initial learning rate of 0.1 for AFHQ and ImageNet-128, as used in the original work, and learning rate of 0.01 for ImageNet-64. We use batch size of 1 for all the experiments. Finally, we extensively tuned the regularization hyperparameter λ to find the value that results in optimal performance across all metrics. We summarize the results of our experiments in Table 5.25 to 5.30. We note that more extensive tuning may be able to find better performing hyperparameters but this goes against the intent of a training-free algorithm.

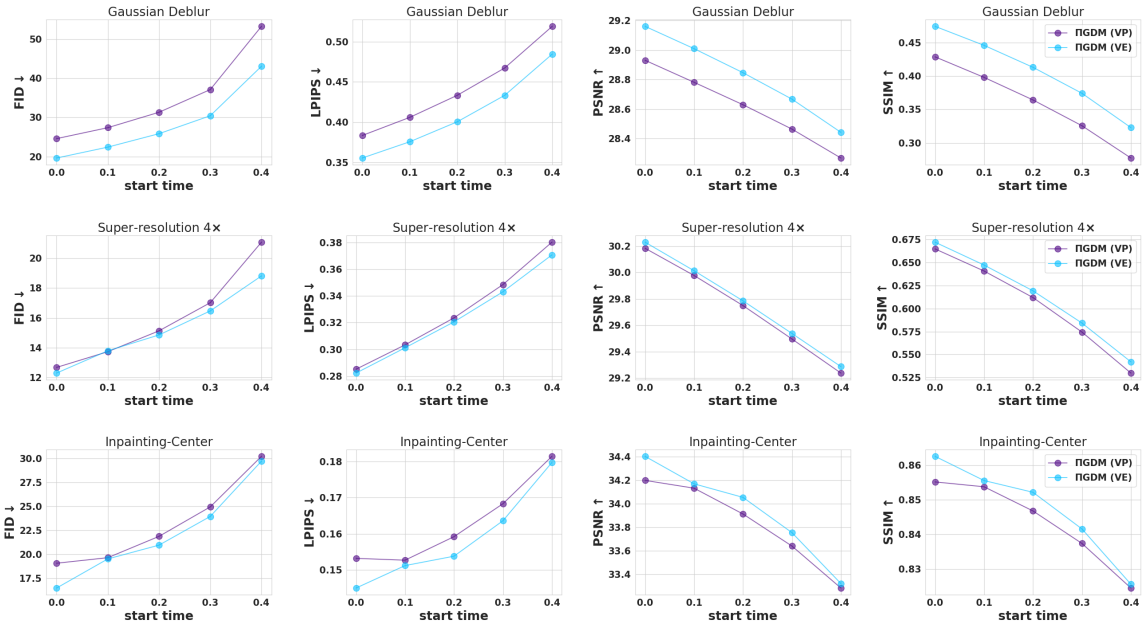


Figure 5.29: Variation in performance ΠGDM sampling with variation in start times on AFHQ dataset. We use pretrained conditional OT model and set $\sigma_y = 0.05$. We observe similar trends with VP-SDE checkpoint. We plot metrics for both choices of r_t^2 that can be derived from variance preserving and variance exploding formulations.

Table 5.25: Hyperparameter search for RED-Diff on face-blurred ImageNet-64 $\times$ 64 with $\sigma_y = 0.05$. We use learning rate of 0.01.

λ	SR 2×, $\sigma_y = 0.05$				Gaussian deblur, $\sigma_y = 0.05$			
	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
0.1	34.09	0.224	30.12	0.798	**46.76**	0.254	29.29	0.715
0.25	28.45	0.206	30.40	0.814	51.20	**0.236**	**30.19**	**0.776**
0.75	**23.02**	**0.187**	**31.22**	**0.839**	73.76	0.287	30.47	0.750
1.5	32.35	0.243	30.80	0.792	82.26	0.335	30.29	0.705
2.0	40.33	0.284	30.41	0.750	86.48	0.358	30.17	0.683
λ	Inpainting-*Center*, $\sigma_y = 0.05$				Denoising, $\sigma_y = 0.05$			
	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
0.1	15.71	0.155	31.74	0.840	12.47	0.085	32.24	0.907
0.25	15.56	0.155	31.73	0.839	11.80	0.083	32.36	0.908
0.75	13.31	0.139	32.65	0.857	8.43	0.062	33.65	0.932
1.5	**12.18**	**0.119**	33.97	0.881	6.11	0.041	35.34	0.958
2.0	12.87	**0.119**	**34.19**	**0.886**	**6.02**	**0.041**	**35.64**	**0.964**

Table 5.26: Hyperparameter search for RED-Diff on face-blurred ImageNet-64 $\times$ 64 with $\sigma_y = 0$. We use learning rate of 0.01.

λ	SR 2×, $\sigma_y = 0$				Gaussian deblur, $\sigma_y = 0$			
	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
0.1	**11.74**	0.224	30.12	0.798	**15.39**	**0.134**	**31.99**	**0.879**
0.25	12.65	**0.130**	**32.34**	**0.886**	29.56	0.236	30.19	0.776
0.75	20.36	0.187	31.22	0.839	55.43	0.287	30.47	0.750
1.5	33.13	0.243	30.80	0.792	71.64	0.335	30.29	0.705
2.0	41.56	0.288	30.46	0.752	78.55	0.358	30.22	0.685

λ	Inpainting-*Center*, $\sigma_y = 0$			
	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
0.1	**7.29**	**0.079**	**39.14**	**0.925**
0.25	7.40	0.155	31.73	0.839
0.75	8.47	0.083	38.59	0.922
1.5	10.75	0.095	37.42	0.916
2.0	12.54	0.119	34.19	0.886

Table 5.27: Hyperparameter search for RED-Diff on face-blurred ImageNet-128 $\times$ 128 with $\sigma_y = 0.05$. We use learning rate of 0.1.

λ	SR 2×, $\sigma_y = 0.05$				Gaussian deblur, $\sigma_y = 0.05$			
	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
0.1	23.25	0.272	30.12	0.731	37.83	0.42	28.54	0.473
0.75	14.56	0.224	30.71	0.782	**21.43**	**0.229**	31.41	0.807
1.5	**10.54**	**0.182**	31.82	0.852	22.85	0.247	**31.65**	**0.809**
2.0	11.65	0.187	**31.93**	**0.859**	24.71	0.259	31.61	0.802

λ	Inpainting-*Center*, $\sigma_y = 0.05$				Denoising, $\sigma_y = 0.05$			
	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
0.1	19.68	0.191	31.75	0.795	12.83	0.134	32.27	0.854
0.75	19.03	0.202	31.36	0.779	12.69	0.14	32.09	0.846
1.5	16.33	0.189	31.81	0.794	10.67	0.121	32.89	0.874
2.0	**14.63**	**0.171**	**32.42**	**0.819**	**9.19**	**0.105**	**33.52**	**0.895**

Table 5.28: Hyperparameter search for RED-Diff on face-blurred ImageNet-128 $\times$ 128 with $\sigma_y = 0$. We use learning rate of 0.1.

λ	SR 2×, $\sigma_y = 0$				Gaussian deblur, $\sigma_y = 0$			
	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
0.1	**3.90**	**0.082**	**34.47**	**0.922**	**4.19**	**0.085**	**34.68**	**0.929**
0.75	6.52	0.105	33.54	0.905	12.59	0.177	32.71	0.864
1.5	10.46	0.142	32.98	0.894	19.29	0.225	32.15	0.831
2.0	13.08	0.165	32.65	0.884	22.57	0.245	31.94	0.816
λ	Inpainting-*Center*, $\sigma_y = 0$				Inpainting-*Freeform*, $\sigma_y = 0$			
	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
0.1	**5.39**	**0.068**	**38.91**	**0.928**	**8.94**	**0.162**	**35.54**	**0.830**
0.75	5.52	0.073	38.11	0.924	9.26	0.166	35.05	0.826
1.5	6.09	0.079	37.32	0.920	10.13	0.172	34.58	0.821
2.0	6.68	0.083	36.87	0.917	10.87	0.176	34.30	0.818

Table 5.29: Hyperparameter search for RED-Diff on AFHQ with $\sigma_y = 0.5$. We use learning rate of 0.1.

λ	SR 4×, $\sigma_y = 0.05$				Gaussian deblur, $\sigma_y = 0.05$			
	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
0.1	21.59	0.385	29.51	0.607	17.36	0.379	29.95	0.639
0.25	22.47	0.374	29.66	0.635	**15.81**	**0.341**	**30.15**	**0.645**
0.75	**20.84**	**0.331**	**29.97**	**0.675**	25.41	0.366	29.76	0.588
1.5	22.46	0.355	29.68	0.642	38.66	0.409	29.34	0.525
2.0	25.02	0.376	29.49	0.618	45.01	0.427	29.18	0.500
λ	Inpainting-*Center*, $\sigma_y = 0.05$				Denoising, $\sigma_y = 0.05$			
	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
0.1	**28.39**	0.216	31.53	0.756	8.32	0.159	32.18	0.827
0.25	28.85	0.217	31.51	0.755	8.35	0.161	32.16	0.826
0.75	28.80	0.218	31.64	0.759	7.94	0.156	32.35	0.833
1.5	28.74	0.205	32.19	0.784	6.63	0.138	33.12	0.862
2.0	28.55	0.190	32.63	0.802	5.71	0.124	33.70	0.882
2.5	28.71	**0.177**	**32.99**	**0.818**	**4.93**	**0.111**	**34.18**	**0.899**

Table 5.30: Hyperparameter search for RED-Diff on AFHQ with $\sigma_y = 0$. We use learning rate (lr) of 0.1 unless mentioned otherwise.

λ	SR 4×, $\sigma_y = 0$				Gaussian deblur, $\sigma_y = 0$			
	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
0.005	11.67	0.197	**32.93**	**0.837**	14.69	0.278	**31.73**	**0.760**
0.05	**8.65**	**0.191**	32.21	0.801	11.67	**0.268**	31.30	0.731
0.1	9.65	0.204	31.84	0.781	**11.53**	0.273	31.05	0.711
0.25	11.65	0.222	31.53	0.768	13.22	0.293	30.63	0.675
0.75	14.98	0.274	30.72	0.726	23.34	0.351	29.91	0.598
1.5	19.40	0.332	29.95	0.665	36.96	0.402	29.39	0.529
2.0	22.72	0.361	29.65	0.632	43.64	0.422	29.22	0.504

λ	Inpainting-*Center*, $\sigma_y = 0$, lr=0.01				Inpainting-*Freeform*, $\sigma_y = 0$			
	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑	FID ↓	LPIPS ↓	PSNR ↑	SSIM ↑
0.005	**8.53**	**0.050**	**38.89**	**0.951**	7.22	0.091	**36.89**	**0.892**
0.05	8.53	0.050	38.89	0.951	7.27	**0.090**	36.88	**0.892**
0.1	8.53	0.050	38.88	0.951	**7.23**	0.091	36.82	0.891
0.25	8.53	0.050	38.83	0.950	7.32	0.094	36.69	0.889
0.75	8.88	0.056	38.60	0.948	7.74	0.102	36.26	0.884
1.5	10.32	0.071	38.04	0.942	8.41	0.112	35.69	0.877
2.0	11.62	0.084	37.54	0.937	8.76	0.119	35.37	0.872

Part II

Efficient Neural Operators with Deep Equilibrium Models

Chapter 6

Deep Equilibrium Based Neural Operators for Steady-State PDEs

Partial differential equations (PDEs) are used to model a wide range of processes in science and engineering. They define a relationship of (unknown) function and its partial derivatives. Most PDEs do not accept a closed form solution and solving them with classical numerical methods can be slow and expensive. Recent work has shown neural operators to be an extremely promising machine learning-based approach to learn solutions of PDEs. These methods train an operator, which takes as input a PDE in some family, and outputs its solution. However, the architectural design space, especially given structural knowledge of the PDE family of interest, is still poorly understood. Motivated by this, we study the benefits of weight-tied neural network architectures for steady-state PDEs. Specifically, we propose a deep equilibrium based operator that directly solves for the solution of a steady-state PDE as the infinite-depth fixed point of an implicit operator layer using a black-box root solver and differentiates analytically through this fixed point resulting in $\mathcal{O}(1)$ training memory. We refer the reader to Sec. 2.1 for a detailed overview on Deep Equilibrium (DEQ) models, and to Sec. 2.3 for an overview on PDEs and neural operators. We introduce our weight-tied architectures in Sec. 6.3, and provide experimental results on two well-known steady-state PDEs: Darcy flow and steady-state Navier Stokes.

6.1 Preliminaries

We now summarize some key concepts and notation used in this chapter. We also refer the reader to Chapter 2 for a more detailed background on PDEs and neural operators.

Definition 1 ($L^2(\Omega; \mathbb{R}^d)$)**.** *For a domain Ω we denote by $L^2(\Omega; \mathbb{R}^d)$ the space of square integrable functions $g : \Omega \to \mathbb{R}^d$ such that $\|g\|_{L^2(\Omega)} < \infty$, where $\|g\|_{L^2(\Omega)} = \left(\int_\Omega \|g(x)\|_{\ell_2}^2 dx\right)^{1/2}$.*

6.1.1 Fourier Neural Operator (FNO)

Neural operators [Lu et al., 2019, Li et al., 2020a, Bhattacharya et al., 2021, Patel et al., 2021, Kovachki et al., 2023] are a deep learning approach to learning solution operators which map a PDE to its solution. The Fourier neural operator (FNO) [Li et al., 2020a] is a particularly successful recent architecture parametrized as a sequence of kernel integral operator layers followed by non-linear activation functions. Each kernel integral operator layer is a convolution-based kernel function that is instantiated through a linear transformation in Fourier domain, making it less sensitive to the level of spatial discretization. Specifically, an L-layered FNO $G_\theta : \mathbb{R}^{d_u} \to \mathbb{R}^{d_u}$ with learnable parameters θ, is defined as

$$G_\theta := \mathcal{Q} \circ \mathcal{L}_L \circ \mathcal{L}_{L-1} \circ \cdots \circ \mathcal{L}_1 \circ \mathcal{P} \tag{6.1}$$

where $\mathcal{P} : L^2(\Omega; \mathbb{R}^{d_u}) \to L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_v})$ and $\mathcal{Q} : L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_v}) \to L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_u})$ are projection operators, and $\mathcal{L}_l : L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_v}) \to L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_v})$ for $l \in [L]$ is the l^{th} FNO layer defined as,

$$\mathcal{L}_l\left(v_l\right) = \sigma\left(W_l v_l + b_l + \mathcal{K}_l(v_l)\right)). \tag{6.2}$$

Here σ is a non-linear activation function, W_l, b_l are the l^{th} layer weight matrix and bias terms. Finally $\mathcal{K}_l$ is the l^{th} integral kernel operator which is calculated using the Fourier transform as introduced in Li et al. [2020a] defined as follows,

$$\mathcal{K}_l(v_l) = \mathcal{F}^{-1}\left(R_l \cdot (\mathcal{F} v_l)\right)(x) \qquad \forall x \in \Omega, \tag{6.3}$$

where $\mathcal{F}$ and $\mathcal{F}^{-1}$ are the Fourier transform and the inverse Fourier transform, with R_l representing the learnable weight-matrix in the Fourier domain. Therefore, ultimately, the trainable parameters θ is a collection of all the weight matrices and biases, i.e, $\theta := \{W_l, b_l, R_l, \cdots, W_1, b_1, R_1\}$.

6.2 Related Work

Neural network based approaches for solving PDEs can broadly be divided into two categories. First are hybrid solvers [Bar-Sinai et al., 2019, Kochkov et al., 2021, Hsieh et al., 2019] which use neural networks in conjunction with existing numerical solvers. The main motivation is to not only improve upon the existing solvers, but to also replace the more computationally inefficient parts of the solver with a learned counter part. Second set of approaches are full machine learning based approaches that aim to leverage the approximation capabilities of neural networks [Hornik et al., 1989] to directly learn the dynamics of the physical system from observations.

Hybrid solvers like Hsieh et al. [2019] use a neural network to learn a correction term to correct over an existing hand designed solver for a Poisson equation, and also provide convergence guarantees of their method to the solution of the PDE. However, the experiments in their paper are limited to linear elliptic PDEs. Further, solvers like Bar-Sinai et al. [2019] use neural networks to derive the discretizations for a given PDE, thus enabling the use of a low-resolution grid in the numerical solver. Furthermore, Kochkov et al. [2021] use neural networks to interpolate differential operators between grid points of a low-resolution grid with high accuracy. This work specifically focuses on solving Navier-Stokes equations, their method is more accurate than numerical techniques like Direct Numerical Simulation (DNS) with a low-resolution grid, and is also $80\times$ more faster. Brandstetter et al.

[2022] introduced a message passing based hybrid scheme to train a hybrid solver and also propose a loss term which helps improve the stability of hybrid solvers for time dependent PDEs. However, most of these methods are equation specific, and are not easily transferable to other PDEs from the same family.

The neural network based approach that has recently garnered the most interest by the community is that of the operator learning framework [Chen and Chen, 1995, Kovachki et al., 2021b, Lu et al., 2019, Li et al., 2020a, Bhattacharya et al., 2021], which uses a neural network to approximate and infinite dimensional operator between two Banach spaces, thus learning an entire family of PDEs at once. Lu et al. [2019] introduces DeepONet, which uses two deep neural networks, referred to as the branch net and trunk net, which are trained concurrently to learn from data. Another line of operator learning framework is that of neural operators Kovachki et al. [2021b]. The most successful methodology for neural operators being the Fourier neural operators (FNO) [Li et al., 2020a]. FNO uses convolution based integral kernels which are evaluated in the Fourier space.

Future works like Tran et al. [2021] introduce architectural improvements that enables one to train deeper FNO networks, thus increasing their size and improving their the performance on a variety of (time-dependent) PDEs. Moreover, the success of Transformers in domains like language and vision has also inspired transformer based neural operators in works like Li et al. [2022b], Hao et al. [2023] and Liu et al. [2022c]. Theoretical results pertaining to the neural operators mostly include universal approximation results Kovachki et al. [2021a], Lanthaler et al. [2022] which show that architectures like FNO and DeepONet can indeed approximate the infinite dimension operators. In this work, we focus on steady-state equations and show the benefits of weight-tying in improving the performance of FNO for steady-state equations. We show that instead of making a network deeper and hence increasing the size of a network, weight-tied FNO architectures can outperform FNO and its variants $4\times$ its size. We further introduce FNO-DEQ, a deep equilibrium based architecture to simulate an infinitely deep weight-tied network (by solving for a fixed point) with $\mathcal{O}(1)$ training memory. Our work takes inspiration from recent theoretical works like Marwah et al. [2021], Chen et al. [2021], Marwah et al. [2022] which derive parametric rates for some-steady state equations, and in fact prove that neural networks can approximate solutions to some families of PDEs with just $\text{poly}(d)$ parameters, thus evading the curse of dimensionality.

6.3 Problem Setup

6.3.1 Steady-State PDE

We first formally define the system of steady-state PDEs that we will solve for:

Definition 2 (Steady-State PDE)**.** *Given a bounded open set $\Omega \subset \mathbb{R}^d$, a steady-state PDE can be written in the following general form:*

$$L(a(x), u(x)) = f(x), \qquad \forall x \in \Omega \tag{6.4}$$

Here L is a continuous operator, the function $u \in L^2\left(\Omega; \mathbb{R}^{d_u}\right)$ is the unknown function for which we wish to solve, and $a \in L^2\left(\Omega; \mathbb{R}^{d_a}\right)$ collects all the coefficients describing the PDE, and $f \in L^2\left(\Omega; \mathbb{R}^{d_f}\right)$ is a function

independent of u. We will, for concreteness, assume periodic boundary conditions, i.e. $\forall z \in \mathbb{Z}^d, \forall x \in \Omega$ we have $u(x+z) = u(x)$. (Equivalently, $\Omega := \mathbb{T}^d = [0, 2\pi]^d$ can be identified with the torus.) [1] *Finally, we will denote $u^\star : \Omega \to \mathbb{R}$ as the solution to the PDE.*

Steady-state models a system at stationarity, *i.e.*, when some quantity of interest like temperature or velocity no longer changes over time. Classical numerical solvers for these PDEs include iterative methods like Newton updates or conjugate gradient descent, typically with carefully chosen preconditioning to ensure benign conditioning and fast convergence. Furthermore, recent theoretical works [Marwah et al., 2021, Chen et al., 2021, Marwah et al., 2022] show that for many families of PDEs (e.g., steady-state elliptic PDEs that admit a variational formulation), iterative algorithms can be efficiently "neuralized", that is, the iterative algorithm can be represented by a compact neural network, so long as the coefficients of the PDE are also representable by a compact neural network. Moreover, the architectures constructed in these works are heavily weight-tied.

We will operationalize these developments through the additional observation that all these iterative schemes can be viewed as algorithms to find a fixed point of a suitably chosen operator. Namely, we can design an operator $\mathcal{G} : L^2(\Omega; \mathbb{R}^{d_u}) \times L^2(\Omega; \mathbb{R}^{d_f}) \to L^2(\Omega; \mathbb{R}^{d_u})$ [2] such that $u^\star = \mathcal{G}(u^\star, f)$ and a lot of common (preconditioned) first and second-order methods are natural ways to recover the fixed points $u^\star$.

6.3.2 Architectures for Steady-State PDEs

Before describing our architectures, we introduce two components that we will repeatedly use.

Definition 3 (Projection and embedding layers)**.** *A projection and embedding layer, respectively $\mathcal{P} : L^2(\Omega; \mathbb{R}^{d_u}) \times L^2(\Omega; \mathbb{R}^{d_f}) \to L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_v}) \times L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_v})$ and $\mathcal{Q} : L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_v}) \to L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_u})$, are defined as*

$$\mathcal{P}(v, f) = \left(\sigma\left(W_P^{(1)} v + b_P^{(1)}\right), \sigma\left(W_P^{(2)} f + b_P^{(2)}\right)\right),$$
$$\mathcal{Q}(v) = \sigma\left(W_Q v + b_Q\right)$$

where $W_P^{(1)} \in \mathbb{R}^{d_u \times d_v}, W_P^{(2)} \in \mathbb{R}^{d_f \times d_v}, W_Q \in \mathbb{R}^{d_v \times d_u}$ and $b_P^{(1)}, b_P^{(2)} \in \mathbb{R}^{d_v}, b_Q \in \mathbb{R}^{d_u}$.

Definition 4 (Input-injected FNO layer)**.** *An input-injected FNO layer $\mathcal{L} : L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_v}) \times L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_v}) \to L^2(\mathbb{R}^{d_v}; \mathbb{R}^{d_v})$ is defined as*

$$\mathcal{L}(v, g) := g + \sigma\left(Wv + b + \mathcal{F}^{-1}(R^{(k)} \cdot (\mathcal{F}v))\right). \tag{6.5}$$

where $W \in \mathbb{R}^{d_v \times d_v}$, $b \in \mathbb{R}^{d_v}$ and $R^{(k)} \in \mathbb{R}^{d_v \times d_v}$ for all $k \in [K]$ are learnable parameters.

Note the difference between the FNO layer specified above, and the standard FNO layer (6.2) is the extra input g to the layer, which in our architecture will correspond to a projection of (some or all) of the PDE coefficients. We also note that this change to the FNO layer also enables us to learn deeper FNO architectures, as shown in Section 6.5. With this in mind, we can discuss the architectures we propose.

[1] This is for convenience of exposition, our methods can readily be extended to other boundary conditions like Dirichet, Neumann etc.

[2] We note that the choice of defining the operator with the forcing function f is made for purely expository purposes the operator $\mathcal{G}$ can be defined as $\mathcal{G} : L^2(\Omega; \mathbb{R}^{d_u}) \times L^2(\Omega; \mathbb{R}^{d_a}) \to L^2(\Omega; \mathbb{R}^{d_u})$ as well.

6.3.2.1 Weight-tied architecture I: Weight-tied FNO

The first architecture we consider is a weight-tied version of FNO (introduced in Section 6.1), in which we repeatedly apply (M times) the same transformation in each layer. More precisely, we have:

Definition 5 (FNO Weight-Tied). *We define a M times weight-tied neural operator G_θ^M as,*

$$G_\theta^M = \mathcal{Q} \circ \underbrace{\mathcal{B}^L \circ \mathcal{B}^L \circ \cdots \circ \mathcal{B}^L}_{M \text{ times}} \circ \mathcal{P}$$

such that: $\mathcal{P}$, $\mathcal{Q}$ are projection and embedding layers as in Definition 3

6.3.2.2 Weight-tied architecture II: FNO-DEQ

In cases where we believe a weight-tied G_θ^M converges to some fixed point as $M \to \infty$, unrolling G_θ^M for a large M requires a lot of hardware memory for training: training the model requires one to store intermediate hidden units for each weight-tied layer for backpropagation, resulting in a $\mathcal{O}(M)$ increase in the amount of memory required.

To this end, we use Deep Equilibrium models (DEQs) which enables us to implicitly train $G_\theta := \lim_{M\to\infty} G_\theta^M$ by directly solving for the fixed point by leveraging black-box root finding algorithms like quasi-Newton methods, [Broyden, 1965, Anderson, 1965]. Therefore we can think of this approach as an infinite-depth (or infinitely unrolled) weight-tied network. We refer to this architecture as **FNO-DEQ**.

