

ABSTRACT

Title of Dissertation: Next-Gen AI: Advancing Watermarking, Algorithm Synthesis
and Diverse Generative Strategies

Arpit Bansal
Doctor of Philosophy, 2025

Dissertation Directed by: Professor Tom Goldstein
Department of Computer Science

This doctoral thesis explores critical advancements in generative artificial intelligence (AI) across three domains: enhancing security through watermarking, advancing algorithm synthesis, and pioneering diverse strategies for generative image tasks. The research contributes novel methodologies and unified frameworks that push the boundaries of AI capabilities.

The first part of the thesis addresses the growing need for AI security by introducing certified watermarking techniques to safeguard deep neural networks (DNNs) from intellectual property violations. These methods establish robust protections for model ownership and integrity, ensuring the secure deployment of AI innovations.

The second part investigates algorithm synthesis, expanding the computational and reasoning capabilities of neural networks. By enabling complex problem-solving and demonstrating advanced adaptability, this work redefines the role of neural networks as tools for sophisticated algorithmic design, transcending traditional applications in pattern recognition.

The third part focuses on diverse generative strategies for image creation and editing. It begins with Cold Diffusion, employing deterministic transformations to expand the operational mechanics of diffusion models. The research further enhances the generative process by enabling the creation of highly specific and contextually relevant imagery with minimal retraining, improving the flexibility and practicality of diffusion-based approaches. Finally, the thesis presents a unified framework for image reasoning and generation, leveraging next-token prediction with a vision encoder that produces discrete, non-lossy image embeddings aligned with language. This innovation enables a transformer-based architecture to support both high-precision image editing and advanced reasoning, paving the way for a cohesive and versatile AI design.

By addressing security, algorithmic adaptability, and generative innovation, this thesis contributes to the development of next-generation AI systems. It establishes a strong foundation for future advancements in AI technologies, ensuring secure, adaptable, and creative solutions for a wide range of applications.

Next-Gen AI: Advancing Watermarking, Algorithm Synthesis, and Diverse
Generative Strategies

by

Arpit Bansal

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2025

Advisory Committee:

Professor Tom Goldstein, Chair/Advisor
Professor Wojciech Czaja, Dean's Representative
Professor Furong Huang
Professor Tianyi Zhou
Professor Jia-Bin Huang

© Copyright by
author
2025

Acknowledgments

This dissertation is the culmination of collective efforts and support from numerous individuals and institutions over the past 4–5 years. I am profoundly grateful to my collaborators, friends, family, administrative staff, and everyone else whose contributions, encouragement, and guidance have been instrumental in bringing this work to fruition.

First and foremost, I extend my deepest gratitude to my advisor, Prof. Tom Goldstein, for his unwavering support throughout my journey. His empowering and enabling guidance has been instrumental in helping me grow to my full potential and enhance my ability to strive for excellence. From the early days of my PhD, when I was just beginning to find my footing, he has been there with his creative and often “crazy” ideas, encouraging me to persevere and work hard to see them through. His belief in my abilities and his willingness to give me the freedom to explore my own ideas have been invaluable. I am especially grateful for the friendly and welcoming work environment he has fostered, complemented by his great sense of humor and cool demeanor.

I am also incredibly thankful to Prof. Micah Goldblum and Prof. Jonas Geiping, whose collaboration and guidance have been instrumental in shaping my research. The stimulating research discussions and their availability for both professional and personal conversations have provided me with immense support and confidence throughout this journey. I am similarly grateful to Vignesh Ramanathan, who gave me the opportunity to intern at Meta and supported

my research during that time.

I would like to extend my heartfelt thanks to my committee members: Prof. Wojciech Czaja, Prof. Tianyi Zhou, Prof. Jia-Bin Huang, and Prof. Furong Huang, for their invaluable feedback. Their insightful comments and suggestions have significantly contributed to the refinement of this dissertation.

My PhD work would not have been possible without the collaboration and contributions of many talented individuals. I deeply appreciate the engaging discussions and enjoyable research projects shared with Ping-Yeh Chiang, Valeriia Cherepanova, Avi Schwarzschild, Eitan Borgnia, Hong-Min Chu, Jie S. Li, Hamid Kazemi, Gowthami Somepalli, Liam Fowl, Roman Levin, Yuxin Wen, Alex Stein, Sean McLeish, Neel Jain, and John Kirchenbauer.

My time at UMD has been nothing short of inspiring. I am immensely proud to be a Terp and deeply grateful for the welcoming environment I found here. A special thanks to Tom Hurst, Migo Gui, and the staff and faculty of the Computer Science Department at UMD and UMIACS, whose exceptional support made navigating bureaucracy and compute-related challenges much smoother.

I owe a great deal to my friends who have been my pillars of support during these years: Siddharth Singh, Divya Kothandaraman, Harsha Walvekar, Soumik Mukhopadhyay, Swarup Kumar Subudhi, Navita Goyal, Sharmila Duppala, Vedant Nanda, and Kishalay Mahato. Their friendship has been a source of joy and comfort.

I am forever indebted to my mother, Anjali Bansal, and my Father, Amit Bansal, for their sacrifices, unwavering love, and encouragement. Their belief in me has been my constant source of strength. Everything I am and everything I will become in this life is a result of their efforts and teachings. To my sister Sonakshi Bansal for coming into my life and showering me with love

and happiness. Finally, I would not have navigated these past few years as smoothly without the unwavering support of my fiancée, Pravali. Her presence has made this journey not only bearable but truly wonderful.

Lastly, I thank God for guiding me at every step and helping me overcome obstacles. Gita's teachings about karma yoga have been a profound influence, reminding me to persevere through challenges with resilience and focus.

Table of Contents

Acknowledgements	ii
Table of Contents	v
List of Tables	viii
List of Figures	x
I Introduction	1
II AI Security	5
Chapter 1: Certified Neural Network Watermarks	6
1.1 Watermarking Neural Networks	6
1.2 Methods	9
1.2.1 Watermarking	9
1.2.2 Watermark Certification	12
1.3 Experiments	16
1.3.1 Experimental Settings	16
1.3.2 Properties of a Good Threat Model	17
1.3.3 Watermark Certificate Evaluation	19
1.3.4 Empirical Watermark Persistence Evaluation	22
III AI Reasoning	25
Chapter 2: End-to-end Algorithm Synthesis with Recurrent Networks: Logical Extrapolation Without Overthinking	26
2.1 Introduction	26
2.2 Related work	29
2.3 Methods	31
2.3.1 Recall architectures	33
2.3.2 Promoting forward progress through optimization	34
2.3.3 Datasets	35
2.4 Extreme extrapolation	37

2.5	The overthinking problem	41
2.5.1	Manipulating feature maps	42
2.5.2	Manipulating inputs	42
2.5.3	Converging to a fixed point	44
Chapter 3:	Transformers Can Do Arithmetic with the Right Embeddings	45
3.1	Introduction	45
3.2	Related Work	47
3.3	Achieving Length Generalization for Addition	50
3.3.1	Abacus Embeddings Help Align Digits	51
3.3.2	Recurrence In Transformers Boosts Performance	54
3.4	Pushing the Limits of Algorithmic Reasoning for Transformers	56
3.4.1	Integer Multiplication	56
3.4.2	Array Sorting	57
3.4.3	Abacus and Relative Embeddings	59
3.5	Discussion & Limitations	60
IV	AI Generation	63
Chapter 4:	Cold Diffusion: Inverting Arbitrary Image Transforms Without Noise	64
4.1	Introduction	64
4.2	Background	66
4.3	Generalized Diffusion	67
4.3.1	Model components and training	67
4.3.2	Sampling from the model	68
4.3.3	Properties of TACoS	69
4.4	Generalized Diffusions with Various Transformations	71
4.4.1	Deblurring	72
4.4.2	Inpainting	73
4.4.3	Super-Resolution	75
4.5	Cold Generation	76
4.5.1	Generation using deterministic noise degradation	76
4.5.2	Image generation using blur	77
4.5.3	Generation using other transformations	78
Chapter 5:	Universal Guidance for Diffusion Models	81
5.1	Introduction	81
5.2	Background	83
5.2.1	Diffusion Models	84
5.2.2	Controlled Image Generation	85
5.3	Universal Guidance	87
5.3.1	Forward Universal Guidance	87
5.3.2	Backward Universal Guidance	89
5.3.3	Universal Stepwise Refinement	91

5.4	Experiments	93
5.4.1	Results for Stable Diffusion	93
5.4.2	Results for ImageNet Diffusion	101

V AI Reasoning + Generation 105

Chapter 6:	DualCode: Discrete Embeddings for Unified Image Generation and Reasoning	107
6.1	Introduction	108
6.2	Background	110
6.3	DualCode	113
6.4	Experiments	115
6.4.1	Training Methodology	116
6.4.2	Training Data	118
6.5	Results	122
6.6	Conclusion	125

List of Tables

1.1	Attack Radius vs Worst Case Accuracy of the Model. It becomes meaningless to defend against a threat model with a radius larger than 1.8 because these models are indistinguishable from any randomly initialized model.	17
1.2	Certified trigger set accuracy at different radius	19
1.3	Trade-off between certified trigger set accuracy and noise level (σ) for CIFAR-10	20
1.4	ℓ_2 distance change in the first epoch	21
1.5	Trigger set accuracy after 50 epochs of removal attacks. We note that this is only a snapshot of the trigger set accuracy. During training, trigger set accuracies could sometimes fluctuate significantly. We use watermarks from [1] as the baseline watermark.	21
3.1	Exact match accuracy for sorting with various positional embeddings. All results are percentages of the test set.	57
3.2	Accuracy for sorting with various architectures. ST denotes standard transformer, ST w/ II denotes standard transformer with input injection, and LT denotes looped transformer models. All results are percentages of the test set.	58
4.1	Quantitative metrics for quality of image reconstruction using deblurring models.	73
4.2	Quantitative metrics for quality of image reconstruction using inpainting models.	74
4.3	Quantitative metrics for quality of image reconstruction using super-resolution models.	75
4.4	FID scores for CelebA and AFHQ datasets using hot (noise) and cold diffusion (blur transformation). Breaking the symmetry within pixels of the same channel further improves FID.	78
5.1	Quantitative analysis of different guidance applied on Stable Diffusion. The reference value in parenthesis is obtained with no external guidance.	96
5.2	Quantitative analysis of forward guidance only versus combination of forward and backward guidance for object detection guidance on ImageNet with bounding boxes in Figure 5.7. The metric is mAP@50.	102
6.1	Comparison of DualCode with other models across four image reasoning tasks. Results are shown for DualCode with and without weakly labeled training, alongside SEED-X-I and SEED-X-Edit optimized for reasoning and editing, and Chameleon-7B. Best results are bold , second best are <u>underlined</u>	123

6.2	Performance comparison of models across image editing tasks: Color Alteration, Object Replacement, Style Alteration, and Object Removal. DualCode is evaluated in two versions: with and without weakly labeled data. Comparisons include VQ-GAN and SEED-X-Edit. Scores are generated using Gemini-1.5-Pro and GPT-4o-mini. Best results are bold , second best are <u>underlined</u>	123
-----	---	-----

List of Figures

1.1	Original	10
1.2	Embedded Content	10
1.3	Gaussian Noise	10
1.4	Unrelated	10
1.5	Samples of the backdoor images used for watermarking.	10
1.6	Graphical illustration of perfect, sufficient, and necessary threat models.	17
2.1	Architecture schematics. Left to right: A feed-forward network, a network containing three recurrent blocks (in green) that share weights, and a recurrent network with recall.	31
2.2	Prefix sum models trained on 32-bit inputs extrapolate to 512-bit data. The value of our recall and progressive loss is clear by how quickly and accurately our models solve this very large problem.	38
2.3	Maze solving models trained on 9×9 inputs extrapolate to 59×59 problems. Mazes this large cannot be solved without recall, and furthermore progressive loss leads to more accurate models.	38
2.4	Chess models trained on the first 600K easiest puzzles extrapolate to 600K-700K. Recall and progressive loss are required to retain accuracy with many iterations.	40
2.5	Left: Prefix sum models trained on 32-bits tested on 48-bit inputs. Right: Maze models trained on 9×9 mazes tested on 13×13 mazes. In these regimes, we can see that the weaker models can extrapolate to slightly harder problems, but importantly only models with recall avoid the overthinking trap.	41
2.6	Maze solving model performance when the start and end of the maze are changed to be closer together after 50 iterations. DT models with recall can self-correct.	43
2.7	Left: How long it takes prefix sum models to recover from perturbation – recall keeps this quantity small. Center: Test accuracy on 13×13 mazes when features are swapped after 50 iterations. Right: The change in the features when solving 13×13 mazes.	44
3.1	Visualization of data formats and positional embeddings. <i>Abacus Embeddings</i> give the same positional embeddings to all digits of the same significance.	49

3.2	Left: Mean exact match accuracy of three models of depth sixteen on size 20 data, varying the architecture and embeddings. Abacus Embeddings improve accuracy for addition over FIRE and NoPE Embeddings. Right: Mean exact match accuracy of three models of effective depth sixteen on size 40 data, varying over NoPE or FIRE embeddings and architectures. Recurrent looped transformer models improve accuracy for addition for both the FIRE and NoPE embeddings. <i>Looped transformer (LT):</i> Weight tied decoder layers, with input injection and progressive loss. <i>Standard Transformer (ST):</i> Stacked decoder only layers. <i>Standard Transformer with Input Injection (ST w/ II):</i> Standard Transformer with input features added to the hidden representation between each decoder layer.	54
3.3	Varying the size of the recurrent block, while maintaining an effective depth of 16 and training on size 20 data. We see that a recurrent model with eight layers in the recurrent block and two recurrences is the most accurate of all effective depth 16 models, halving the error rate of a standard model with input injection in the OOD evaluation. (See Figure ?? for results with FIRE and NoPE.)	56
3.4	Exact match accuracy of looped transformer models trained on multiplication, with four layers in the recurrent block and four recurrences. The red square denotes in distribution testing on up to 15 digit operands. We see the models with Abacus Embeddings achieve near perfect in distribution accuracy. Combining Abacus Embeddings with FIRE also improves in distribution accuracy on the hardest in distribution problems (bottom right), comparing to the FIRE-only baseline.	58
3.5	Exact match accuracy of standard transformer of depth 16 with input injection, trained on up to size 20 data. The red square denotes in distribution testing. Combining Abacus Embeddings with FIRE or RoPE embeddings improves out of distribution accuracy for addition, over the baseline models without Abacus Embeddings.	60
4.1	Demonstration of the forward and backward processes for both hot and cold diffusions. While standard diffusions are built on Gaussian noise (top row), we show that generative models can be built on arbitrary and even noiseless/cold image transforms, including the ImageNet-C <i>snowification</i> operator, and an <i>animorphosis</i> operator that adds a random animal image from AFHQ.	64
4.2	Comparison of sampling methods for unconditional generation using cold diffusion on the CelebA dataset. Iterations 2, 4, 8, 16, 32, 64, 128, 192, and 256 are presented. Top: Algorithm 3 produces compounding artifacts and fails to generate a new image. Bottom: TACoS succeeds in sampling a high quality image without noise.	70
4.3	Deblurring models trained on the MNIST, CIFAR-10, and CelebA datasets. Left to right: degraded inputs $D(x_0, T)$, direct reconstruction $R(D(x_0, T))$, sampled reconstruction with TACoS described in Algorithm 4, and original image.	73
4.4	Inpainting models trained on the MNIST, CIFAR-10, and CelebA datasets. Left to right: Degraded inputs $D(x_0, T)$, direct reconstruction $R(D(x_0, T))$, sampled reconstruction with TACoS described in Algorithm 4, and original image.	74

4.5	Superresolution models trained on the MNIST, CIFAR-10, and CelebA datasets. Left to right: degraded inputs $D(x_0, T)$, direct reconstruction $R(D(x_0, T))$, sampled reconstruction with TACoS described in Algorithm 4, and original image.	75
4.6	Examples of generated samples from 128×128 CelebA and AFHQ datasets using cold diffusion with blur transformation	78
4.7	Preliminary demonstration of the generative abilities of other cold diffusins on the 128×128 CelebA dataset. The top row is with <i>animorphosis</i> models, the middle row is with inpainting models, and the bottom row exhibits super-resolution models.	80
5.1	Diffusion guided by off-the-shelf networks. Top left: segmentation guidance, top-right: face recognition guidance, bottom-left: object detection guidance, bottom-right: style-transfer.	81
5.2	Illustration of the Universal Stepwise Refinement (USR) in face-guided generation. The left-most image is the guidance face, while the adjacent images are generated with USR using $k = 1$ and $k = 2$, respectively. A modest increment in USR steps from 1 (the trivial value) to 2 significantly enhances both guidance adherence and image fidelity.	91
6.1	Demonstrating the capability of DualCode embeddings for image editing. On the left , it transfers a bookshelf from one image to another, replacing the painting. In multi-turn editing (right), when asked to remove the apple, it performs the task correctly. Furthermore, when asked to remove the black purse instead, the autoregressive nature of the model allows it to retain the history of inputs, remembering the removed apple and thus only removing the black purse.	107
6.2	Overview of the Unified Vision Embedding Framework: Top Left: The DualCode framework integrates features from CLIP, DinoV2, and VQGAN through a Mixer architecture, generating unified embeddings of size $N_v \times D$. A quantization module processes these embeddings to produce discrete outputs, which are then concatenated with the original VQGAN embeddings and projected into the LLM input space via fully connected layers. Top Right: The Mixer architecture combines input embeddings from different sources (e.g., CLIP: $N_c \times D_c$, DinoV2: $N_d \times D_d$, VQGAN: $N_v \times D_v$) into a cohesive embedding of size $N \times D$. Bottom Left: The Mixer architecture uses Separable Linear Blocks, each comprising two MLPs. The first MLP projects the input from $N_i \times D_i$ to $N_i \times D_o$, followed by a transpose operation. The second MLP transforms the transposed input from $D_o \times N_i$ to $D_o \times N_o$ before transposing it back to $N_o \times D_o$, ensuring adaptable and consistent embedding transformation. Bottom Right: Each MLP block within the Separable Linear Block includes a linear layer followed by layer normalization, facilitating stability and efficient processing.	112

6.3	Comparison of DualCode, VQ-GAN, and SEED-X-Edit models across various tasks. Top Left: DualCode achieves precise color alteration, outperforming VQ-GAN in consistency and SEED-X in content preservation. Top Right: DualCode excels in object removal with clean inpainting, while VQ-GAN struggles with precision. Bottom Left: DualCode produces natural style transfer results and effective object replacements, though SEED-X outshines in the latter task due to its pretrained SDXL model. Bottom Right: DualCode demonstrates superior multi-image editing capabilities, surpassing VQ-GAN. SEED-X is excluded from multi-turn editing due to its single image-to-image training limitations.	120
6.4	Performance of various vision embeddings: LLM aligned Continuous Embeddings, Discrete VQ-GAN embeddings, and DualCode Embeddings. For image reasoning tasks, performance is evaluated using four benchmarks: MME, MMBench, POPE, and MMStar. For image editing tasks, we evaluate the embeddings on three scenarios: Single Image-to-Image, Multi-Image, and Multi-Turn Image Editing. The editing capabilities are assessed using Gemini-1.5-Pro and GPT-4o-mini. Continuous embeddings are lossy and hence unsuitable for high-precision image editing.	121
6.5	Demonstrating multi-turn image editing with end-to-end autoregressive models trained on VQGAN and DualCode embeddings. Unlike traditional models, which modify only the last image and ‘forget’ previous edits, an end-to-end autoregressive model retains image history and attends to prior edits. In this example, the model first removes object A (e.g., the Christmas tree, cup, or dog), then removes object B instead of A, all while referencing the original image. DualCode outperforms VQGAN, which sometimes misses the target object or removes unintended elements, showing superior performance in multi-turn image editing.	121

Part I

Introduction

In the rapidly evolving landscape of artificial intelligence (AI), the 21st century has witnessed unprecedented advancements, marked by the emergence of sophisticated algorithms and generative models capable of surpassing human creativity. This technological progress heralds the dawn of the next generation of AI, poised to redefine digital innovation and automate complex cognitive tasks. At the forefront of this transformative era lies generative AI, a branch of artificial intelligence that leverages advanced neural network architectures to create diverse content, spanning from textual narratives to intricate visual artworks. As we navigate through this era of rapid technological advancement, it becomes increasingly imperative to address the multifaceted challenges and opportunities inherent in the development and deployment of generative models.