Definition 6 (FNO-DEQ). *Given $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{B}^L$ in Definition 5, FNO-DEQ is trained to parametrize the fixed point equation $\mathcal{B}^L(v^\star, \mathcal{P}(f)) = v^\star$ and outputs $u^\star = \mathcal{Q}(v^\star)$.*

Usually, it is non-trivial to differentiate through these black-box root solvers. DEQs enable us to implicitly differentiate through the equilibrium fixed point efficiently without any need to backpropagate through these root solvers, therefore resulting in $\mathcal{O}(1)$ training memory. We refer the readers to Sec. 2.1 for further details.

6.4 Details of Experimental Setup

6.4.1 Network architectures

We consider the following network architectures for our experiments.

FNO: We closely follow the architecture proposed by Li et al. [2020a] and construct this network by stacking four FNO layers and four convolutional layers, separated by GELU activation [Hendrycks and Gimpel, 2016]. Note that in our current set up, we recover the original FNO architecture if the input to the l^{th} layer is the output of $(l-1)^{\text{th}}$ layer *i.e.*, $v_l = \mathcal{B}_{l-1}(v_{l-1})$.

Improved FNO (FNO++): The original FNO architecture suffers from vanishing gradients, which prohibits it from being made deeper [Tran et al., 2021]. We overcome this limitation by introducing residual connections within each block of FNO, with each FNO block $\mathcal{B}_l$ comprising of three FNO layers $\mathcal{L}$ *i.e.*, $\mathcal{B}_l = \mathcal{L}^l_{L_1} \circ \mathcal{L}^l_{L_2} \circ \mathcal{L}^l_{L_3}$ and three convolutional layers, where $\mathcal{L}$ is defined in Eq. (6.5).

Weight-tied network (FNO-WT): This is the weight-tied architecture introduced in Definition 5, where we initialize $v_0(x) = 0$ for all $x \in \Omega$.

FNO-DEQ: As introduced in Definition 6, FNO-DEQ is a weight-tied network where we explicitly solve for the fixed point in the forward pass with a root finding algorithm. We use Anderson acceleration [Anderson, 1965] as the root solver. For the backward pass, we use approximate implicit gradients [Geng et al., 2021b] which are light-weight and more stable in practice, compared to implicit gradients computed by inverting Jacobian.

Note that both weight-tied networks and FNO-DEQs leverage weight-tying but the two models differ in the ultimate goal of the forward pass: DEQs explicitly solve for the fixed point during the forward pass, while weight-tied networks trained with backpropagation may or may-not reach a fixed point [Anil et al., 2022]. Furthermore, DEQs require $\mathcal{O}(1)$ memory, as they differentiate through the fixed point implicitly, whereas weight-tied networks need to explicitly create the entire computation graph for backpropagation, which can become very large as the network depth increases. Additionally, FNO++ serves as a non weight-tied counterpart to a weight-tied input-injected network. Like weight-tied networks, FNO++ does not aim to solve for a fixed point in the forward pass.

6.4.1.1 Implementation details

The width of an FNO layer set to 32 across all the networks. Additionally, we retain only 12 Fourier modes in FNO layer, and truncate higher Fourier modes. We use the code provided by Li et al. [2020a] to replicate the results for FNO, and construct rest of the networks on top of this as described above.

For FNO-DEQ, we use Anderson solver [Anderson, 1965] to solve for the fixed point in the forward pass. The maximum number of Anderson solver steps is kept fixed at 32 for Dary Flow, and 16 for Navier Stokes. For the backward pass, we use phantom gradients [Geng et al., 2021b] which are computed as

$$u^\star = \tau G_\theta(u^\star, a) + (1 - \tau)u^\star, \tag{6.6}$$

where τ is a tunable damping factor and $u^\star$ is the fixed point computed using Anderson solver in the forward pass. This step can be repeated S times. We use $\tau = 0.5$ and $S = 1$ for Darcy Flow, and $\tau = 0.8$ and $S = 3$ for Navier-Stokes.

For the S-FNO-DEQ used in Table 6.1, we use Broyden's method [Broyden, 1965] to solve for the fixed point in the forward pass and use exact implicit gradients, computed through implicit function theorem as shown in Eq. (2.3), for the backward pass through DEQ. The maximum number of solver steps is fixed at 32.

For weight-tied networks, we repeatedly apply the FNO block to the input 12 times for Darcy flow, and 6 times for Navier-Stokes.

6.4.1.2 Training details

We train all the networks for 500 epochs with Adam optimizer. The learning rate is set to 0.001 for Darcy flow and 0.005 for Navier-Stokes. We use learning rate weight decay of 1e-4 for both Navier-Stokes and Darcy flow. The batch size is set to 32. In case of Darcy flow, we also use cosine annealing for learning rate scheduling. We run all our experiments on a combination of NVIDIA RTX A6000, NVIDIA GeForce RTX 2080 Ti and 3080 Ti. All networks can easily fit on a single NVIDIA RTX A6000, but training time varies between the networks.

6.4.2 Methodology

We test the aforementioned network architectures on two families of steady-state PDEs: Darcy Flow equation and steady-state Navier-Stokes equation for incompressible fluids. For experiments with Darcy Flow, we use the dataset provided by Li et al. [2020a], and generate our own dataset for steady-state Navier-Stokes.

For each family of PDE, we train networks under 3 different training setups: clean data, noisy inputs and noisy observations. For experiments with noisy data, both input and observations, we add noise sampled from a sequence of standard Gaussians with increasing values of variance $\{\mathcal{N}(0, (\sigma_k^2))\}_{k=0}^{M-1}$, where M is the total number of Gaussians we sample from. We set $\sigma_0^2 = 0$ and $\sigma_{\text{max}}^2 = \sigma_{M-1}^2 \leq 1/r$, where r is the resolution of the grid. Thus, the training data includes equal number of PDEs with different levels of Gaussian noise added to their input or observations.

We add noise to training data, and always test on clean data. We follow prior work [Li et al., 2020b] and report the relative L_2 norm between ground truth $u^\star$ and prediction on test data. The total depth of all networks besides FNO is given by $6B + 4$, where B is the number of FNO blocks. Each FNO block has 3 FNO layers and convolutional layers. In addition, we include depth due to $\mathcal{P}$, $\mathcal{Q}$, and an additional final FNO layer and a convolutional layer.

6.4.3 Datasets

6.4.3.1 Darcy Flow

As mentioned in the previous section, we use the dataset provided by Li et al. [2020a] for our experiments with steady-state Darcy-Flow. All models are trained on 1024 data samples and tested on 500 samples. The resolution of the original images is 421×421, which we downsample to 85×85 for our experiments. For experiments with noisy inputs/observations, the variance of Gaussian noise that we add to PDEs are [0, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3]. We provide visualization of random samples from the dataset in Figure 6.2.

6.4.3.2 Steady-State Incompressible Fluid Navier-Stokes

$$\begin{aligned} u \cdot \nabla \omega &= \nu \Delta \omega + f, \qquad && x \in \Omega \\ \nabla \cdot u &= 0 && x \in \Omega \end{aligned}$$

To generate the dataset for steady-state Navier-Stokes, instead of solving the steady state PDE using steady-state solvers like the SIMPLE algorithm [Patankar and Spalding, 1983], we first choose the solution $\omega^\star := \nabla \times u^\star$ of the PDE and then generate the corresponding equation, *i.e.*, calculate the corresponding force term $f = u^\star \cdot \nabla \omega^\star - \nu \Delta \omega^\star$.

To generate the solutions $\omega^\star$, we forward propagate a relatively simple initial distribution of ω_0 (sampled from a Gaussian random field) through a time-dependent Navier-Stokes equation in the vorticity form for a short period of time. This ensures our dataset contains solutions ω^* that are rich and complex. Precisely, recall the Navier-Stokes equations in their vorticity form:

$$\begin{aligned} \partial_t \omega(x,t) + u(x,t) \cdot \nabla \omega(x,t) &= \nu \Delta \omega(x,t) + g(x) \qquad x \in (0, 2\pi)^2, t \in [0,T] \\ \nabla \cdot u(x,t) &= 0 \qquad x \in (0, 2\pi)^2, t \in [0,T] \\ \omega(x,0) &= \omega_0(x) \qquad x \in (0, 2\pi)^2 \end{aligned} \tag{6.7}$$

where $g(x) = \nabla \times \tilde{g}(x)$ and $\tilde{g}(x) = \sin(5x_1)\hat{x}_2$ is a divergence free forcing term and $x = (x_1, x_2)$ are the two coordinates of the input vector. We forward propagate the equations (6.7) using a pseudo-spectral method using the functions provided in JAX-CFD [Kochkov et al., 2021, Dresdner et al., 2022] package. The initial vorticity ω_0 is sampled from a Gaussian random field $\mathcal{N}(0, (5^{3/2}(I + 25\Delta)^{-2.5}))$, which is then made divergence free.

We forward propagate the Navier-Stokes equation in (6.7) for time $T = 0.5$ with $dt = 0.002$ to get $\omega(1, x)$, which we choose as the solution to the steady-state PDE in (6.9), i.e, $\omega^\star$ for Equation 6.9.

Subsequently, we use the stream function Ψ [Batchelor and Batchelor, 1967] to calculate $u = (\partial\Psi/\partial x_1, \partial\Psi/\partial x_2)$ by solving the Poisson equation $\Delta\Psi = \omega$ in the Fourier domain. Furthermore, since $f = u^\star \cdot \nabla \omega^\star - \nu \Delta \omega^\star$, we use the stream function to calculate (f_1, f_2), *i.e.*, the different components of the force term.

We use 4500 training samples and 500 testing samples. The input to the network is the vector field $\tilde{f} = (f_1, f_2)$ and we learn a map that outputs the vorticity $\omega^\star$. The resolution of grid used to generate the dataset is 256×256 which we downsample to 128×128 while training the models. For experiments with noisy inputs/observations, we consider two values of maximum variance of Gaussian noise: 1e-3 and 4e-3. The variances of the Gaussian noise that we add to the PDEs for the latter case are [0, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 2e-3, 4e-3]. However, when conducting experiments with a variance of 1e-3, we exclude the last two values of variance from this list. We provide visualization of random samples from the dataset in Figure 6.3 for viscosity $\nu = 0.001$ and in Figure 6.4 for viscosity $\nu = 0.01$.

6.5 Experimental Results

6.5.1 Darcy Flow

For our first set of experiments we consider stationary Darcy Flow equations, a form of linear elliptic PDE with in two dimensions. The PDE is defined as follows,

$$\begin{aligned} -\nabla \cdot (a(x)\nabla u(x)) &= f(x), \qquad && x \in (0,1)^2 \\ u(x) &= 0 && x \in \partial(0,1)^2. \end{aligned} \tag{6.8}$$

Note that the diffusion coefficient $a \in L^\infty(\Omega)(\Omega; \mathbb{R}_+)$, i.e., the coefficients are always positive, and $f \in L^2(\Omega; \mathbb{R}^{d_f})$ is the forcing term. These PDEs are used to model the steady-state pressure of fluids flowing through a porous media. They can also be used to model the stationary state of the diffusive process with $u(x)$ modeling the temperature distribution through the space with a defining the thermal conductivity of the medium. The task is to learn an operator $G_\theta : L^2(\Omega; \mathbb{R}^{d_u}) \times L^2(\Omega; \mathbb{R}^{d_a}) \to L^2(\Omega; \mathbb{R}^{d_u})$ such that $u^\star = G_\theta(u^\star, a)$.

We report the results of our experiments on Darcy Flow in Table 6.1. The original FNO architecture does not improve its performance with increased number of FNO blocks $\mathcal{B}$. FNO++ with residual connections scales better but saturates at around 4 FNO blocks. In contrast, FNO-WT and FNO-DEQ with just a *single* FNO block outperform deeper non-weight-tied architectures on clean data and on data with noisy inputs. When observations are noisy, FNO-WT and FNO-DEQ outperform FNO++ with a similar number of parameters, and perform comparably to FNO++ with $4\times$ parameters.

We also report results on shallow FNO-DEQ, FNO-WT and FNO++ architectures. An FNO block in these shallow networks has a single FNO layer instead of three layers. In our experiments, shallow weight-tied networks outperform non-weight-tied architectures including FNO++ with $7\times$ parameters on clean data and on data with noisy inputs, and perform comparably to deep FNO++ on noisy observations. In case of noisy observations, we encounter training instability issues in FNO-DEQ. We believe that this shallow network lacks sufficient representation power and cannot accurately solve for the fixed point during the forward pass. These errors in fixed point estimation accumulate over time, leading to incorrect values of implicit gradients, which in turn result in training instability issues.

6.5.2 Steady-state Navier-Stokes Equations for Incompressible Flow

We consider the steady-state Navier-Stokes equation for an incompressible viscous fluid in the vorticity form defined on a torus, i.e., with periodic boundary condition,

$$\begin{aligned} u \cdot \nabla \omega &= \nu \Delta \omega + f, \qquad && x \in \Omega \\ \nabla \cdot u &= 0 && x \in \Omega \end{aligned} \tag{6.9}$$

where $\Omega := (0, 2\pi)^2$, and $u : \Omega \to \mathbb{R}^2$ is the velocity and $\omega : \Omega \to \mathbb{R}$ where $\omega = \nabla \times u$, $\nu \in \mathbb{R}_+$ is the viscosity and $f : \Omega \to \mathbb{R}$ is the external force term. We learn an operator $G_\theta : L^2(\Omega; \mathbb{R}^{d_u}) \times L^2(\Omega; \mathbb{R}^{d_f}) \to L^2(\Omega; \mathbb{R}^{d_u})$, such that $u^\star = G_\theta(u^\star, f)$. We train all the models on data with viscosity values $\nu = 0.01$ and $\nu = 0.001$, and create a dataset for steady-state incompressible Navier-Stokes, which we will make public as a community benchmark for steady-state PDE solvers.

Architecture	Parameters	#Blocks	Test error ↓		
			$\sigma^2_{\max} = 0$	$(\sigma^2_{\max})^i = 0.001$	$(\sigma^2_{\max})^t = 0.001$
FNO	2.37M	1	0.0080 ± 5e-4	0.0079 ± 2e-4	0.0125 ± 4e-5
FNO	4.15M	2	0.0105 ± 6e-4	0.0106 ± 4e-4	0.0136 ± 2e-5
FNO	7.71M	4	0.2550 ± 2e-8	0.2557 ± 8e-9	0.2617 ± 2e-9
FNO++	2.37M	1	0.0075 ± 2e-4	0.0075 ± 2e-4	0.0145 ± 7e-4
FNO++	4.15M	2	0.0065 ± 2e-4	0.0065 ± 9e-5	0.0117 ± 5e-5
FNO++	7.71M	4	0.0064 ± 2e-4	0.0064 ± 2e-4	**0.0109 ± 5e-4**
S-FNO++	1.78M	0.66	0.0093 ± 5e-4	0.0094 ± 7e-4	0.0402 ± 6e-3
FNO-WT	2.37M	1	**0.0055 ± 1e-4**	**0.0056 ± 5e-5**	0.0112 ± 4e-4
FNO-DEQ	2.37M	1	**0.0055 ± 1e-4**	**0.0056 ± 7e-5**	0.0112 ± 4e-4
S-FNO-WT	1.19M	0.33	0.0057 ± 3e-5	0.0057 ± 5e-5	0.0112 ± 1e-4
S-FNO-DEQ	1.19M	0.33	0.0056 ± 4e-5	0.0056 ± 5e-5	0.0136 ± 0.011

Table 6.1: Results on Darcy flow: clean data (Col 4),noisy inputs (Col 5) and noisy observations (Col 6) with max variance of added noise being $(\sigma^2_{\max})^i$ and $(\sigma^2_{\max})^t$, respectively. Reported test error has been averaged on three different runs with seeds 0, 1, and 2. Here, S-FNO++, S-FNO-WT and S-FNO-DEQ are shallow versions of FNO++, FNO-WT and FNO-DEQ respectively.

Results for Navier-Stokes equation have been reported in Table 6.2 and Table 6.3. For both values of viscosity, FNO-DEQ outperforms other architectures for all three cases: clean data, noisy inputs and noisy observations. FNO-DEQ is more robust to noisy inputs compared to non-weight-tied architectures. For noisy inputs, FNO-DEQ matches the test-error of noiseless case in case of viscosity 0.01 and almost matches the test-error of noiseless case in case of viscosity 0.001. We provide additional results for noise level 0.004 in Table 6.4 and Table 6.5. FNO-DEQ and FNO-WT consistently outperform non-weight-tied architectures for higher levels of noise as well.

In general, DEQ-based architectures are slower to train (upto $\sim$2.5$\times$ compared to feedforward networks of similar size) as we solve for the fixed point in the forward pass. However, their inductive bias provides performance gains that cannot be achieved by simply stacking non-weight-tied FNO layers. In general, we observe diminishing returns in FNO++ beyond 4 blocks. Additionally, training the original FNO network on more than 4 FNO blocks is challenging, with the network often diverging during training, and therefore we do not include these results in the paper.

6.5.3 Convergence analysis of fixed point

We report variations in test error, absolute residual $\|G_\theta(\mathbf{z}_t) - \mathbf{z}_t\|_2$, and relative residual $\frac{\|G_\theta(\mathbf{z}_t)-\mathbf{z}_t\|_2}{\|\mathbf{z}_t\|_2}$ with an increase in the number of solver steps while solving for the fixed point in FNO-DEQ, for both Darcy Flow (See Table 6.6) and Steady-State Navier Stokes (See Table 6.7). We observe that all these values decrease with increase in the number of fixed point solver iterations and eventually saturate once we have a reasonable estimate of the fixed point. We observe that increasing the number of fixed point solver iterations results in a better estimation of the fixed point. For steady state PDEs, we expect the test error to reduce as the estimation of the fixed point improves. Furthermore, at inference

Architecture	Parameters	#Blocks	Test error ↓		
			$\sigma^2_{\text{max}} = 0$	$(\sigma^2_{\text{max}})^i = 0.001$	$(\sigma^2_{\text{max}})^t = 0.001$
FNO	2.37M	1	0.184 ± 0.002	0.218 ± 0.003	0.184 ± 0.001
FNO	4.15M	2	0.162 ± 0.024	0.176 ± 0.004	0.152 ± 0.005
FNO	7.71M	4	0.157 ± 0.012	0.187 ± 0.004	0.166 ± 0.013
FNO++	2.37M	1	0.199 ± 0.001	0.230 ± 0.001	0.197 ± 0.001
FNO++	4.15M	2	0.154 ± 0.005	0.173 ± 0.003	0.154 ± 0.006
FNO++	7.71M	4	0.151 ± 0.003	0.165 ± 0.004	0.149 ± 0.003
FNO-WT	2.37M	1	$\mathbf{0.123 \pm 0.004}$	$\mathbf{0.129 \pm 0.004}$	0.124 ± 0.005
FNO-DEQ	2.37M	1	$\mathbf{0.123 \pm 0.005}$	$\mathbf{0.129 \pm 0.004}$	$\mathbf{0.123 \pm 0.006}$

Table 6.2: Results on incompressible steady-state Navier-Stokes (viscosity=0.001): clean data (Col 4), noisy inputs (Col 5) and noisy observations (Col 6) with max variance of added noise being $(\sigma^2_{\text{max}})^i$ and $(\sigma^2_{\text{max}})^t$, respectively. Reported test error has been averaged on three different runs with seeds 0, 1, and 2.

Architecture	Parameters	#Blocks	Test error ↓		
			$\sigma^2_{\text{max}} = 0$	$(\sigma^2_{\text{max}})^i = 0.001$	$(\sigma^2_{\text{max}})^t = 0.001$
FNO	2.37M	1	0.181 ± 0.005	0.186 ± 0.003	0.178 ± 0.006
FNO	4.15M	2	0.138 ± 0.007	0.150 ± 0.006	0.137 ± 0.012
FNO	7.71M	4	0.152 ± 0.006	0.163 ± 0.002	0.151 ± 0.008
FNO++	2.37M	1	0.188 ± 0.002	0.207 ± 0.004	0.187 ± 0.003
FNO++	4.15M	2	0.139 ± 0.004	0.153 ± 0.002	0.140 ± 0.005
FNO++	7.71M	4	0.130 ± 0.005	0.151 ± 0.004	0.128 ± 0.009
FNO-WT	2.37M	1	0.089 ± 0.004	$\mathbf{0.089 \pm 0.003}$	0.089 ± 0.004
FNO-DEQ	2.37M	1	$\mathbf{0.085 \pm 0.005}$	0.090 ± 0.003	$\mathbf{0.087 \pm 0.007}$

Table 6.3: Results on incompressible steady-state Navier-Stokes (viscosity=0.01): clean data (Col 4), noisy inputs (Col 5) and noisy observations (Col 6) with max variance of added noise being $(\sigma^2_{\text{max}})^i$ and $(\sigma^2_{\text{max}})^t$, respectively. Reported test error has been averaged on three different runs with seeds 0, 1, and 2.

time we observe that the test error improves (i.e. reduces) with increase in the number of fixed point solver iterations even though the FNO-DEQ is trained with fewer solver steps. For Navier-Stokes with viscosity 0.01, at inference time we get a test MSE loss of 0.0744 with 48 solver steps from 0.0847 when used with 24 solver steps.

This further bolsters the benefits of DEQs (and weight-tied architectures in general) for training neural operators for steady-state PDEs. Moreover, performance saturates after a certain point once we have a reasonable estimate of the fixed point, hence showing that more solver steps stabilize to the same solution.

Architecture	Parameters	#Blocks	Test error ↓		
			$\sigma^2_{\max} = 0$	$(\sigma^2_{\max})^i = 0.004$	$(\sigma^2_{\max})^t = 0.004$
FNO	2.37M	1	0.184 ± 0.002	0.238 ± 0.008	0.179 ± 0.004
FNO	4.15M	2	0.162 ± 0.024	0.196 ± 0.011	0.151 ± 0.010
FNO	7.71M	4	0.157 ± 0.012	0.216 ± 0.002	0.158 ± 0.009
FNO++	2.37M	1	0.199 ± 0.001	0.255 ± 0.002	0.197 ± 0.004
FNO++	4.15M	2	0.154 ± 0.005	0.188 ± 0.006	0.157 ± 0.006
FNO++	7.71M	4	0.151 ± 0.003	0.184 ± 0.008	0.147 ± 0.004
FNO-WT	2.37M	1	$\mathbf{0.123 \pm 0.004}$	0.141 ± 0.003	$\mathbf{0.125 \pm 0.007}$
FNO-DEQ	2.37M	1	$\mathbf{0.123 \pm 0.005}$	$\mathbf{0.139 \pm 0.007}$	0.127 ± 0.002

Table 6.4: **Results on incompressible Steady-State Navier-Stokes (viscosity=0.001)**: clean data (Col 4), noisy inputs (Col 5) and noisy observations (Col 6) with max variance of added noise being $(\sigma^2_{\max})^i$ and $(\sigma^2_{\max})^t$, respectively. Reported test error has been averaged on three different runs with seeds 0, 1, and 2.
‡ indicates that the network diverges during training for one of the seeds.

Architecture	Parameters	#Blocks	Test error ↓		
			$\sigma^2_{\max} = 0$	$(\sigma^2_{\max})^i = 0.004$	$(\sigma^2_{\max})^t = 0.004$
FNO	2.37M	1	0.181 ± 0.005	0.207 ± 0.003	0.178 ± 0.008
FNO	4.15M	2	0.138 ± 0.007	0.163 ± 0.003	0.137 ± 0.006
FNO	7.71M	4	0.152 ± 0.006	0.203 ± 0.055	0.151 ± 0.008
FNO++	2.37M	1	0.188 ± 0.002	0.217 ± 0.001	0.187 ± 0.005
FNO++	4.15M	2	0.139 ± 0.004	0.170 ± 0.005	0.138 ± 0.005
FNO++	7.71M	4	0.130 ± 0.005	0.168 ± 0.007	0.126 ± 0.007
FNO-WT	2.37M	1	0.089 ± 0.004	0.097 ± 0.008	$\mathbf{0.087 \pm 0.003}$
FNO-DEQ	2.37M	1	$\mathbf{0.085 \pm 0.005}$	$\mathbf{0.096 \pm 0.008}$	$\mathbf{0.087 \pm 0.004}$

Table 6.5: **Results on incompressible Steady-State Navier-Stokes (viscosity=0.01)**: clean data (Col 4), noisy inputs (Col 5) and noisy observations (Col 6) with max variance of added noise being $(\sigma^2_{\max})^i$ and $(\sigma^2_{\max})^t$, respectively. Reported test error has been averaged on three different runs with seeds 0, 1, and 2.
‡ indicates that the network diverges during training for one of the seeds.

6.5.4 Train and Test Loss Curves

We visualize training and test loss curves for steady-state Navier-Stokes with viscosity 0.01 in Figure 6.1. We observe that while all the models converge to approximately the same MSE loss value while training, DEQs and weight-tied networks get a better test loss in fewer epochs.

Solver steps	Absolute residual ↓	Relative residual ↓	Test Error ↓
2	212.86	0.8533	0.0777
4	18.166	0.0878	0.0269
8	0.3530	0.00166	0.00567
16	0.00239	1.13e-5	0.00566
32	0.000234	1.1e-6	0.00566

Table 6.6: Convergence analysis of fixed point for noiseless Darcy Flow: The test error, absolute residual $\|G_\theta(\mathbf{z}_t) - \mathbf{z}_t\|_2$ and relative residual $\frac{\|G_\theta(\mathbf{z}_t)-\mathbf{z}_t\|_2}{\|\mathbf{z}_t\|_2}$ decrease with increase in the number of fixed point solver iterations. The performance saturates after a certain point once we have a reasonable estimate of the fixed point. We consider the noiseless case, where we do not add any noise to inputs or targets.

Solver steps	Absolute residual ↓	Relative residual ↓	Test Error ↓
4	544.16	0.542	0.926
8	397.75	0.408	0.515
16	150.33	0.157	0.147
24	37.671	0.0396	0.0847
48	5.625	0.0059	0.0744
64	3.3	0.0034	0.0746

Table 6.7: Convergence analysis of fixed point for noiseless incompressible Steady-State Navier-Stokes with viscosity=0.01: The test error, absolute residual $\|G_\theta(\mathbf{z}_t) - \mathbf{z}_t\|_2$ and relative residual $\frac{\|G_\theta(\mathbf{z}_t)-\mathbf{z}_t\|_2}{\|\mathbf{z}_t\|_2}$ decrease with increase in the number of fixed point solver iterations. The performance saturates after a certain point once we have a reasonable estimate of the fixed point. We consider the noiseless case, where we do not add any noise to inputs or targets.

6.6 Universal Approximation and Fast Convergence of FNO-DEQ

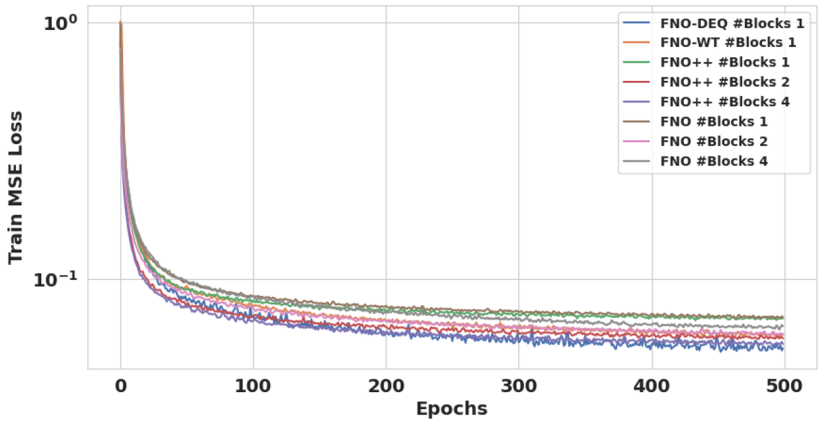
Though the primary contribution of our paper is empirical, we show (by fairly standard techniques) that FNO-DEQ is a universal approximator, under mild conditions on the steady-state PDEs. Moreover, we also show that in some cases, we can hope the fixed-point solver can converge rapidly.

As noted in Definition 2, we have $\Omega := \mathbb{T}^d$. We note that all continuous function $f \in L^2(\Omega; \mathbb{R})$ and $\int_\Omega |f(x)|dx < \infty$ can be written as, $f(x) = \sum_{\omega \in \mathbb{Z}^d} e^{ix^T\omega} \hat{f}_w$. where $\{\hat{f}_\omega\}_{\omega \in \mathbb{Z}^d}$ are the Fourier coefficients of the function f. We define as $L^2_N(\Omega)$ as the space of functions such that for all $f_N \in L^2_N(\Omega)$ with Fourier coefficients that vanish outside a bounded ball. Finally, we define an orthogonal projection operator $\Pi_N : L^2(\Omega) \to L^2_N(\Omega)$, such that for all $f \in L^2(\Omega)$ we have,

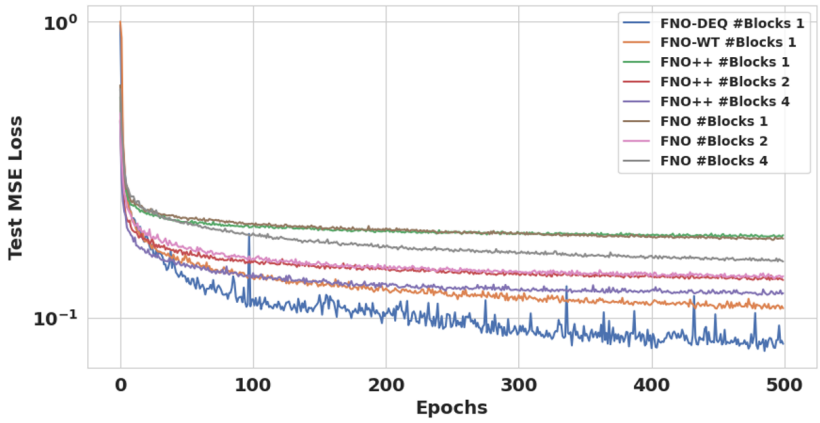
$$f_n = \Pi_N(f) = \Pi_N \left(\sum_{\omega \in \mathbb{Z}^d} f_\omega e^{ix^T\omega} \right) = \sum_{\|\omega\|_\infty \leq N} \hat{f}_\omega e^{ix^T\omega}. \tag{6.10}$$

That is, the projection operator Π_N takes an infinite dimensional function and projects it to a finite dimensional space. We prove the following universal approximation result:

Theorem 1. *Let $u^\star \in L^2(\Omega; \mathbb{R}^{d_u})$ define the solution to a steady-state PDE in Definition 2, Then there exists*



(a) Training Loss Curve



(b) Test Loss Curve

Figure 6.1: Training and Test Loss Curves for Steady-State Navier-Stokes with viscosity 0.01. The x axis is the number of epochs and y axis is the MSE loss in log scale. Note that while all the models converge to approximately the same MSE loss value while training, DEQs and weight-tied networks get a better test loss in fewer epochs.

an operator $\mathcal{G} : L^2(\Omega; \mathbb{R}^{d_u}) \times L^2(\Omega; \mathbb{R}^{d_f}) \to L^2(\Omega; \mathbb{R}^{d_u})$ *such that,* $u^\star = \mathcal{G}(u^\star, f)$*. Furthermore, for every* $\epsilon > 0$ *there exists an* $N \in \mathbb{N}$ *such that for compact sets* $K_u \subset L^2(\Omega; \mathbb{R}^{d_u})$ *and* $K_f \subset L^2(\Omega; \mathbb{R}^{d_f})$ *there exists a neural network* $G_\theta : L^2_N(\Omega; \mathbb{R}^{d_u}) \times L^2_N(\Omega; \mathbb{R}^{d_f}) \to L^2_N(\Omega; \mathbb{R}^{d_u})$ *with parameters* θ*, such that,*

$$\sup_{u \in K_u, f \in K_f} \|u^\star - G_\theta(\Pi_N u^\star, \Pi_N f)\|_{L^2(\Omega)} \leq \epsilon.$$

The proof for the above theorem is relatively straightforward and provided in Appendix 6.6.1. The proof uses the fact that $u^\star$ is a fixed-point of the operator $G(u, f) = u - (L(u) - f)$, allowing us to use the the results in Kovachki et al. [2021a] that show a continuous operator can be approximated by a network as defined in (6.1). Note that the choice of G is by no means unique: one can "universally approximate" any operator $G(u, f) = u - A(L(u) - f)$, for a continuous operator A. Such a G can be thought of as a form of "preconditioned" gradient descent, for a preconditioner A. For example, a Newton update has the form $G(u, f) = u - L'(u)^{-1}(L(u) - f)$, where $L' : L^2(\Omega; \mathbb{R}^{d_u}) \to L^2(\Omega; \mathbb{R}^{d_u})$ is the Frechet derivative of the operator L.

The reason this is relevant is that the DEQ can choose to universally approximate a fixed-point equation for which the fixed-point solver it is trained with also converges rapidly. As an example, the following classical result shows that under Lax-Milgram-like conditions (a kind of strong convexity condition), Newton's method converges doubly exponentially fast:

Lemma 1 (Faragó and Karátson [2002], Chapter 5)**.** *Consider the PDE defined Definition 2, such that $d_u = d_v = d_f = 1$. such that $L'(u)$ defines the Frechet derivative of the operator L. If for all $u, v \in L^2(\Omega; \mathbb{R})$ we have $\|L'(u)v\|_{L^2(\Omega)} \geq \lambda \|v\|_{L^2(\Omega)}$ and $\|L'(u) - L'(v)\|_{L^2(\Omega)} \leq \Lambda \|u - v\|_{L^2(\Omega)}$ for $0 < \lambda \leq \Lambda < \infty$, then for the Newton update, $u_{t+1} \leftarrow u_t - L'(u_t)^{-1}\left(L(u_t) - f\right)$, with $u_0 \in L^2(\Omega; \mathbb{R})$, there exists an $\epsilon > 0$, such that $\|u_T - u^\star\|_{L^2(\Omega)} \leq \epsilon$ if $T \geq \log\left(\log\left(\frac{1}{\epsilon}\right) / \log\left(\frac{2\lambda^2}{\Lambda \|L(u_0) - f\|_{L^2(\Omega)}}\right)\right)$.*

We note that the conditions of the above lemma are satisfied for elliptic PDEs like Darcy Flow, as well as many variational non-linear elliptic PDEs (e.g., those considered in Marwah et al. [2022]). Hence, we can expect FNO-DEQs to quickly converge to the fixed point, since they employ quasi-Newton methods like Broyden and Anderson methods [Broyden, 1965, Anderson, 1965]

6.6.1 Proof of Universal Approximation of FNO-DEQ

The proof of the universal approximation essentially follows from the result on the universal approximation capabilities of FNO layers in Kovachki et al. [2021a], applied to $\mathcal{G}(v, f) = v - (Lv - f)$. For the sake of completeness, we reiterate the key steps.

For simplicity, we will assume that $d_u = d_v = d_f = 1$. (The results straightforwardly generalize.) We will first establish some key technical lemmas and introduce some notation and definitions useful for the proof for Theorem 1.

Definition 7. *An operator $T : L^2(\Omega; \mathbb{R}) \to L^2(\Omega; \mathbb{R})$ is continuous at $u \in L^2(\Omega; \mathbb{R})$ if for every $\epsilon > 0$, there exists a $\delta > 0$, such that for all $v \in L^2(\Omega)$ with $\|u - v\|_{L^2(\Omega)} \leq \delta$, we have $\|L(u) - L(v)\|_{L^2(\Omega)} \leq \epsilon$.*

First, we approximate the infinite-dimensional operator $\mathcal{G} : L^2(\Omega) \times L^2(\Omega) \to L^2(\Omega)$ by projecting the functions in $L^2(\Omega)$ to a finite-dimensional approximation $L^2_N(\Omega)$, and considering the action of the operator in this subspace. The linear projection we use is the one introduced in (6.10). More precisely we show the following result,

Lemma 2. *Given a continuous operator $L : L^2(\Omega) \to L^2(\Omega)$ as defined in (6.4), let us define an operator $\mathcal{G} : L^2(\Omega) \times L^2(\Omega) \to L^2(\Omega)$ as $\mathcal{G}(v, f) := v - (L(v) - f)$. Then, for every $\epsilon > 0$ there exists an $N \in \mathbb{N}$ such that for all v, f in any compact set $K \subset L^2(\Omega)$, the operator $\mathcal{G}_N = \Pi_N \mathcal{G}(\Pi_N v, \Pi_N f)$ is an ϵ-approximation of $\mathcal{G}(v, f)$, i.e., we have,*

$$\sup_{v,f \in K} \|\mathcal{G}(v, f) - \mathcal{G}_N(v, f)\|_{L^2(\Omega)} \leq \epsilon.$$

Proof. Note that for an $\epsilon > 0$ there exists an $N = N(\epsilon, d)$ such that for all $v \in K$ we have

$$\sup_{v \in K} \|v - \Pi_N v\|_{L^2(\Omega)} \leq \epsilon.$$

Therefore, using the definition of $\mathcal{G}_N$ we can bound the $L^2(\Omega)$ norm of the difference between $\mathcal{G}$ and $\mathcal{G}_N$ as follows,

$$\begin{aligned}
&\|\mathcal{G}(v,f) - \Pi_N\mathcal{G}(v_n,f_n)\|_{L^2(\Omega)} \\
&\leq \|\mathcal{G}(v,f) - \Pi_N\mathcal{G}(v,f)\|_{L^2(\Omega)} + \|\Pi_N\mathcal{G}(v,f) - \Pi_N\mathcal{G}(\Pi_N v,\Pi_N f)\|_{L^2(\Omega)} \\
&\leq \underbrace{\|\mathcal{G}(v,f) - \Pi_N\mathcal{G}(v,f)\|_{L^2(\Omega)}}_{I} + \underbrace{\|\mathcal{G}(v,f) - \mathcal{G}(\Pi_N v,\Pi_N f)\|_{L^2(\Omega)}}_{II}
\end{aligned}$$

We first bound the term I as follows:

$$\begin{aligned}
&\|\mathcal{G}(v,f) - \Pi_N\mathcal{G}(v,f)\|_{L^2(\Omega)} \\
&= \|v - (L(v) - f) - \Pi_N\left(v - (L(v) - f)\right)\|_{L^2(\Omega)} \\
&= \|v - \Pi_N v\|_{L^2(\Omega)} + \|f - \Pi_N f\|_{L^2(\Omega)} + \|L(v) - \Pi_N L(v)\|_{L^2(\Omega)} \\
&= \epsilon + \epsilon + \|L(v) - \Pi_N L(v)\|_{L^2(\Omega)} \qquad (6.11)
\end{aligned}$$

Since L is continuous, for all compact sets $K \subset L^2(\Omega)$, $L(K)$ is compact as well. This is because: (1) for any $u \in K$, $\|L(u)\|_{L^2(\Omega)}$ is finite; (2) for any $v \in K$, $\|L(v)\|_{L^2(\Omega)} \leq \|L(u)\|_{L^2(\Omega)} + C\|u - v\|_{L^2(\Omega)}$. Therefore, for every $\epsilon > 0$, there exists an $N \in \mathbb{N}$ such that

$$\sup_{v\in K} \|L(v) - \Pi_N L(v)\|_{L^2(\Omega)} \leq \epsilon.$$

Substituting the above result in (6.11), we have

$$\|\mathcal{G}(v,f) - \Pi_N\mathcal{G}(v,f)\|_{L^2(\Omega)} \leq 3\epsilon. \qquad (6.12)$$

Similarly, for all $v \in K$ where K is compact, we can bound Term II as following,

$$\begin{aligned}
&\|\mathcal{G}(v,f) - \mathcal{G}(\Pi_N v,\Pi_N f)\|_{L^2(\Omega)} \\
&\leq \|v - (L(v) - f) - \Pi_N v - (L(\Pi_N v) - \Pi_N f)\|_{L^2(\Omega)} \\
&\leq \|v - \Pi_N v\|_{L^2(\Omega)} + \|f - \Pi_N f\|_{L^2(\Omega)} + \|L(v) - L(\Pi_N v)\|_{L^2(\Omega)} \\
&\leq \epsilon + \epsilon + \|L(v) - L(\Pi_N v)\|_{L^2(\Omega)}. \qquad (6.13)
\end{aligned}$$

Now, since $v \in K$ and $L : L^2(\Omega) \to L^2(\Omega)$ is a continuous operator, there exists a modulus of continuity (an increasing real valued function) $\alpha \in [0,\infty)$, such that for all $v \in K$, we have

$$\|L(v) - L(\Pi_N v)\|_{L^2(\Omega)} \leq \alpha\left(\|v - \Pi_N v\|_{L^2(\Omega)}\right)$$

Hence for every $\epsilon > 0$ there exists an $N \in \mathbb{N}$ such that,

$$\alpha(\|v - \Pi_N v\|_{L^2(\Omega)}) \leq \epsilon.$$

Plugging these bounds in (6.13), we get,

$$\|\mathcal{G}(v,f) - \mathcal{G}(\Pi_N v,\Pi_N f)\|_{L^2(\Omega)} \leq 3\epsilon. \qquad (6.14)$$

Therefore, combining (6.12) and (6.14) then for $\epsilon > 0$, there exists an $N \in \mathbb{N}$, such that for all $v, f \in K$ we have

$$\sup_{v,f\in K} \|\mathcal{G}(v,f) - \Pi_N\mathcal{G}(v_n,f_n)\|_{L^2(\Omega)} \leq 6\epsilon. \qquad (6.15)$$

Taking $\epsilon' = 6\epsilon$ proves the claim. □

Proof of Theorem 1. For Lemma 2 we know that there exists a finite dimensional projection for the operator $\mathcal{G}$, defined as $\mathcal{G}_N(v, f)$ such that for all $v, f \in L^2(\Omega)$ we have

$$\|\mathcal{G}(v, f) - \mathcal{G}_N(v, f)\|_{L^2(\Omega)} \leq \epsilon.$$

Now using the definition of $\mathcal{G}_N(v, f)$ we have

$$\begin{aligned}\mathcal{G}_N(v, f) &= \Pi_N \mathcal{G}(\Pi_N v, \Pi_N f) \\ &= \Pi_N v - (\Pi_N L(\Pi_N v) - \Pi_N f)\end{aligned}$$

From Kovachki et al. [2021a], Theorem 2.4 we know that there exists an FNO network G_{θ^L} of the form defined in (6.1) such that for all $v \in K$, where K is a compact set, there exists an ϵ^L we have

$$\sup_{v \in K} \|\Pi_N L(\Pi_N v) - G_{\theta^L}\|_{L^2(\Omega)} \leq \epsilon^L \tag{6.16}$$

Finally, note that from Lemma D.1 in Kovachki et al. [2021a], we have that for any $v \in K$, there exists an FNO layers $G_{\theta^f} \in L^2(\Omega)$ and $G_{\theta^v} \in L^2(\Omega)$ defined in (6.2) such that

$$\sup_{v \in K} \|\Pi_N v - G_{\theta^v}\|_{L^2(\Omega)} \leq \epsilon^v \tag{6.17}$$

and

$$\sup_{f \in K} \|\Pi_N f - G_{\theta^f}\|_{L^2(\Omega)} \leq \epsilon^f \tag{6.18}$$

for $\epsilon^v > 0$ and $\epsilon^f > 0$.

Therefore there exists an $\tilde{\epsilon} >$ such that there is an FNO network $G_\theta : L^2(\Omega) \times L^2(\Omega) \to L^2(\Omega)$ where $\theta := \{\theta^L, \theta^v, \theta^f\}$ such that

$$\sup_{v \in K, f \in L^2(\Omega)} \|\mathcal{G}_N(v, f) - G_\theta(v, f)\|_{L^2(\Omega)} \leq \tilde{\epsilon} \tag{6.19}$$

Now, since we know that $u^\star$ is the fixed point of the operator $\mathcal{G}$ we have from Lemma 2 and (6.19),

$$\begin{aligned}\|\mathcal{G}(u^\star, f) - G_\theta(u^\star, f)\|_{L^2(\Omega)} &\leq \|u^\star - \mathcal{G}_N(u^\star, f)\|_{L^2(\Omega)} + \|\mathcal{G}_N(u^\star, f) - G_\theta(u^\star, f)\|_{L^2(\Omega)} \\ &\leq \tilde{\epsilon} + \epsilon.\end{aligned}$$

□

6.6.2 Proof of Fast Convergence for Newton Method

Definition 8 (Frechet Derivative in $L^2(\Omega)$)**.** *For a continuous operator $F : L^2(\Omega) \to L^2(\Omega)$, the Frechet derivative at $u \in L^2(\Omega)$ is a linear operator $F'(u) : L^2(\Omega) \to L^2(\Omega)$ such that for all $v \in L^2(\Omega)$ we have*

$$\lim_{\|v\|_{L^2(\Omega)} \to 0} \frac{\|F(u + v) - F(u) - F'(u)(v)\|_{L^2(\Omega)}}{\|v\|_{L^2(\Omega)}} = 0.$$

Lemma 3. *Given the operator $L : L^2(\Omega) \to L^2(\Omega)$ with Frechet derivative L', such that for all $u, v \in L^2(\Omega)$, we have $\|L'(u)(v)\|_{L^2(\Omega)} \geq \lambda \|v\|_{L^2(\Omega)}$, then $L'(u)^{-1}$ exists and we have, for all $v_1, v_2 \in L^2(\Omega)$:*

1. $\|L'(u)^{-1}(v_1)\|_{L^2(\Omega)} \leq \frac{1}{\lambda}\|v_1\|_{L^2(\Omega)}$.
2. $\|v_1 - v_2\|_{L^2(\Omega)} \leq \frac{1}{\lambda}\|L(v_1) - L(v_2)\|_{L^2(\Omega)}$

Proof. Note that for all $u, v' \in L^2(\Omega)$ we have,

$$\|L'(u)v'\|_{L^2(\Omega)} \geq \lambda\|v'\|_{L^2(\Omega)}$$

Taking $v = L'(u)^{-1}(v')$, we have

$$\begin{aligned}\|L'(u)\left(L'(u)^{-1}(v)\right)\|_{L^2(\Omega)} &\geq \lambda\|L^{-1}(u)(v)\|_{L^2(\Omega)} \\ \implies \frac{1}{\lambda}\|v\|_{L^2(\Omega)} &\geq \|L^{-1}(u)(v)\|_{L^2(\Omega)}.\end{aligned}$$

For part 2, note that there exists a $c \in [0,1]$ such that

$$\|L(v_1) - L(v_2)\|_{L^2(\Omega)} \geq \inf_{c\in[0,1]} \|L'(cv_1 + (1-c)v_2)\|_2\|v_1 - v_2\|_{L^2(\Omega)} \geq \lambda\|v_1 - v_2\|_{L^2(\Omega)}.$$

□

We now show the proof for Lemma Lemma 1. The proof is standard and can be found in Faragó and Karátson [2002], however we include the complete proof here for the sake of completeness.

We restate the Lemma here for the convenience of the reader.

Lemma 4 (Faragó and Karátson [2002], Chapter 5)**.** *Consider the PDE defined Definition 2, such that $d_u = d_v = d_f = 1$. such that $L'(u)$ defines the Frechet derivative of the operator L. If for all $u, v \in L^2(\Omega; \mathbb{R})$ we have $\|L'(u)v\|_{L^2(\Omega)} \geq \lambda\|v\|_{L^2(\Omega)}$ [3] and $\|L'(u) - L'(v)\|_{L^2(\Omega)} \leq \Lambda\|u - v\|_{L^2(\Omega)}$ for $0 < \lambda \leq \Lambda < \infty$, then for the Newton update, $u_{t+1} \leftarrow u_t - L'(u_t)^{-1}\left(L(u_t) - f\right)$, with $u_0 \in L^2(\Omega; \mathbb{R})$, there exists an $\epsilon > 0$, such that $\|u_T - u^\star\|_{L^2(\Omega)} \leq \epsilon$ if [4] $T \geq \log\left(\log\left(\frac{1}{\epsilon}\right) / \log\left(\frac{2\lambda^2}{\Lambda\|L(u_0)-f\|_{L^2(\Omega)}}\right)\right)$.*

Proof of Lemma 1. Re-writing the updates in Lemma 1 as,

$$u_{t+1} = u_t + p_t \tag{6.20}$$
$$L'(u_t)p_t = -(L(u_t) - f) \tag{6.21}$$

Now, upper bounding $L(u_{t+1}) - f$ for all $x \in \Omega$ we have,

$$\begin{aligned}&L(u_{t+1}(x)) - f(x) \\ &= L(u_t(x)) - f(x) + \int_0^1 \left(L'(u_t(x) + t(u_{t+1}(x) - u_t(x)))\right)(u_{t+1}(x) - u_t(x))dt \\ &= L(u_t(x)) - f(x) + L'(u_t(x))p_t(x) + \int_0^1 \left(L'(u_t(x) + t(u_{t+1}(x) - u_t(x))) - L'(u_t(x))\right)p_t(x)dt \\ &= \int_0^1 \left(L'(u_t(x) + t(u_{t+1}(x) - u_t(x))) - L'(u_t(x))\right)p_t(x)dt\end{aligned}$$

[3] We note that this condition is different from the condition on the inner-product in the submitted version of the paper, which had. $\langle L'(u), v\rangle_{L^2(\Omega)} \geq \lambda\|v\|_{L^2(\Omega)}$.

[4] We note that this rate is different from the one in the submitted version of the paper.

where we use (6.21) in the final step.

Taking $L^2(\Omega)$ norm on both sides and using the fact that $\|L'(u) - L'(v)\|_{L^2(\Omega)} \leq \Lambda \|u - v\|_{L^2(\Omega)}$, we have

$$\|L(u_{t+1}) - f\|_{L^2(\Omega)} \leq \int_0^1 \Lambda t \|u_{t+1} - u_t\|_{L^2(\Omega)} \|p_t\|_{L^2(\Omega)} dt$$

Noting that for all $x \in \Omega$, we have $u_{t+1} - u_t = p_t$, and using the fact that for all u, v $\|L'(u)^{-1} v\|_{L^2(\Omega)} \leq \frac{1}{\lambda} \|v\|_{L^2(\Omega)}$ we have, $\|L'(u_t) p_t\|_{L^2(\Omega)} \leq \frac{1}{\lambda} \|p_t\|_{L^2(\Omega)}$

$$\begin{aligned}
\|L(u_{t+1}) - f\|_{L^2(\Omega)} &\leq \int_0^1 \Lambda t \|u_{t+1} - u\|_{L^2(\Omega)} \|p_t\|_{L^2(\Omega)} dt \\
&\leq \Lambda/2 \|p_t\|^2_{L^2(\Omega)} \\
&\leq \Lambda/2 \| - L'(u_t)^{-1} (L(u_t) - f)\|^2_{L^2(\Omega)} \\
&\leq \frac{\Lambda}{2\lambda^2} \|L(u_t) - f)\|^2_{L^2(\Omega)}
\end{aligned}$$

where we use the result from Lemma 3 in the last step.