The doctoral research presented herein makes significant contributions to the field across various dimensions:

1. Advancement in Certified Watermarking:

With the exponential growth in the value of AI models and their pivotal role in driving innovation across industries, ensuring the security and integrity of these models has become paramount. The initial segment of the research focuses on advancing deep network watermarking techniques as a proactive measure against theft and unauthorized use. This research introduces certified watermarking, a novel approach that embeds indelible signatures within AI models, providing robust and tamper-evident security measures. By establishing a certification mechanism, this approach ensures the integrity of the watermark, effectively thwarting attempts to bypass or alter it. Rigorous experimentation and validation on benchmark datasets such as CIFAR-10 validate the effectiveness and reliability of certified watermarking techniques in safeguarding AI models and intellectual property.

2. Exploration of Algorithm Learning Capabilities:

Going beyond mere pattern recognition, the research delves into whether neural networks can learn algorithms and extrapolate from simple to complex tasks. Drawing inspiration from cognitive science and prior research in deep thinking, this segment investigates the evolution of recurrent networks in solving increasingly intricate problems over time. Systematic training on elementary datasets and exposure to progressively challenging scenarios unveil the remarkable potential of AI systems in advanced reasoning and problem-solving. By showcasing the adaptability of recurrent networks in solving diverse tasks ranging from mazes to chess, this research sheds light on the cognitive depth and versatility of AI systems.

3. Enhancement of Image Generation with Diffusion Models:

In the realm of image generation, the research explores innovative approaches to enhance the capabilities of diffusion models. Building upon foundational research in generative modeling, this effort seeks to challenge traditional assumptions and push the boundaries of what is achievable in image generation. Through a comprehensive examination of diffusion model mechanisms and theory, the research uncovers novel insights into the generation of realistic images. Introducing the concept of "cold diffusion," the study demonstrates the feasibility of deterministic transformations as viable alternatives to noise-based techniques. Furthermore, the research proposes an algorithm for image generation based on intricate guidance, leveraging segmentation maps, facial identities, object locations, and image styles. This algorithm, requiring minimal training and leveraging pretrained models, streamlines the image generation process, enhancing its versatility and applicability across diverse

domains.

4. Unified Image Reasoning and Generation:

Multimodal Large Language Models (MLLMs) are transforming AI with their ability to generate text and images, performing tasks like image reasoning and high-precision editing. While decoder-only architectures aligned with large language models (LLMs) have proven effective, they require vision embeddings that balance semantic understanding and detailed visual fidelity. Existing models like CLIP prioritize semantic alignment but are lossy, while VQGAN excels in visual reconstruction but lacks linguistic coherence, limiting their versatility. To address this, we introduce DualCode, a lightweight framework that augments pre-trained VQGAN embeddings with semantic tokens from models like CLIP and DINOv2, creating unified embeddings that are both non-lossy and semantically rich. DualCode eliminates the need for costly retraining, supports next-token prediction in decoder-only architectures, and achieves scalable, high-fidelity reasoning and editing. Demonstrating superior performance against models like SEED-X and Chameleon, DualCode bridges the gap between visual fidelity and semantic alignment, advancing watermarking, algorithm synthesis, and diverse generative strategies for next-generation AI systems.

Part II

AI Security

Chapter 1: Certified Neural Network Watermarks

1.1 Watermarking Neural Networks

With the rise of deep learning, there has been an extraordinary growth in the use of neural networks in various computer vision and natural language understanding tasks. In parallel with this growth in applications, there has been exponential growth in terms of the cost required to develop and train state-of-the-art models [2]. For example, the GPT-3 generative language model [3] is estimated to cost around 4.6 million dollars [4] in TPU cost alone. This does not include the cost of acquiring and labeling data or paying engineers, which may be even greater. With up-front investment costs growing, if access to models is offered as a service, the incentive is strong for an adversary to try to steal the model, sidestepping the costly training process. Incentives are equally strong for companies to protect such a significant investment.

Watermarking techniques have long been used to protect the copyright of digital multimedia [5]. The copyright holder hides some imperceptible information in images, videos, or sound. When they suspect a copyright violation, the source and destination of the multimedia can be identified, enabling appropriate follow-up actions.

Watermarking in Neural Networks

Recently, watermarking has been extended to deter the theft of machine learning models

[1, 6]. The model owner either imprints a predetermined signature into the parameters of the model [6] or trains the model to give predetermined predictions [1] for a certain trigger set (e.g., images superimposed with a predetermined pattern).

A strong watermark must also resist removal by a motivated adversary. Even though the watermarks in [1, 6, 7] initially claimed some resistance to various watermark removal attacks, it was later shown in [8, 9] that these watermarks can, in fact, be removed with more sophisticated methods, using a combination of distillation, parameter regularization, and finetuning.

Certified Watermarking

To address concerns regarding the effectiveness of existing watermarking techniques, we propose a certifiable watermark: unless the attacker changes the model parameters by more than a certain ℓ_2 distance, the watermark is guaranteed to remain. Our proposed watermarking technique is the first to provide a certificate against an ℓ_2 adversary. We also analyzed whether ℓ_2 adversary is a reasonable threat model as well as the magnitude of appropriate defense radius. Surprisingly, we find our certified radius to be quite substantial relative to the range of meaningful radius that one could certify. Additionally, we empirically find that our certified watermark is more resistant to previously proposed watermark removal attacks compared to its counterparts – it is thus valuable even when a certificate is not required.

Related Work

[6] proposed the first method of watermarking neural networks: they embed the watermark into the parameters of the network during training through regularization. However, the proposed approach requires explicit inspection of the parameters for ownership verification. Later, [1, 10] improved upon this approach, such that the watermark can be verified through API-only access

to the model. Specifically, they embed the watermark by forcing the network to deliberately misclassify certain “backdoor” images. The ownership can then be verified through the adversary’s API by testing its predictions on these images.

In light of later and stronger watermark removal techniques, several papers have proposed methods to improve neural network watermarking. However, one concern for all these watermark methods is that a sufficiently motivated adversary may attempt to remove the watermark. Even though existing methods claim resistance to watermark removal attacks, such as finetuning, other authors later show that by adding regularization, finetuning and pruning, their watermarks can be removed without compromising the prediction accuracy of the stolen model. To avoid this cycle, we propose a certifiably unremovable watermark: given that parameters are not modified more than a given threshold ℓ_2 distance, the watermark will be preserved.

Our work is inspired by recent work on certified adversarial robustness. Certified adversarial robustness involves not only training the model to be robust to adversarial attacks under particular threat models, but also proving that no possible attacks under a particular constraint could possibly succeed. Specifically, in this paper, we used the randomized smoothing technique first developed for classifiers, and later extended to deal with regression models. However, as opposed to defending against an ℓ_2 -bounded threat models in the image space, we are now defending against an ℓ_2 -bounded adversary in the parameter space. Surprisingly, even though the certificate holds only when randomized smoothing is applied, empirically, when our watermark is evaluated in a black-box setting on the non-smoothed model, it also exhibits stronger persistence compared to previous methods.

1.2 Methods

Below, we introduce the formal model for neural network watermarking, and the watermark removal adversaries that we are concerned with. Then, we describe some background related to randomized smoothing, and show that by using randomized smoothing we can create a watermark that provably cannot be removed by an ℓ_2 adversary.

1.2.1 Watermarking

White box vs black box We first introduce the distinction between black box and white box settings from the perspective of the owner of the model. In a white box setting, parameters are known. In a black box setting, the model parameters are hidden behind an API. We consider cases where the owner may have either black box or white box access to verify their watermarks.

Black-box watermarking In backdoor-based watermarking, the owner employs a “trigger set” of specially chosen images that has disjoint distribution compared to the original dataset. If another model makes correct predictions on this trigger set, then this is evidence that the model has been stolen. A backdoor-based watermark can be verified in a black-box setting.

The trigger set may be chosen in various ways. [1] considered three different methods of generating the trigger set: embedded content, pre-specified noise, and abstract images. Embedded content methods embed text over existing datasets and assigns all examples with the text overlay the same fixed label. Pre-specified noise overlays Gaussian noise on top of existing dataset and again assigns the examples with the same fixed label. For abstract images, a set of images from a different domain is additionally used to train the network. For example, MNIST images could

form the trigger set for a CIFAR-10 network, so if an adversary’s model performs exceedingly well on MNIST images, then the adversary must have used the stolen model. Examples of trigger set images are presented can be found in 1.5.



Figure 1.1: Original



Figure 1.2: Embedded Content

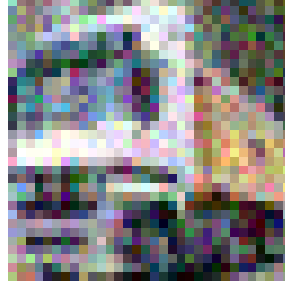


Figure 1.3: Gaussian Noise

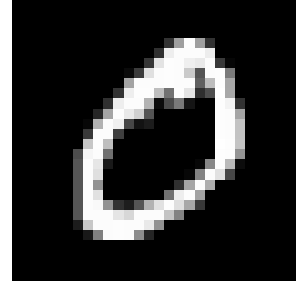


Figure 1.4: Unrelated Noise

Figure 1.5: Samples of the backdoor images used for watermarking.

Our proposed method builds upon such backdoor-based watermarks, so our marked model can also naturally be verified in the black-box manner even though our certificate is only valid in the white-box setting described in the next section.

White-box watermarking White-box watermarks in general embed information directly into the parameters. Our proposed watermark does not directly embed information into parameters, but parameter access is required for verification, so it is still a white-box watermark. The rationale for using such a white-box watermark is detailed below.

In the black-box setting, to verify model ownership, we generally check that the trigger set accuracy function from parameters to accuracy $f(\theta)$ is larger than a threshold [8]. The trigger set accuracy function takes in model parameter as input and outputs the accuracy on the trigger set. Since directly certifying the function is hard, we first convert the trigger set accuracy function $f(\theta)$ to its smoothed counterpart $h(\theta)$, and then check that $h(\theta)$ is greater than the threshold t for ownership verification. Practically, one converts the base function to the smoothed function

by injecting random noise into the parameters during multiple trigger set evaluations, and then taking the median trigger set accuracy as $\hat{h}$. Note that this verification process *requires* access to parameters, so ownership verification using $\hat{h}$ is considered a *white-box* watermark.

Watermark Removal Threat Model

In our experiments, we consider three different threat models to the watermark verification:

1) distillation, 2) finetuning, and 3) an ℓ_2 adversary.

In the distillation threat model (1), we assume that the adversary initializes their model with our original model, and then trains their model with distillation using unlabeled data that comes from the same distribution. In other words, the adversary uses our original model to label the unlabeled data for finetuning. [8] propose first adding some regularization during the initial part of the attack to remove the watermark, and then later turning off the regularization to fully recover the test accuracy of the model. We experiment with this distillation attack both with and without regularization.

In the finetuning threat model (2), the adversary has its own labeled dataset from the original data-generating distribution. This adversary is strictly stronger compared to the distillation threat model. In our experiments, we make the conservative assumption that the adversary has exactly the same amount of data as the model owner.

The ℓ_2 adversary (3) obtains the original model parameters, and then is allowed to move the parameters at most a certain ℓ_2 distance to maximally decrease trigger set accuracy. Even though the ℓ_2 adversary is not a completely realistic threat model, we argue similarly to the adversarial robustness literature [11] that being able to defend against a small ℓ_2 adversary is a requirement for defending against more sophisticated attacks. In our experiments, we empirically find that a

large shift of parameters in ℓ_2 distance is indicative of the strength of the adversary. For example, training the models for more time, with a larger learning rate, or using ground truth labels as opposed to distillation are all stronger attacks, and as expected, they both remove the watermark faster and move the parameters by a greater ℓ_2 distance (Table 1.4). Additionally, given a local Lipschitz constant of L and a learning rate of r , the number of steps required to move outside of the ϵ - ℓ_2 ball can be upper bounded by $\epsilon/(rL)$, and we think the number of steps is a good proxy for the computational budget of the adversary.

1.2.2 Watermark Certification

For our certificates, we focus on the ℓ_2 adversary described above: the goal of certification is to bound the worst-case decrease in trigger set accuracy, given that the model parameters do not move too far in ℓ_2 distance. Doing this directly is in general quite difficult [12], but using techniques from [13, 14], we show that by adding random noise to the parameters it is possible to define a smoothed version of the model and bound the change in its trigger set accuracy.

Deriving the certificate

Before we start describing the watermark certificate, we will first introduce the percentile smoothed function from [13].

Definition 1.2.1. *Given $f : \mathbb{R}^d \rightarrow \mathbb{R}$ and $G \sim N(0, \sigma^2 I)$, we define the percentile smoothing of f as*

$$p(x) = \sup\{y \in \mathbb{R} \mid \mathbb{P}[f(x + G) \leq y] \leq p\} \quad (1.2.2.1)$$

$$p(x) = \inf(\{y \in \mathbb{R} \mid \mathbb{P}[f(x + G) \leq y] \geq p\}) \quad (1.2.2.2)$$

As mentioned in [13], the two forms p and p are needed to handle edge cases with discrete distributions. While p may not admit a closed form, we can approximate it by Monte Carlo sampling [14].

There are some differences from existing adversarial robustness work in how we apply these bounds. First, while the robustness literature applies the smoothing results to bound outputs of the classifier itself, we apply smoothing over the trigger set accuracy function to bound changes in trigger set accuracy. Second, we are applying smoothing over parameters as opposed to input. Our trigger set accuracy function $f(X, \theta)$ in general takes in two arguments: X , a set of images, and θ , the model parameters. In the case of adversarial robustness, the model parameters θ are constant after training while the attacker perturbs the image x . But in our case, the trigger set X remains constant and the adversary can only change θ . Therefore, to defend against our specific adversary, we apply smoothing over θ as opposed to X . Since the trigger set X is constant for our case, we simply write the trigger set accuracy function as $f(\theta)$ for the remaining part of the paper.

In our proposed watermark, we use the median smoothed version ($h_{50\%}$) of the trigger set accuracy function for ownership verification. Empirically evaluating $h_{50\%}$ essentially involves adding noise to several copies of the model parameters, calculating trigger set accuracy for all of them, and taking the median trigger set accuracy.

Even though the evaluation process of $h_{50\%}$ is more involved compared to the base trigger set accuracy function, the smoothed version allows us to use Lemma 1 from [13] to bound the worst case change in the trigger set accuracy given bounded change in parameters, as shown in Corollary 1.

Corollary 1.2.2. *Given a measurable trigger set accuracy function $f(\theta)$, the median smoothed trigger set accuracy function $_{50\%}(\theta)$ can be lower bounded as follows*

$$\Phi_{(-\epsilon/\sigma)}(\theta) \leq_{50\%} (\theta + \delta) \quad \forall \|\delta\|_2 < \epsilon, \quad (1.2.2.3)$$

when the adversary does not modify the model parameters θ by more than ϵ in terms of ℓ_2 norm.

Φ is the standard Gaussian CDF.

Using the above corollary, we can then bound the worst case trigger set accuracy given the ϵ adversary by evaluating $\Phi_{(-\epsilon/\sigma)}(x)$. Even though $\Phi_{(-\epsilon/\sigma)}(x)$ does not have a closed form, we can calculate an empirical estimator that would lower bound it with sufficient confidence c .

Trigger set accuracy and model ownership In this paper, we assume a sufficiently high trigger set accuracy implies ownership with high probability. However, there are some scenarios where the assumption does not hold, which we will clarify below. Whether high trigger set accuracy implies ownership depends heavily on the trigger set selected. For example, if the trigger set (X, Y) selected has labels corresponding to what most people would consider to be correct classes, then a model developed independently by someone else would likely classify such trigger sets correctly, leading to incorrect ownership assignment. However, if the trigger set consists of wrong or meaningless labels (such as dog images paired with cat labels), then an independently developed model is very unlikely to classify such trigger sets correctly. In our paper, we assume that the trigger set examples selected have a probability less than random chance of being classified correctly by a random model, and that a sufficiently high trigger set accuracy implies ownership. Our certificate is only focused on proving the preservation of trigger set accuracy when the adversary is allowed to move parameters within a certain ℓ_2 norm ball.

Algorithm 1 Embed Certifiable Watermark

Required: training samples X , trigger set samples $X_{trigger}$, learning rate τ , maximum noise level ϵ , replay count k , noise sample count t

for epoch = 1, ... , N **do**

for $B \subset X$ **do**

$$g_\theta \leftarrow E_{(x,y) \in B} [\nabla_\theta l(x, y, \theta)]$$

$$\theta \leftarrow \theta - \tau g_\theta$$

for $B \subset X_{trigger}$ **do**

$$g_\theta = 0$$

for $i = 1$ to k **do**

$$\sigma \leftarrow \frac{i}{k} \epsilon$$

for $j = 1$ to t **do**

$$G \sim N(0, \sigma^2 I)$$

$$g_\theta \leftarrow g_\theta + E_{(x,y) \in B} [\nabla_\theta l(x, y, \theta + G)]$$

$$g_\theta \leftarrow g_\theta / (kt)$$

$$\theta \leftarrow \theta - \tau g_\theta$$

Embedding the Certifiable Watermark

To embed the watermark during training, we add Gaussian noise and train on the trigger set images with the desired labels. For a given trigger set image, we average gradients across several (in our experiments, 100) draws of noise to better approximate the gradient of the smoothed classifier. Directly adding a large amount of noise into all parameters makes training unstable, so we incrementally increase the levels of noise within each epoch. In our experiments, we inject Gaussian noise with a range of standard deviations σ ranging from 0 to 1. Empirically, we notice that the test accuracy drops when using this technique to embed the watermark, so to recover some of the lost test accuracy, we warm up the model with regular training and only begin embedding the watermark after the fifth epoch. We note that using warm-up epochs to recover clean accuracy is a common practice in the robustness literature [15, 16]. The detailed training method is described in Algorithm 1.

1.3 Experiments

In our first set of experiments, we investigate the strength of our certificate under two datasets and three watermark schemes. In our second set of experiments, we evaluate the watermark’s empirical robustness to removal compared to previous methods that claimed resistance to removal attacks. The code for all these experiments is publically available ¹.

1.3.1 Experimental Settings

To produce the trigger sets themselves, we consider the three schemes from [1]: images with embedded content (superimposed text), images with random noise, or images from an unrelated dataset (CIFAR-10 for MNIST and vice versa) (Figure 1.5). While we generated certificates for all three schemes, we focus on embedded content watermark for empirical persistency evaluation.

To train the watermarked model, we used ResNet-18, SGD with learning rate of .05, momentum of .9, and weight decay of 1e-4. The model is trained for 100 epochs, and the learning rate is divided by 10 every 30 epochs. Only 50% of the data is used for training, since we reserve the other half for the adversary. For our watermark models, we select σ of 1, replay count of 20, and noise sample count of 100. Given these training parameters, embedding the watermark increase the compute time by two times compared to regular training. For certification, we use 10000 instances of Monte Carlo sampling to perform smoothing.

To attack the model, we used Adam with learning rates of .1, .001 or .0001 for 50 epochs.

¹A PyTorch implementation of Certified Watermarks is available at https://github.com/arpitbansal297/Certified_Watermarks

Attack Radius	0.2	0.4	0.6	0.8	1.0	1.2	1.4	1.6	1.8
Worst Case Accuracy	85.8%	82.5%	80.5%	76.2%	67.1%	56.1%	32.0%	18.4%	8.4%

Table 1.1: Attack Radius vs Worst Case Accuracy of the Model. It becomes meaningless to defend against a threat model with a radius larger than 1.8 because these models are indistinguishable from any randomly initialized model.

We test three different types of attacks: finetuning, hard-label distillation, and soft-label distillation. Soft-label distillation takes the probability distribution of the original model as labels, whereas hard-label distillation takes only the label with maximum probability. We always give the adversary the same amount of data as the owner (labeled for finetuning, unlabeled for distillation) to err on the conservative side for our evaluation.