Therefore we have

$$\begin{aligned}
&\|L(u_{t+1}) - f\|_{L^2(\Omega)} \leq \left(\frac{\Lambda}{2\lambda^2}\right)^{2^t - 1} (L(u_0) - f)^{2^t} \\
\implies &\|L(u_{t+1}) - f\|_{L^2(\Omega)} \leq \left(\frac{\Lambda}{2\lambda^2}\right)^{2^t - 1} (L(u_0) - L(u^\star))^{2^t} \\
\implies &\|u_{t+1} - u^\star\|_{L^2(\Omega)} \leq \frac{1}{\lambda} \left(\frac{\Lambda}{2\lambda^2}\right)^{2^t - 1} \|L(u_0) - L(u^\star)\|^{2^t}_{L^2(\Omega)}.
\end{aligned}$$

Therefore, if

$$\frac{\Lambda}{2\lambda^2} \|L(u_0) - L(u^\star)\|_{L^2(\Omega)} \leq 1,$$

then we have

$$\|u_{t+1} - u^\star\|_{L^2(\Omega)} \leq \epsilon,$$

for

$$T \geq \log\left(\log\left(\frac{1}{\epsilon}\right) / \log\left(\frac{2\lambda^2}{\Lambda \|L(u_0) - f\|_{L^2(\Omega)}}\right)\right).$$

□

6.7 Visualization of samples from datasets

We visualize random pairs of input and output $a(x)$ and the output $u(x)$ for Darcy flow Figure 6.2 and Steady-State Navier Stokes for viscosity = 0.001 in Figure 6.3, and viscosity = 0.01 in Figure 6.4.

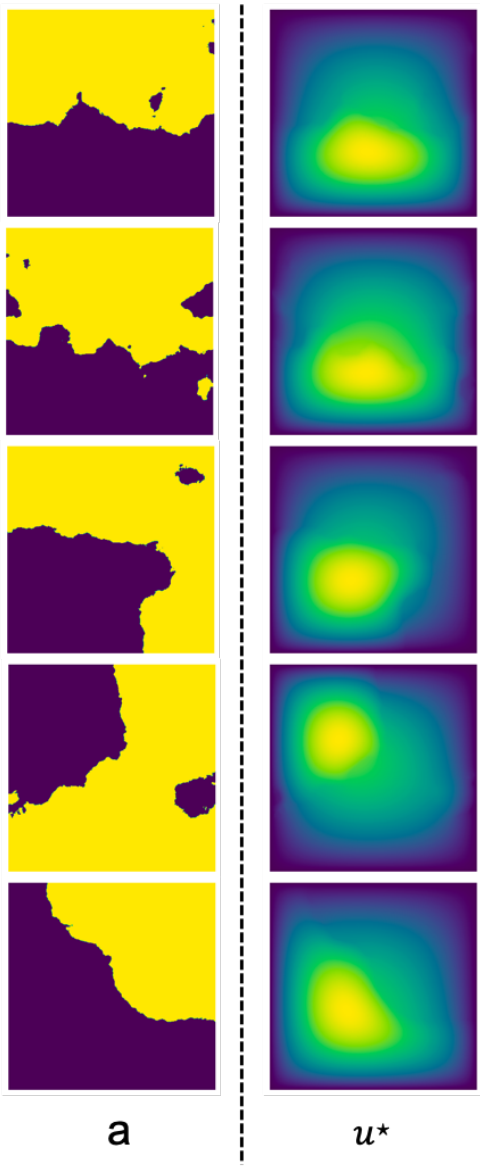


Figure 6.2: Samples from Darcy Flow PDE dataset

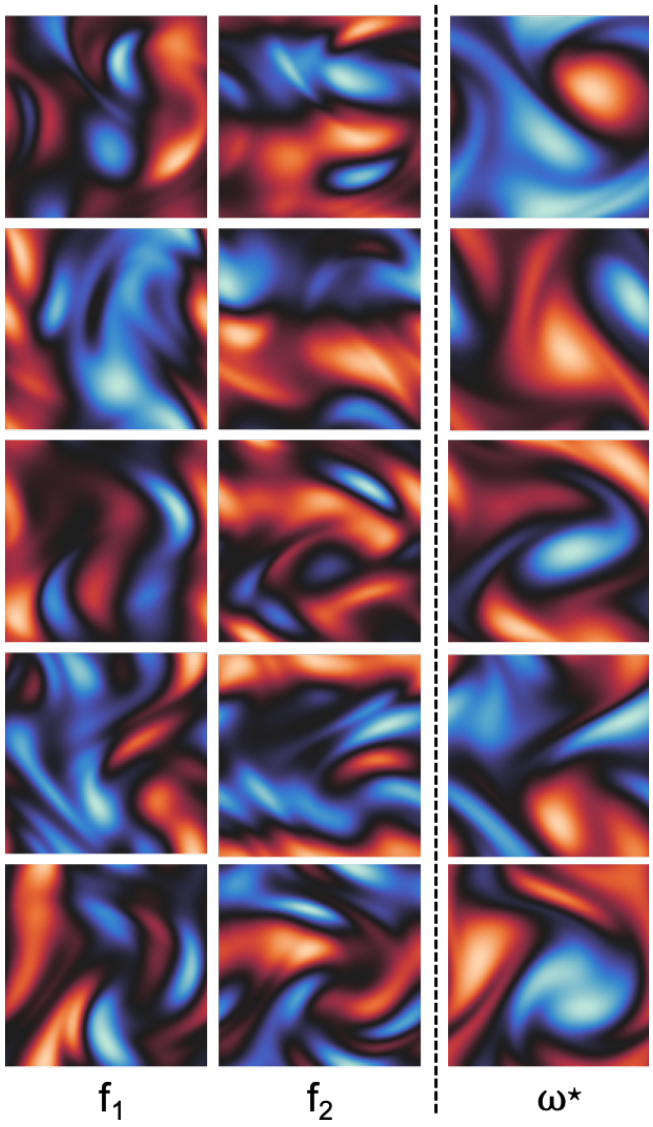


Figure 6.3: Samples from Steady-state Navier-Stokes dataset with viscosity 0.001. Each triplet visualizes the inputs f_1, f_2 and the ground truth output i.e. $\omega^\star$.

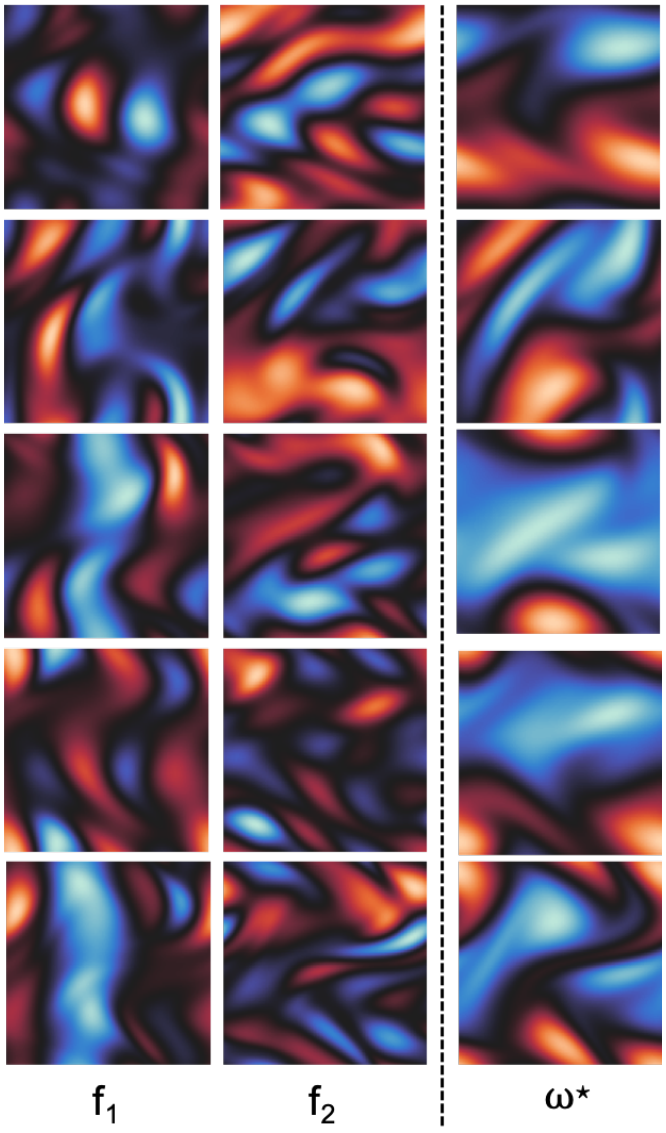


Figure 6.4: Samples from Steady-state Navier-Stokes dataset with viscosity 0.01. Each triplet visualizes the inputs f_1, f_2 and the ground truth output i.e. $\omega^\star$.

6.8 Discussion

We demonstrate that the inductive bias of deep equilibrium models—and weight-tied networks in general—makes them ideal architectures for approximating neural operators for steady-state PDEs. Our experiments on steady-state Navier-Stokes equation and Darcy flow equations show that weight-tied models and FNO-DEQ perform outperform FNO models with $\sim 4\times$ the number of parameters and depth. Our findings indicate that FNO-DEQ and weight-tied architectures are, in general, more robust to both input and observation noise compared to non-weight-tied architectures, including FNO. We believe that our results complement any future progress in the design and development of PDE solvers [Tran et al., 2021, Li et al., 2022a] for steady-state PDEs, and hope that our work motivates the study of relevant inductive biases that could be used to improve them.

Part III

Algorithmic Generalization with Deep Equilibrium Models

Chapter 7

Understanding Upwards Generalization with Deep Equilibrium Models

One of the main challenges limiting the practical applicability of modern deep learning systems is the ability to generalize outside the training distribution [Koh et al., 2021]. One particularly important type of out-of-distribution (OOD) generalization is *upwards generalization*, or the ability to generalize to more difficult problem instances than those encountered at training time [Selsam et al., 2018, Bansal et al., 2022, Schwarzschild et al., 2021b, Nye et al., 2021]. Often, good performance on more difficult instances will require a larger amount of test-time computation, so a natural question arises: how can we design neural net architectures which can reliably exploit additional test-time computation to achieve better accuracy? In this chapter, we show that a broad class of architectures named *equilibrium models* display strong upwards generalization, and find that stronger performance on harder examples (which require more iterations of inference to get correct) strongly correlates with the *path independence* of the system—its tendency to converge to the same steady-state behaviour regardless of initialization, given enough computation.

7.1 Preliminary

In this chapter, we will examine upward generalization capability of DEQ models and depthwise recurrent models. We point the reader to Sec. 2.1 for a detailed overview of Deep Equilibrium (DEQ) models. In this section, we further list primary distinction between these two network architectures.

7.1.1 Distinction between DEQs and Depthwise Recurrent Networks

Both deep equilibrium (DEQ) models and input-injected depthwise recurrent (i.e. weight-tied, fixed-depth) networks leverage weight-tying *i.e.,* they apply the same transformation at each layer, $f_\theta^{[i]} =$

$f_\theta \; \forall i$. The two models differ in the ultimate aim of the forward pass: while depthwise recurrent models compute a (weight-tied) fixed depth computation (which may or may not approach a fixed point), the stated *goal* of equilibrium models is explicitly to find a fixed point. Weight-tied fixed depth networks, by definition, require backpropagation through an explicit stack of layers. Deep equilibrium models, however, directly solve for fixed points using (potentially black-box) solvers during the forward pass and may be trained using implicit differentiation.

7.2 Related Work

There is a long line of research on neural networks that can adapt their computational budget based on the complexity of the task they are learning to solve—akin to the intrinsic mechanism in humans to reason and solve problems. Schmidhuber [2012] introduced self-delimiting neural networks which are a type of recurrent neural networks (RNNs) that adapt their compute based on the output of a special ”halt” neuron. Adaptive computation time (ACT) [Graves, 2016a] also uses the output of a sigmoidal halting unit to determine the termination condition of an RNN, but it avoids long ”thinking” time by explicitly penalizing it. Subsequent works have successfully applied variants of ACT in image classification and object detection [Figurnov et al., 2017], visual reasoning [Eyzaguirre and Soto, 2020], Transformers [Vaswani et al., 2017b] for language modelling [Dehghani et al., 2019, Elbayad et al., 2020a, Liu et al., 2021], and recognizing textual entailment [Neumann et al., 2016]. PonderNet [Banino et al., 2021] reformulates the halting policy of ACT as a probabilistic model, and adds a regularization term in the loss objective to encourage exploration. With these additions, PonderNet can extrapolate to more difficult examples on the parity task, first proposed by Graves [2016b]: in a vector with entries of 0, -1, and 1, output 1 for odd number of ones, and 0 otherwise. In this work, we do not optimize or penalize the network for the number of computational steps. Our main goal is to understand the underlying mechanism that results in scalable generalization of equilibrium models on harder problem instances. Our current work is closely related to previous work by Schwarzschild et al. [2021b] and [Bansal et al., 2022] that propose architectural choices and training mechanisms that enable weight tied networks to generalize on harder problem instances.

Another family of models with the property of adaptive inference compute budget is early exit networks [Teerapittayanon et al., 2016, Laskaridis et al., 2021]. These networks have multiple additional “exit” prediction heads along their depth. At inference time, the result that satisfies an exit policy is selected as the prediction output. This approach of designing adaptive networks has been adapted both in natural language processing [Schwartz et al., 2020, Soldaini and Moschitti, 2020, Elbayad et al., 2020b, Zhou et al., 2020, Liu et al., 2020] and vision [Li et al., 2017, Wang et al., 2018, Xing et al., 2020, Kouris et al., 2021]. Most of these architectures have complex sub-modules that are trained in multiple stages, and require complex exit policies. In contrast, equilibrium models have a simple architecture, and can use root solvers to efficiently solve for the fixed point at inference.

More complex transformer-based language models like GPT-3 also struggle to generalize well on simple algorithmic tasks like addition [Brown et al., 2020a]. Recent work by Nye et al. [2021] shows that transformers can be trained to perform well on algorithmic tasks and generalize on OOD data by emitting the intermediate steps of an algorithm to a buffer called “scratchpad”. Using a scratchpad enables the model to revisit its errors and correct them.

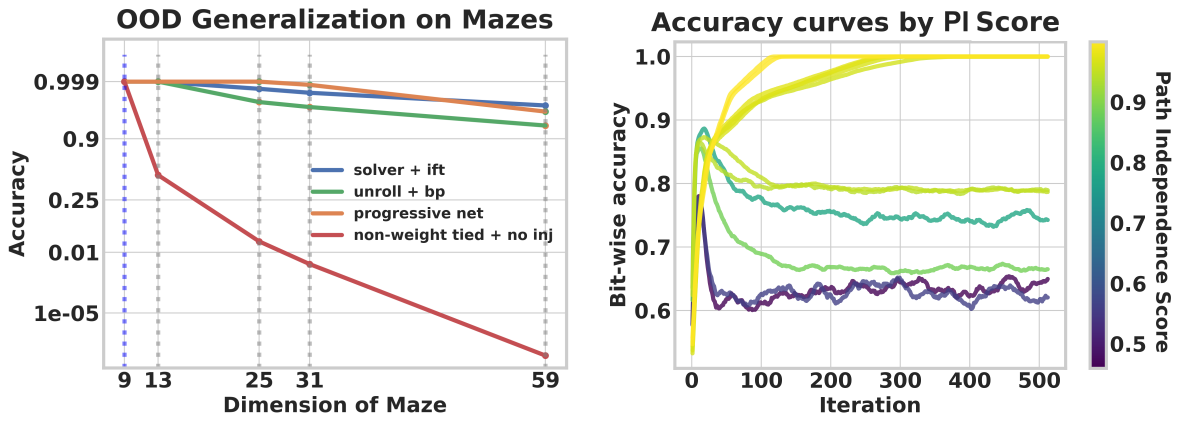
Relation between Path Independence and the Concepts of Global Convergence and Stability Path independence is closely related to the concept of *global stability and global convergence* in control theory and optimization. This concept is somewhat overloaded, as it sometimes requires convergence to a single point [Slotine et al., 1991], and sometimes implies the system is convergent everywhere, even if to different points [Wang et al., 2003, Sriperumbudur and Lanckriet, 2009]. We thus choose the term *path independence* to refer specifically to the fact that the system will converge to the same limiting behavior (whatever that might be) regardless of the initial state of the system.

7.3 Upwards Generalization with Deep Equilibrium Models

In this section, we establish that equilibrium models are capable of strong upwards generalization. To study the effects of test time computation, it is useful to consider tasks with an explicit difficulty parameter, so that the learned models can be tested on more difficult instances which require a large number of iterations to solve correctly. We focus on multiple algorithmic generalization tasks: **prefix sum** and **mazes** by Schwarzschild et al. [2021a,b], **blurry MNIST**, **matrix inversion** and **edge copy** by Du et al. [2022]. Taken together, these tasks cover a wide range of problems from different domains, namely sequence prediction, visual reasoning, image classification, continuous optimization and graph regression. Below we describe individual tasks in detail.

7.3.1 Algorithmic Tasks to Test Upwards Generalization

- **Prefix-sum** is a sequence-to-sequence task whereby the network is given a sequence of 0-1 bits, and is trained to output, for each bit, the parity of all of the bits received since the beginning of the sequence until the current bit. We train on 10,000 unique 32-bit binary strings, and report results on binary strings of other lengths.
- **Mazes** task is an image-to-image task, where the input is a three-channel RGB image. The 'start' and 'finish' positions are marked by a red and a green square respectively; walls are marked in black. The output is the optimal path in the maze that connects these two points without passing through the walls. We train on 50,000 small mazes of size 9×9, and report upward generalization results on larger mazes.
- **Blurry MNIST** [Liang et al., 2021] is a robustness-to-corruption task: one has to learn to do MNIST classification from lightly blurred images and generalize zero-shot to highly blurred ones. The images are blurred with Gaussian blur filters. Training data uses Gaussian blur filters with standard deviation equal to $2, 2.5, 3, 3.5$ and testing data uses Gaussian blur filters with slightly higher standard deviation equal to $4, 4.5, 5, 5.5$.
- In the **matrix inversion task** [Du et al., 2022], the goal is to learn to invert 20×20 matrices in a way that generalizes to matrices that have worse condition number than those observed during training. The well conditioned invertible matrix $\mathbf{M}$ that is used for training is represented as $\mathbf{M} = \mathbf{R} + \mathbf{R}^\top + 0.5\mathbf{I}$, where $\mathbf{R}$ is a random matrix with individual elements sampled between -1 and 1. The more difficult less well-conditioned matrix used for testing is constructed as $\mathbf{M} = \mathbf{R} + \mathbf{R}^\top + 0.1\mathbf{I}$.



(a) (b)

Figure 7.1: (**left**) Strong upward generalization on mazes by PI models. Models were trained on 9×9 sized mazes and tested for upward generalization on larger mazes. y-axis uses probit transformation. (**right**) PI models are better able to make use of additional test-time computation. We trained models with varying number of training-time iterations, learning rate and weight norm application. Bit-wise accuracies are evaluated and averaged over different string-lengths.

- **Edge copy** [Du et al., 2022] is a simple graph regression task that requires learning to output the input edge features, in a way that generalizes to larger graph size.The value of each edge in a fully connected graph is randomly sampled with a uniform value between -1 and 1. We train on graphs with 2-10 nodes and evaluate on graphs with 15 nodes. See [Du et al., 2022] for more details. **[asp: Add dataset sizes for individual tasks]**

Note that the training and test data for the latter two tasks are generated with noise added on-the-fly, as done by Du et al. [2022]. To maintain clarity and focus, we run our detailed analysis on the prefix sum and mazes tasks in the following sections, and provide additional results with the remaining tasks in **[asp: Add section here]**. **[asp: Add example images for each of the individual tasks]**

7.3.2 Empirical Evidence of Strong Upwards Generalization

Figure 7.1a shows that equilibrium models demonstrate very strong upward generalization performance compared to non weight-tied fixed-depth models. Moreover, Figure 7.1b shows that increasing inference depth consistently improves performance—especially on harder problem instances.

7.4 Path Independence

We argue that a key determiner of whether a learned model can exploit additional test-time computation is whether the dynamical system corresponding to the model is *path independent*; that is, whether the learned model's hidden layer activations converge to the same asymptotic behaviour (i.e. fixed point or limit cycle), regardless of the initialization of the system. For example, a simple integrator $x_{t+1} = x_t + 1$ is clearly *not* path independent, as its final state depends on the initial state x_0 and the

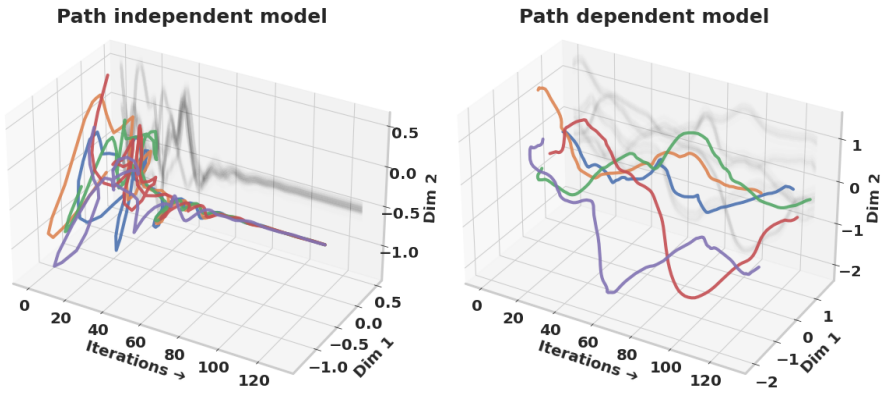


Figure 7.2: Trajectories of path independent models converge to the same hidden state for a given input, regardless of initialization, whereas the trajectories of path dependent models depend on initialization. Here, we display five trajectories with different initializations obtained from a path independent (left) and path dependent model (right) on the prefix-sum task, projected onto two random directions.

number of iterations run; conversely, the system $x_{t+1} = (x_t + 1)/2$ *is* path independent, as it will converge to the solution $x_T = 1$ as $T \to \infty$ regardless of the initial condition of x_0. Path independence is closely related to the concept of *global stability* from control theory (see Section Sec. 7.2 for more).

Intuitively, path independent systems can more easily take advantage of additional test-time iterations than path dependent ones. For instance, gradient descent applied to a convex objective is path independent, and correspondingly when confronted with a more ill-conditioned problem instance, one can compensate by increasing the number of iterations. Conversely, a weather simulation is path dependent, and extending the simulation won't yield more accurate predictions of a given day's weather. Based on this intuition, we hypothesize that path independence of a learned model is a key determiner of whether it can take advantage of an increased test-time iteration budget when generalizing to harder problem instances:

> *Path Independence Hypothesis:* Models which successfully fit the training distribution with a path independent function are better able to exploit more test-time iterations to achieve higher accuracy, compared to those which fit the training distribution with a path-dependent function.

More formally, we say that the computation performed by a recurrent operator computing function f_θ on an input $\boldsymbol{x}$ is path independent if it converges to the same limiting behaviour regardless of the current state $\boldsymbol{z}_t$. As a special case, if the computation is convergent, this property is equivalent to the existence of a unique fixed point $\boldsymbol{z}^*$ such that $f_\theta^\infty(\boldsymbol{x}, \boldsymbol{z}_0) = \boldsymbol{z}^*$ for any $\boldsymbol{z}_0$. However, our definition allows for other behaviors such as limit cycles (see Sec. 7.7).

Some architectures guarantee the path independence property (see Sec. 2.1). However, most common DEQ architectures—and the ones we use throughout this paper—have the expressive power to learn multiple fixed points per input. Since it is unclear whether architectures enforcing the contraction property lose expressiveness [Bai et al., 2021], we focus our investigation on unrestricted architectures.

7.4.1 Architectural Components Necessary for Path Independence

Past work has observed that *weight tying* and *input injection* are both crucial for upwards generalization [Bansal et al., 2022]. We observe that both architectural components are also necessary for a learned model to be PI.[1] Without weight tying, the network is constrained to have a fixed forward depth, so it is meaningless to talk about the limiting behavior in large depth. Input injection ensures that the equilibrium point depends on the input despite having an "infinite depth". Without input injection, a PI network would necessarily forget the input; hence, any model which successfully fits the training distribution must be path dependent.

Interestingly, both architectural motifs are also key components of deep equilibrium models [Bai et al., 2019]; in that work, the motivation was to enable efficient gradient estimation via the implicit function theorem (IFT) — a concept closely related to path independence, since the premise of the IFT gradient estimator is that only the final hidden state matters, not the path taken to get there. It is striking that two separate lines of work would converge on the same architectural motifs, one motivated by generalization and the other by a variant of path independence.

Reproducing the results of Bansal et al. [2022], in Figure 7.1a we show upward generalization performance using both equilibrium models and progressive nets [Bansal et al., 2022] – and the lack thereof using non-input-injected networks. For the remainder of this paper, we focus on architectures with both input injection and weight tying.

7.4.2 Quantifying Path Independence: Asymptotic Alignment Score (AA Score)

We propose a simple metric to quantify path independence based on the directional alignment of the fixed points computed with the same input, but different initializations. We name this metric the *Asymptotic Alignment (AA) score*. Pseudocode to compute the metric is given in Algorithm 8. The AA score is the average cosine similarity between the fixed points obtained with the training time initialization (often simply the zero vector) and the fixed points obtained when one initializes the solver *using the fixed points computed on different inputs*. Higher AA scores (with 1 being the highest value) imply higher degrees of path independence. In Sec. 7.5, we show a strong correlation between path independence and accuracy using the AA score.

The AA score is cheap to compute, is a reliable indicator of path independence (see below), and is unitless, meaning that networks obtained from different training runs can be compared on equal footing. See Sec. 7.9 for other metrics we have considered for quantifying path independence and why we found AA score to be preferable.

7.4.3 Stress-testing the AA score

To stress-test the extent to which the AA score really measures path independence, we search for *adversarial initializations* that are optimized to result in distinct fixed points, hence low AA values. (Unlike adversarial examples, this attack is not constrained to an ε-ball.) We use the L-BFGS [Liu and

[1] Our definition also admits non-input-injected models to be path independent if they're representing constant functions (i.e. input independent). We don't consider such cases in our analyses.

Algorithm 8 Asymptotic Alignment Score

Input: A batched input $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, an operator f_w

Initialize: $\begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \mathbf{0}$

Define: $h(y_1, y_2) = \frac{y_1}{\|y_1\|_2} \cdot \frac{y_2}{\|y_2\|_2}$

Compute $\begin{bmatrix} z_1' \\ z_2' \end{bmatrix} = \text{FIX}_{f_w}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}\right)$

Interchange and reinitialize iterates

Compute $\begin{bmatrix} z_1'' \\ z_2'' \end{bmatrix} = \text{FIX}_{f_w}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \begin{bmatrix} z_2' \\ z_1' \end{bmatrix}\right)$

return average$(h(z_1'', z_1'), h(z_2'', z_2'))$

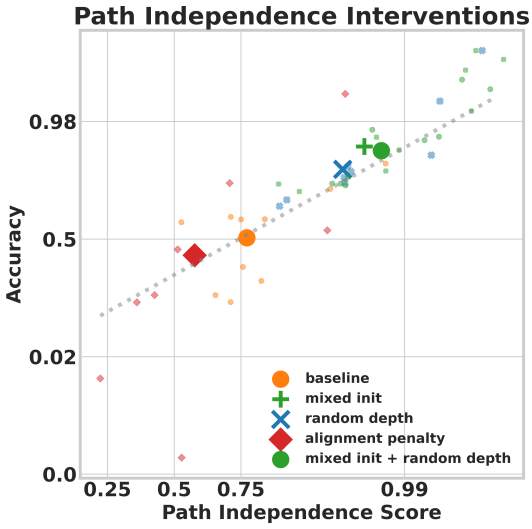


Figure 7.3: **(left) AA Score Algorithm:** We provide the algorithm for a simple illustrative case of two inputs. In practice, we consider larger batches. **(right) Promoting path independence improves generalization in the prefix sum task:** Interventions that are designed to promote path independence (initializing fixed points with random noise or running the fixed point solver with stochastic budget) improves generalization. Conversely, those that hurt path independence (penalty term that directly penalizes fixed point alignment) leads to poorer generalization.

Model	Task	AA ↑	Accuracy (%)	Attacked AA ↑	Attacked Acc. (%)
Non-PI network	Maze	0.32	87.12	0.09	0
PI network	Maze	1.00	100	1.00	100
Non-PI network	Prefix sum	0.62	66.66	0.18	0
PI network	Prefix sum	0.99	100	0.99	100

Table 7.1: **Stress-testing the AA Scores:** AA scores for PI vs non-PI networks computed on 13×13 mazes and 64 bit prefix sum. Attacked AA refers to the cosine similarity between the fixed point from zero initialization and an adversarial initialization. Non-PI networks can be easily steered away from the initial fixed point estimate through adversarial initializations but it is difficult to do so for PI networks with high AA scores.

Nocedal, 1989] optimizer, and repeat the search multiple times starting from different fixed point initializations. We include pseudocode in the supplementary material.

Results of the adversarial stress test can be seen in Table 7.1. The results corroborate that the AA score is indeed a reliable measure of path independence; while it isn't possible to find adversarial initializations for high AA score networks (indicating high path independence), low AA score networks can easily be adversarially initialized to be steered away from the original fixed point estimate.

Experimental details The non-path independent network used in this table is a 8-layer weight-tied input-injected network trained with backpropagation gradients. We set the maximum number of iterations for fixed point computation to 12 as we found that this choice gives the highest test accuracy on 13×13 mazes. While selecting the optimal value for maximum iterations, we increased the value up to 60 iterations. The path independent network is a 32-layer weight-tied input injected network trained with backpropagation gradients. The maximum iterations for this network are set to 500. We used a batch size of 1 and set the maximum number of L-BFGS [Liu and Nocedal, 1989] updates per batch to 50 for both the networks. We implemented this code in PyTorch and use these values of hyperparameters: `lr=1`, `tolerance_grad=1e-7`, `tolerance_change=1e-9`, `line_search_fn="strong_wolfe"`. We report AA scores and accuracy computed on 500 examples. We provide pseudocode in Algorithm Box 9

Algorithm 9 Stress testing AA Scores: An algorithm to find adversarial initializations

Input: A trained network $f_{\boldsymbol{w}}$, an input $\boldsymbol{x}$, N max number of LBFGS updates

Define: $h(\boldsymbol{y}_1, \boldsymbol{y}_2) := \frac{\boldsymbol{y}_1}{\|\boldsymbol{y}_1\|_2} \cdot \frac{\boldsymbol{y}_2}{\|\boldsymbol{y}_2\|_2}$

random init$(\cdot)$: A method that returns a random vector initialized from $\mathcal{N}(\mathbf{0}, \mathbf{I})$

Initialize: $\boldsymbol{z} = \boldsymbol{x}$ or random init$(\boldsymbol{x})$

▷ Disable gradients

$\boldsymbol{z}_1 = \text{FIX}_{f_{\boldsymbol{w}}}(\boldsymbol{z}, \boldsymbol{x})$

▷ Clone $\boldsymbol{z}_1$ and Enable gradients on $\boldsymbol{z}_1$

for trials from 1 to N **do**

  $\boldsymbol{z}_2 = \text{FIX}_{f_{\boldsymbol{w}}}(\boldsymbol{z}_1, \boldsymbol{x})$

  Perform L-BFGS update on $\boldsymbol{z}_1$ to minimize $h(\boldsymbol{z}_1, \boldsymbol{z}_2)$

end for

Output: $\boldsymbol{z}_1$

7.5 Path Independence Correlates with Upward Generalization

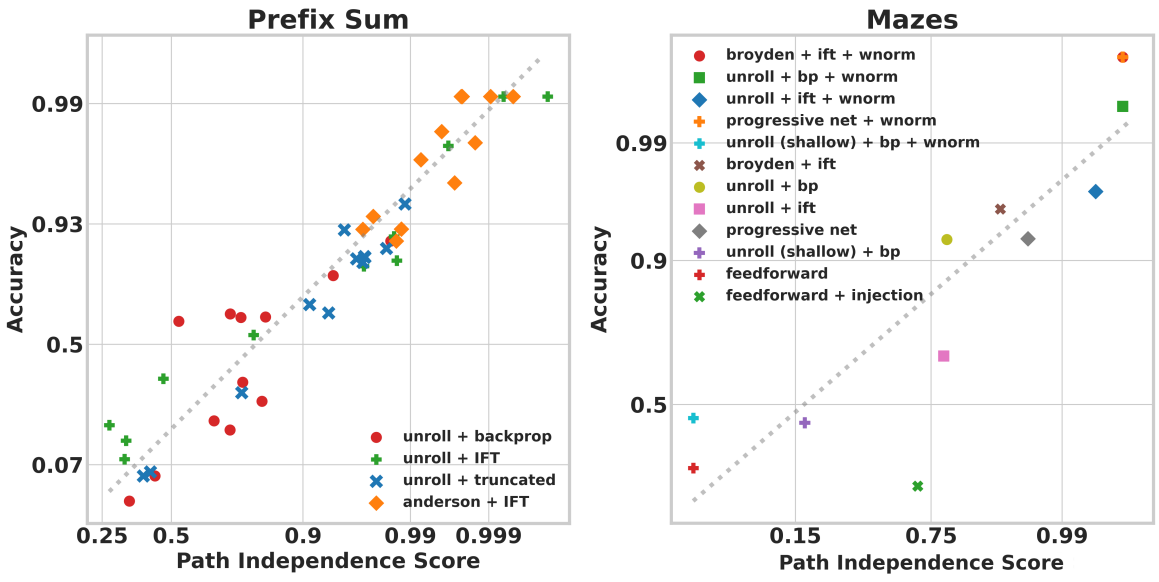
Is path independence (as measured by the AA score) a strong predictor of upwards generalization? We took the trained networks from Sec. 7.3, computed their average AA scores on in- and out-of-distribution splits and inspected whether the AA scores are correlated with upward generalization.

On prefix sum experiments, we varied 1) network depth, 2) whether or not weight norm (wnorm) [Salimans and Kingma, 2016] was used or not,[2] 3) learning rate (one of [0.01, 0.001, 0.0001]), 4) forward solver (fixed point iterations or Anderson acceleration [Anderson, 1965], and 5) the gradient estimator (backprop or implicit gradients).[3] On the maze experiments, we varied 1) network depth, 2) use of weight norm, 3) forward solver (fixed point iterations or Broyden solver [Broyden, 1965]), and 5) the gradient estimator (backprop or implicit gradients).

Figure 7.4 displays our findings. We evaluated performance on a mixture of in- and OOD validation data; results on individual data splits can be found in the supplementary material. The results show a strong correlation between AA score and accuracy when the inference depth is large enough. This shows that PI networks allow for scaling test-time compute to improve test-time accuracy (see also

[2] Bai et al. [2019] report that weight norm helps stabilize the training of DEQ models.

[3] Note that the deep equilibrium model (DEQ) setup [Bai et al., 2019] correspond to using a root solver (such as Anderson) for the forward pass and implicit gradients for the backward pass.



(a) Prefix Sum

(b) Mazes

Figure 7.4: High AA scores correlate with good upward generalization. For a given choice of an architecture and a task, the reported numbers are averaged over problem instances of different dimensions. We apply the probit transformation along both axes, following Miller et al. [2021]. Accuracies and AA scores are capped at 0.999 for compatibility with the probit transform.

Figure 7.1b). The in-distribution validation performance of non-PI networks degrades with deeper inference depths. Unsurprisingly, these networks generalize poorly on harder problem instances that require deeper inference depths (i.e. problem instances that provably require at least a given number of layers to handle). Further results on the BlurryMNIST, matrix inversion and edge copy tasks can be found in 7.14.1, 7.14.2 and 7.14.3.

7.6 Experimental Manipulations of Path Independence

The previous section demonstrates a strong correlation between path independence and the ability to exploit additional test-time iterations. Unfortunately, we can't make a causal claim based on these studies: the observed effect could have been due to an unobserved confounder. In this section, we intervene directly on path independence by imposing regularizers which directly encourage or penalize path independence. We find that interventions designed to *promote* path independence also improve generalization, while interventions designed to *reduce* path independence also hurt generalization.

7.6.1 Promoting Path Independence via Randomized Forward Passes

A straightforward way to encourage path independence is simply to initialize the hidden states with random noise during training. To this end, we experimented with initializing the hidden states with zeros on half of the examples in the batch, and with standard Gaussian noise on the rest of the examples. The reason to include the zero-initializations at training time is that we initialize from zeros at test time

- not including this initialization during training time causes a distribution shift.

Another way to promote path independence is simply running the forward solver with randomized compute budgets/depths during training time. While a path independent solution can be expected to be robust against this intervention, a path dependent one will fail.

We took the training configurations of the 12 prefix-sum networks described in Sec. 7.5 that use fixed point iterations in their forward pass, and backpropagation gradient in their backward pass, and retrained them separately with the aforementioned mixed initialization and randomized depth strategies without modifying any other experimental conditions. As can be seen in Figure 7.3, the interventions lead to strong test-time path independent neural networks, while also reliably improving in- and out-of-distribution validation accuracy. We especially emphasize that shallow networks trained with mixed initialization actually remain far from having high AA scores using the training-time forward pass conditions due to lack of convergence. However, since the mixed initialization strategy results in path independent networks, scaling up test-time compute budget leads to high AA scores, and therefore high upwards generalization.

7.6.2 Penalizing Path Independence via the Fixed Point Alignment Penalty

Does an intervention that results in less path independence also result in poorer upwards generalization? Like in the mixed initialization experiment, we retrained the 12 unroll + backpropagation networks with an additional auxiliary loss term that penalizes the dot product between the fixed points computed from the same input, but different initializations sampled from standard Gaussian noise. Figure 7.3 shows that this intervention succeeded in pushing the AA scores down, while also keeping the accuracy on the same trend line.

7.7 Disambiguating Convergence and Path Independence

Is convergence necessary for path independence? We answer this statement in the negative, and show that neither training-time convergence nor test-time convergence is required for path independence. Instead convergence to the same limiting behavior regardless of initialization is important.

Training Time Convergence We consider two implicitly trained equilibrium models trained on the mazes task—one trained with implicit gradients computed via implicit function theorem (IFT), and the other trained with an approximation of the (inverse) Jacobian, called phantom gradients [Geng et al., 2021b]. We report the values of residuals (*i.e.*, $\|f_\theta(\boldsymbol{x}, \boldsymbol{z}) - \boldsymbol{z}\|_2$), AA scores and accuracies observed for in- and out-of-distribution data for mazes in Table 7.2. We observe that DEQs trained with phantom gradients have higher values of in-distribution residuals but are path independent, as indicated by their high AA scores, and show strong upward generalization as indicated by their good accuracy.

The mixed-initialization intervention described in Sec. 7.6.1 also leads to a separation between training-time convergence and path independence. We found that it is possible to train very shallow (*i.e.*, 5 layer) unrolled networks that, while being very far from converging during training and attaining poor in-distribution generalization, are able to converge and achieve perfect performance when run

Model	Residual ↓		AA score ↑		Accuracy (%) ↑	
	In-dist	OOD	In-dist	OOD	In-dist	OOD
DEQ (phantom grad.)	11.83	0.016	0.96	0.99	99.96	99.88
DEQ (IFT)	1.4	0.011	0.99	0.99	99.99	100

Table 7.2: Training-time convergence is not needed for path independence: models might show poor training-time convergence (as shown by high values of residuals) but still be path independent. Residual, AA score, and Accuracy for DEQ trained with IFT vs phantom gradients. In-distribution (In-dist) results were computed on 9×9 mazes, and OOD results were computed on 25×25 mazes.

for many more iterations during test time. Details are provided in the supplementary material.

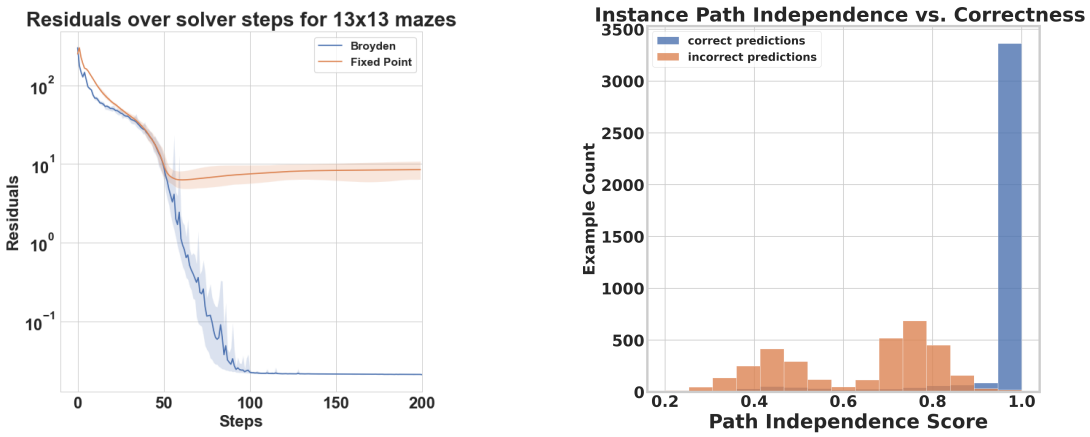
Test Time Convergence From Table 7.2, one might conclude that test time convergence is important for path independence. However, we show that this connection is not necessary, and convergence to the same fixed point is not a required condition for path independence. We study test time convergence properties of an unrolled weight-tied input-injected network trained with backpropagation under different solvers. This network is highly path independent using either the Broyden solver or fixed point iterations, as indicated by its high AA scores (0.99) on both in- and out-of-distribution data. We visualize the values of test-time residuals with fixed point iterations and Broyden's method in Figure 7.5a. Both these solvers converge to different limit cycles but still show good upward generalization.

7.8 Validity of Path Independence on a Per-Example Level

The connection between path independence and prediction correctness also largely holds on a per-instance basis. Using the prefix-sum networks trained with the mixed-initialization strategy (the most performant group of networks in our intervention experiments), we plotted the distribution of per-instance fixed point alignment scores, colored by whether the prediction on that instance was correct or not in Figure 7.5b. This suggests that path independence can be used as a valuable sanity-check to determine whether a prediction is correct or not without the need for any label data, both in- and out-of-distribution. We provide a more in-depth per-instance analysis in the supplementary material.

7.9 Alternative Approaches for Quantifying Path Independence

We review alternative methods for quantifying how path-independent a given equilibrium model is, and why the Asymptotic Alignment score (AA Score) is the most suitable one amongst them for our analyses. We want a path independence metric to satisfy three criteria: 1) **Dimensionless:** Metrics computed from different training runs should be directly comparable with each other. To ensure this, one has to make sure that one uses dimensionless metrics (i.e. doesn't have units). 2) **local and global path independence:** The metric should test for path independence not just in a local neighborhood of



(a) (b)

Figure 7.5: (**Left**) Different solvers display differing asymptotic behaviours but still achieve good upwards generalization. Here, the network has an adversarial AA score of 0.99, and achieves accuracy of 99.98% (fixed point iterations) and 99.97% (Broyden solver) respectively on the mazes task; (**Right**) Per-instance path independence is highly correlated with correctness of predictions for prefix sum task.

fixed points, but over a sufficiently large portion of the initialization domain. 3) **efficiency:** It should be computationally tractable.

With these in mind, we review three alternative ways of quantifying path independence. The summary of the analysis is described in Table :

- **Jacobian Norm:** The Frobenius norm of the Jacobian of the output of an equilibrium model with respect to its fixed point initialization (i.e. $||\frac{\partial \text{FIX}(\boldsymbol{x}, \boldsymbol{z}_0)}{\partial \boldsymbol{z}_0}||_2$) gives a robust measure of how locally sensitive the network is to initializations. If this quantity is very small (and approaching 0 with more root finding steps), one can conclude that the given equilibrium model is path-independent on the given input. The main issues with this approach are: 1) It's extremely computationally intensive to compute (since both the input and the output of FIX are high-dimensional, neither forward nor backward differentiation can estimate this Jacobian efficiently) 2) It has units, meaning it isn't comparable across different training runs, 3) It only measures local path independence.

- **Agreement with Adversarial Initializations:** This method quantifies to what extent it is possible to directly optimize initializations such that they result in different fixed points. If the network is path independent, there shouldn't exist an adversary that can find adversarial initializations. The exact procedure for how adversarial fixed points can be found in Sec. 7.4.3. The downsides with this approach is that 1) it's expensive - one has to solve an optimization problem for every problem instance, potentially using different optimizers and initializations, and 2) it only checks for local path independence.

- **Agreement with Swapped Initializations (Asymptotic Alignment Score):** The main idea behind this approach is to initialize the forward pass of an equilibrium model with fixed points obtained from other problem instances, and checking if the network still displays the same

asymptotic behaviour (i.e. finds the same fixed point). If one uses cosine similarity to measure consistency, then one recovers Fixed Point Alignment score proposed in Section 7.4.2. Another option is to use the average Euclidean distance to check for consistency. This is identical to computing the trace of the covariance matrix of the distance between the computed fixed points. This approach – especially the one that utilizes cosine similarity to quantify consistency – is appealing, as 1) because it's unitless, it can be compared between training runs on the same task, 2) it checks for non-local convergence by sampling from a distribution of relevant fixed points 3) it's very efficient, requiring only two forward passes. The version that utilizes Euclidean distance does have units, hence cannot be used for cross-model comparisons [4]) To further make sure that this metric is correct, and does subsume with local path independence, we ran stress tests described in Section 7.4.2 to make sure this metric produces results consistent with the *agreement with adversarial initializations* method described above.

Alternatives to Cosine Similarity in Measuring Directional Alignment While cosine similarity is a conceptually clean way of quantifying directional alignment, one can consider alternative kernels as well. Ideally, our main results should not depend on the specific implementation details of path independence metrics (as long as they satisfy the criteria we set out above).

To check how sensitive our results are on the precise functional form of the AA score, we replaced it with three other kernels and re-assessed whether the results pointed to the same high-level takeaways. Concretely, we tried the Gaussian kernel $\phi_g(r) = \exp(-(\epsilon r)^2)$, Laplacian kernel $\phi_l(r) = \exp(-|\epsilon r|)$ and inverse multiquadratic kernel $\phi_i(r) = \frac{1}{\sqrt{1+(\epsilon r)^2}}$ where the input to the kernel function is the Euclidean distance of the normalized fixed points $r = ||\frac{z_1}{||z_1||} - \frac{z_2}{||z_2||}||$. We used $\epsilon = 5000$, which gave a reasonable dynamic range in the resulting values. The results can be seen in Figure 7.6. The takeaways from the plots remain identical: *path independence is strongly correlated with generalization, regardless of the specific details of how path independence is quantified* (as long as it satisfies the criteria we set out above).

Importance of using dimensionless metrics: Being unitless is necessary for a metric to be comparable across training runs, though obviously not sufficient. Consider two equilibrium models M1 and M2, where the fixed points computed by M2 have the same direction as those computed by M1, but have twice the Euclidean norm. The behaviour of this M2 is qualitatively the same as that of M1, but any metric that depends on the Euclidean metric (or Jacobian L2 norm, or any non-unitless metric) would report this network to be less path independent. Note that this is not a purely theoretical consideration: simply adding L2 regularization, or penalizing the magnitude of fixed points to encourage convergence will directly impact the scale of the fixed points (things that practitioners often use), thereby rendering non-unitless metrics unreliable.

[4] If one uses LayerNorm [Ba et al., 2016b] or similar normalization layers, we could expect the Euclidean distance based metric to behave similar to the cosine similarity based one.

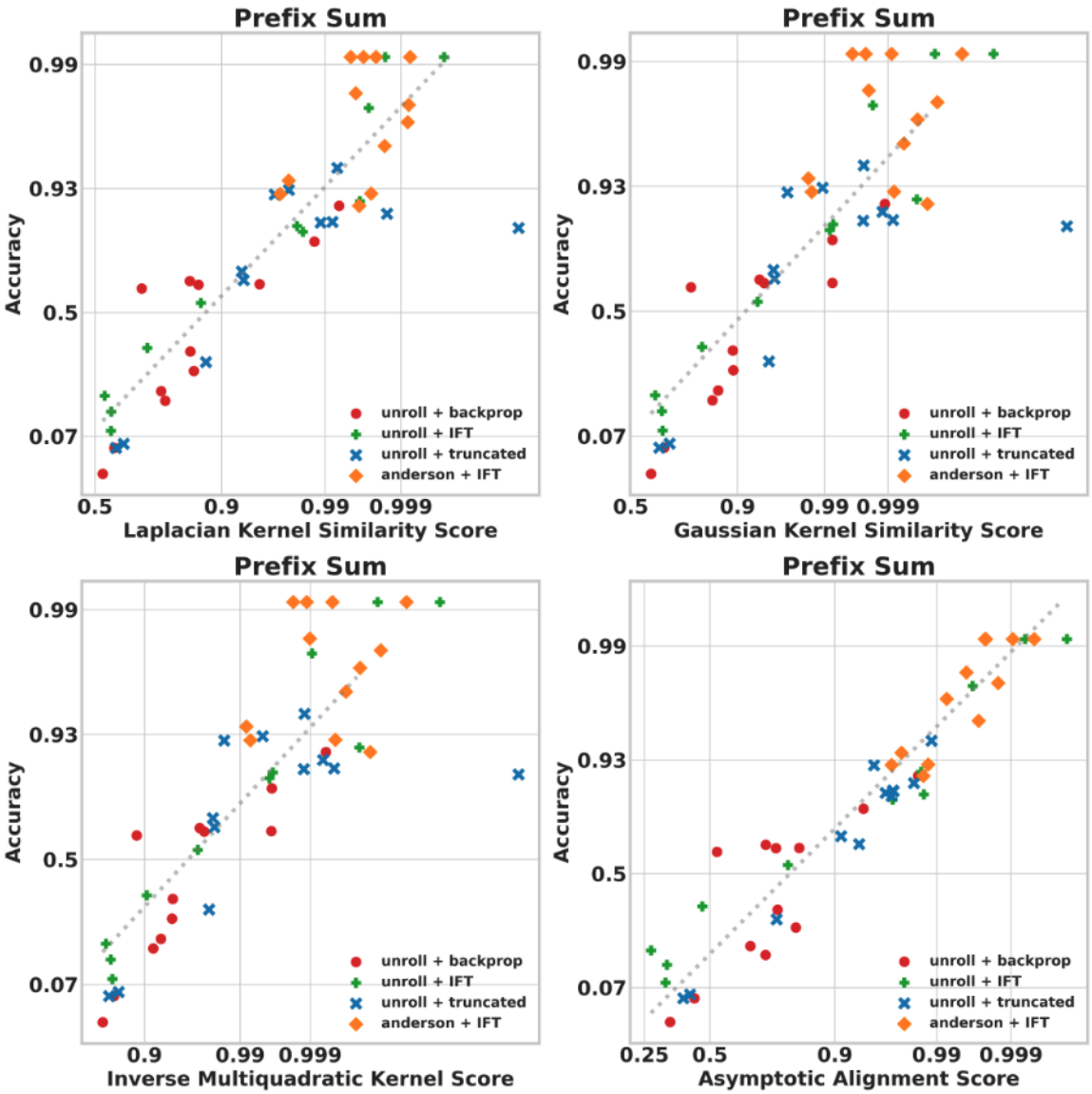


Figure 7.6: **Alternatives to Cosine Similarity in AA Score:** Replacing cosine similarity (right bottom) with alternative kernels like Gaussian (left top), Laplacian (right top) and inverse multiquadratic kernel yields qualitatively similar results. This demonstrates that the takeaways from our experiments don't depend on the particular implementation details of the ways path-independence is quantified.