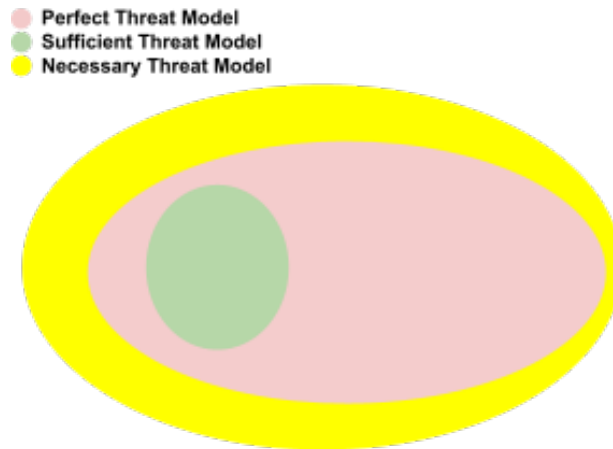


Figure 1.6: Graphical illustration of perfect, sufficient, and necessary threat models.

1.3.2 Properties of a Good Threat Model

Before we present our experimental results for watermark certification, we first briefly discuss properties of a good threat model and what an appropriate radius to certify would be.

In [17], the author defines a *perfect threat model* as being able to capture all items that are similar to the current example. In the case of adversarial attacks, we would like to capture all modified images that are similar to the original image. In the case of watermark removal, we

would like to cover all attacked models that have similar test accuracy as the original model.

However, specifying a perfect threat model is often impossible for an obvious reason: we do not have an oracle measure of human perceptual similarities nor can we easily specify constraints that capture all models with similar test accuracy. As a result, researchers often have to trade off between two different imperfect threat models: a *sufficient threat model* and a *necessary threat model*. The sufficient threat model is a subset of the perfect threat model whereas the necessary threat model is a superset of the perfect threat model as illustrated in the Figure 1.6.

Most prior works prioritize the sufficiency criteria over the necessary criteria since being able to defend against a sufficient threat model is a requirement for being able to defend against the perfect threat model. In the watermark setting, being able to retain the trigger set accuracy (watermark) within the ℓ_2 norm ball of the parameter space is thus a prerequisite for defending against the perfect threat model which would include models outside of the ℓ_2 norm ball.

Appropriate Radius to Certify Here, we analyze the appropriate ℓ_2 norm constraint based on the sufficiency condition and find that a certified radius between 0 and 1.8 is appropriate and that our certified radius is indeed at a similar magnitude. In Table 1.1 on the right, we use a standard PGD attack in the parameter space to gauge how much the accuracy of the model could decrease within the specified radius on the CIFAR-10 dataset. We ran 40 steps of PGD with a learning rate of 0.001, and the parameter gradient is calculated over 2560 examples. If we consider all models with higher than 80% test accuracy to be similar, then an ℓ_2 radius of 0.6 would satisfy the sufficiency condition. This is similar in magnitude to our certified radius presented in the next section, which is between 0.2 to 1.2. The determination of the appropriate radius is still dependent on some subjective judgement on what one would consider to be a well performing

test accuracy. However, what we do know is that the appropriate radius should lie somewhere between 0 and 1.8 – defending against a radius larger than 1.8 is meaningless, as a model with only 8% accuracy on CIFAR-10 is indistinguishable from any randomly initialized model.

1.3.3 Watermark Certificate Evaluation

In this section, we investigate the certified trigger set accuracy that our watermarking is able to guarantee against ℓ_2 adversaries of various strengths. To further contextualize the meaning of a certified ℓ_2 radius, we consider the size of the empirical changes in parameters observed after performing various watermark removal attacks. Finally, we also study how one can increase the certificate by modifying the noise level.

As shown in Table 1.2, we are able to certify trigger set accuracy for radii up to 0.4 for all datasets and watermark schemes considered. This is quite a substantial radius when considering the sufficiency condition, which suggests a meaningful certificate does not exceed a radius of 1.8. Our certificate seems to be similarly effective across all trigger set types. In the best scenario for CIFAR-10, we can certify that the trigger set accuracy does not drop below 51% as long as parameters do not move more than an ℓ_2 distance of 1.

Dataset	Watermark	ℓ_2 Radius (ϵ)					
		0.2	0.4	0.6	0.8	1	1.2
MNIST	Embedded content	100%	95%	47%	3%	0%	0%
MNIST	Noise	100%	91%	7%	0%	0%	0%
MNIST	Unrelated	100%	94%	45%	4%	0%	0%
CIFAR-10	Embedded content	100%	100%	100%	93%	51%	5%
CIFAR-10	Noise	100%	100%	100%	100%	47%	0%
CIFAR-10	Unrelated	100%	100%	100%	97%	35%	0%

Table 1.2: Certified trigger set accuracy at different radius

To see how long our certificates can persist in the face of attack, we measure the approximate

Noise Level (σ)	Test Accuracy	Certified Watermark Accuracy ℓ_2 radius (ϵ)						
		0.2	0.4	0.6	0.8	1	1.2	1.4
1	86.00%	100.00%	100.00%	100.00%	93.00%	51.00%	5.00%	0.00%
1.1	84.56%	100.00%	100.00%	100.00%	97.00%	63.00%	13.00%	0.00%
1.2	84.18%	100.00%	100.00%	100.00%	100.00%	98.00%	74.00%	24.00%

Table 1.3: Trade-off between certified trigger set accuracy and noise level (σ) for CIFAR-10 amount of ℓ_2 parameter change in the first epoch under different attack settings. In Table 1.4, with learning rate 0.0001, parameters change by ℓ_2 distance of approximately 2-3. In other words, it would require approximately 1/3 to 1/2 of an epoch to move outside of a certified radius of 1.

Interestingly, attacks considered to be stronger correspond to changes of a greater distance. This relationship helps support the use of ℓ_2 radius as a proxy for the strength of the adversary. For example, fine-tuning has been found to be a stronger attack compared to hard label distillation, and correspondingly [8], fine-tuning moves the network by a larger distance in the first epoch compared to hard label distillation. Similarly, an attack that is stronger due to a higher learning rate moves the parameters much faster compared to an attack with a lower learning rate.

In Table 1.3, we show that one can obtain larger certificates by increasing the noise level. However, as one makes the model more robust against watermark removal, the model’s test accuracy also decreases. This trade-off is similar to the trade-off observed in the adversarial robustness literature [18]. As the level of noise increases, training also becomes more unstable. For example, using the same hyperparameters as our other experiments, we were unable to train models with $\sigma = 1.5$. However, this is not to say that it is impossible to train a model with $\sigma = 1.5$. We did find an alternative setting where $\sigma = 1.5$ is trainable and offers higher robustness compared to $\sigma = 1.0 - 1.2$. However, since the hyperparameters are not the same, we do not list the results here as we don’t think they are directly comparable.

Attack Type	Finetuning	Distillation Hard Label	Distillation Soft Label	Finetuning	Distillation Hard Label	Distillation Soft Label
Learning Rate	0.0001	0.0001	0.0001	0.001	0.001	0.001
MNIST	2.67	2.39	1.56	19.39	17.58	20.35
CIFAR-10	2.85	2.41	2.06	19.93	19.40	19.29

Table 1.4: ℓ_2 distance change in the first epoch

Dataset	Attack	lr	Baseline Watermark	Black-box Watermark	White-box Watermark
MNIST	Finetuning	0.0001	45.31%	59.38%	100.00%
MNIST	Finetuning	0.001	50.00%	54.70%	100.00%
MNIST	Hard-Label Distillation	0.001	42.19%	50.00%	100.00%
MNIST	Soft-Label Distillation	0.001	96.88%	100.00%	100.00%
CIFAR-10	Finetuning	0.0001	17.20%	9.40%	100.00%
CIFAR-10	Finetuning	0.001	14.06%	10.94%	100.00%
CIFAR-10	Hard-Label Distillation	0.001	29.69%	81.25%	100.00%
CIFAR-10	Soft-Label Distillation	0.001	81.25%	100.00%	100.00%
CIFAR-100	Finetuning	0.0001	18.75%	23.44%	100.00%
CIFAR-100	Finetuning	0.001	0.00%	0.00%	0.00%
CIFAR-100	Hard-Label Distillation	0.001	7.81%	12.5%	5.00%
CIFAR-100	Soft-Label Distillation	0.001	96.88%	96.88%	98.44%
MNIST	Hard-Label Distillation + Reg	0.1	40.63%	32.81%	0.00%
CIFAR-10	Hard-Label Distillation + Reg	0.1	8.00%	27.00%	0.00%
CIFAR-100	Hard-Label Distillation + Reg	0.1	0.00%	0.00%	0.00%

Table 1.5: Trigger set accuracy after 50 epochs of removal attacks. We note that this is only a snapshot of the trigger set accuracy. During training, trigger set accuracies could sometimes fluctuate significantly. We use watermarks from [1] as the baseline watermark.

Overall, it would take approximately 0.03 to 0.3 epochs for the attacker to escape the certified radius, depending on the type of attack, watermark schemes, and dataset. Our certified bounds are substantial when considering the sufficiency criteria, but they are still quite small when compared to a non- ℓ_2 bounded attack. In the next section, we show that even though our certificates are not large when considering the optimization trajectory, the watermarks are empirically stronger than the certificate is able to guarantee: in most cases, our watermarks are more resistant to removal attacks compared to previous methods in both the white-box and black-

box settings.

1.3.4 Empirical Watermark Persistence Evaluation

In this section, we evaluate the persistence of our proposed watermarking methods and the model’s performance on the original dataset. For all experiments in this section, we use the embedded content method to produce the trigger set. We compare our watermark method with the baseline method from [1], which is the same as our watermark method but without noise injection during training.

For persistence evaluation, we focus on two main attacks: the distillation attack and the finetuning attack, as both of these have been shown to be very effective in [8, 9]. In addition, we tested the effect of different learning rates and label smoothing levels, which have also been shown to influence the effectiveness of watermark removal techniques [8]. To make our attacks more similar to [8], we also experimented with adding parameter regularization during attack.

We first evaluate our proposed watermark against finetuning attacks. In Table 1.5, we see that our proposed watermark is much more robust with respect to finetuning attacks than the baseline method on CIFAR-10, and is comparably resistant on MNIST. In the case of CIFAR-10, the baseline watermark is completely removed within less than 10 epochs, but our white-box watermark is still visible after finetuning for up to 50 epochs. In the case of MNIST, both the proposed method and the baseline are quite resistant. However, our proposed method achieves slightly higher trigger set accuracy for both white-box watermarks and black-box watermarks throughout the 50 epochs of the finetuning attack. In the case of CIFAR-100, neither watermark is very resistant to removal. However, our blackbox watermark slightly outperforms the baseline

method.

In the face of the distillation attack, we find our white-box watermark to be extremely resistant. The trigger set accuracy remains 100% even after 50 epochs of attack. However, our black-box watermark works more effectively on CIFAR-10 than MNIST. In the case of CIFAR-10, the black-box watermark remains at 81.25% after 50 epochs of distillation attack, whereas only 50.00% of trigger set accuracy remains for MNIST. In the case of CIFAR-100, our proposed watermark slightly outperforms the baseline method. However, they are both quite susceptible to removal attack.

When regularization is added in addition to distillation, we find that our white-box watermark is completely removed. This could be due to regularization moving the parameters further in terms of ℓ_2 norm. However, we note that our black-box watermark still persists similarly to the baseline.

In some cases, the baseline watermark persists quite strongly. For example, in the case of soft-label distillation, the baseline watermark still achieves higher than 75% accuracy after attack. We tried a variety of settings, but we had difficulty completely removing the watermark as described in [8]. Differences in performance could be due to architecture, regularization, or other factors – experimental code was not released by [8], so it is hard to know exactly what might be the cause. However, we note that our main goal is to show that our proposed watermark is more resistant to removal, and our trigger set accuracy is consistently higher compared to the baseline throughout the attack.

Even though our watermark is generally more resistant in both the white-box and black-box settings, our proposed method does slightly decrease the accuracy of the model on the original dataset. Test accuracies are decreased by 0.1% (from 99.5% to 99.4%), 3.3% (from 89.3% to

86.0%), 1.1% (from 68.28% to 67.23%) for MNIST, CIFAR-10, and CIFAR-100 respectively.

The decrease in clean accuracy has been historically observed for other forms of robust training [18], and recovery of the test accuracy in robust training is still an active area of research [15].

However, it is worth noting that the decrease in accuracy does not scale with the difficulty of the dataset. For example, even though CIFAR-100 is a much more challenging dataset compared to CIFAR-10, we actually observe smaller accuracy decrease for CIFAR-100.

Part III

AI Reasoning

Chapter 2: End-to-end Algorithm Synthesis with Recurrent Networks: Logical Extrapolation Without Overthinking

2.1 Introduction

Humans solve complex logical reasoning problems through *logical extrapolation* – they assemble simple logical primitives into complex strategies. For example, a person taught to prove simple lemmas can in turn prove more complex theorems simply by expending more cognitive effort.

Neural networks have achieved great success at pattern matching tasks, often exceeding human performance, but they struggle to solve complex reasoning tasks in a scalable, algorithmic way. Recently, Deep Thinking systems have been proposed as a way to represent and learn scalable reasoning processes using recurrent neural networks [19]. The word ‘thinking’ in this context refers to sequential processing to solve discrete/logical problems. These systems train recurrent models (networks that recycle parameters between layers) to solve reasoning problems. Unlike traditional feed-forward models, which are limited in the complexity of problems they can solve by their finite depth, the effective depth of recurrent models can be expanded after training simply by iterating the recurrent unit for longer.

When trained properly, thinking systems learn scalable algorithms for solving classes of

problems. After training to solve small/easy problem instances with few recurrent iterations, the algorithm is then extended to run for more iterations at test time. In doing so, the system can achieve algorithmic extrapolation, solving problems of greater difficulty than those in the training set.

To date, the level of algorithmic extrapolation observed in thinking systems has been quite modest. For example it has been demonstrated that a system trained on 9×9 mazes can extrapolate to solve a 13×13 maze. These systems fail to achieve greater extrapolation because of a problem we call *overthinking*; recurrent systems, when extended too far outside their training regime, often deteriorate and fail to produce interpretable outputs.

In this work, we design purpose-built neural architectures and specialized training loops to make it possible to train systems that do not suffer from overthinking and instead converge to a fixed point when iterated for thousands of iterations. By doing so, we are able to build thinking systems that exhibit extreme algorithmic extrapolation behaviors, and leap from solving small/simple training problems with tens of iterations to solving large and complex problem instances at test time using thousands of iterations.

We experiment on benchmark problems for measuring extrapolation behavior. These tasks include computing prefix sums, finding optimal solutions for two-dimensional mazes, and solving chess puzzles [20]. Our architectures and training routines significantly outperform existing methods on all tasks in the benchmark suite. Additionally, we demonstrate that iterative methods for these reasoning problems are susceptible to overthinking, a problem that is overcome by our new architectures and training loops.

The main contributions can be summarized as follows:

- We provide a recurrent architecture for algorithmic extrapolation, in which the problem input is concatenated directly to the feature stack of certain layers in the recurrent thinking module. This prevents the problem instance from being forgotten if deep features become noisy, corrupted, or lossy.
- We develop a new training routine that incentivizes recurrent networks to make incremental improvements towards a solution, improving the feature representation after each iteration. This training process removes information about how many times the recurrent module has been applied, preventing the network from learning iteration-specific behaviors and instead allowing models to learn scalable behaviors that can be iterated indefinitely for extrapolation.
- We analyze the overthinking problem and show that our models overcome this phenomenon. In some cases, the algorithms learned by the proposed networks appear to be capable of solving problems of arbitrary size, despite training only on very small problem instances.

Our improvements in performance on the easy-to-hard benchmark datasets can be categorized in several ways. First, our models yield uniformly higher accuracy across the most difficult tasks used for testing in previous work. Second, we show that our models can extrapolate to much harder/larger examples than are considered in previous work, where the prior methods generalize poorly, if at all. Lastly, we show that our models do not forget solutions in settings where previous models overthink.

2.2 Related work

Preliminary work [19] shows that simple recurrent architectures, when trained to solve various reasoning problems, can exhibit algorithmic extrapolation while their feed-forward counterparts cannot. In this section, we contextualize this approach amongst prior work on algorithm learning, adaptive neural models, and logical extrapolation.

Algorithm learning describes models that learn scalable processes from data. Early works on this topic study the ability of recurrent neural networks (RNNs) to process input strings of arbitrary lengths [21, 22]. More recent work by [23] introduces neural Turing machines designed to mimic programmable computers, and [24] propose a parallel version inspired by massively parallel graphical processing units. These methods, and various improvements to them, show promising results on bit string to bit string tasks, including copying inputs and adding integers, and even demonstrate the ability to generalize from shorter training strings to longer ones at testing [23, 24, 25]. Since they are based on classical RNNs, however, the amount of computation they perform is directly linked to the length of the input string, which prevents them from executing more or less computation independently of the input size (or corresponding to the difficulty of the problem). Moreover, classical RNNs are often trained incrementally to produce one bit at a time, rather than synthesizing an algorithm for solving an entire problem end-to-end. This makes it difficult to apply them to problems where the solution cannot be decomposed into incremental parts (e.g., chess).

Constraint satisfiability problem (CSP) solving networks disentangle the amount of work from the input size [26]. Specifically, message passing neural networks can execute more passes to solve harder CSPs [26]. These systems are specific to the problem of constraint satisfaction

for boolean expressions, but they are an early demonstration of scalable algorithmic behavior.

Adaptive neural networks are designed to expend varying amounts of computation on different inputs, thus overcoming the limitation of classical RNNs. Self delimiting neural networks use one neuron to determine when to stop updating the hidden state in RNNs, and in doing so, they perform more or less computation for each token in an input sequence [27]. Adaptive compute time (ACT) is an algorithm that provides RNNs with a halting unit, which estimates the probability that computation should continue. This algorithm penalizes ‘ponder time’ during training to encourage the network to solve problems quickly [28]. [29] exhibit strong performance on visual question answering by introducing a differentiable version of ACT. Adaptive transformer-based language models also exist, notably Universal Transformers and Depth-Adaptive Transformers, which utilize ACT to determine the work required for each input [30, 31]. Similarly, iterative residual networks like NAIS-Nets repeat blocks in stages and perform well on image classification [32]. All of these works test their methods *in-distribution*, i.e. where the training and testing data are sampled from the same distribution; logical extrapolation outside the training domain is not considered.

Logical extrapolation describes the task of generalizing to test sets which comprise more computationally complex samples than the training data. [33] claims that “neural network models with end-to-end training pipelines ... cannot, on their own, successfully perform algorithmic reasoning,” and instead proposes a hybrid hand-crafted and learned approach. Similarly, [34] propose recurrent relational networks, which operate on graphs by iteratively passing messages. They also claim that classical architectures lack the inductive bias to reason about relationships between objects. Several recent works call these claims into question. [19] employ recurrent networks based on weight-sharing architectures, which can be made deeper at test time independent

of the input size. These systems exhibit logical extrapolation behavior in several domains. More details on these methods are discussed in Section 2.3 as our algorithms build on these directly. [35] reformulate the halting unit in ACT leading to probabilistic RNN models with improved performance called PonderNet, and importantly their method outperforms ACT on logical extrapolation tasks for prefix sums.

The ‘thinking’ systems proposed by [19] depart from classical recurrent networks for text, which learn from step-by-step supervision to produce output tokens one at a time. In contrast, ‘thinking’ systems autonomously synthesize a scalable algorithm end-to-end with no supervision over what each algorithmic step should do. Even more importantly, they can be applied to solve complex problems that are difficult or impossible to decompose by hand. In other words, sequence-to-sequence models, even adaptive RNNs, have a severe limitation in that the problem needs to be represented as a sequential input and that each token in the output is generated by the same function. ‘Thinking’ systems, on the other hand, provide a mechanism for domains where this type of decomposition is unnatural, difficult, or even impossible.

2.3 Methods

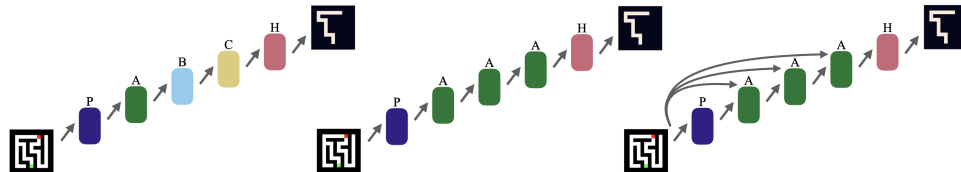


Figure 2.1: Architecture schematics. Left to right: A feed-forward network, a network containing three recurrent blocks (in green) that share weights, and a recurrent network with recall.

We begin with some terms and definitions. We study networks that share weights across blocks of layers during training. For example, instead of three distinct residual blocks, a single

residual block is repeated three times (see Figure 2.1 for a graphical depiction). At test time, networks trained this way can be made “deeper” to extend their compute budget simply by repeating the block more times. We refer to the number of layers applied in a recurrent network as its “depth,” and this quantity grows as the number of recurrent iterations increases. The number of feature maps produced by each layer (or the number of filters in a convolutional layer) is referred to as its “width.”

More formally, let r be a function representing a recurrent block, e.g. a ResNet block [36], and let r^n denote n recurrences of that function, e.g. $r^2(x) = r(r(x))$. Let ϕ denote a feature map, or an output of r , and let $\phi_n = r^n(x)$. We also consider an initial “embedding function” denoted by p , which projects an input instance into feature space, and also a final “output head” denoted by h , which maps features to outputs. A Deep Thinking (DT) network with m iterations of the recurrent block can then be expressed as follows.

$$f(x; m) := h(r^m(p(x))) \quad (2.3.0.1)$$

In our systems p comprises a convolutional layer followed by a ReLU, r is a single four-layer residual block, and h is a set of three convolutional layers with ReLUs after the first and second. We fix m during training and compute gradients for optimization by backpropagating through the unrolled network. Then, m can be increased for testing, allowing these networks to increase their processing power and solve larger and harder problems. Below, we consider two new approaches for achieving improved extrapolation: recall architectures and a modified optimization process.

2.3.1 Recall architectures

When humans think for a long time to solve a problem, we often stop to reread the question or review the task at hand. We improve DT architectures to periodically recall the input exactly. We incorporate this capability into architectures by concatenating the input problem to the features output from each instance of the recurrent block.

Popular architectures in computer vision typically incorporate skip connections that similarly pass information from earlier layers forward. In fact, empirical evidence suggests that skip connections, for example in highway networks, ResNets, and DenseNets, stabilize training [36, 37, 38]. Our architectural modification is driven by the intuition that a noisy training process creates thinking networks that are imperfect and may leak or distort information over time as features are iteratively fed back through the recurrent unit thousands of times. *Recall* allows the system to reproduce any missing or damaged features, and makes it impossible to “forget” the problem being solved.

To delineate the modification to the architecture by incorporating *recall* functionality into the network, we employ the previously defined notation in a concise manner. The function $f_{\text{recall}}(x; m)$ is essentially defined as $h(r_{\text{recall}}^m(p(x), x))$. Here, the term “where” introduces two fundamental definitions: firstly, $r_{\text{recall}}(\phi, x)$ is formulated as $r([\phi, x])$, which combines ϕ and x within a list structure for processing. Secondly, $r_{\text{recall}}^m(\phi, x)$ evolves recursively as r_{recall} acts upon the outcome of its previous iteration, $r_{\text{recall}}^{m-1}(\phi, x)$, combined again with x , ensuring a continuity in the recall mechanism across iterations.

In practical terms, during the k -th iteration, the standard input to the function r would be ϕ_{k-1} , signifying the output from the preceding iteration. However, with the introduction of

recall, this input transforms into $[\phi_{k-1}, x]$, a composite input that merges the original input x with the feature map ϕ_{k-1} generated in the prior cycle. This amalgamation is then processed through an additional convolutional layer tailored to convert the combined input $[\phi_{k-1}, x]$ back into a feature map that retains the same dimensional characteristics as ϕ_{k-1} itself. Networks that adopt this approach of integrating concatenating skip connections are referred to as DT-Recall models, highlighting their enhanced capacity for “remembering” past inputs alongside current information through the network’s iterations.

2.3.2 Promoting forward progress through optimization

We propose a training objective to encourage the system to incrementally make progress from any starting point. We do this by inputting a problem instance and running the recurrent module for some random number of iterations. We then take the output of this process, restart the recurrence in the network with these features as if iterations had just begun (discarding gradients from the initial iterations), and train the model to produce the solution after a random number of additional iterations.

This incremental training process has two benefits. First, it trains the network to continue improving the quality of partial solutions, even when they contain errors or distortions that creep in from running many iterations. Second, by choosing features from a random iteration to serve as the initial state for the training step, we discourage the network from internally counting iterations and learning iteration-specific behaviors, such as behaviors that get executed only on iteration five, for example. Rather, the network is encouraged to learn iteration-agnostic behaviors that are effective at any stage of the problem solving process.

Algorithm 2 Incremental Progress Training Algorithm

Input: parameter vector θ , integer m , weight α

for batch = 1, ... , N **do**

 Sample inputs x and targets y .

 Choose $n \sim U\{0, m - 1\}$ and $k \sim U\{1, m - n\}$

 Compute ϕ_n with n iterations w/o tracking gradients

 Compute $\hat{y}_{\text{prog}}$ with additional k iterations

 Compute $\hat{y}_m$ with new forward pass of m iterations

 Compute $\mathcal{L}_{\text{max_iters}}$ with $\hat{y}_m$ and $\mathcal{L}_{\text{progressive}}$ with $\hat{y}_{\text{prog}}$.

 Compute $\mathcal{L} = (1 - \alpha) \cdot \mathcal{L}_{\text{max_iters}} + \alpha \cdot \mathcal{L}_{\text{progressive}}$

 Compute $\nabla_{\theta}\mathcal{L}$ and update θ

In our implementation, we randomly sample the number of iterations used to generate a partial solution, n , and the number of training iterations, k , budgeted for the network to improve this partial solution. We then update the network’s parameters to minimize loss after $n + k$ total iterations when it starts with the partial solution. This is done by detaching the recurrent module’s output after n iterations from the computation graph before computing the gradient of the loss at iteration $n + k$. The process above is an analog of truncated backpropagation through time [39], with a random starting and end point. During training, we ensure that the sum of n and k is less than a fixed maximum number of iteration m , which we call the *training regime*. The incremental loss described above is added to the standard loss computed with a full forward and backward pass through the unrolled m -iteration network.

2.3.3 Datasets

We evaluate our methods on the benchmark problems available in the Python package `easy-to-hard-data`, which generates reasoning problems of various difficulties. The three problems considered are computing prefix sums, finding the optimal path in a two-dimensional maze, and solving chess puzzles. We briefly review the input and output structures for each

problem and refer the reader to [20] for more detail, including the data generation process. Note that the architectures we consider are fully convolutional, and produce outputs of the same dimension as their inputs. Furthermore, the training and testing datasets we consider have labels of the same dimension as their inputs. Therefore, a network trained on inputs of one size can then trivially be applied to inputs of a different size.

We begin with the toy problem of computing prefix sums modulo two. The inputs and targets for this problem consist of bit strings. The i^{th} bit of the target is the mod two sum of all bits prior to and including the i^{th} bit of the input. We control the “difficulty” of the problem by changing the length of the bit string. Note that computing prefix sums of greater length is known to require a greater number of sequential operations [40]. All of our training is done on 32-bit strings and we explore the behavior of our models on longer strings, even showing strong performance on 512-bits.

The mazes we consider are two-dimensional square images where the walls are black, the permissible paths are white, and the start and end positions are denoted with red and green squares. The targets for this problem are maps of the same dimension as the input maze, but with ones on the optimal path and zeros elsewhere. We make more challenging datasets by increasing the size of the mazes. All training in our work is done on 9×9 mazes. Despite this small training size, we benchmark extrapolation performance on mazes of size 59×59 , and observe models solving mazes as large as 201×201 with high success rates.

The chess puzzles we consider are mid-game boards represented by twelve 8×8 planes indicating the positions of all 12 types of pieces (there are six distinct piece classes and two colors on a chess board). The goal of this task is to find the best next move, and each target encodes this information in an 8×8 array with zeros everywhere except the origin and destination of the

piece to be moved, which are populated with ones. The chess dataset is sorted by difficulty rating, as determined by Elo scores computed via human trials on Lichess.org. Problems are sorted by difficulty and we train on the first 600K easiest puzzles and test our models on indices ranging from 700K to 1.1M.

Our experiments have two categories: (I) Testing extrapolation at extreme leaps in problem difficulty/size and (II) Testing scenarios that emphasize the overthinking trap and how we avoid it. The following two sections describe each of these experiments, respectively.

2.4 Extreme extrapolation

In each problem domain, we show that our models with recall and trained with progressive loss perform best.¹ We compare our models to a baseline of DT models trained with $m = 30$ and problem-specific widths (detailed below), which are both wider and deeper than the models previously studied. However, to provide a fair comparison, we use consistent values for these parameters across our experiments since we find that all architectures benefit from additional width and depth. We also compare to feed-forward models without weight sharing of the same effective depth as a 30-iteration DT network, which are ResNets [36]. We include ablation studies lending credence to our claim that both the architecture and the loss modifications are instrumental in boosting extrapolation power. The code to reproduce all of our experiments is publicly available.²

Prefix Sums Though prior DT systems show extrapolation behavior on strings with up to

¹Note that the DT models presented in this section use a maximum confidence exit rule for fair comparison. This exit rule marginally mitigates overthinking in the regime initially tested by [19]. Models with our new architecture and/or loss do not require this rule and are presented with default outputs.

²A PyTorch implementation of the code is available at github.com/aks2203/deep-thinking.

48 bits, we use a 512-bit testing set to illustrate the extent of our performance improvements over prior work. We train our models on 32-bit strings with a maximum of only $m = 30$ recurrent iterations (indicated in the figures as the shaded *train regime* section), 400 convolutional filters per layer, and incremental loss weight $\alpha = 1$ when progressive loss is used.

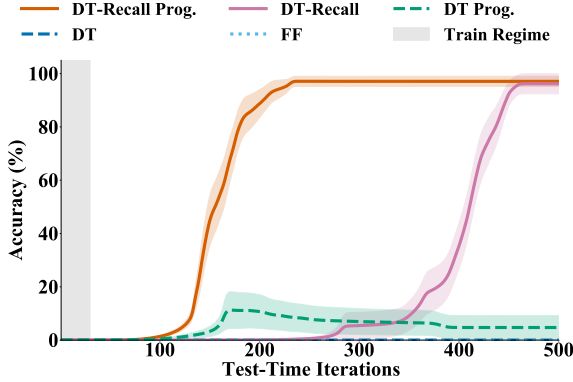


Figure 2.2: **Prefix sum models trained on 32-bit inputs extrapolate to 512-bit data.** The value of our recall and progressive loss is clear by how quickly and accurately our models solve this very large problem.

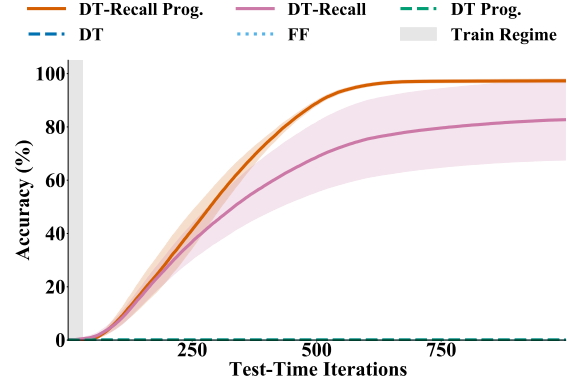


Figure 2.3: **Maze solving models trained on 9×9 inputs extrapolate to 59×59 problems.** Mazes this large cannot be solved without recall, and furthermore progressive loss leads to more accurate models.

In Figure 2.2, we show that without our techniques, DT networks are unable to solve very long binary strings, achieving 0% accuracy on 512-bit data, while DT-Recall models with progressive loss can solve more than 97% after approximately 200 iterations.

To better understand the individual effects of our proposed approach, we perform an ablation study. Figure 2.2 also makes clear that DT-Recall networks trained *without* progressive loss achieve the same final accuracy as models trained *with* it; but they require approximately twice the number of iterations to get there. Because our proposed objective succeeds in making models iteration agnostic, solutions are often found sooner than DT models that are trained to solve 32-bit problems in 30 iterations specifically. DT models trained with progressive loss outperform vanilla DT networks, however neither of these models (without recall) can solve more than a

handful of 512-bit testing examples.

One other study in this domain reveals the importance of randomly setting n in Algorithm 2. To show this, we modify the algorithm slightly, to always use $n = 0$, rather than randomly choosing that value. While this change seems minor, and indeed makes a small impact on the results, it does lower the accuracy. Specifically, DT networks with recall on average solve 90.27% of 512-bit test samples when trained with the $n = 0$ loss, and with randomly sampled values of n , as described in Algorithm 2, they achieve 97.12% accuracy. From this ablation experiment, we conclude that randomly sampling these values is, in fact, beneficial.

Mazes As with the prefix sum problem, we can improve performance on hard mazes by combining incremental training and recall architectures. In particular, while Figure 2.2 may not convince the reader that our proposed loss is critical since DT-Recall models without it perform very well, with more complex data a drastic difference emerges.

We show in Figure 2.3 that on the significantly harder test set of 59×59 mazes, our models exhibit strong algorithmic extrapolation, while previous methods, both feed-forward and DT systems, completely fail. Not only do our models achieve a higher peak accuracy, but they do not overthink, as can be seen by the flat spans in the curves in Figure 2.3. In fact, we can push these systems to their limits (and the limits of our hardware) and find that our models can solve 74% of the 201×201 mazes.³ See Appendices ?? and ?? for examples of 201×201 and 801×801 mazes, which our models solve using 2,000 and 20,000 iterations, respectively (the equivalent of 10,004 and 100,004 convolutional layers in depth).

The best models in Figure 2.3 are DT-Recall models trained with 30 maximum iterations

³The seemingly modest performance of 74% accuracy should be understood in context. Of the 100 mazes in this test set, 20 have one pixel (out of 166,464) wrong and no single maze has more than seven mistakes.

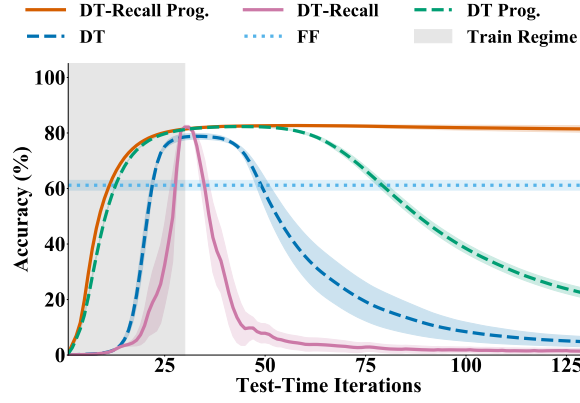


Figure 2.4: **Chess models trained on the first 600K easiest puzzles extrapolate to 600K-700K.** Recall and progressive loss are required to retain accuracy with many iterations.

and a weighting in the loss of $\alpha = 0.01$. In order to see how critical recall is in overcoming the overthinking problem, we show in the same figure that without recall all models fail to extrapolate. The benefit of progressive loss is also highlighted by the fact that DT-Recall models with progressive loss achieve on average 97% accuracy while DT-Recall models without progressive loss only reach an average of 83%. Interestingly, in this domain, we find that the best model has $\alpha = 0.01$ in the loss, a much smaller weight than we use for prefix sums.

Chess Puzzles We further find that using recall and progressive loss yields notable improvement on chess puzzles. In Figure 2.4, we see that our techniques lead to a 4% accuracy improvement compared to networks without them. We show that either removing recall or training without the incremental progress loss will hurt performance. Our best models are DT-Recall networks with 512 convolutional filters in each layer trained with a maximum of 30 iterations and a weight of $\alpha = 0.5$. Moreover, their accuracy is preserved as the number of iterations increases, while that of the other DT networks decays seriously after about 70 iterations – the sign of overthinking and the subject of the following section.

2.5 The overthinking problem

Deep Thinking networks boast impressive capabilities to solve harder problems by thinking deeper, but they are prone to overthinking. In particular, by testing models on data closer in complexity to the training data we can better compare our methods to prior work. Figure 2.5 shows that some recurrent networks that can perform algorithmic extrapolation may collapse entirely when performing too many iterations.

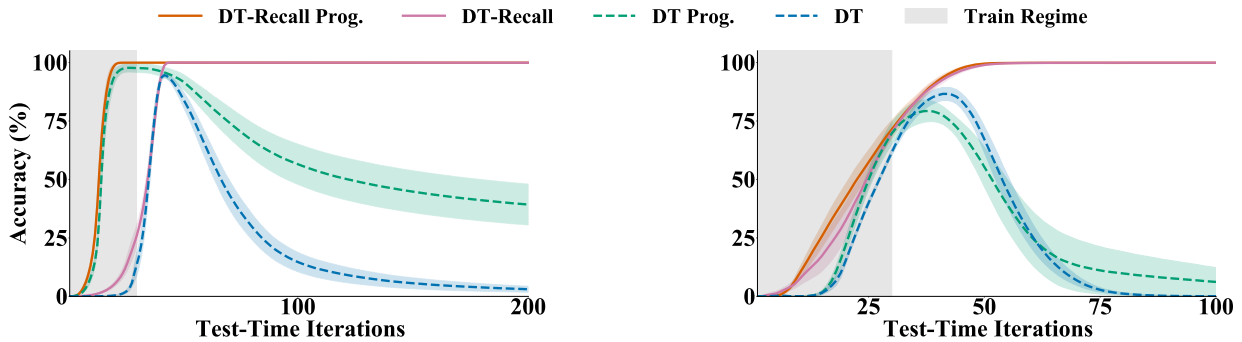


Figure 2.5: **Left:** Prefix sum models trained on 32-bits tested on 48-bit inputs. **Right:** Maze models trained on 9×9 mazes tested on 13×13 mazes. In these regimes, we can see that the weaker models can extrapolate to slightly harder problems, but importantly only models with recall avoid the overthinking trap.

Our models do not suffer from the sharp decline in accuracy as the number of iterations increases. In Figure 2.5, the decay in each dashed curve and the stability with high iteration counts of the solid curves shows that recall is critical to avoid overthinking. See Figure 2.4 for an example where progressive loss is needed too. The monotonic increase in accuracy with added iterations in our models is practically useful, as it allows for pre-defining a large iteration number at which to terminate the recurrence rather than having to carefully choose a stopping iteration to maximize performance while avoiding degradation.

One observation that can be made from these results is that the overthinking problem often seems to disappear when skip connections are added to provide networks with an uninterrupted

view of the input. Another way to describe this is that our models seem to converge to a fixed-point solution as they iterate rather than becoming unstable. This property is desirable and using representative models trained to solve mazes, we explore how robust models are when we manipulate the features during the thinking process.

2.5.1 Manipulating feature maps

First, we investigate sensitivity to adding noise in the feature maps before concatenating the inputs. We examine model behavior when we add Gaussian noise (with mean zero and standard deviation one) to the features after one iteration of maze solving. Models with recall can still solve 13×13 mazes even when we perturb the features, but models without recall cannot.

Next, we ask whether the initial feature maps (after the projection layer p) carry any important information. We test this by replacing the feature maps with zeros after 50 iterations – at this point the model has solved the maze and by annihilating the features we remove all information from p . In this case, our DT-recall models naturally regenerate their features and recover to solve the problem again, indicating that the learned algorithm is able to find a solution using the input without the initial projection.

2.5.2 Manipulating inputs

With the notion that our models may embody a convergent process, we turn our attention to investigating how the networks determine when to stop manipulating the representation. Perhaps there is something in the feature maps output by late iterations that tells the network to stop working. We can test this in two ways. We perturb the input (which we concatenate onto the

features) after some number of iterations – first subtly, then by swapping it with an entirely different example.

To start, we flip single bits in the input string for prefix sums. We explore the response of the model to flipping bits at different indices after 50 iterations (when the models have already solved the initial 48-bit problem). The left panel of Figure 2.7 shows recovery time as a function of index of the flipped bit. We observe that our models can indeed recover from single bit flips, and recovery time decays linearly with index. Higher indices are closer to the end and affect fewer bits in the output than lower ones.

Similarly, with mazes, we change the input by moving the end of the maze two steps closer to the start. We use this new input concatenated with features generated after 50 iterations of solving the original maze. In this case, we see in Figure 2.6 that models with recall self-correct and solve the new problem in very few iterations.

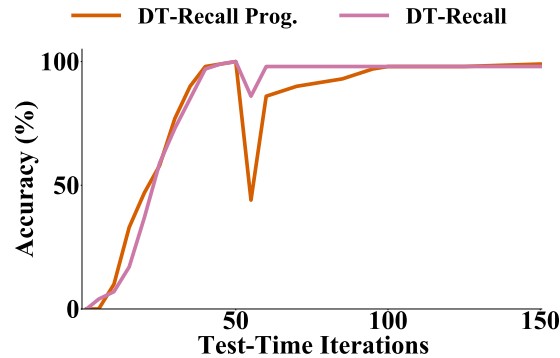


Figure 2.6: Maze solving model performance when the start and end of the maze are changed to be closer together after 50 iterations. DT models with recall can self-correct.