Path Independence Metrics	Dimensionless	Local & Global Coverage	Efficient
IO Jacobian Norm	✗	✗	✗
Using Adversarial Initializations	✓✓	✓	✗
Asymptotic Alignment (AA) Score	✓✓	✓	✓✓

Table 7.3: **Comparing different ways of quantifying path independence:** We compare three different methods of quantifying how path independent a network is in terms of their correctness (whether they actually measure path independence), dimensionlessness (whether the quantity is unitless, allowing comparisons between between different networks meaningful), local and global coverage (whether the metric checks for path independence locally or globally) and efficiency (whether it's computationally cheap to compute the metric). Among the methods we've considered (See Section 7.9), the Asymptotic Alignment (AA) Score is the most suited one for our purposes. Note that no metric considered here has perfect global coverage (i.e. can verify that a given network is globally path independent or not).

7.10 Path Independence vs. Accuracy Plots for Different Difficulty Levels

The path-independence vs. accuracy plots in Figure 7.4b uses averaged values over different problem difficulties. In Figure 7.7, we show how these correlations look like at different lengths on the prefix sum task. The same figure also shows how the accuracy differs if one does inference using the training time forward pass budget. The main takeaways from this plot are:

- Out-of-distribution problem lengths (right top of the Figure) do indeed require larger inference depths - all models perform very poorly when inference is run with training-time forward pass budget.
- The correlation between path independence and accuracy - as the definition implies - is more apparent in the very large forward pass limit. In this regime, the correlation persists for all lengths.
- While networks with low PI values using the training-time budget occasionally perform worse when the test-time compute is increased, that doesn't happen with path-independent networks.

7.11 Intervention Results on Different Difficulty Levels

Mirroring Section 7.10, we provide how the accuracy correlates with path independence on 1) different problem difficulties (32 is in-distribution) and different forward pass budgets in Figure 7.8.

We especially emphasize the *mixed initialization* results (mixed initialization is one of the interventions that promotes path independence). Note that some of these networks (half of which are as shallow as 6 layers) do poorly in and out-of-distribution when the forward pass uses the training forward pass budget. Unsurprisingly, the AA scores for these networks are low in this condition. However, when additional test-time compute is provided, these networks reach perfect in-distribution accuracy,

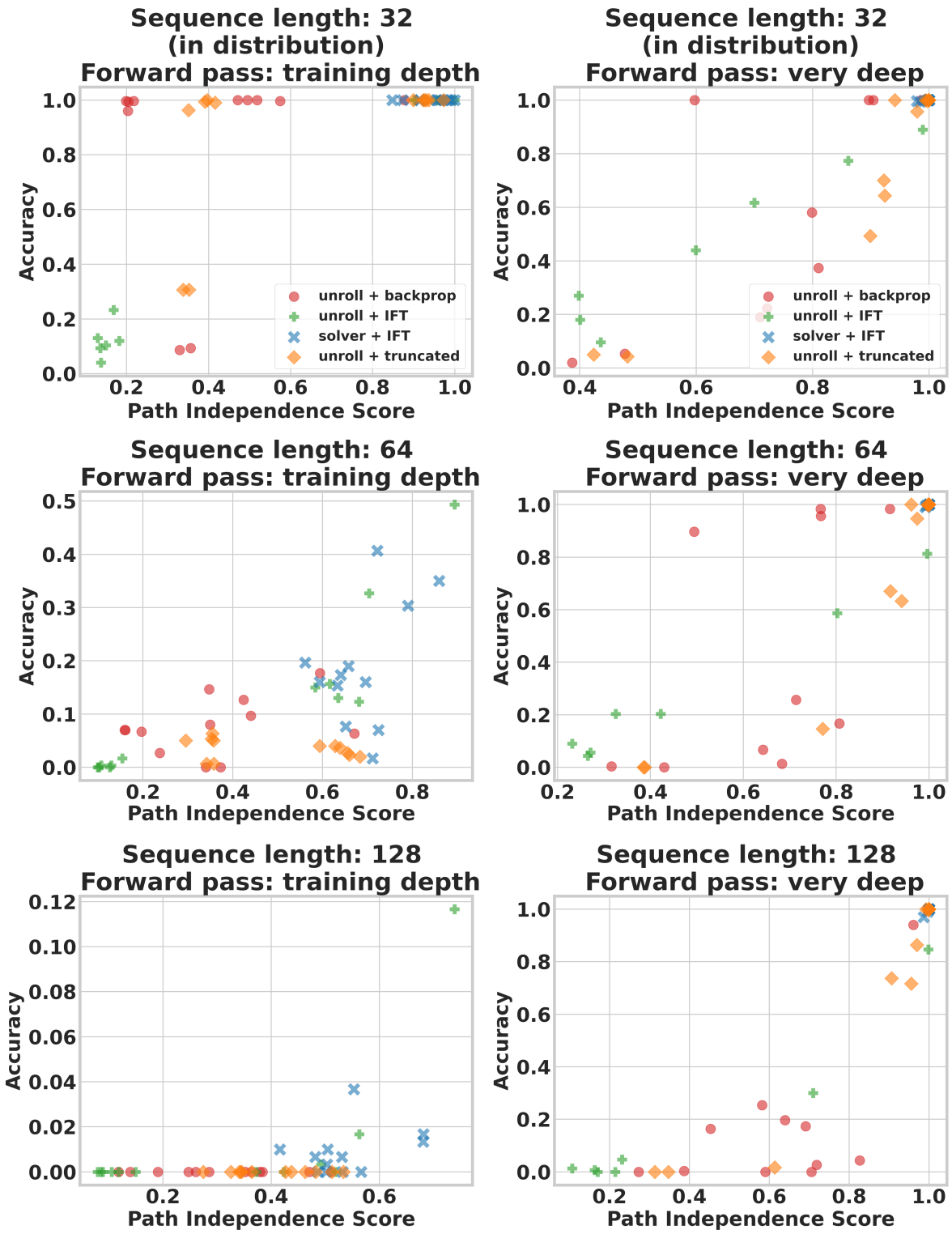


Figure 7.7: **Accuracy vs. Path Independence Correlation on Different Problem Difficulties and Inference Depths on Prefix Sum:** The data on the plots in the left column is obtained when the forward pass is run with the training forward pass budged. Likewise, the data on the right column is obtained in the large inference-time budget limit. Each row corresponds to a different problem difficulty, with the first row (length = 32) corresponding to in-distribution data.

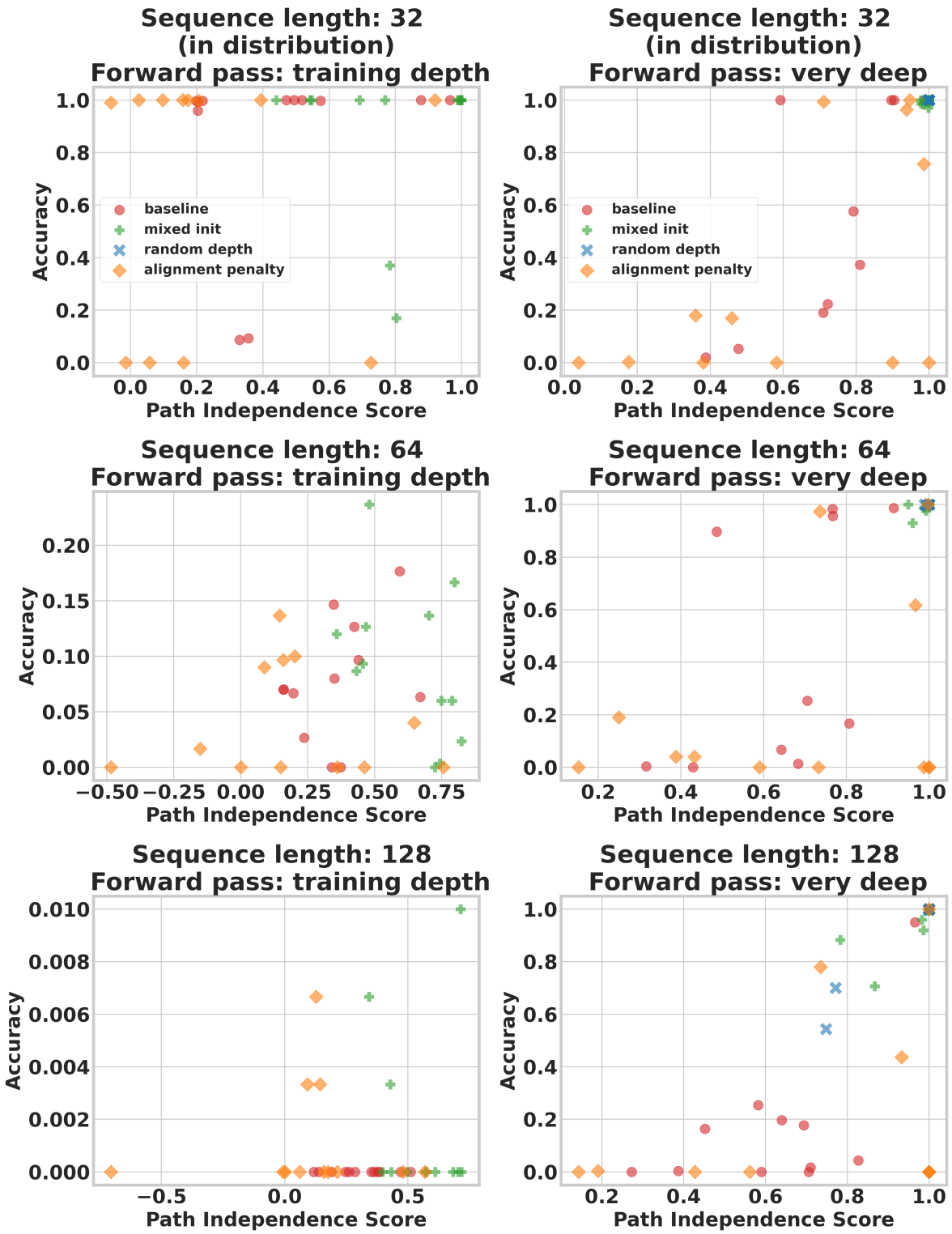


Figure 7.8: **Accuracy vs. Path Independence Correlation on Different Problem Difficulties and Inference Depths on Prefix Sum:** The data on the plots in the left column is obtained when the forward pass is run with the training forward pass budged. Likewise, the data on the right column is obtained in the large inference-time budget limit. Each row corresponds to a different problem difficulty, with the first row (length = 32) corresponding to in-distribution data.

and near-perfect OOD accuracy (especially on length = 64 split). This demonstrates that training time convergence is not required for path independence. We find it surprising that with the help of a path-independence-promoting regularization procedure, it's possible to train very shallow (i.e. less than 6 layers) networks such that they can exploit additional test-time compute to achieve very high accuracy.

7.12 Per-Instance Path Independence Analyses - Convergence vs. Path Independence

In addition to the analysis in Section 7.8, we also analyzed how test-time convergence correlates with path independence on a per-instance basis. The per-instance AA score vs. convergence (measured in terms of the L2 distance between the last two root solver iterates) can be seen in Figure 7.9. We used the 12 networks trained using the mixed initialization strategy (since these networks yielded the best in and OOD performance), and used 300 samples from the length splits 16, 32, 64, 128 and 256.

This analysis reaffirms our past observation that test-time convergence is not needed for path independence. The plot contains data from three different regimes:

- **Full convergence to a fixed point:** These datapoints, observed on the right bottom part of Figure 7.9, corresponds to samples where the solver nearly converges to a fixed point. This is almost always associated with high FPA scores, and good per-instance accuracy.
- **Limit cycles:** These datapoints, corresponding to the right-top part of Figure 7.9, correspond to cases where the solver doesn't converge to a fixed points, but enters an orbit around it. AA scores, as well as the per-instance accuracy values remain high in this regime as well, indicating that test-time convergence (to a fixed point) is not necessary for path independence, and good accuracy. Note that the boundary between the full-convergence regime and the limit cycle regime is gradual.
- **Divergence:** On a number of the samples, the solver diverges. These are associated with high residuals, low AA scores and low per-instance accuracies. This shows that the main source of per-instance lack of path independence on networks that are otherwise overwhelmingly path independent (on other samples) is almost always solver divergence.

7.13 Test Time Convergence and Path independence

We provide a per-instance analysis of test-time convergence in Figure 7.10. We display plots for per-instance residuals i.e. $\|f(x,z) - z\|_2$ for Broyden's method and naive fixed point iterations over solver steps. In addition, we also plot L2 norm between the fixed points of these solvers at every step. We provide plots for both in-distribution data i.e. 9×9 mazes and on more difficult problem instances i.e. 13×13 and 25×25 mazes. Across all the mazes, we observe that there are problem instances where both the solvers converge to the same limiting behavior (as indicated by low values of residuals and L2 norms). However, there are a considerable number of points where naive fixed

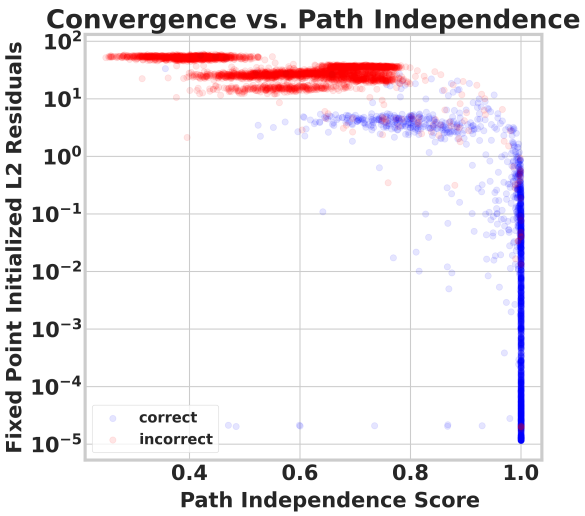


Figure 7.9: **Per-Instance Test-time Convergence vs. Path Independence**: The per-instance AA score vs. convergence (measured in terms of the L2 distance between the last two root solver iterates). Each example is labelled based on whether the network's prediction on it was correct (blue) or not (red). We used the 12 networks trained using the mixed initialization strategy (since these networks yielded the best in and OOD performance), and used 300 samples from the length splits 16, 32, 64, 128 and 256. While good convergence is almost always associated with high AA score values, the converse is not necessarily true: there are many samples on which convergence is poor, yet the AA score is high.

point iterations converge to a limit cycle but still output correct predictions. This behavior can be seen on both in-distribution data and on harder problem instances. We find that the absolute residuals between the points in the limit cycles are a small percentage of the Euclidean norms of the points: 0.9% for 9x9 mazes, 0.49% for 13x13 mazes and 1.5% for 25x25 mazes which indicates that these limit cycles are localized. These examples of problem instances where both the solvers converge to different limiting behavior reaffirm our observation that convergence to the same fixed point at test-time is not a necessary condition for path independence.

7.14 Additional Experimental Results

7.14.1 Results on the Blurry MNIST Task

We tested our path-independence hypothesis on task we call BlurryMNIST [Liang et al., 2021]. This task involves training an image classifier on MNIST digits corrupted with small degrees of Gaussian blur, and testing the performance on significantly more corrupted ones. The blur corruption is implemented by convolving each image with Gaussian filters of differing standard deviations. We used standard deviations from 2 to 5.5 (in increments of 0.5) to generate each split. This dataset allows for testing in and out-of-distribution performance by way of training on a subset of the splits (i.e. the four lowest blur splits) and testing performance on all splits.

We trained a number of fully connected equilibrium models on the BlurryMNIST task and inspected

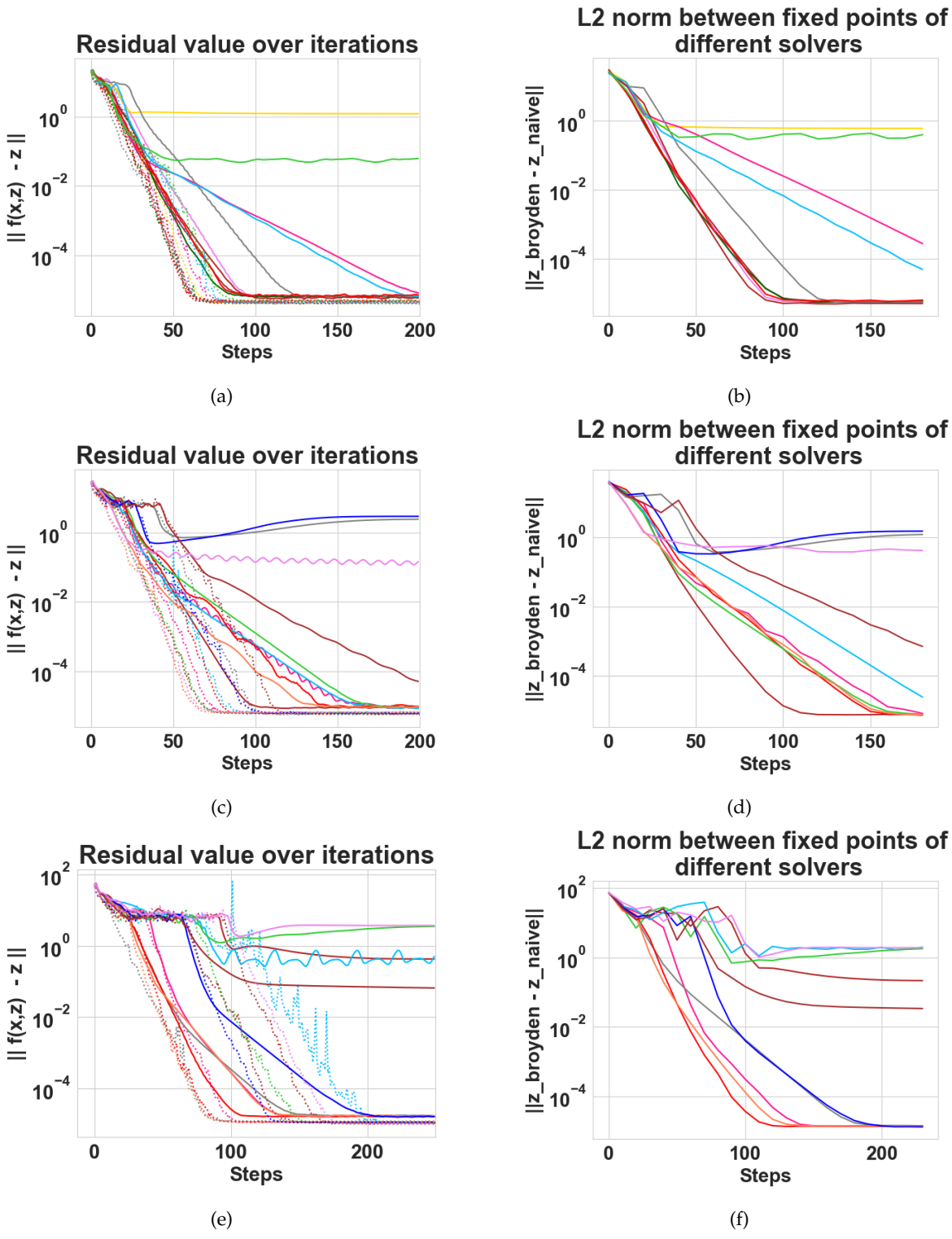


Figure 7.10: Different solvers display differing asymptotic behaviour but still achieve good upwards generalization (**Left column**) We display plots for the values of $||f(x,z) - z||_2$ over multiple solver steps for Broyden's method (dotted lines) and naive fixed point iterations (solid lines). (**Right column**)We also plot L2 norm between the fixed points obtained through fixed point solver and Broyden's method. Each line indicates one problem instance and for a given row, lines with same color are the same problem instance. The network was trained on 9×9 mazes, has an adversarial FPA score of 0.99, and achieves accuracy of 100% with both the solvers on all the displayed problem instances of mazes; **(first row)** 9×9 mazes i.e. in-distribution, **(second row)** 13×13 mazes. **(third row)** 25×25 mazes

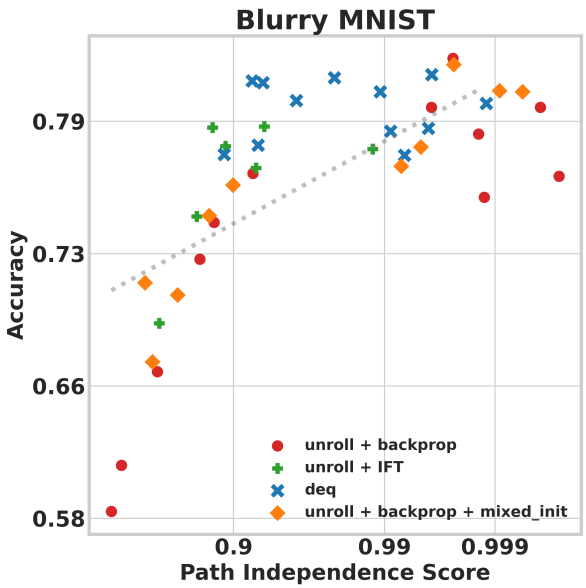


Figure 7.11: **BlurryMNIST results:** Path independence and generalization correlate on the BlurryMNIST dataset. This task involves training an image classifier on MNIST digits corrupted with small degrees of Gaussian blur, and testing the performance on significantly more corrupted ones.

whether path independence still correlates with accuracy. The results can be found in Figure 7.11. The correlation we reported in the main body of the paper also holds in the BlurryMNIST dataset. Note that the in-distribution error rates of the trained models vary between 1 and 5 percent.

7.14.2 Results on Matrix Inversion Task

We also tested the connection between path independence and generalization on the Matrix Inversion task proposed by Du et al. [2022]. This task is concerned with learning to invert 20x20 matrices. Success is defined by how well the trained model works on matrices with worse condition numbers than those observed during training. Note that this task is qualitatively very different from all the others we considered before. We have summarized the results in Figure 7.12.

Using a fully-connected ResNet block as the equilibrium model cell of width 512, we trained a number of equilibrium models where we varied 1) the forward pass (fixed point iterations vs. solver) 2) the backwards pass (backprop gradients or implicit gradients) 3) learning rate (0.001, 0.0001 and 0.00001), 4) learning rate schedule (step decays of magnitude 0.5 at different points during training) and 5) whether layer normalization [Ba et al., 2016b] was used or not. The results can be seen in Figure 7.12. We see a similar pattern that we saw on earlier tasks: lack of path independence correlates with poor generalization. Note that our best model matches the performance of the energy based model approach proposed by Du et al. [2022] in the matrix inversion task and significantly beats their baselines.

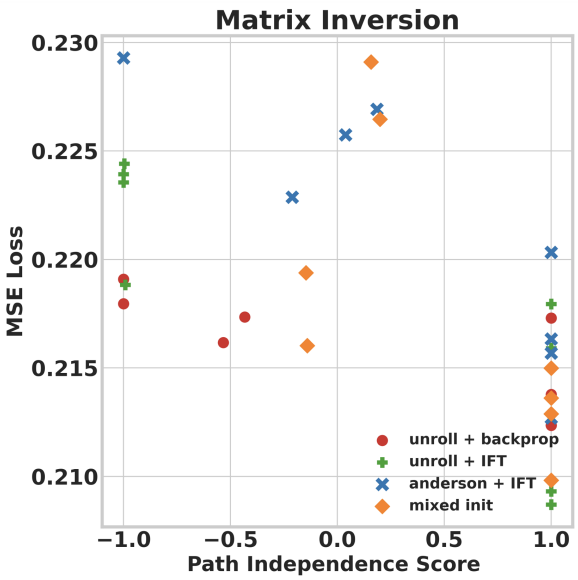


Figure 7.12: **Matrix inversion results:** We also tested the connection between path independence and generalization on the Matrix Inversion task proposed by Du et al. [2022]. Lack of path independence correlates with poor generalization in this task as well (note that **lower is better** in this task).