The last way we test the hypothesis that networks are continually comparing their solution to the problem instance is by partly solving Maze “A,” and then swapping the features with those obtained from 50 iterations of trying to solve Maze “B.” In other words: if we concatenate the input problem (A) with the features from iteration 50 corresponding to a different maze (B),

which maze will the network solve? Clearly, a system without recall will solve maze B. However, with recall, networks will recover and pull the features back to representing a solution for maze A. The center panel of Figure 2.7 shows the effect of swapping feature maps.

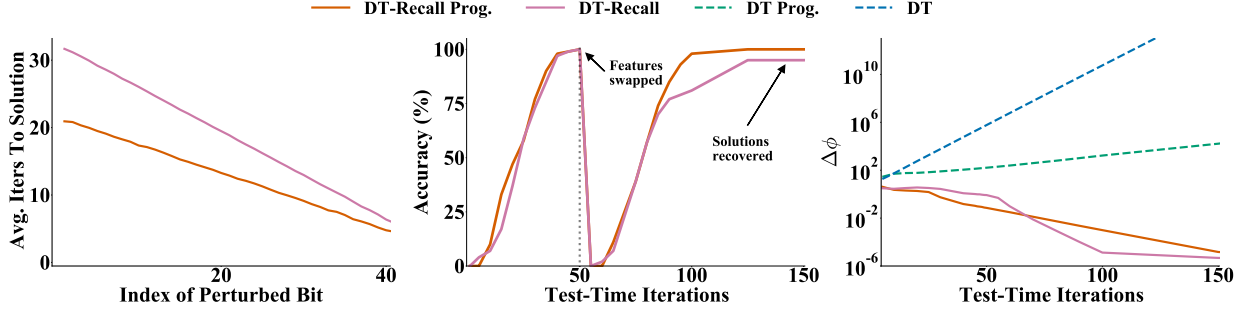


Figure 2.7: **Left:** How long it takes prefix sum models to recover from perturbation – recall keeps this quantity small. **Center:** Test accuracy on 13×13 mazes when features are swapped after 50 iterations. **Right:** The change in the features when solving 13×13 mazes.

2.5.3 Converging to a fixed point

We can also study the convergence by measuring the change in the output at each iteration. A decreasing change in the feature maps with each additional iteration suggests that the network manipulates the representation, moving it closer to one that solves the problem, and that it will hold onto this representation (or stay nearby) once it is reached. Moreover, we seek to qualitatively categorize each model type (architecture/loss pair) as convergent or non-convergent. To do we measure the change in the solution at each iteration with ℓ_2 -norm of the difference. The right-hand panel of Figure 2.7 shows $\Delta\phi(n) := \|\phi_n - \phi_{n-1}\|_{\ell_2}$. We see that our models appear to converge, while DT networks without recall explode, providing another view into overthinking.

Chapter 3: Transformers Can Do Arithmetic with the Right Embeddings

3.1 Introduction

Much of the recent work on Large Language Models (LLMs) focuses on their ability to solve problems in natural language and code generation. Despite progress in these domains, transformers still struggle to perform complex multi-step and algorithmic reasoning tasks in a zero shot setting without resorting to tool use. To study algorithmic reasoning in a sterile laboratory setting, the academic community focuses on simple arithmetic test problems like addition. Addition is simple enough that modest-sized LLMs can (in principle) be trained from scratch to do it without running into capacity and training budget limitations, yet complex enough that even large industrial models fail on large numbers without a code interpreter [41].

Training transformers for arithmetic enables us to study several important questions. First, we ask what architectural design choices, dataset characteristics, and training pipeline variants are required to learn a many-step reasoning process like multi-digit addition? Going deeper, we then investigate whether these models are capable of *logical extrapolation*—can they solve problems of greater size and difficulty than those that appear in their training set?

Prior studies indicate that addition is hard for transformers [42, 43, 44, 45]. Our experiments indicate that this difficulty stems from their inability to clearly represent the exact position of a digit within a long sequence of digits. To address this problem, we propose a simple modification

to the data representation that directly addresses this shortcoming. Our *Abacus Embeddings* are simple learned positional embeddings that are used to encode positions within each span of numerical tokens. Combining Abacus Embeddings and standard positional embeddings, we observe dramatic improvements in accuracy such that models trained with at most 20 digit operands can generalize to problems with 120 digit operands. This represents a state-of-the-art generalization factor of $6\times$, with the previous state of the art being only $2.5\times$. To the best of our knowledge, these are the longest sequences on which learned addition has ever been demonstrated.

We also study several other methods of improving arithmetic and generalization in transformers. We find that incorporating *input injection*—skip connections inserted between the input layer and each decoder layer—can reduce generalization errors by 50% over the Abacus Embedding baseline. We also find that together with our embeddings looped transformer architectures, which contain recurrent layers in which the same parameters are re-used multiple times, can achieve near-perfect generalization on addition problems we consider.

Since our proposed methods solve large addition problems successfully, we evaluate whether the same approaches can be used to improve other kinds of algorithmic learning. We explore multiplication problems of up to 15 digit numbers and sorting over arrays of up to 10 numbers, making this the first study of extreme length generalization techniques for addition that transfer to other algorithmic tasks. Our contributions can be summarized as follows.

- We propose a new positional embedding called *Abacus Embeddings* to better capture the significance of each digit, which leads to near-perfect in-distribution generalization.
- We show that when we combine Abacus Embeddings with input injection and looped

transformers performance further improves, increasing from 92.9% to 99.1% in out of distribution accuracy, an 87% reduction in error compared to using the embeddings with standard architectures alone.

- We push length generalization beyond existing work and show that our models can solve problems with six times as many digits as the largest samples in the training set, whereas the previous state of the art is only two and a half times.
- We extend our findings to more complex problems including multiplication and sorting where we show length generalization in these domains.

3.2 Related Work

Arithmetic and Algorithmic Reasoning. Solving arithmetic with next token prediction is a difficult problem that attracts a lot of attention [e.g. 46]. However, in zero-shot settings, even incredibly strong commercial API models struggle with very large addition problems (e.g. up to 100 digits) without access to tools. Among attempts to improve arithmetic performance of transformer-based models, reversing the digits so the arguments are written with the least significant digit first is popular [42, 43, 44, 45]. Furthermore, changing the data format by adding explicit index characters improves model capability for addition [44, 45, 47]. Other work approaches arithmetic by embedding real numbers by scaling a single fixed token-embedding for numbers [48]. Moreover, Dziri et al. [49] show multiplication is a hard problem for GPT-3 [50] even when finetuned on this task. Dziri et al. [49] further show that GPT-4 [51] struggles to obtain high in-distribution accuracy on multiplication, even with a scratchpad. However, Lee et al. [42] find that with a detailed scratchpad, small transformers can perform multiplication

in-distribution.

Arithmetic is a subset of the larger class of algorithmic reasoning problems that focus on the ability to learn and execute algorithms and generalize to longer problems [52, 53, 54, 55, 56]. The more general algorithmic reasoning field includes work on various architectures and data modalities aimed at learning algorithms from data. Veličković et al. [55] and Rodionov and Prokhorenkova [56], for example, train neural networks to execute specific algorithmic tasks by training on input-output pairs as well as intermediate steps and hints. In a similar vein and although initially appreciated for efficiency, weight sharing and recurrence can be used to make models adaptive and help generalize to harder problems [30, 57, 58, 59]. Schwarzschild et al. [60] and Bansal et al. [61] explore an end-to-end learning approach using recurrent convolutional neural networks to learn algorithms from input-output pairs, tackling algorithmic tasks like prefix sums, mazes, and chess. Weight sharing for algorithmic reasoning is also helpful with transformers and we use the *looped transformer* in some of our experiments below. A looped transformer has a transformer block called recurrently on its own output lending itself to executing iterative algorithms [62, 63, 64]. Additionally, recent work aims to improve reasoning in LLMs [44], but McLeish et al. [65] demonstrate that LLMs, even with code interpreters, are less than perfect at algorithmic reasoning tasks, indicating a crucial need for advancements in our methodologies. This paper takes a step towards improving LLM arithmetic and algorithmic capabilities without tool use.

Positional Embeddings. Indicating the position of tokens in a sequence to transformer models is critical for language modeling [66]. Absolute positional embeddings (APE) are learned embeddings that are added to token embeddings before the first layer of the transformer [66]. However, these

Least Significant Digit First:	1	2	3	4	+	1	2	3	4	=	2	4	6	8
Most Significant Digit First:	4	3	2	1	+	4	3	2	1	=	8	6	4	2
Abacus Embeddings:	1	2	3	4	0	1	2	3	4	0	1	2	3	4
Absolute Embeddings:	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Figure 3.1: Visualization of data formats and positional embeddings. *Abacus Embeddings* give the same positional embeddings to all digits of the same significance.

absolute embeddings inhibit length generalization [67]. To address this issue, Shaw et al. [68] propose relative embeddings (RPE) which are embedded during the attention computation, a mechanism further simplified by Raffel et al. [69]. Others build on these works to improve length generalization including Sandwich [70], Kerple [71], and Alibi [67] positional embeddings. Additionally, Kazemnejad et al. [72] show that decoder layers can still learn positional information with no explicit positional embeddings. No positional embeddings (NoPE) can achieve good length generalization performance for small algorithmic tasks and even outperform some specialized embeddings. Rotary Positional Embeddings (RoPE) [73] are commonly used in state-of-the-art open source transformers [e.g. 74]. However, RoPE does limit the length generalization as models are trained only using rotations based on training data length [67, 72]. For improved length generalization, one can add post-training extensions [75]. The latest and most useful for arithmetic is Functional Interpolation for Relative Position Embeddings (FIRE) [76]. FIRE shows the strongest length generalization to date, which leads to length generalization by $2.5\times$ on addition [45] when combined with randomized embeddings [77]. In this work, we focus on NoPE and FIRE embeddings since these are the best performers for addition in reversed format among existing embeddings [45].

3.3 Achieving Length Generalization for Addition

We study a range of methods for improving the arithmetic capabilities of language models trained from scratch centering on two main hypotheses: (1) the positional information for individual digits within numbers is being lost and (2) recurrence can improve the reasoning abilities of transformer architectures on multi-step arithmetic reasoning problems. We briefly discuss the training and evaluation setup before describing each of our improvements in detail.

Experimental Setup. We train decoder-only causal language models to solve addition problems. Following prior work [42, 43, 44, 45, 72], inputs are formatted least significant digit first, e.g. $98282 + 3859172 = 2787472$. Unlike prior work, we do not add any padding between digits [43] and do not pad any numbers with zeros, neither in the case of carry digits [45], nor to make all operands the same length [43]. We train on all combinations of operand lengths less than or equal to i and j where i and j are the maximum lengths of the first and second operands, respectively. For this study all training sets have 20 million samples and $i = j$, hence we can use one number to define the dataset i , where i is the maximum length of either operand. We sample data with replacement and we stratify the data, so that all length pairs (i, j) are equally sampled during training. To facilitate training of many models from scratch, we use a language model cramming setup [78] and limit each training run to 8 exaFLOP of compute (a single Nvidia RTX4000 GPU for 24 hours); for multiplication results we allow 64 exaFLOP (eight Nvidia RTX4000 GPUs for 24 hours). During training, we mask the input question and only compute loss on the answer digits. For further details on data construction and training we refer to Appendix ??.

We report model accuracy for each (i, j) length pair and unlike most existing work, we also include accuracy for pairs where $i \neq j$ to highlight all instances of extrapolation. This extensive

tabulation is costly and makes inference the main computational burden of this study. Since our training pipeline produces fairly consistent results, we report the mean over three runs (rather than using a best-of-ten reporting scheme [45]). We measure accuracy in the strict sense where only exact matches of all output digits are counted as correct, i.e. if a single digit is incorrect then the example is marked as wrong and we refer to this as *exact match accuracy*. We have the following three evaluation categories: (i) in distribution (ID) where the models are tested on problems up to the maximum size seen during training; (ii) out of distribution (OOD) where the models are tested on problems greater than the maximum size seen during training but both operands are at most 100 digits; (iii) and extreme out of distribution (100+ digit OOD) where the models are tested on problems where both operands are of the same length and are both more than 100 digits and less than 160 digits. In the 100+ OOD setting, we only analyze problems where the operands are the same length ($i = j$) due to inference costs at this scale.

We consider two standard transformer architectures. First, we use a standard autoregressive transformer model where multiple decoder layers are stacked in a feedforward manner. Second, we enhance this standard transformer model by incorporating *input injection*, where the embedded inputs are added to the input of each decoder layer [61, 79, 80]. We visually describe the architectures in the Appendix Figure ??.

3.3.1 Abacus Embeddings Help Align Digits

From prior work and our own initial experiments, we observe that even when input numbers are presented least-significant digit first and training data is stratified and abundant (several million examples), standard transformers struggle to learn multi-digit addition. We also observe

that humans do long addition by first aligning the digits of the same significance into columns. Thus, our first hypothesis is that the significance of each digit (i.e. each digit’s position relative to the beginning of the number) is not easy for transformers to represent, and that this sub-problem presents more of a hurdle than the actual addition itself.

Prior work addresses this by proposing explicit index hints in the inputs and outputs of the addition, for example $a6b7c5 + a1b6c3 = a7b3c9$, finding that transformers perform much better on addition with the information provided by such hints [44, 45]. However, index hints of this form increase the input context length required and *double* the output length and inference cost of solving a given addition problem. Furthermore, Zhou et al. [45] find that the ability of models trained with index hints to generalize is sensitive to the particular random initialization. Zhou et al. [45] highlight this by training models with different random seeds, varying weight initialization and data input order seeds, showing the variance in the performance of these models can vary from near perfect on 100 digit addition to 0% accuracy at 90 digit addition.

To address the limitations of transformers at representing positional information, we design a specially built positional embedding that encodes the location of each digit relative to the start of the current number. We call this *Abacus Embeddings*. We apply the same positional embedding to all digits of the same significance, providing an explicit signal that the model can use to align digits. We visually describe these embeddings in Figure 3.1.

We take inspiration from *Randomized Embeddings* [77] but instead of using random ascending indices to represent positions in a sample, we use consecutive ascending indices with a random starting position to allow for length generalization. Specifically, during training we give consecutive positional embeddings to each digit in a number, starting from a randomly chosen offset value from $U[1, k]$, where k is a hyperparameter. Unless otherwise stated the default value for k in this

study is 100. For example, if the input is 123, the positional encodings are $\beta, \beta + 1, \beta + 2$ where $\beta \sim U[1, 100]$, which are then passed through a learned embedding matrix. The value sampled from $U[1, k]$ is the same for all numbers in a batch, meaning all digits of the same significance obtain the same positional embedding. This training scheme allows the model to see a wide range of positional embeddings, even when training sequences are short. At test time, each positional embedding begins from one, i.e. $\beta = 1$.

Abacus Embeddings Solve Addition. Abacus Embeddings improve generalization performance up to 100 digits and beyond for standard transformer architectures. In Figure 3.2 (left), we highlight the comparative boost Abacus Embeddings have over standard transformer architectures and embeddings for performing addition, taking the mean accuracy of three models in all cases. The accuracy results for the standard transformer models trained with FIRE and Abacus, tested both in-domain (ID) and out-of-domain (OOD), are also shown in Figure ???. Additionally, in Appendix ??, we present similar 2D grid plots for several other experiments that are depicted as bar charts in the main text. Zhou et al. [45] find that operand lengths of up to forty digits are required during training for good generalization to 100 digit addition during testing (albeit not robustly). We find that with our Abacus Embeddings, we can achieve similar accuracy and larger extrapolation using a standard model with input injection trained on maximum operand sizes of 20 digits.

As Abacus Embeddings are a variant of absolute positional embeddings, technically they cannot generalize beyond the relative positions seen during training. However the hyperparameter k that randomizes the starting offset used for each individual addition example can be increased to enable generalization by training a larger range of embeddings for a given computational budget.

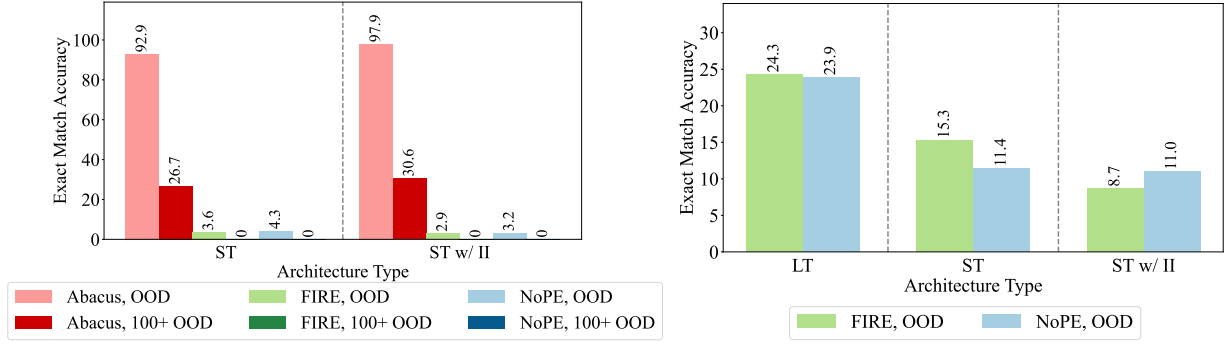


Figure 3.2: **Left:** Mean exact match accuracy of three models of depth sixteen on size 20 data, varying the architecture and embeddings. Abacus Embeddings improve accuracy for addition over FIRE and NoPE Embeddings. **Right:** Mean exact match accuracy of three models of effective depth sixteen on size 40 data, varying over NoPE or FIRE embeddings and architectures. Recurrent looped transformer models improve accuracy for addition for both the FIRE and NoPE embeddings.

Looped transformer (LT): Weight tied decoder layers, with input injection and progressive loss. *Standard Transformer (ST):* Stacked decoder only layers. *Standard Transformer with Input Injection (ST w/ II):* Standard Transformer with input features added to the hidden representation between each decoder layer.

Relatedly, Appendix Figure ?? shows that training on larger datasets improves performance, even for operands with fewer than 100 digits.

3.3.2 Recurrence In Transformers Boosts Performance

With positional embeddings addressed, next we explore whether recurrent architectures can further improve the ability of transformers to perform multi-digit addition. We use the term *recurrent block* to refer to a set of decoder layers with distinct weights and *recurrences* to refer to the number of times the recurrent block is repeated. We use the term *effective depth* to mean the number of layers used in a transformer, whether their weights are unique or not. Unless otherwise stated, we use a maximally recurrent architecture, i.e. only one unique layer recurred to achieve the effective depth. We also employ input injection, skip-connections that propagate a copy of the input to each layer in the network.

The Benefits of Recurrence. In Figure 3.2 (right), we compare all architecture variants using both FIRE and NoPE embeddings trained on addition over operands with up to 40 digits. Despite having approximately $10\times$ fewer parameters than the other models, we see that the looped transformer (recurrent, with input injection and progressive loss), achieves the best out of distribution performance using either position embedding. In Figure ?? in the Appendix, we show this result is robust across multiple training data sizes.

With recurrent models, we can choose to vary the number of recurrences for each forward pass while training. This tends to improve generalization to harder tasks at test time and is also referred to as *progressive loss* computation [61]. This loss function is a convex combination of the loss values from two forward passes, one with the nominal number of recurrences (so 16 for a 1×16 model) and one with a random smaller number of recurrences.

Next, we explore the effect of varying the size of the recurrent block while keeping the effective depth fixed. We perform this ablation by halving the number of layers in the recurrent block and doubling the number of recurrences, sweeping from a model with sixteen layers in the block and a single recurrence (16×1 , i.e. a standard transformer), through to one layer in the block but with sixteen recurrences (1×16). Analyzing these results in Figure 3.3, we show further performance improvements are possible in some cases with the combination of both recurrence and Abacus Embeddings. In particular, a model with two recurrences (8×2) incurs half the error of the purely non-recurrent model (16×1) for OOD problems and enjoys increased accuracy on 100+ OOD problems.

Although the experiments presented in Figure 3.3 are a fair comparison across depth, the purely standard transformer models have many more parameters than their recurrent counterparts. In Table ?? in the appendix, we record the parameter counts to the nearest million.

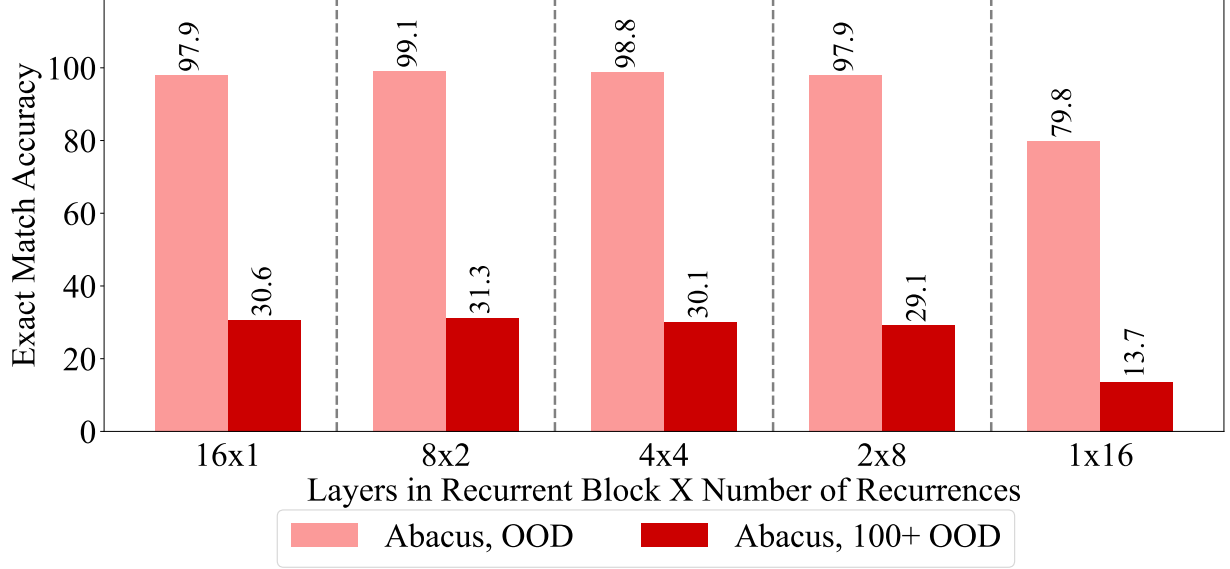


Figure 3.3: Varying the size of the recurrent block, while maintaining an effective depth of 16 and training on size 20 data. We see that a recurrent model with eight layers in the recurrent block and two recurrences is the most accurate of all effective depth 16 models, halving the error rate of a standard model with input injection in the OOD evaluation. (See Figure ?? for results with FIRE and NoPE.)

3.4 Pushing the Limits of Algorithmic Reasoning for Transformers

While there is an emphasis on addition as a difficult problem in existing work, our methods perform so well that we look beyond addition and apply our tools to even more difficult problems, including multiplication and sorting.

3.4.1 Integer Multiplication

We now study a harder task, multiplication of natural numbers, where the length of the output may be the sum of the lengths of the operands. Compared to addition, where the output is at most one digit more than the longest operand, multiplication has longer-distance dependency and the output length scales much faster as problem size increases.

To adapt from addition to multiplication, we make some small changes to our set-up. First, we remove the input injection from inside the recurrent block and second, we divide the gradients in the recurrent block by the number of recurrences, down-weighting the gradient update from batches with many recurrences [61]. (We analyze the impact of these design decisions for addition models in Appendix Figure ??.) We only examine looped transformers as the compute required for training and hyperparameter search for multiplication is far greater than for addition, limiting us to a much smaller scale analysis.

Abacus Embeddings help looped transformers reach near-perfect accuracy in-distribution for multiplication. In Figure 3.4, we show how the training distribution, surrounded by the red square fully saturates with Abacus Embeddings. In fact, models with our Abacus Embeddings achieve higher in distribution accuracy on 15 digit multiplication than prior work [43] and do not require padding each operand to the same length with zeros. In particular, we highlight that the specific problems that models trained with FIRE embeddings struggle to solve are the hardest problems in the training set and Abacus Embeddings outperform them in this key area (see the lower right corner of the red boxes in Figure 3.4).

3.4.2 Array Sorting

Table 3.1: Exact match accuracy for sorting with various positional embeddings. All results are percentages of the test set.

	FIRE	Abacus	Abacus + FIRE
OOD (number length - 30)	55.32	68.63	67.28
OOD (array length - 30)	21.35	9.67	21.11
All OOD (30×30)	3.73	2.65	4.48
All OOD (20×20)	14.65	9.78	16.91

Table 3.2: Accuracy for sorting with various architectures. ST denotes standard transformer, ST w/ II denotes standard transformer with input injection, and LT denotes looped transformer models. All results are percentages of the test set.

	ST	ST w/ II	LT
All OOD (exact string match)	4.48	3.84	2.60
All OOD (min. elem. only)	49.73	60.09	68.51

While both addition and multiplication accept only two operands, we now analyze the task of sorting arrays of multiple variable length numbers, a more challenging testbed for evaluating the generalization abilities of our Abacus Embeddings. We present each sorting problem using alphabetical indices for each (reversed) number in an input array where the expected output is the alphabetical indices in ascending order. For example, $a : 64957, b : 99963, c : 10218, d : 7141, e : 05781 = d, e, b, a, c$. We train with arrays of up to 10 numbers each having up to 10 digits and then evaluate with arrays of up to 30 numbers each having up to 30 digits. We give more detail on the sorting data construction process in Appendix ??.

In this setting, we explore two axes of generalization. First, we increase the maximum

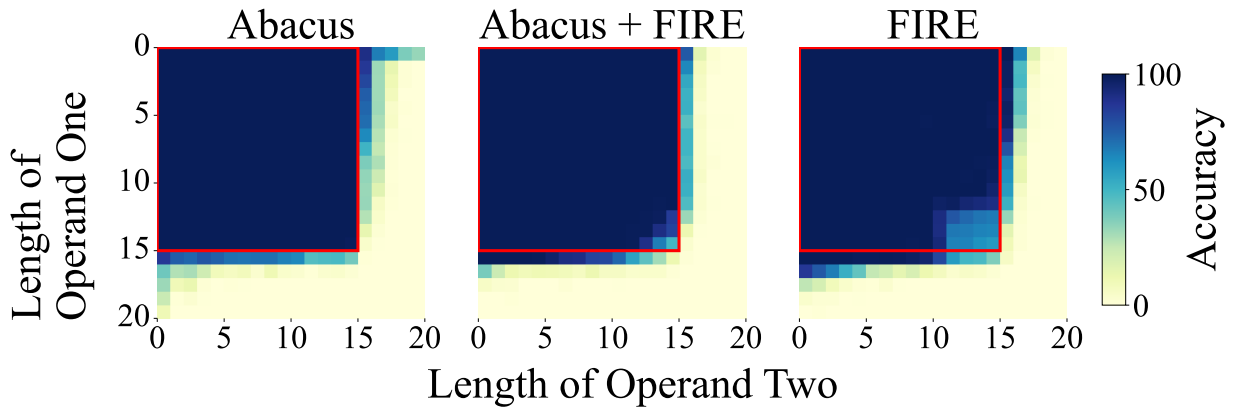


Figure 3.4: Exact match accuracy of looped transformer models trained on multiplication, with four layers in the recurrent block and four recurrences. The red square denotes in distribution testing on up to 15 digit operands. We see the models with Abacus Embeddings achieve near perfect in distribution accuracy. Combining Abacus Embeddings with FIRE also improves in distribution accuracy on the hardest in distribution problems (bottom right), comparing to the FIRE-only baseline.

possible length of the input numbers to 30 digits while maintaining the maximum array length to 10 and refer to this scenario as “OOD (number length - 30).” Second, we increase the number of inputs in the array to be sorted to 30 while keeping the maximum digit length of each number at 10 and term this scenario “OOD (array length - 30).” Finally, we consider a scenario where both axes are increased simultaneously, referred to as “all OOD.”

In Table 3.1, we illustrate the performance of a standard transformer (eight layers) trained with different embeddings—FIRE, Abacus, and their combination. Again, our results demonstrate that the combined embedding approach enhances the model’s ability to generalize, surpassing the performance of either embedding alone in the “all OOD” setting. However, in Table 3.2, we observe mixed results when pairing the Abacus+FIRE Embeddings combination with different model architectures with effective depth eight. For sorting, different architectures appear to be better suited to different types of extrapolation, for example the looped transformer is best at extrapolating for finding the minimum element but not for sorting the whole array.

Overall, the superior sorting performance of the Abacus Embeddings underscores their potential utility across a broader spectrum of algorithmic tasks beyond basic arithmetic. Abacus Embeddings may be instrumental in use cases requiring transformer models to perform a variety of complex positional, numerical, and/or relational reasoning tasks.

3.4.3 Abacus and Relative Embeddings

As Abacus Embeddings are only applied to numbers, to incorporate Abacus Embeddings into a general purpose model, they must be compatible with other relative embeddings to maintain good downstream performance on non-arithmetic tasks. We examine these types of combinations

here and conclude that Abacus Embeddings complement techniques that are good for natural language well, suggesting that these combinations could be powerful for large-scale general models.

Although Abacus Embeddings are implicitly combined with NoPE (no positional embeddings) embeddings for all experiments seen so far, most state-of-the-art open source models use Rotary Embeddings. Rotary Embeddings are weak for length generalization. We show that combining Abacus Embeddings with RoPE does, in fact, yield improvement in operand length generalization. However, in Figure 3.5, we demonstrate the true potential for integrating Abacus Embeddings into a more general system, showing that the combination of Abacus Embeddings with FIRE unlocks generalization well beyond the problems that FIRE embeddings can solve on their own.

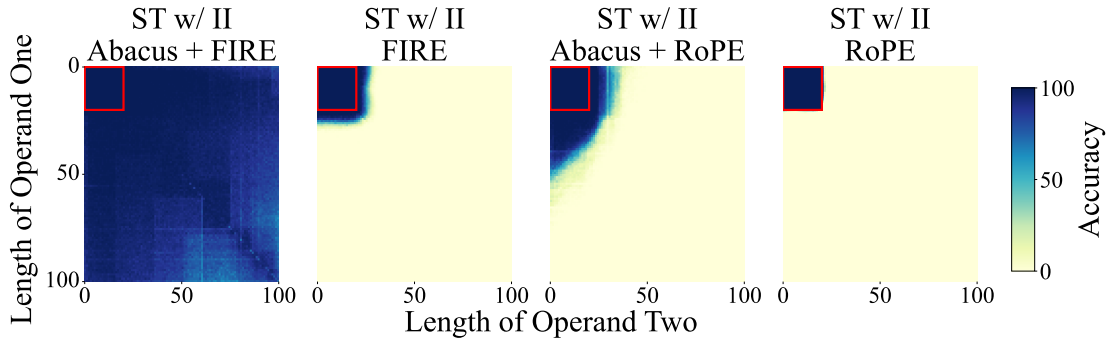


Figure 3.5: Exact match accuracy of standard transformer of depth 16 with input injection, trained on up to size 20 data. The red square denotes in distribution testing. Combining Abacus Embeddings with FIRE or RoPE embeddings improves out of distribution accuracy for addition, over the baseline models without Abacus Embeddings.

3.5 Discussion & Limitations

While the capabilities of LLMs have advanced far enough to encompass complex tasks including code generation and mathematical reasoning, stress testing the limits of these models remains a challenge. In this paper, we study mathematical reasoning tasks including addition,

multiplication, and sorting to evaluate these capabilities in a controlled setting. We analyze the ability of specialized language models to learn algorithmic tasks in a zero shot setting, without access to outside tools like code interpreters, etc., exploring the benefits of various architectural improvements like improved embeddings and recurrent layers.

Across our experiments, we find that our novel Abacus Embeddings improve performance dramatically both when applied to standard transformers as well as recurrent variants. We repeatedly achieve length generalizations of at least $6\times$ (capped by the context length) more than doubling the extrapolation demonstrations in prior work, achieving near perfect results on addition of up to 100 digits, with repeatable results across multiple training runs. We demonstrate the complementary properties of our Abacus Embeddings with other relative embeddings like FIRE, achieving dramatic improvements in in-distribution multiplication performance, and making headway on the challenging problem of variable length array sorting.

Contrasting with prior work, our experiments explore types of extrapolation well beyond just length generalization for addition, presenting an architecture modification that improves performance on multiple algorithmic reasoning tasks simultaneously. We hope that our work deepens the community’s understanding of these problems and paves the way for further advancements in the algorithmic reasoning capabilities of large language models.

Limitations There are some intrinsic limitations that accompany any study involving language model training from scratch under compute constraints. However, the primary point of relevance for this study is that although we show the compatibility of Abacus Embeddings with FIRE and RoPE embeddings, we do not actually explore any natural language tasks. In the future, a larger scale study including natural language would be needed to understand further how

Abacus Embeddings would perform on heterogeneous tasks comprising both numerical and natural language inputs.

Part IV

AI Generation

Chapter 4: Cold Diffusion: Inverting Arbitrary Image Transforms Without Noise

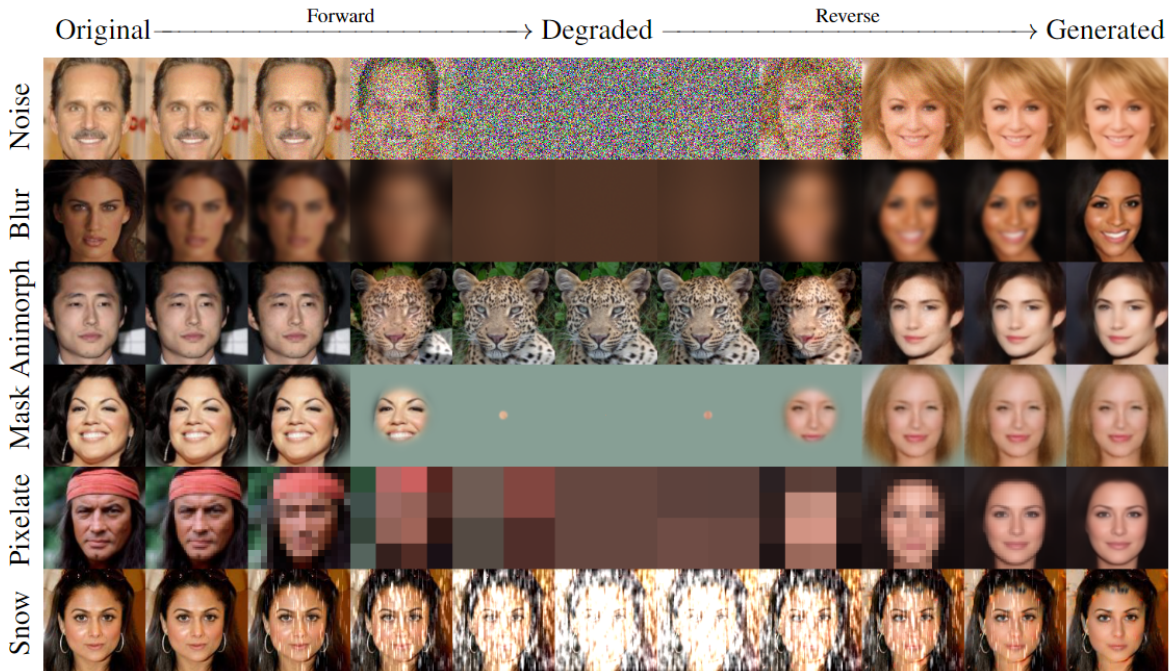


Figure 4.1: Demonstration of the forward and backward processes for both hot and cold diffusions. While standard diffusions are built on Gaussian noise (top row), we show that generative models can be built on arbitrary and even noiseless/cold image transforms, including the ImageNet-C *snowification* operator, and an *animorphosis* operator that adds a random animal image from AFHQ.

4.1 Introduction

Diffusion models have recently emerged as powerful tools for generative modeling [81].

Diffusion models come in many flavors, but all are built around the concept of random noise removal; one trains an image restoration/denoising network that accepts an image contaminated

with Gaussian noise, and outputs a denoised image. At test time, the denoising network is used to convert pure Gaussian noise into a photo-realistic image using an update rule that alternates between applying the denoiser and adding Gaussian noise. When the right sequence of updates is applied, complex generative behavior is observed.

The origins of diffusion models, and also our theoretical understanding of these models, are strongly based on the role played by Gaussian noise during training and generation. Diffusion has been understood as a random walk around the image density function using *Langevin dynamics* [82, 83], which requires Gaussian noise in each step. The walk begins in a high temperature (heavy noise) state, and slowly anneals into a “cold” state with little if any noise. Another line of work derives the loss for the denoising network using variational inference with a Gaussian prior [84, 85, 86].

In this work, we examine the need for Gaussian noise, or any randomness at all, for diffusion models to work in practice. We consider *generalized diffusion models* that live outside the confines of the theoretical frameworks from which diffusion models arose. Rather than limit ourselves to models built around Gaussian noise, we consider models built around arbitrary image transformations like blurring, downsampling, etc. We train a restoration network to invert these deformations using a simple ℓ_p loss. When we apply a sequence of updates at test time that alternate between the image restoration model and the image degradation operation, generative behavior emerges, and we obtain photo-realistic images.

The existence of *cold diffusions* that require no Gaussian noise (or any randomness) during training or testing raises questions about the limits of our theoretical understanding of diffusion models. It also unlocks the door for potentially new types of generative models with very different properties than conventional diffusion seen so far.

4.2 Background

Both the Langevin dynamics and variational inference interpretations of diffusion models rely on properties of the Gaussian noise used in the training and sampling pipelines. From the score-matching generative networks perspective [83, 87], noise in the training process is critically thought to expand the support of the low-dimensional training distribution to a set of full measure in ambient space. The noise is also thought to act as data augmentation to improve score predictions in low density regions, allowing for mode mixing in the stochastic gradient Langevin dynamics (SGLD) sampling. The gradient signal in low-density regions can be further improved during sampling by injecting large magnitudes of noise in the early steps of SGLD and gradually reducing this noise in later stages.

[88] propose a method to learn a noise schedule that leads to faster optimization. Using a classic statistical result, [89] show the connection between removing additive Gaussian noise and the gradient of the log of the noisy signal density in deterministic linear inverse problems. Here, we shed light on the role of noise in diffusion models through theoretical and empirical results in applications to inverse problems and image generation.

Iterative neural models have been used for various inverse problems [90, 91]. Recently, diffusion models have been applied to them [87] for the problems of deblurring, denoising, super-resolution, and compressive sensing [89, 92, 93, 94].

Although not their focus, previous works on diffusion models have included experiments with deterministic image generation [85, 95, 96] and in selected inverse problems [97]. Recently, [98] use a combination of Gaussian noise and blurring as a forward process for diffusion. Though they show the feasibility of a different degradation, here we show definitively that noise is not

a *necessity* in diffusion models, and we observe the effects of removing noise for a number of inverse problems.

Despite prolific work on generative models in recent years, methods to probe the properties of learned distributions and measure how closely they approximate the real training data are by no means closed fields of investigation.

Indirect feature space similarity metrics such as Inception Score [99], Mode Score [100], Frechet inception distance (FID) [101], and Kernel inception distance (KID) [102] have been proposed and adopted to some extent, but they have notable limitations [103]. To adopt a popular frame of reference, we will use FID as the feature similarity metric for our experiments.

4.3 Generalized Diffusion

Standard diffusion models are built around two components. First, there is an image degradation operator that contaminates images with Gaussian noise. Second, a trained restoration operator is created to perform denoising. The image generation process alternates between the application of these two operators. In this work, we consider the construction of generalized diffusions built around arbitrary degradation operations. These degradations can be randomized (as in the case of standard diffusion) or deterministic.

4.3.1 Model components and training

Given an image $x_0 \in \mathbb{R}^N$, consider the *degradation* of x_0 by operator D with severity t , denoted $x_t = D(x_0, t)$. The output distribution $D(x_0, t)$ of the degradation should vary continuously in t , and the operator should satisfy $D(x_0, 0) = x_0$.

In the standard diffusion framework, D adds Gaussian noise with variance proportional to t . In our generalized formulation, we choose D to perform various other transformations such as blurring, masking out pixels, downsampling, and more, with severity that depends on t . We explore a range of choices for D in Section 4.4.