7.14.3 Results on the Edge Copy Task

We also test our path independence hypothesis on tasks that take in a graph as an input. We consider the following the edge copy task proposed by Du et al. [2022], where the goal is to learn to simply output the input edge features, in a way that generalizes to larger graph sizes.

We used an equilibrium model cell that's compatible with graph tasks in the Edge Copy experiments, whose structure is shown in Figure 7.14. This cell is especially suited for edge regression tasks, since each application of the cell refines the edge features.

We trained a number of equilibrium models where we varied 1) the forward pass (fixed point iterations vs. solver) 2) the backwards pass (backprop gradients or implicit gradients) 3) learning rate (0.0001, , 0.000333 and 0.0001) and 4) whether layer normalization [Ba et al., 2016b] was used or not. Each hyperparameter configuration was run twice with different seeds. The results can be seen in Figure 7.13. We see a similar pattern that we saw on earlier tasks: lack of path independence correlates with poor generalization. Note that our best equilibrium model outperforms the the energy based model approach proposed by Du et al. [2022] in this task.

7.15 Discussion

Being able to attain better levels of performance using a larger inference-time compute budget is a feat that eludes most standard deep learning architectures. This is especially relevant for tasks that require *upwards generalization*, *i.e.,* the ability to generalize from easy problem instances to hard ones. We show that equilibrium models are capable of displaying upwards generalization by exploiting

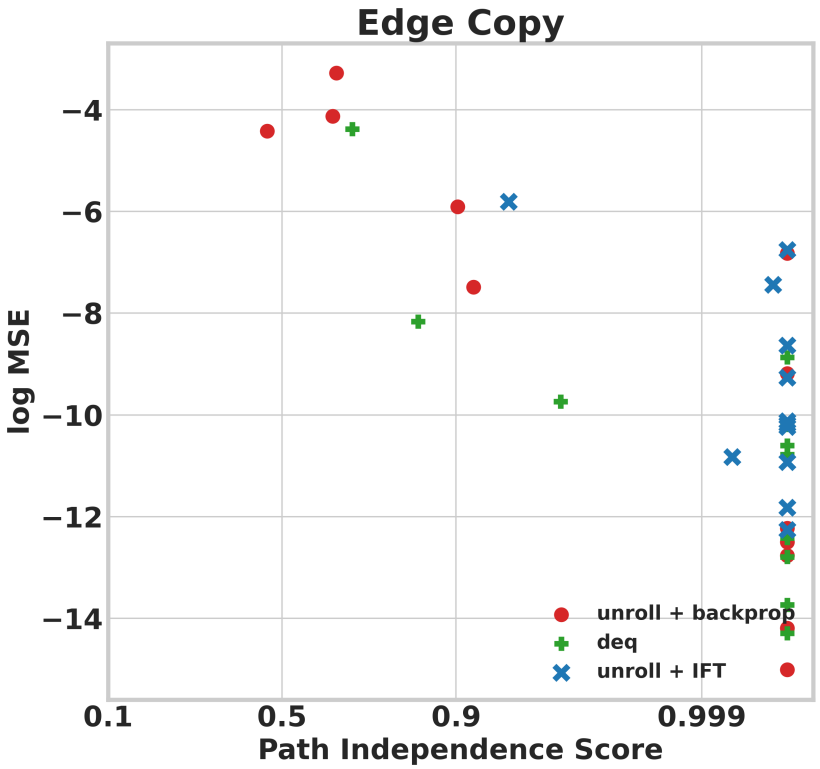


Figure 7.13: **Edge copy task:** The edge copy task requires learning to simply output the input edge features, in a way that generalizes to larger graph sizes. Lack of path independence correlates with poor generalization in the edge copy task as well. Note that **lower in better** in this task.

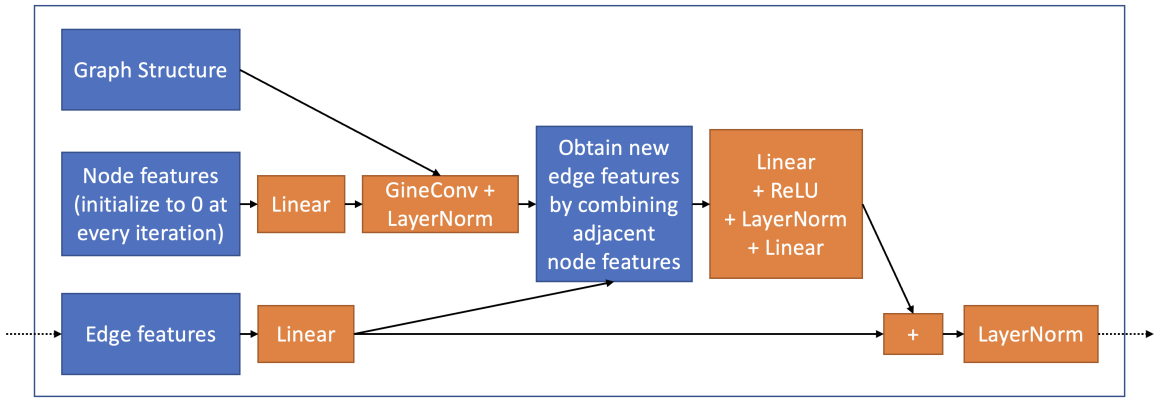


Figure 7.14: **A graph-processing equilibrium model cell:** In the edge-copy experiments, we used the equilibrium model cell illustrated above. The workhorse of this cell is the GINEConv operation [Hu et al., 2019], which fuses node and edge features to produce updated node features. This cell is especially suited for edge regression tasks, since each application of the cell refines the edge features.

scalable test-time compute. We link this to a phenomenon we call *path independence*: the tendency of an equilibrium network to converge to the same limiting behavior given an input, regardless of the initial conditions. We investigate this phenomenon through careful experiments and verify that path independent networks indeed generalize well on harder problem instances by exploiting more test time compute. Moreover, interventions on training conditions that promote path independence also improve upwards generalization, while those that penalize it hurt this capability. Our findings suggest that path independent equilibrium models are a promising direction towards building general purpose learning systems whose test-time performance improves with more compute.

Bibliography

R. Aggarwal, V. Sounderajah, G. Martin, D. S. Ting, A. Karthikesalingam, D. King, H. Ashrafian, and A. Darzi. Diagnostic accuracy of deep learning in medical imaging: a systematic review and meta-analysis. *NPJ digital medicine*, 4(1):65, 2021. 1

S. Agrawal, H. Kim, D. Sanz-Alonso, and A. Strang. A variational inference approach to inverse problems with gamma hyperpriors. *SIAM/ASA Journal on Uncertainty Quantification*, 10(4):1533–1559, 2022. 70

M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022. 1

M. S. Albergo, N. M. Boffi, and E. Vanden-Eijnden. Stochastic interpolants: A unifying framework for flows and diffusions. *arXiv preprint arXiv:2303.08797*, 2023. 50, 51, 53

B. D. Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12 (3):313–326, 1982. 7

D. G. Anderson. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 1965. 6, 17, 18, 19, 22, 103, 104, 113, 130

C. Anil, A. Pokle, K. Liang, J. Treutlein, Y. Wu, S. Bai, J. Z. Kolter, and R. B. Grosse. Path independent equilibrium models can better exploit test-time computation. *Advances in Neural Information Processing Systems*, 35:7796–7809, 2022. 104

Anthropic. Claude sonnet 3.5. *https://www.anthropic.com/news/claude-3-5-sonnet*, 2024. 1

U. M. Ascher and L. R. Petzold. *Computer methods for ordinary differential equations and differential-algebraic equations*, volume 61. Siam, 1998. 7, 41

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016a. 40, 41

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016b. 135, 143, 144

S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. *Neural Information Processing Systems (NeurIPS)*, 2019. 2, 5, 6, 15, 21, 23, 31, 128, 130

S. Bai, V. Koltun, and J. Z. Kolter. Stabilizing equilibrium models by jacobian regularization. *arXiv preprint arXiv:2106.14342*, 2021. 6, 127

S. Bai, Z. Geng, Y. Savani, and J. Z. Kolter. Deep equilibrium optical flow estimation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 620–630, 2022. 6, 21

J. Baldridge, J. Bauer, M. Bhutani, N. Brichtova, A. Bunner, K. Chan, Y. Chen, S. Dieleman, Y. Du, Z. Eaton-Rosen, et al. Imagen 3. *arXiv preprint arXiv:2408.07009*, 2024. 1

A. Banino, J. Balaguer, and C. Blundell. Pondernet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021. 124

A. Bansal, A. Schwarzschild, E. Borgnia, Z. Emam, F. Huang, M. Goldblum, and T. Goldstein. End-to-end algorithm synthesis with recurrent networks: Logical extrapolation without overthinking. *arXiv preprint arXiv:2202.05826*, 2022. 123, 124, 128

Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019. 100

C. K. Batchelor and G. K. Batchelor. *An introduction to fluid dynamics*. Cambridge university press, 1967. 106

M. Benning and M. Burger. Modern regularization methods for inverse problems. *Acta numerica*, 27: 1–111, 2018. 70

C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019. 1

M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester. Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 417–424, 2000. 70

D. Berthelot, A. Autef, J. Lin, D. A. Yap, S. Zhai, S. Hu, D. Zheng, W. Talbott, and E. Gu. Tract: Denoising diffusion models with transitive closure time-distillation, 2023. 38, 45, 48

K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart. Model reduction and neural networks for parametric PDEs. *The SMAI journal of computational mathematics*, 7:121–157, 2021. 100, 101

A. Bora, A. Jalal, E. Price, and A. G. Dimakis. Compressed sensing using generative models. In *International conference on machine learning*, pages 537–546. PMLR, 2017. 71

J. P. Boyd. *Chebyshev and Fourier spectral methods*. Courier Corporation, 2001. 11

J. Brandstetter, D. Worrall, and M. Welling. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022. 100

A. Brock, J. Donahue, and K. Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018. 47

T. Brooks, B. Peebles, C. Homes, W. DePue, Y. Guo, L. Jing, D. Schnurr, J. Taylor, T. Luhman, E. Luhman, C. W. Y. Ng, R. Wang, and A. Ramesh. Video generation models as world simulators. 2024. URL `https://openai.com/research/video-generation-models-as-world-simulators`. 1

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020a. 124

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020b. 1, 44

C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of computation*, 1965. 6, 31, 103, 104, 113, 130

S. H. Chan, X. Wang, and O. A. Elgendy. Plug-and-play admm for image restoration: Fixed-point convergence and applications. *IEEE Transactions on Computational Imaging*, 3(1):84–98, 2016. 70

C.-H. Chao, W.-F. Sun, B.-W. Cheng, Y.-C. Lo, C.-C. Chang, Y.-L. Liu, Y.-L. Chang, C.-P. Chen, and C.-Y. Lee. Denoising likelihood score matching for conditional score-based data generation. *arXiv preprint arXiv:2203.14206*, 2022. 47

R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018. 3, 9, 51, 52

R. T. Q. Chen. torchdiffeq, 2018. URL `https://github.com/rtqichen/torchdiffeq`. 59

T. Chen and H. Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995. 101

T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016. 42

Z. Chen, J. Lu, and Y. Lu. On the representation of solutions to elliptic pdes in barron spaces. *Advances in neural information processing systems*, 34:6454–6465, 2021. 101, 102

J. Choi, S. Kim, Y. Jeong, Y. Gwon, and S. Yoon. Ilvr: Conditioning method for denoising diffusion probabilistic models. *arXiv preprint arXiv:2108.02938*, 2021. 70

Y. Choi, Y. Uh, J. Yoo, and J.-W. Ha. Stargan v2: Diverse image synthesis for multiple domains. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020. 59

K. Choudhary, B. DeCost, C. Chen, A. Jain, F. Tavazza, R. Cohn, C. W. Park, A. Choudhary, A. Agrawal, S. J. Billinge, et al. Recent advances and applications of deep learning methods in materials science. *npj Computational Materials*, 8(1):59, 2022. 1

H. Chung, J. Kim, M. T. Mccann, M. L. Klasky, and J. C. Ye. Diffusion posterior sampling for general noisy inverse problems. *arXiv preprint arXiv:2209.14687*, 2022a. 50, 54, 59, 70, 71

H. Chung, B. Sim, D. Ryu, and J. C. Ye. Improving diffusion models for inverse problems using manifold constraints. *arXiv preprint arXiv:2206.00941*, 2022b. 70, 71

H. Chung, B. Sim, and J. C. Ye. Come-closer-diffuse-faster: Accelerating conditional diffusion models for inverse problems through stochastic contraction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12413–12422, 2022c. 56

H. Chung, J. Kim, S. Kim, and J. C. Ye. Parallel diffusion models of operator and image for blind inverse problems. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6059–6069, 2023. 71

R. Cohen, M. Elad, and P. Milanfar. Regularization by denoising via fixed-point projection (red-pro). *SIAM Journal on Imaging Sciences*, 14(3):1374–1406, 2021. 70

G. Daras, J. Dean, A. Jalal, and A. G. Dimakis. Intermediate layer optimization for inverse problems using deep generative models. *arXiv preprint arXiv:2102.07364*, 2021. 71

M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and L. Kaiser. Universal transformers. 2019. URL https://openreview.net/pdf?id=HyzdRiR9Y7. 124

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 57

N. S. Detlefsen, J. Borovec, J. Schock, A. H. Jha, T. Koker, L. Di Liello, D. Stancl, C. Quan, M. Grechkin, and W. Falcon. Torchmetrics-measuring reproducibility in pytorch. *Journal of Open Source Software*, 7 (70):4101, 2022. 59

P. Dhariwal and A. Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34, 2021. 1, 16

M. Ding, C. Deng, J. Choo, Z. Wu, A. Agrawal, A. Schwarzschild, T. Zhou, T. Goldstein, J. Langford, A. Anandkumar, et al. Easy2hard-bench: Standardized difficulty labels for profiling llm performance and generalization. *arXiv preprint arXiv:2409.18433*, 2024. 2

J. Dong, R. Yin, X. Sun, Q. Li, Y. Yang, and X. Qin. Inpainting of remote sensing sst images with deep convolutional generative adversarial network. *IEEE geoscience and remote sensing letters*, 16(2): 173–177, 2018. 70

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021. 3, 38, 40, 49

A. Doury, S. Somot, S. Gadat, A. Ribes, and L. Corre. Regional climate model emulator based on deep learning: Concept and first evaluation of a novel hybrid downscaling approach. *Climate Dynamics*, 60(5):1751–1779, 2023. 1

G. Dresdner, D. Kochkov, P. Norgaard, L. Zepeda-Núñez, J. A. Smith, M. P. Brenner, and S. Hoyer. Learning to correct spectral methods for simulating turbulent flows. 2022. doi: 10.48550/ARXIV. 2207.00556. URL https://arxiv.org/abs/2207.00556. 106

Y. Du, S. Li, J. Tenenbaum, and I. Mordatch. Learning iterative reasoning through energy minimization. In *International Conference on Machine Learning*, pages 5570–5582. PMLR, 2022. 125, 126, 143, 144

P. Dubois, T. Gomez, L. Planckaert, and L. Perret. Machine learning for fluid flow reconstruction from limited measurements. *Journal of Computational Physics*, 448:110733, 2022. 2

M. Elad, B. Kawar, and G. Vaksman. Image denoising: The deep learning revolution and beyond—a survey paper. *SIAM Journal on Imaging Sciences*, 16(3):1594–1654, 2023. 70

M. Elbayad, J. Gu, E. Grave, and M. Auli. Depth-adaptive transformer. In *International Conference on Learning Representations*, 2020a. URL https://openreview.net/forum?id=SJg7KhVKPH. 124

M. Elbayad, J. Gu, E. Grave, and M. Auli. Depth-adaptive transformer. *ArXiv*, abs/1910.10073, 2020b. 124

C. Eyzaguirre and A. Soto. Differentiable adaptive computation time for visual reasoning. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, pages 12817–12825, 2020. 124

T. Falk, D. Mai, R. Bensch, Ö. Çiçek, A. Abdulkadir, Y. Marrakchi, A. Böhm, J. Deubner, Z. Jäckel, K. Seiwald, et al. U-net: deep learning for cell counting, detection, and morphometry. *Nature methods*, 16(1):67–70, 2019. 22

I. Faragó and J. Karátson. *Numerical solution of nonlinear elliptic problems via preconditioning operators: Theory and applications*, volume 11. Nova Publishers, 2002. 113, 116

M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov. Spatially adaptive computation time for residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1039–1048, 2017. 124

S. W. Fung, H. Heaton, Q. Li, D. McKenzie, S. Osher, and W. Yin. Fixed point networks: Implicit depth models with jacobian-free backprop. *arXiv e-prints*, pages arXiv–2103, 2021. 21

S. W. Fung, H. Heaton, Q. Li, D. McKenzie, S. Osher, and W. Yin. Jfb: Jacobian-free backpropagation for implicit networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 6648–6656, 2022. 6

Y. Gal and Z. Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. *Advances in neural information processing systems*, 29, 2016. 6

W. Gan, Y. Hu, J. Liu, H. An, U. Kamilov, et al. Block coordinate plug-and-play methods for blind inverse problems. *Advances in Neural Information Processing Systems*, 36, 2024. 71

C. Gao, X. Lan, N. Li, Y. Yuan, J. Ding, Z. Zhou, F. Xu, and Y. Li. Large language models empowered agent-based modeling and simulation: A survey and perspectives. *Humanities and Social Sciences Communications*, 11(1):1–24, 2024. 1

Z. Geng, M.-H. Guo, H. Chen, X. Li, K. Wei, and Z. Lin. Is attention better than matrix decomposition? In *International Conference on Learning Representations (ICLR)*, 2021a. 21

Z. Geng, X.-Y. Zhang, S. Bai, Y. Wang, and Z. Lin. On training implicit models. *Advances in Neural Information Processing Systems*, 34, 2021b. 6, 21, 104, 132

Z. Geng, A. Pokle, and J. Z. Kolter. One-step diffusion distillation via deep equilibrium models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. 37

D. Gilton, G. Ongie, and R. Willett. Neumann networks for linear inverse problems in imaging. *IEEE Transactions on Computational Imaging*, 6:328–343, 2019. 70

X. Gong, S. Chang, Y. Jiang, and Z. Wang. Autogan: Neural architecture search for generative adversarial networks. In *IEEE International Conference on Computer Vision (ICCV)*, 2019. 44

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 1, 15, 16

A. Graikos, N. Malkin, N. Jojic, and D. Samaras. Diffusion models as plug-and-play priors. *Advances in Neural Information Processing Systems*, 35:14715–14728, 2022. 71

S. R. Granter, A. H. Beck, and D. J. Papke Jr. Alphago, deep learning, and the future of the human microscopist. *Archives of pathology & laboratory medicine*, 141(5):619–621, 2017. 1

W. Grathwohl, R. T. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018. 9

A. Graves. Adaptive computation time for recurrent neural networks. *CoRR*, abs/1603.08983, 2016a. URL http://arxiv.org/abs/1603.08983. 124

A. Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016b. 124

J. Gu, S. Zhai, Y. Zhang, L. Liu, and J. M. Susskind. Boot: Data-free distillation of denoising diffusion models with bootstrapping. In *ICML 2023 Workshop on Structured Probabilistic Inference {\&} Generative Modeling*, 2023. 38

D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024. 1

G. Gupta, X. Xiao, and P. Bogdan. Multiwavelet-based operator learning for differential equations. *Advances in neural information processing systems*, 34:24048–24062, 2021a. 12

R. Gupta, D. Srivastava, M. Sahu, S. Tiwari, R. K. Ambasta, and P. Kumar. Artificial intelligence to deep learning: machine intelligence approach for drug discovery. *Molecular diversity*, 25:1315–1360, 2021b. 1

Z. Hao, Z. Wang, H. Su, C. Ying, Y. Dong, S. Liu, Z. Cheng, J. Song, and J. Zhu. Gnot: A general neural operator transformer for operator learning. In *International Conference on Machine Learning*, pages 12556–12569. PMLR, 2023. 101

D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016. 103

M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 23, 25, 42, 59

G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 37

J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. 1, 2, 8, 16, 22, 26, 45, 50, 52

J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022. 1, 44, 45

S. Hong, I. Park, and S. Y. Chun. On the robustness of normalizing flows for inverse problems in imaging. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10745–10755, 2023. 71

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. 100

J.-T. Hsieh, S. Zhao, S. Eismann, L. Mirabella, and S. Ermon. Learning neural pde solvers with convergence guarantees. *arXiv preprint arXiv:1906.01200*, 2019. 100

W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019. 145

L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 2024. 2

B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024. 1

A. Hyvärinen and P. Dayan. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(4), 2005. 9

A. Jalal, M. Arvinte, G. Daras, E. Price, A. G. Dimakis, and J. Tamir. Robust compressed sensing mri with deep generative priors. *Advances in Neural Information Processing Systems*, 34:14938–14954, 2021. 70

K. H. Jin, M. T. McCann, E. Froustey, and M. Unser. Deep convolutional neural network for inverse problems in imaging. *IEEE transactions on image processing*, 26(9):4509–4522, 2017. 70

M. Jin, W. Luo, S. Cheng, X. Wang, W. Hua, R. Tang, W. Y. Wang, and Y. Zhang. Disentangling memory and reasoning ability in large language models. *arXiv preprint arXiv:2411.13504*, 2024. 4

J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021. 1

Z. Kadkhodaie and E. P. Simoncelli. Solving linear inverse problems using the prior implicit in a denoiser. *arXiv preprint arXiv:2007.13640*, 2020. 70

U. S. Kamilov, H. Mansour, and B. Wohlberg. A plug-and-play priors approach for solving nonlinear imaging inverse problems. *IEEE Signal Processing Letters*, 24(12):1872–1876, 2017. 70

T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, and T. Aila. Training generative adversarial networks with limited data. *Advances in neural information processing systems*, 33:12104–12114, 2020a. 45, 47

T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8110–8119, 2020b. 45

T. Karras, M. Aittala, T. Aila, and S. Laine. Elucidating the design space of diffusion-based generative models. In *Proc. NeurIPS*, 2022. 1, 3, 7, 9, 41, 42, 45, 47, 49, 51, 53, 56

T. Karras, M. Aittala, J. Lehtinen, J. Hellsten, T. Aila, and S. Laine. Analyzing and improving the training dynamics of diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24174–24184, 2024. 1

B. Kawar, G. Vaksman, and M. Elad. Snips: Solving noisy inverse problems stochastically. *Advances in Neural Information Processing Systems*, 34:21757–21769, 2021. 70

B. Kawar, M. Elad, S. Ermon, and J. Song. Denoising diffusion restoration models. *Advances in Neural Information Processing Systems*, 35:23593–23606, 2022. 59, 70, 71

D. Kim, Y. Kim, W. Kang, and I.-C. Moon. Refining generative process with discriminator guidance in score-based diffusion models. *arXiv preprint arXiv:2211.17091*, 2022. 47

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 22

D. P. Kingma and R. Gao. Vdm++: Variational diffusion models for high-quality synthesis, 2023. 52

D. P. Kingma, T. Salimans, B. Poole, and J. Ho. Variational diffusion models. *arXiv preprint arXiv:2107.00630*, 2021. 52

C.-Y. Ko, S. Dai, P. Das, G. Kollias, S. Chaudhury, and A. Lozano. Memreasoner: A memory-augmented llm architecture for multi-hop reasoning. In *The First Workshop on System-2 Reasoning at Scale, NeurIPS'24*, 2024. 4

D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021. 1, 100, 106

P. W. Koh, S. Sagawa, H. Marklund, S. M. Xie, M. Zhang, A. Balsubramani, W. Hu, M. Yasunaga, R. L. Phillips, I. Gao, et al. Wilds: A benchmark of in-the-wild distribution shifts. In *International Conference on Machine Learning*, pages 5637–5664. PMLR, 2021. 123

J. Kohler, A. Pumarola, E. Schönfeld, A. Sanakoyeu, R. Sumbaly, P. Vajda, and A. Thabet. Imagine flash: Accelerating emu diffusion models with backward distillation. *arXiv preprint arXiv:2405.05224*, 2024. 2, 38

Z. Kong and W. Ping. On fast sampling of diffusion probabilistic models. *arXiv preprint arXiv:2106.00132*, 2021. 45, 49

D. A. Kopriva. *Implementing spectral methods for partial differential equations: Algorithms for scientists and engineers*. Springer Science & Business Media, 2009. 11

A. Kouris, S. I. Venieris, S. Laskaridis, and N. D. Lane. Multi-exit semantic segmentation networks. *arXiv preprint arXiv:2106.03527*, 2021. 124

N. Kovachki, S. Lanthaler, and S. Mishra. On universal approximation and error bounds for Fourier neural operators. *The Journal of Machine Learning Research*, 22(1):13237–13312, 2021a. 101, 112, 113, 115

N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021b. 101

N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Learning maps between function spaces with applications to PDEs. *Journal of Machine Learning Research*, 24(89):1–97, 2023. 12, 100