We also require a *restoration* operator R that (approximately) inverts D . This operator has the property that $R(x_t, t) \approx x_0$. In practice, this operator is implemented via a neural network parameterized by θ . The restoration network is trained via the minimization problem

$$\min_{\theta} \mathbb{E}_{x \sim \mathcal{X}} \|R_{\theta}(D(x, t), t) - x\|,$$

where x denotes a random image sampled from distribution $\mathcal{X}$ and $\|\cdot\|$ denotes a norm, which we take to be ℓ_1 in our experiments. We have so far used the subscript R_{θ} to emphasize the dependence of R on θ during training, but we will omit this symbol for simplicity in the discussion below.

4.3.2 Sampling from the model

After choosing a degradation D and training a model R to perform the restoration, these operators can be used in tandem to invert severe degradations by using standard methods borrowed from the diffusion literature. For small

Algorithm 3 Naive Sampling (Eg. DDIM)

Input: A degraded sample x_t $s = t, t - 1, \dots, 1$ $\hat{x}_0 \leftarrow R(x_s, s)$ $x_{s-1} = D(\hat{x}_0, s - 1)$

Return: x_0

Algorithm 4 Transformation Agnostic Cold Sampling (TACoS)

Input: A degraded sample x_t $s = t, t - 1, \dots, 1$ $\hat{x}_0 \leftarrow R(x_s, s)$ $x_{s-1} = x_s - D(\hat{x}_0, s) + D(\hat{x}_0, s - 1)$

degradations ($t \approx 0$), a single application of R can be used to obtain a restored image in one

shot. However, because R is typically trained using a simple convex loss, it yields blurry results when used with large t . Rather, diffusion models [84, 85] perform generation by iteratively applying the denoising operator and then adding noise back to the image, with the level of added noise decreasing over time. This is the standard update sequence in Algorithm 3.

When the restoration operator is perfect, *i.e.* when $R(D(x_0, t), t) = x_0$ for all t , one can easily see that Algorithm 3 produces exact iterates of the form $x_s = D(x_0, s)$. But what happens for imperfect restoration operators? In this case, errors can cause the iterates x_s to wander away from $D(x_0, s)$, and inaccurate reconstruction may occur.

We find that the standard sampling approach in Algorithm 3 works well for noise-based diffusion, possibly because the restoration operator R has been trained to correct (random Gaussian) errors in its inputs. However, we find that it yields poor results in the case of cold diffusions with smooth/differentiable degradations as demonstrated for a deblurring model in Figure 4.2. We propose Transformation Agnostic Cold Sampling (TACoS) in Algorithm 4, which we find to be superior for inverting smooth, cold degradations.

This sampler has important mathematical properties that enable it to recover high quality results. Specifically, for a class of linear degradation operations, it can be shown to produce exact reconstruction (*i.e.* $x_s = D(x_0, s)$) even when the restoration operator R fails to perfectly invert D . We discuss this in the following section.

4.3.3 Properties of TACoS

It is clear from inspection that both Algorithms 3 and 4 perfectly reconstruct the iterate $x_s = D(x_0, s)$ for all $s < t$ if the restoration operator is a perfect inverse for the degradation operator. Hence in this section, we will discuss the reconstruction operator that fails to reconstruct

the image perfectly i.e. incurs error. We first analyze the stability of these algorithms to errors in the restoration operator and then theoretically show that for a simple blur degradation, the error incurred using algorithm 3 is always greater than algorithm 4.

For small values of x and s , TACoS as described in 4 is tolerant of error in the restoration operator R . To see why, consider a problem with linear degradation function of the form $D(x, s) \approx x + s \cdot e$ for a constant vector e . We choose this ansatz because the Taylor expansion of

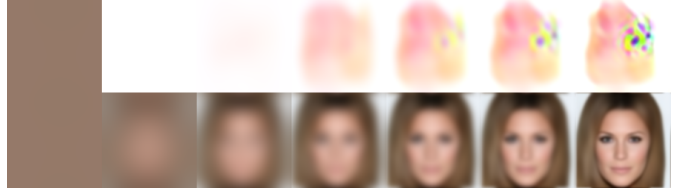


Figure 4.2: Comparison of sampling methods for unconditional generation using cold diffusion on the CelebA dataset. Iterations 2, 4, 8, 16, 32, 64, 128, 192, and 256 are presented. **Top:** Algorithm 3 produces compounding artifacts and fails to generate a new image. **Bottom:** TACoS succeeds in sampling a high quality image without noise.

any smooth degradation $D(x, s)$ around $x = x_0, s = 0$ has the form $D(x, s) \approx x + s \cdot e(x) + \text{HOT}$ where HOT denotes higher order terms. Note, however, the analysis below requires e to be a constant that does not depend on x . The constant/zeroth-order term in this Taylor expansion is zero because we assumed above that the degradation operator satisfies $D(x, 0) = x$.

For a degradation $D(x, s)$ and any restoration operator R , the term x_{s-1} in TACoS becomes

$$\begin{aligned} x_s - D(R(x_s, s), s) + D(R(x_s, s), s-1) &= D(x_0, s) - D(R(x_s, s), s) + D(R(x_s, s), s-1) \\ &= x_0 + s \cdot e - R(x_s, s) - s \cdot e + R(x_s, s) + (s-1) \cdot e = x_0 + (s-1) \cdot e = D(x_0, s-1) \end{aligned}$$

By induction, we see that the algorithm outputs the value $x_s = D(x_0, s)$ for all $s < t$, regardless of the choice of R . In other words, for *any* choice of R , the iteration behaves the same as it would when R is a perfect inverse for the degradation D .

By contrast, Algorithm 3 does not enjoy this behavior even for small values of s . In fact,

when R is not a perfect inverse for D , x_0 is not a fixed point of the update rule in Algorithm 3 because $x_0 \neq D(R(x, 0), 0) = R(x, 0)$ and hence errors compound. If R does not perfectly invert D , we should expect Algorithm 3 to incur errors, even for small values of s . Meanwhile, for small values of s , the behavior of D approaches its first-order Taylor expansion, and Algorithm 4 becomes immune to errors in R . Figure 4.2 demonstrates the stability of TACoS described in Algorithm 4 vs. Algorithm 3 for a deblurring model.

The above analysis is not a complete convergence theory but rather highlights a desirable theoretical property of our method that a naive sampler lacks. However, we can prove that for a *toy* problem in which the blur operator removes one frequency at a time, the error incurred by sampling using Algorithm 3 is greater than the error incurred from using Algorithm 4.

4.4 Generalized Diffusions with Various Transformations

In this section, we take the first step towards cold diffusion by reversing different degradations and hence performing conditional generation. We will extend our methods to perform unconditional (i.e. from scratch) generation in Section 4.5. We empirically evaluate generalized diffusion models trained on different degradations with TACoS proposed in Algorithm 4. We perform experiments on the vision tasks of deblurring, inpainting, and super-resolution. We perform our experiments on MNIST [104], CIFAR-10 [105], and CelebA [106]. In each of these tasks, we gradually remove the information from the clean image, creating a sequence of images such that $D(x_0, t)$ retains less information than $D(x_0, t - 1)$. For these different tasks, we present both qualitative and quantitative results on a held-out testing dataset and demonstrate the importance of the sampling technique described in Algorithm 4. For all quantitative results in this section,

the Frechet inception distance (FID) scores [101] for degraded and reconstructed images are measured with respect to the testing data.

4.4.1 Deblurring

We consider a generalized diffusion based on a Gaussian blur operation (as opposed to Gaussian noise) in which an image at step t has more blur than at $t - 1$. The forward process given the Gaussian kernels $\{G_s\}$ and the image x_{t-1} at step $t - 1$ can thus be written as

$$x_t = M_t \circ x_{t-1} = M_t \circ \dots \circ M_1 \circ x_0 = \bar{M}_t \circ x_0 = D(x_0, t)$$

where $*$ denotes the convolution operator, which blurs an image using a kernel.

We train a deblurring model by minimizing the loss (4.3.1.1), and then use TACoS to invert this blurred diffusion process for which we trained a DNN to predict the clean image $\hat{x}_0$. Qualitative results are shown in Figure 4.3 and quantitative results in Table 4.1. Qualitatively, we can see that images created using the sampling process are sharper and in some cases completely different as compared to the direct reconstruction of the clean image. Quantitatively we can see that the reconstruction metrics such as RMSE and PSNR get worse when we use the sampling process, but on the other hand FID with respect to held-out test data improves. The qualitative improvements and decrease in FID show the benefits of the generalized sampling routine, which brings the learned distribution closer to the true data manifold. Note: we compare the images reconstructed via Algorithm 4, with direct generation as compared to Algorithm 3. This is because the image reconstruction via Algorithm 3 is much worse than both direct generation and Algorithm 4.

In the case of blur operator, the sampling routine can be thought of adding frequencies at each step. This is because the sampling routine involves the term $D(\hat{x}_0, t) - D(\hat{x}_0, t - 1)$ which in the case of blur becomes $\tilde{G}_t * x_0 - \tilde{G}_{t-1} * x_0$. This results in a difference of Gaussians, which is a band pass filter and contains frequencies that were removed at step t . Thus, in the sampling process, we sequentially add the frequencies that were removed during the degradation process.

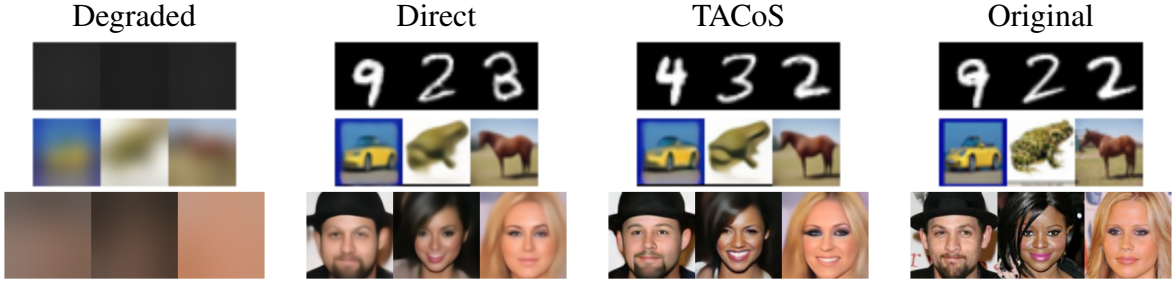


Figure 4.3: Deblurring models trained on the MNIST, CIFAR-10, and CelebA datasets. **Left to right:** degraded inputs $D(x_0, T)$, direct reconstruction $R(D(x_0, T))$, sampled reconstruction with TACoS described in Algorithm 4, and original image.

Table 4.1: Quantitative metrics for quality of image reconstruction using deblurring models.

Dataset	Degraded			Sampled			Direct		
	FID	SSIM	RMSE	FID	SSIM	RMSE	FID	SSIM	RMSE
MNIST	438.59	0.287	0.287	4.69	0.718	0.154	5.10	0.757	0.142
CIFAR-10	298.60	0.315	0.136	80.08	0.773	0.075	83.69	0.775	0.071
CelebA	382.81	0.254	0.193	26.14	0.568	0.093	36.37	0.607	0.083

4.4.2 Inpainting

We define a schedule of transforms that progressively grays-out pixels from the input image. We remove pixels using a Gaussian mask as follows: For input images of size $n \times n$ we start with a 2D Gaussian curve of variance β , discretized into an $n \times n$ array. We normalize so the peak of the curve has value 1, and subtract the result from 1 so the center of the mask as value 0. We randomize the location of the Gaussian mask for MNIST and CIFAR-10, but keep it

centered for CelebA. We denote the final mask by z_β .

Input images x_0 are iteratively masked for T steps via multiplication with a sequence of masks $\{z_{\beta_i}\}$ with increasing β_i . We can control the amount of information removed at each step by tuning the β_i parameter. In the language of Section 4.3, $D(x_0, t) = x_0 \cdot \prod_{i=1}^t z_{\beta_i}$, where the operator $\cdot$ denotes entry-wise multiplication.

Figure 4.4 presents results on test images and compares the output of the inpainting model to the original image. The reconstructed images display reconstructed features qualitatively consistent with the context provided by the unperturbed regions of the image. We quantitatively assess the effectiveness of the inpainting models on each of the datasets by comparing distributional similarity metrics before and after the reconstruction. Our results are summarized in Table 4.2. Note, the FID scores here are computed with respect to the held-out validation set.

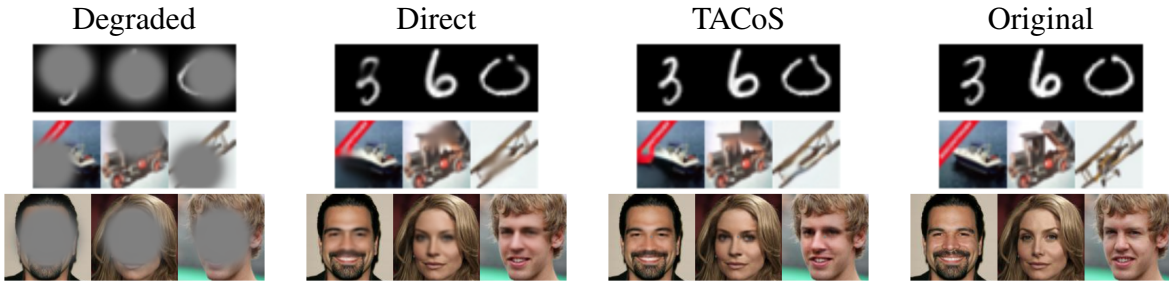


Figure 4.4: Inpainting models trained on the MNIST, CIFAR-10, and CelebA datasets. **Left to right:** Degraded inputs $D(x_0, T)$, direct reconstruction $R(D(x_0, T))$, sampled reconstruction with TACoS described in Algorithm 4, and original image.

Table 4.2: Quantitative metrics for quality of image reconstruction using inpainting models.

Dataset	Degraded			Sampled			Direct		
	FID	SSIM	RMSE	FID	SSIM	RMSE	FID	SSIM	RMSE
MNIST	108.48	0.490	0.262	1.61	0.941	0.068	2.24	0.948	0.060
CIFAR-10	40.83	0.615	0.143	8.92	0.859	0.068	9.97	0.869	0.063
CelebA	127.85	0.663	0.155	5.73	0.917	0.043	7.74	0.922	0.039

4.4.3 Super-Resolution

For this task, the degradation operator downsamples the image by a factor of two in each direction. The final resolution of x_T is 4×4 for MNIST and CIFAR-10 and 2×2 in the case of Celeb-A. After each down-sampling, the lower-resolution image is resized to the original image size, using nearest-neighbor interpolation.

Figure 4.5 presents example testing data inputs for all datasets and compares the output of the super-resolution model to the original image. Though the reconstructed images are not perfect for the more challenging datasets, the reconstructed features are qualitatively consistent with the context provided by the low resolution image. Table 4.3 compares the distributional similarity metrics between degraded/reconstructed images and test samples.

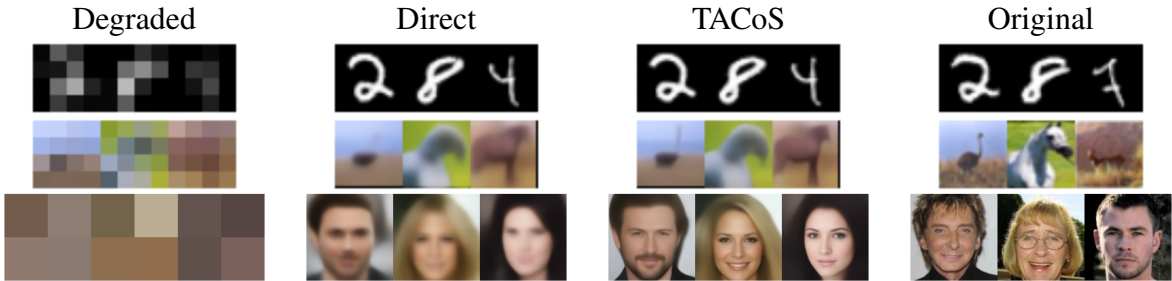


Figure 4.5: Superresolution models trained on the MNIST, CIFAR-10, and CelebA datasets. **Left to right:** degraded inputs $D(x_0, T)$, direct reconstruction $R(D(x_0, T))$, sampled reconstruction with TACoS described in Algorithm 4, and original image.

Table 4.3: Quantitative metrics for quality of image reconstruction using super-resolution models.

Dataset	Degraded			Sampled			Direct		
	FID	SSIM	RMSE	FID	SSIM	RMSE	FID	SSIM	RMSE
MNIST	368.56	0.178	0.231	4.33	0.820	0.115	4.05	0.823	0.114
CIFAR-10	358.99	0.279	0.146	152.76	0.411	0.155	169.94	0.420	0.152
CelebA	349.85	0.335	0.225	96.92	0.381	0.201	112.84	0.400	0.196

4.5 Cold Generation

Diffusion models can successfully learn the underlying distribution of training data, and thus generate diverse, high quality images [85, 95, 107, 108]. We will first discuss deterministic generation using Gaussian noise and then discuss in detail unconditional generation using deblurring. Finally, we provide a proof of concept that the TACoS described in Algorithm 4 can be extended to other degradations.

4.5.1 Generation using deterministic noise degradation

Here we discuss image generation using a noise-based degradation presented in our notation from Section 4.3, which we will later prove is equivalent to DDIM [85]. We use the following degradation operator: $D(x, t) = \sqrt{\alpha_t}x + \sqrt{1 - \alpha_t}z$.

D is an interpolation between the data point x and a sampled noise pattern $z \in \mathcal{N}(0, 1)$. During training, D is applied *once* and thus z is sampled once for every image in every batch. However, sampling involves iterative applications of the degradation operator D , which poses the question of how to pick z for the sequence of degradations D applied in a single image generation.

There are three possible choices for z . The first would be to resample z for each application of D , but this would make the sampling process nondeterministic for a fixed starting point. Another option is to sample a noise pattern z *once* for each *separate* image generation and reuse it in each application of D . In Table 4.4 we refer to this approach as *Fixed Noise*. Finally, one can calculate the noise vector z to be used in step t of reconstruction by using the formula

$$\hat{z}(x_t, t) = \frac{x_t - \sqrt{\alpha_t}R(x_t, t)}{\sqrt{1 - \alpha_t}}.$$

This method denoted *Estimated Noise* in Table 4.4 turns out to be equivalent to the deterministic sampling proposed in [85].

4.5.2 Image generation using blur

The forward diffusion process in noise-based diffusion models has the advantage that the degraded image distribution at the final step T is simply an isotropic Gaussian. One can therefore perform (unconditional) generation by first drawing a sample from the isotropic Gaussian, and sequentially denoising it with backward diffusion.

When using blur as a degradation, the fully degraded images do not form a nice closed-form distribution that we can sample from. They do, however, form a simple enough distribution that can be modeled with simple methods. Note that every image x_0 degenerates to an x_T that is constant (i.e., every pixel is the same color) for large T . Furthermore, the constant value is exactly the channel-wise mean of the RGB image x_0 , and can be represented with a 3-vector. This 3-dimensional distribution is easily represented using a Gaussian mixture model (GMM). This GMM can be sampled to produce the random pixel values of a severely blurred image, which can be deblurred using cold diffusion to create a new image.

Our generative model uses a blurring schedule where we progressively blur each image with a Gaussian kernel of size 27×27 over 300 steps. The standard deviation of the kernel starts at 1 and increases exponentially at the rate of 0.01. We then fit a simple GMM with one component to the distribution of channel-wise means. To generate an image from scratch, we sample the channel-wise mean from the GMM, expand the 3D vector into a 128×128 image with three channels, and then apply TACoS.

Empirically, the presented pipeline generates images with high fidelity but low diversity, as reflected quantitatively by comparing the perfect symmetry column with results from hot diffusion in Table 4.4. We attribute this to the perfect correlation between pixels of x_T sampled from the channel-wise mean Gaussian mixture model. To break the symmetry between pixels, we add a small amount of Gaussian noise (of standard deviation 0.002) to each sampled x_T . As shown in Table 4.4, the simple trick drastically improves the quality of generated images. We also present the qualitative results for cold diffusion using blur transformation in Figure Table 4.4: FID scores for CelebA and AFHQ datasets using hot (noise) and cold diffusion (blur transformation). Breaking the symmetry within pixels of the same channel further improves FID.