V. M. Krasnopolsky. Neural network applications to solve forward and inverse problems in atmospheric and oceanic satellite remote sensing. In *Artificial Intelligence Methods in the Environmental Sciences*, pages 191–205. Springer, 2009. 70

V. M. Krasnopolsky and H. Schiller. Some neural network applications in environmental sciences. part i: forward and inverse problems in geophysical remote measurements. *Neural Networks*, 16(3-4): 321–334, 2003. 70

D. Krishnan and R. Fergus. Fast image deconvolution using hyper-laplacian priors. *Advances in neural information processing systems*, 22, 2009. 70

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 21, 41, 43

S. Lanthaler, S. Mishra, and G. E. Karniadakis. Error estimates for deeponets: A deep learning framework in infinite dimensions. *Transactions of Mathematics and Its Applications*, 6(1):tnac001, 2022. 101

C. Laroche, A. Almansa, and E. Coupete. Fast diffusion em: a diffusion model for blind inverse problems with application to deconvolution. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 5271–5281, 2024. 71

S. Laskaridis, A. Kouris, and N. D. Lane. Adaptive inference through early-exit networks: Design, challenges and directions. In *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, pages 1–6, 2021. 124

H. W. Leung and J. Bovy. Towards an astronomical foundation model for stars with a transformer-based model. *Monthly Notices of the Royal Astronomical Society*, 527(1):1494–1520, 2024. 1

R. J. LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007. 11

J. Li, S. Consul, E. Zhou, J. Wong, N. Farooqui, Y. Ye, N. Manohar, Z. Wei, T. Wu, B. Echols, et al. Banishing llm hallucinations requires rethinking generalization. *arXiv preprint arXiv:2406.17642*, 2024. 2

X. Li and X. Qiu. Mot: Memory-of-thought enables chatgpt to self-improve. *arXiv preprint arXiv:2305.05181*, 2023. 4

X. Li, Z. Liu, P. Luo, C. Change Loy, and X. Tang. Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3193–3202, 2017. 124

Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020a. 12, 13, 100, 101, 103, 104, 105

Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020b. 12, 105

Z. Li, D. Z. Huang, B. Liu, and A. Anandkumar. Fourier neural operator with learned deformations for pdes on general geometries. *arXiv preprint arXiv:2207.05209*, 2022a. 121

Z. Li, K. Meidani, and A. B. Farimani. Transformer for partial differential equations' operator learning. *arXiv preprint arXiv:2205.13671*, 2022b. 101

D. Liang, J. Cheng, Z. Ke, and L. Ying. Deep magnetic resonance image reconstruction: Inverse problems meet neural networks. *IEEE Signal Processing Magazine*, 37(1):141–151, 2020. 70

K. Liang, C. Anil, Y. Wu, and R. Grosse. Out-of-distribution generalization with deep equilibrium models. *Uncertainty and Robustness in Deep Learning, ICML 2021*, 2021. 125, 141

R. Liao, Y. Xiong, E. Fetaya, L. Zhang, K. Yoon, X. Pitkow, R. Urtasun, and R. Zemel. Reviving and improving recurrent back-propagation. In *International Conference on Machine Learning*, pages 3082–3091. PMLR, 2018. 5

S. Lin, B. Liu, J. Li, and X. Yang. Common diffusion noise schedules and sample steps are flawed. *arXiv preprint arXiv:2305.08891*, 2023. 53

S. Lin, A. Wang, and X. Yang. Sdxl-lightning: Progressive adversarial diffusion distillation. *arXiv preprint arXiv:2402.13929*, 2024. 38

Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022. 3, 10, 50, 51, 52, 53, 57, 59

D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1):503–528, 1989. 128, 130

G. Liu, H. Sun, J. Li, F. Yin, and Y. Yang. Accelerating diffusion models for inverse problems through shortcut sampling. *arXiv preprint arXiv:2305.16965*, 2023a. 71

H. Liu, C. Sferrazza, and P. Abbeel. Chain of hindsight aligns language models with feedback. *arXiv preprint arXiv:2302.02676*, 2023b. 1

L. Liu, Y. Ren, Z. Lin, and Z. Zhao. Pseudo numerical methods for diffusion models on manifolds. *arXiv preprint arXiv:2202.09778*, 2022a. 49

W. Liu, P. Zhou, Z. Zhao, Z. Wang, H. Deng, and Q. Ju. Fastbert: a self-distilling bert with adaptive inference time. *arXiv preprint arXiv:2004.02178*, 2020. 124

X. Liu, C. Gong, and Q. Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022b. 3, 50, 51, 53

X. Liu, B. Xu, and L. Zhang. Mitigating spectral bias for the multiscale operator learning with hierarchical attention. Oct. 2022c. URL `http://arxiv.org/abs/2210.10890`. 101

X. Liu, Y. Xie, S. Diao, S. Tan, and X. Liang. A diffusion probabilistic prior for low-dose ct image denoising. *arXiv preprint arXiv:2305.15887*, 2023c. 71

Y. Liu, F. Meng, J. Zhou, Y. Chen, and J. Xu. Faster depth-adaptive transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13424–13432, 2021. 124

Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015. 21

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 42

C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *arXiv preprint arXiv:2206.00927*, 2022a. 45

C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *arXiv preprint arXiv:2206.00927*, 2022b. 49

C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *arXiv preprint arXiv:2211.01095*, 2022c. 49

L. Lu, P. Jin, and G. E. Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019. 100, 101

E. Luhman and T. Luhman. Knowledge distillation in iterative generative models for improved sampling speed. *ArXiv*, abs/2101.02388, 2021. 1, 3, 37, 38, 45, 48

W. Luo, T. Hu, S. Zhang, J. Sun, Z. Li, and Z. Zhang. Diff-instruct: A universal approach for transferring knowledge from pre-trained diffusion models. *Neural Information Processing Systems (NeurIPS)*, 2024. 38

M. Mardani, J. Song, J. Kautz, and A. Vahdat. A variational perspective on solving inverse problems with diffusion models. *arXiv preprint arXiv:2305.04391*, 2023. 50, 51, 59, 70, 71, 92

T. Marwah, Z. Lipton, and A. Risteski. Parametric complexity bounds for approximating PDEs with neural networks. *Advances in Neural Information Processing Systems*, 34:15044–15055, 2021. 101, 102

T. Marwah, Z. C. Lipton, J. Lu, and A. Risteski. Neural network approximations of PDEs beyond linearity: Representational perspective. *arXiv preprint arXiv:2210.12101*, 2022. 101, 102, 113

R. J. McCann. A convexity principle for interacting gases. *Advances in mathematics*, 128(1):153–179, 1997. 51

T. Meinhardt, M. Moller, C. Hazirbas, and D. Cremers. Learning proximal operators: Using denoising networks for regularizing inverse imaging problems. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1781–1790, 2017. 70

C. Meng, R. Gao, D. P. Kingma, S. Ermon, J. Ho, and T. Salimans. On distillation of guided diffusion models. *arXiv preprint arXiv:2210.03142*, 2022. 37, 38, 47, 48

J. P. Miller, R. Taori, A. Raghunathan, S. Sagawa, P. W. Koh, V. Shankar, P. Liang, Y. Carmon, and L. Schmidt. Accuracy on the line: on the strong correlation between out-of-distribution and in-distribution generalization. In *International Conference on Machine Learning*, pages 7721–7735. PMLR, 2021. 131

I. Mirzadeh, K. Alizadeh, H. Shahrokhi, O. Tuzel, S. Bengio, and M. Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024. 2

Y. Mishin. Machine-learning interatomic potentials for materials science. *Acta Materialia*, 214:116980, 2021. 1

T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018. 6

F. Moukalled, L. Mangani, M. Darwish, F. Moukalled, L. Mangani, and M. Darwish. *The finite volume method*. Springer, 2016. 11

M. Neumann, P. Stenetorp, and S. Riedel. Learning to reason with adaptive computation. *arXiv preprint arXiv:1610.07647*, 2016. 124

T. H. Nguyen and A. Tran. Swiftbrush: One-step text-to-image diffusion model with variational score distillation. *arXiv preprint arXiv:2312.05239*, 2023. 1, 38

M. Nye, A. J. Andreassen, G. Gur-Ari, H. Michalewski, J. Austin, D. Bieber, D. Dohan, A. Lewkowycz, M. Bosma, D. Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021. 123, 124

OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023. 1, 44

X. Pan, X. Zhan, B. Dai, D. Lin, C. C. Loy, and P. Luo. Exploiting deep generative prior for versatile image restoration and manipulation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7474–7489, 2021. 71

M. Pandey, M. Fernandez, F. Gentile, O. Isayev, A. Tropsha, A. C. Stern, and A. Cherkasov. The transformational role of gpu computing and deep learning in drug discovery. *Nature Machine Intelligence*, 4(3):211–221, 2022. 1

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-performance Deep Learning Library. In *Neural Information Processing Systems (NeurIPS)*, 2019. 41, 48

S. V. Patankar and D. B. Spalding. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. In *Numerical prediction of flow, heat transfer, turbulence and combustion*, pages 54–73. Elsevier, 1983. 106

R. G. Patel, N. A. Trask, M. A. Wood, and E. C. Cyr. A physics-informed operator regression framework for extracting data-driven continuum models. *Computer Methods in Applied Mechanics and Engineering*, 373:113500, 2021. 100

W. Peebles and S. Xie. Scalable diffusion models with transformers. *arXiv preprint arXiv:2212.09748*, 2022. 38, 40, 44, 47, 48, 49

A. Pokle, Z. Geng, and J. Z. Kolter. Deep equilibrium approaches to diffusion models. *Advances in Neural Information Processing Systems*, 35:37975–37990, 2022. 15

A. Pokle, M. J. Muckley, R. T. Chen, and B. Karrer. Training-free linear image inversion via flows. *arXiv preprint arXiv:2310.04432*, 2023. 50

B. Poole, A. Jain, J. T. Barron, and B. Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv*, 2022. 38

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017. 3, 12

A. Raj, Y. Li, and Y. Bresler. Gan-based projector for faster recovery with convergence guarantees in linear inverse problems. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5602–5611, 2019. 71

S. Rasp, M. S. Pritchard, and P. Gentine. Deep learning to represent subgrid processes in climate models. *Proceedings of the national academy of sciences*, 115(39):9684–9689, 2018. 1

M. Revay, R. Wang, and I. R. Manchester. Lipschitz bounded equilibrium networks. *arXiv:2010.01732*, 2020. 5

A. Ribes and F. Schmitt. Linear inverse problems in imaging. *IEEE Signal Processing Magazine*, 25(4): 84–99, 2008. 70

J. Rick Chang, C.-L. Li, B. Poczos, B. Vijaya Kumar, and A. C. Sankaranarayanan. One network to solve them all–solving linear inverse problems using deep projection models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5888–5897, 2017. 70

H. E. Robbins. An empirical bayes approach to statistics. In *Breakthroughs in Statistics: Foundations and basic theory*, pages 388–394. Springer, 1992. 54

Y. Romano, M. Elad, and P. Milanfar. The little engine that could: Regularization by denoising (red). *SIAM Journal on Imaging Sciences*, 10(4):1804–1844, 2017. 70

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015. 57

C. Saharia, J. Ho, W. Chan, T. Salimans, D. J. Fleet, and M. Norouzi. Image super-resolution via iterative refinement. arxiv. *arXiv preprint arXiv:2104.07636*, 2, 2021. 71

C. Saharia, W. Chan, H. Chang, C. Lee, J. Ho, T. Salimans, D. Fleet, and M. Norouzi. Palette: Image-to-image diffusion models. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–10, 2022a. 50, 59, 71

C. Saharia, J. Ho, W. Chan, T. Salimans, D. J. Fleet, and M. Norouzi. Image super-resolution via iterative refinement. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4713–4726, 2022b. 50

T. Salimans and J. Ho. Progressive distillation for fast sampling of diffusion models. In *International Conference on Learning Representations (ICLR)*, 2022. 1, 3, 37, 38, 42, 45, 46, 47, 48

T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *CoRR*, abs/1602.07868, 2016. URL `http://arxiv.org/abs/1602.07868`. 130

T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016. 42

A. Sauer, K. Schwarz, and A. Geiger. Stylegan-xl: Scaling stylegan to large diverse datasets. In *ACM SIGGRAPH 2022 conference proceedings*, pages 1–10, 2022. 45

A. Sauer, D. Lorenz, A. Blattmann, and R. Rombach. Adversarial diffusion distillation. *arXiv preprint arXiv:2311.17042*, 2023. 38

S. Scher. Toward data-driven weather and climate forecasting: Approximating a simple general circulation model with deep learning. *Geophysical Research Letters*, 45(22):12–616, 2018. 1

J. Schmidhuber. Self-delimiting neural networks. *arXiv preprint arXiv:1210.0118*, 2012. 124

J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020. 1

R. Schwartz, G. Stanovsky, S. Swayamdipta, J. Dodge, and N. A. Smith. The right tool for the job: Matching model and instance complexities. *arXiv preprint arXiv:2004.07453*, 2020. 124

A. Schwarzschild, E. Borgnia, A. Gupta, A. Bansal, Z. Emam, F. Huang, M. Goldblum, and T. Goldstein. Datasets for studying generalization from easy to hard examples. *arXiv preprint arXiv:2108.06011*, 2021a. 125

A. Schwarzschild, E. Borgnia, A. Gupta, F. Huang, U. Vishkin, M. Goldblum, and T. Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. *Advances in Neural Information Processing Systems*, 34, 2021b. 123, 124, 125

D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018. 123

V. Shah and C. Hegde. Solving linear inverse problems using gan priors: An algorithm with provable guarantees. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 4609–4613. IEEE, 2018. 71

N. Shaul, R. T. Chen, M. Nickel, M. Le, and Y. Lipman. On kinetic optimal probability paths for generative models. In *International Conference on Machine Learning*, pages 30883–30907. PMLR, 2023. 51, 53

N. Shinn, F. Cassano, B. Labash, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning.(2023). *arXiv preprint cs.AI/2303.11366*, 2023. 1

D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017. 1

J.-J. E. Slotine, W. Li, et al. *Applied nonlinear control*, volume 199. Prentice hall Englewood Cliffs, NJ, 1991. 125

J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning (ICML)*, 2015. 16, 50

L. Soldaini and A. Moschitti. The cascade transformer: an application for efficient answer sentence selection. *CoRR*, abs/2005.02534, 2020. URL https://arxiv.org/abs/2005.02534. 124

J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020a. 3, 15, 16, 17, 22, 23, 26, 27, 45, 52, 70

J. Song, A. Vahdat, M. Mardani, and J. Kautz. Pseudoinverse-guided diffusion models for inverse problems. In *International Conference on Learning Representations*, 2022. 50, 51, 54, 56, 59, 60, 70, 71, 89, 90

Y. Song and P. Dhariwal. Improved techniques for training consistency models. *arXiv preprint arXiv:2310.14189*, 2023. 2

Y. Song and S. Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019. 1, 2, 16, 50, 52, 70

Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020b. 1, 2, 7, 8, 9, 16, 45, 50, 52, 53, 59, 70

Y. Song, C. Durkan, I. Murray, and S. Ermon. Maximum likelihood training of score-based diffusion models. *Advances in Neural Information Processing Systems*, 34:1415–1428, 2021a. 52

Y. Song, L. Shen, L. Xing, and S. Ermon. Solving inverse problems in medical imaging with score-based generative models. *arXiv preprint arXiv:2111.08005*, 2021b. 70

Y. Song, P. Dhariwal, M. Chen, and I. Sutskever. Consistency models. *arXiv preprint arXiv:2303.01469*, 2023. 2, 38, 42, 45, 46, 48

B. K. Sriperumbudur and G. R. Lanckriet. On the convergence of the concave-convex procedure. In *Nips*, volume 9, pages 1759–1767. Citeseer, 2009. 125

R. Stephany and C. Earls. Pde-learn: Using deep learning to discover partial differential equations from noisy, limited data. *Neural Networks*, 174:106242, 2024. 2

S. Suganyadevi, V. Seethalakshmi, and K. Balasamy. A review on deep learning in medical image analysis. *International Journal of Multimedia Information Retrieval*, 11(1):19–38, 2022. 1

G. Team, R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023. 1

S. Teerapittayanon, B. McDanel, and H.-T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016. 124

S. Toshniwal, W. Du, I. Moshkov, B. Kisacanin, A. Ayrapetyan, and I. Gitman. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data. *arXiv preprint arXiv:2410.01560*, 2024. 2

H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 1

A. Tran, A. Mathews, L. Xie, and C. S. Ong. Factorized Fourier neural operators. *arXiv preprint arXiv:2111.13802*, 2021. 101, 104, 121

H. Tran, Z. Yao, J. Wang, Y. Zhang, Z. Yang, and H. Yu. Rare: Retrieval-augmented reasoning enhancement for large language models. *arXiv preprint arXiv:2412.02830*, 2024. 4

D. Ulyanov, A. Vedaldi, and V. Lempitsky. Deep image prior. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9446–9454, 2018. 70

A. Vahdat, K. Kreis, and J. Kautz. Score-based generative modeling in latent space. *Advances in Neural Information Processing Systems*, 34:11287–11302, 2021. 45

D. Van Veen, A. Jalal, M. Soltanolkotabi, E. Price, S. Vishwanath, and A. G. Dimakis. Compressed sensing with deep image prior and learned regularization. *arXiv preprint arXiv:1806.06438*, 2018. 70

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *Neural Information Processing Systems (NeurIPS)*, 2017a. 38, 40, 49

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Neural Information Processing Systems (NeurIPS)*, 2017b. 124

S. V. Venkatakrishnan, C. A. Bouman, and B. Wohlberg. Plug-and-play priors for model based reconstruction. In *2013 IEEE global conference on signal and information processing*, pages 945–948. IEEE, 2013. 70

Veo-Team, :, A. Gupta, A. Razavi, A. Toor, A. Gupta, D. Erhan, E. Shaw, E. Lau, F. Belletti, G. Barth-Maron, G. Shaw, H. Erdogan, H. Sidahmed, H. Nandwani, H. Moraldo, H. Kim, I. Blok, J. Donahue, J. Lezama, K. Mathewson, K. David, M. K. Lorrain, M. van Zee, M. Narasimhan, M. Wang, M. Babaeizadeh, N. Papalampidi, N. Pezzotti, N. Jha, P. Barnes, P.-J. Kindermans, R. Hornung, R. Villegas, R. Poplin, S. Zaiem, S. Dieleman, S. Ebrahimi, S. Wisdom, S. Zhang, S. Fruchter, S. Nørly, W. Hua, X. Yan, Y. Du, and Y. Chen. Veo 2. 2024. URL `https://deepmind.google/technologies/veo/veo-2/`. 1

A. F. Vidal, V. De Bortoli, M. Pereyra, and A. Durmus. Maximum likelihood estimation of regularization parameters in high-dimensional inverse problems: An empirical bayesian approach part i: Methodology and experiments. *SIAM Journal on Imaging Sciences*, 13(4):1945–1989, 2020. 70

C. Villani et al. *Optimal transport: old and new*, volume 338. Springer, 2009. 9

P. Vincent. A connection between score matching and denoising autoencoders. *Neural computation*, 23 (7):1661–1674, 2011. 9

R. Vinuesa and S. L. Brunton. Enhancing computational fluid dynamics with machine learning. *Nature Computational Science*, 2(6):358–366, 2022. 1

Y. Wan, W. Wang, Y. Yang, Y. Yuan, J.-t. Huang, P. He, W. Jiao, and M. R. Lyu. A & b== b & a: Triggering logical reasoning failures in large language models. *arXiv preprint arXiv:2401.00757*, 2024. 2

Q.-G. Wang, T. H. Lee, and C. Lin. Global stability of limit cycles. In *Relay Feedback*, pages 57–83. Springer, 2003. 125

W. Wang, L. Dong, H. Cheng, X. Liu, X. Yan, J. Gao, and F. Wei. Augmenting language models with long-term memory. *Advances in Neural Information Processing Systems*, 36, 2024a. 4

X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018. 124

Y. Wang, J. Yu, and J. Zhang. Zero-shot image restoration using denoising diffusion null-space model. *arXiv preprint arXiv:2212.00490*, 2022. 50, 63, 70, 71, 87

Z. Wang, C. Lu, Y. Wang, F. Bao, C. Li, H. Su, and J. Zhu. Prolificdreamer: High-fidelity and diverse text-to-3d generation with variational score distillation. *Advances in Neural Information Processing Systems*, 36, 2024b. 1, 38

J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022. 4

J. Whang, Q. Lei, and A. Dimakis. Solving inverse problems with a flow-based noise model. In *International Conference on Machine Learning*, pages 11146–11157. PMLR, 2021a. 70, 71

J. Whang, E. Lindgren, and A. Dimakis. Composing normalizing flows for inverse problems. In *International Conference on Machine Learning*, pages 11158–11169. PMLR, 2021b. 70, 71

S. Williams and J. Huckle. Easy problems that llms get wrong. *arXiv preprint arXiv:2405.19616*, 2024. 2

E. Winston and J. Z. Kolter. Monotone operator equilibrium networks. *Advances in neural information processing systems*, 33:10718–10728, 2020. 5, 6

Q. Xing, M. Xu, T. Li, and Z. Guan. Early exit or not: Resource-efficient blind quality enhancement for compressed images. In *European Conference on Computer Vision*, pages 275–292. Springer, 2020. 124

R. Xiong, Y. Yang, D. He, K. Zheng, S. Zheng, C. Xing, H. Zhang, Y. Lan, L. Wang, and T. Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning (ICML)*, 2020. 40

P. Xu, X. Ji, M. Li, and W. Lu. Small data machine learning in materials science. *npj Computational Materials*, 9(1):42, 2023a. 2

Y. Xu, Y. Zhao, Z. Xiao, and T. Hou. Ufogen: You forward once large scale text-to-image generation via diffusion gans. *arXiv preprint arXiv:2311.09257*, 2023b. 38

S. Xue, M. Yi, W. Luo, S. Zhang, J. Sun, Z. Li, and Z.-M. Ma. Sa-solver: Stochastic adams solver for fast sampling of diffusion models. *Neural Information Processing Systems (NeurIPS)*, 2024. 49

A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Li, D. Liu, F. Huang, H. Wei, H. Lin, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Lin, K. Dang, K. Lu, K. Bao, K. Yang, L. Yu, M. Li, M. Xue, P. Zhang, Q. Zhu, R. Men, R. Lin, T. Li, T. Xia, X. Ren, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Wan, Y. Liu, Z. Cui, Z. Zhang, and Z. Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024. 1

J. Yang, J. Wright, T. S. Huang, and Y. Ma. Image super-resolution via sparse representation. *IEEE transactions on image processing*, 19(11):2861–2873, 2010. 70

K. Yang, J. H. Yau, L. Fei-Fei, J. Deng, and O. Russakovsky. A study of face obfuscation in imagenet. In *International Conference on Machine Learning*, pages 25313–25330. PMLR, 2022. 57

S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022. 1

S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. *URL https://arxiv. org/pdf/2305.10601. pdf*, 2023. 4

T. Yin, M. Gharbi, R. Zhang, E. Shechtman, F. Durand, W. T. Freeman, and T. Park. One-step diffusion with distribution matching distillation. *arXiv preprint arXiv:2311.18828*, 2023. 38

F. Yu, Y. Zhang, S. Song, A. Seff, and J. Xiao. LSUN: construction of a large-scale image dataset using deep learning with humans in the loop. *CoRR*, abs/1506.03365, 2015. 21

K. Zhang, W. Zuo, S. Gu, and L. Zhang. Learning deep cnn denoiser prior for image restoration. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3929–3938, 2017. 70

Q. Zhang and Y. Chen. Fast sampling of diffusion models with exponential integrator. *arXiv preprint arXiv:2204.13902*, 2022. 49

Q. Zhang and Y. Chen. Fast sampling of diffusion models with exponential integrator. In *The Eleventh International Conference on Learning Representations*, 2023. 45

R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018. 48, 59

W. Zhao, L. Bai, Y. Rao, J. Zhou, and J. Lu. Unipc: A unified predictor-corrector framework for fast sampling of diffusion models. *Neural Information Processing Systems (NeurIPS)*, 2024. 49

X. Zhao, Y.-S. Ting, K. Diao, and Y. Mao. Can diffusion model conditionally generate astrophysical images? *Monthly Notices of the Royal Astronomical Society*, 526(2):1699–1712, 2023. 1

H. Zheng, W. Nie, A. Vahdat, K. Azizzadenesheli, and A. Anandkumar. Fast sampling of diffusion models via operator learning. *arXiv preprint arXiv:2211.13449*, 2022. 38, 45, 48

M. Zhou, Z. Wang, H. Zheng, and H. Huang. Long and short guidance in score identity distillation for one-step text-to-image generation. *ArXiv 2406.01561*, 2024a. URL `https://arxiv.org/abs/2406.01561`. 38

M. Zhou, H. Zheng, Z. Wang, M. Yin, and H. Huang. Score identity distillation: Exponentially fast distillation of pretrained diffusion models for one-step generation. In *International Conference on Machine Learning*, 2024b. 38

W. Zhou, C. Xu, T. Ge, J. McAuley, K. Xu, and F. Wei. Bert loses patience: Fast and robust inference with early exit. *Advances in Neural Information Processing Systems*, 33:18330–18341, 2020. 124