Dataset	Hot Diffusion		Cold Diffusion	
	Fixed Noise	Estimated Noise	Perfect symmetry	Broken symmetry
CelebA	59.91	23.11	97.00	49.45
AFHQ	25.62	20.59	93.05	54.68



Figure 4.6: Examples of generated samples from 128×128 CelebA and AFHQ datasets using cold diffusion with blur transformation

4.5.3 Generation using other transformations

In this section, we provide a proof of concept that generation can be extended to other transformations. Specifically, we show preliminary results on inpainting, super-resolution, and *animorphosis*. Inspired by the simplicity of the degraded image distribution for the blurring

routine presented in the previous section, we use degradation routines with predictable final distributions here as well.

To use the Gaussian mask transformation for generation, we modify the masking routine so the final degraded image is completely devoid of information. One might think a natural option is to send all of the images to a completely black image x_T , but this would not allow for any diversity in generation. To get around this maximally non-injective property, we instead make the mask turn all pixels to a random, solid color. This still removes all of the information from the image, but it allows us to recover different samples from the learned distribution via Algorithm 4 by starting off with different color images. More formally, a Gaussian mask $G_t = \prod_{i=1}^t z_{\beta_i}$ is created in a similar way as discussed in the Section 4.4.2, but instead of multiplying it directly to the image x_0 , we create x_t as $G_t \cdot x_0 + (1 - G_t) \cdot c$, where c is an image of a randomly sampled color.

For super-resolution, the routine down-samples to a resolution of 2×2 , or 4 values in each channel. These degraded images can be represented as one-dimensional vectors, and their distribution is modeled using one Gaussian distribution. Using the same methods described for generation using blurring described above, we sample from this Gaussian-fitted distribution of the lower-dimensional degraded image space and pass this sampled point through the generation process trained on super-resolution data to create one output.

Additionally to show one can invert nearly any transformation, we include a new transformation deemed *animorphosis*, where we iteratively transform a human face from CelebA to an animal face from AFHQ. Though we chose CelebA and AFHQ for our experimentation, in principle such interpolation can be done for any two initial data distributions.

More formally, given an image x and a random image z sampled from the AFHQ manifold,

x_t can be written as $x_t = \sqrt{\alpha_t}x + \sqrt{1 - \alpha_t}z$. Note this is essentially the same as the noising procedure, but instead of adding noise we are adding a progressively higher weighted AFHQ image. In order to sample from the learned distribution, we sample a random image of an animal and use TACoS.

We present results for the CelebA dataset, and hence the quantitative results in terms of FID scores for inpainting, super-resolution and *animorphosis* are 90.14, 92.91 and 48.51 respectively. We further show some qualitative samples in Figure 4.7, and in Figure 4.1.

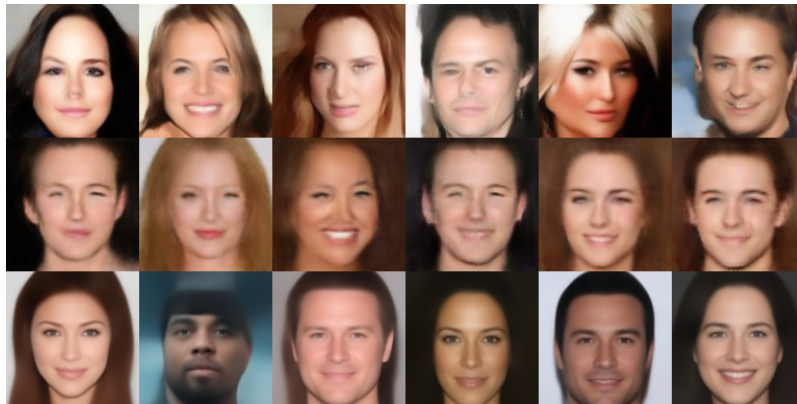


Figure 4.7: Preliminary demonstration of the generative abilities of other cold diffusins on the 128×128 CelebA dataset. The top row is with *animorphosis* models, the middle row is with inpainting models, and the bottom row exhibits super-resolution models.

Chapter 5: Universal Guidance for Diffusion Models

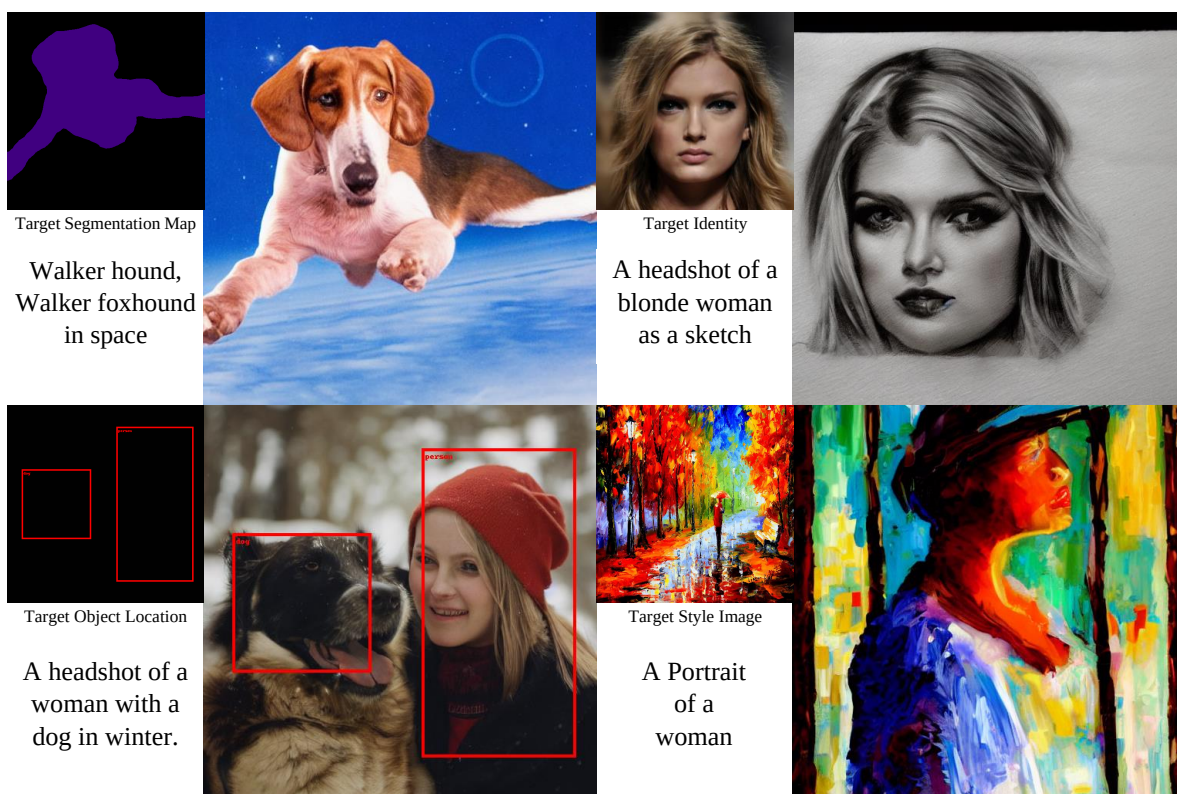


Figure 5.1: Diffusion guided by off-the-shelf networks. Top left: segmentation guidance, top-right: face recognition guidance, bottom-left: object detection guidance, bottom-right: style-transfer.

5.1 Introduction

Diffusion models are powerful tools for creating digital art and graphics. Much of their success stems from our ability to carefully control their outputs, customizing results for each

user’s individual needs. Most models today are controlled through *conditioning*. With conditioning, the diffusion model is built from the ground up to accept a particular modality of input from the user, be it descriptive text, segmentation maps, class labels, etc. While conditioning is a powerful tool, it results in models that are handcuffed to a single conditioning modality. If another modality is needed, a new model has to be trained, often from scratch. Unfortunately, the high cost of training makes this prohibitive for most users.

A more flexible approach to controlling model outputs is to use *guidance*. In this approach, the diffusion model acts as a generic image generator, and is not required to understand a user’s instructions. The user pairs this model with a guidance function that measures whether some criterion has been met. For example, one could guide the model to minimize the CLIP score between the generated image and a text description of the user’s choice. During each iteration of image creation, the iterates are nudged down the gradient of the guidance function, causing the final generated image to satisfy the user’s criterion.

In this paper, we study guidance methods that enable any off-the-shelf model or loss function to be used as guidance for diffusion. Because guidance functions can be used without re-training or modification, this form of guidance is *universal* in that it enables a diffusion model to be adapted for nearly any purpose.

From a user perspective, guidance is superior to conditioning, as a *single* diffusion network is treated like a foundational model that provides universal coverage across many use cases, both commonplace and bespoke. Unfortunately, it is widely believed that this approach is infeasible. While early diffusion models relied on classifier guidance [95], the community quickly turned to classifier-free schemes [109] that require a model to be trained from scratch on class labels with a particular frozen ontology that cannot be changed [110, 111, 112].

The difficulty of using guidance stems from the domain shift between the noisy images used by the diffusion sampling process and the clean images on which the guidance models are trained. When this gap is closed, guidance can be performed successfully. For example, [110] successfully use a CLIP model as guidance, but only after re-training CLIP from scratch using noisy inputs. Noisy retraining closes the domain gap, but at a very high financial and engineering cost. To avoid the additional cost, we study methods for closing this gap by changing the sampling scheme, rather than the model.

To this end, our contributions are summarized as follows:

- We propose an algorithm that enables universal guidance for diffusion models. Our proposed sampler evaluates the guidance models only on denoised images, rather than noisy latent States. By doing so, we close the domain gap that has plagued standard guidance methods. This strategy provides the end-user with the flexibility to work with a wide range of guidance modalities and even multiple modalities simultaneously. The underlying diffusion model remains fixed and no fine-tuning of any kind is necessary.
- We demonstrate the effectiveness of our approach for a variety of different constraints such as *classifier labels*, *human identities*, *segmentation maps*, *annotations from object detectors*, and constraints arising from *inverse linear problems*.

5.2 Background

We first briefly review the recent literature on the core framework behind diffusion models. Then, we define the problem setting of controlled image generation and discuss previous related works.

5.2.1 Diffusion Models

Diffusion models are strong generative models that proved powerful even when first introduced for image generation [83, 84]. The approach has been successfully extended to a number of domains, such as audio and text generation [113, 114, 115, 116].

We introduce (unconditional) diffusion formally, as it is helpful in describing the nuances of different types of models. A diffusion model is defined as a combination of a T -step forward process and a T -step reverse process. Conceptually, the forward process gradually adds Gaussian noise of different magnitudes to a clean data point z_0 , while the reverse process attempts to gradually denoise a noisy input in hopes of recovering a clean data point. More concretely, given an array of scalars representing noise scales $\{\alpha_t\}_{t=1}^T$ and an initial, clean data point z_0 , applying t steps of the forward process to z_0 yields a noisy data point

$$z_t = \sqrt{\alpha_t}z_0 + (\sqrt{1 - \alpha_t})\epsilon, \epsilon \sim \mathcal{N}(0, I). \quad (5.2.1.1)$$

A diffusion model is a learned denoising network ϵ_θ . It is trained so that for any pair (z_0, t) and any sample of ϵ ,

$$\epsilon_\theta(z_t, t) \approx \epsilon = \frac{z_t - \sqrt{\alpha_t}z_0}{\sqrt{1 - \alpha_t}}. \quad (5.2.1.2)$$

The reverse process takes the form $q(z_{t-1}|z_t, z_0)$ with various detail definitions, where $q(\cdot|\cdot)$ is generally parameterized as a Gaussian distribution. Different works also studied different approximations of the unknown $q(z_{t-1}|z_t, z_0)$ used to perform sampling. For example, denoising

diffusion implicit model (DDIM) [85] first computed a *predicted* clean data point

$$\hat{z}_0 = \frac{z_t - (\sqrt{1 - \alpha_t})\epsilon_\theta(z_t, t)}{\sqrt{\alpha_t}}, \quad (5.2.1.3)$$

and sample z_{t-1} from $q(z_{t-1}|z_t, \hat{z}_0)$ by replacing unknown z_0 with $\hat{z}_0$. On the other hand, while the details of individual sampling methods vary, all sampling methods produce z_{t-1} based on current sample z_t , current time step t and a predicted noise $\hat{\epsilon}$. To ease the notation burden, we define a function $S(\cdot, \cdot, \cdot)$ as an abstraction of the sampling method, where $z_{t-1} = S(z_t, \hat{\epsilon}, t)$.

5.2.2 Controlled Image Generation

In this paper, we focus on controlled image generation with various constraints. Consider a differentiable guidance function f , for example a CLIP feature extractor or a segmentation network. When applied to an image, we obtain a vector $c = f(x)$. We also consider a function $\ell(\cdot, \cdot)$ that measures the closeness of two vectors c and c' . Given a particular choice of c , which we call a *prompt*, the corresponding constraint (based on c , ℓ , and f) is formalized as $\ell(c, f(z)) \approx 0$, and we aim to generate a sample z from the image distribution satisfying the constraint. In plain words, we want to generate an in-distribution image that matches the prompt.

Prior work on controlled generative diffusion falls into two main categories. We refer to the first category as conditional image generation, and the second category as guided image generation. Next, we discuss the characteristics of each category and better situate our work among existing methods.

Conditional Image Generation.

Methods from this category require training new diffusion models that accept the prompt as

an additional input [109, 110, 112, 117, 118, 119, 120]. For example, [109] proposed classifier-free guidance using class labels as prompts, and trained a diffusion model by linear interpolation between unconditional and conditional outputs of the denoising networks. [112] studied the case where the guidance function is a known linear degradation operator, and trained a conditional model to solve linear inverse problems. [110] further extended classifier-free guidance to text-conditional image generation with descriptive phrases as prompts, and trained a diffusion model to enforce the similarity between the CLIP [121] representations of the generated images and the text prompts. These methods are successful across different types of constraints, however the requirement to retrain the diffusion model makes them computationally intensive.

Guided Image Generation. Works in this category employed a frozen pre-trained diffusion model as a foundation model, but modify the sampling method to guide the image generation with feedback from the guidance function. Our method falls into this category. Prior work that studied guided image generation did so with a variety of restrictions and external guidance functions [95, 97, 122, 123, 124, 125, 126]. For example, [95] proposed classifier guidance, where they trained a classifier on images of different noise scales as the guidance function f , and included gradients of the classifier during the sampling process. However, a classifier for noisy images is domain-specific and generally not readily available – an issue our method circumvents. [122] assumed the external guidance functions to be linear operators, and generated the component of images residing in the null space of linear operators with the foundation model. Unfortunately, extending that method to handle non-linear guidance functions is non-trivial. [123] studied general guidance functions, and modified the sampling process with the gradient of guidance function calculated on the expected denoised images. Nevertheless, the authors only presented results with simpler

non-linear guidance functions such as non-linear blurring.

In this work, we study universal guidance algorithms for guided image generation with diffusion models using any off-the-shelf guidance functions f , such as object detection or segmentation networks.

5.3 Universal Guidance

We propose a guidance algorithm that augments the image sampling method of a diffusion model to include guidance from an off-the-shelf auxiliary network. Our algorithm is motivated by an empirical observation that the reconstructed clean image $\hat{z}_0$ obtained by Equation (5.2.1.3), while naturally imperfect, is still appropriate for a generic guidance function to provide informative feedback to guide the image generation. In Section 5.3.1, we motivate our *forward universal guidance* by extending classifier guidance [95] to leverage this observation and handle generic guidance functions. In Section 5.3.2, we propose a supplementary *backward universal guidance* to help enforce the generated image to satisfy the constraint based on the guidance function f . In Section 5.3.3, we discuss a simple yet helpful stepwise refinement trick to empirically improve the fidelity of generated images.

5.3.1 Forward Universal Guidance

To guide the generation with information from the external guidance function f and the loss function ℓ , an immediate thought is to extend classifier guidance [95] to accept any general guidance function. Concretely, given a class prompt c , classifier guidance performs classification-

guided sampling by replacing $\epsilon_\theta(z_t, t)$ in each sampling step $S(z_t, t)$ with

$$\hat{\epsilon}_\theta(z_t, t) = \epsilon_\theta(z_t, t) - \sqrt{1 - \alpha_t} \nabla_{z_t} \log p(c|z_t). \quad (5.3.1.1)$$

Defining $\ell_{ce}(\cdot, \cdot)$ to be the cross-entropy loss and f_{cl} to be the guidance function that outputs classification probability, Equation (5.3.1.1) can be re-written as

$$\hat{\epsilon}_\theta(z_t, t) = \epsilon_\theta(z_t, t) + \sqrt{1 - \alpha_t} \nabla_{z_t} \ell_{ce}(c, f_{cl}(z_t)). \quad (5.3.1.2)$$

However, directly replacing f_{cl} and ℓ_{ce} with any off-the-shelf guidance and loss functions does not work in practice, as f is most likely trained on clean images and fails to provide meaningful guidance when the input is noisy.

To address the issue, we observe that

$$p(c|z_t) = \int p(c|z_0, z_t) p(z_0|z_t) dz_0 = \mathbb{E}_{z_0 \sim p(z_0|z_t)} [p(c|z_0)]. \quad (5.3.1.3)$$

where c is conditionally-independent with z_t given z_0 . Leveraging the fact that we can obtain a *predicted* clean image $\hat{z}_0$ by Equation (5.2.1.3) with $\epsilon_\theta(z_t, t)$, we approximate the expectation in Equation (5.3.1.3) as $\mathbb{E}_{z_0 \sim p(z_0|z_t)} [p(c|z_0)] \approx p(c|\hat{z}_0)$. This leads to our proposed guided sampling procedure

$$\hat{\epsilon}_\theta(z_t, t) = \epsilon_\theta(z_t, t) + s(t) \cdot \nabla_{z_t} \ell(c, f(\hat{z}_0)) \quad (5.3.1.4)$$

where $s(t)$ controls the guidance strength for each sampling step and

$$\nabla_{z_t} \ell(c, f(\hat{z}_0)) = \nabla_{z_t} \ell \left(c, f \left(\frac{z_t - \sqrt{1 - \alpha_t} \epsilon_\theta(z_t, t)}{\sqrt{\alpha_t}} \right) \right)$$

as in Equation (5.2.1.3). We term Equation (5.3.1.4) forward universal guidance, or forward guidance in short. In practice, applying forward guidance effectively brings the generated image closer to the prompt while keeping the generation trajectory in the data manifold. We note that a related approach is also studied in [123], where the guidance step is computed based on $E[z_0|z_t]$. The approach drew inspiration from the score-based generative framework [87], but resulted in a different update method.

5.3.2 Backward Universal Guidance

As will be shown in Section 5.4.2, we observe that forward guidance sometimes over-prioritizes maintaining the “realness” of the image, resulting in an unsatisfactory match with the given prompt. Simply increasing the guidance strength $s(t)$ is suboptimal, as this often results in instability as the image moves off the manifold faster than the denoiser can correct it.

Algorithm 5 Universal Guidance

Parameter: Refinement steps k , backward guidance steps m , and guidance strength $s(t)$,
Required: z_t the noisy vector at a given time-step t , diffusion model ϵ_θ , noise scale α_t ,
guidance function f , loss function ℓ , and prompt c
for $n = 1, 2, \dots, k$ **do**
 Calculate $\hat{z}_0$ as in Equation (5.2.1.3).
 Calculate $\hat{\epsilon}_\theta$ using forward universal guidance as in Equation (5.3.1.4).
 if $m > 0$ **then**
 Calculate Δz_0 by minimizing Equation (5.3.2.1) with m steps of gradient descent.
 Perform backward universal guidance by $\hat{\epsilon}_\theta \leftarrow \hat{\epsilon}_\theta - \sqrt{\alpha_t/(1 - \alpha_t)}\Delta z_0$
 $z_{t-1} \leftarrow S(z_t, \hat{\epsilon}_\theta, t)$.
 $\epsilon' \sim \mathcal{N}(0, I)$.
 $z_t \leftarrow \sqrt{\alpha_t/\alpha_{t-1}}z_{t-1} + \sqrt{1 - \alpha_t/\alpha_{t-1}}\epsilon'$.
Return z_{t-1}

To address the issue, we propose backward universal guidance, or backward guidance in short, to supplement forward guidance and help enforce the generated image to satisfy the constraint. The key idea of backward guidance is to optimize for a clean image that best matches the prompt based on $\hat{z}_0$, and linearly translate the guided change back to the noisy image space at step t . Concretely, instead of directly calculating $\nabla_{z_t}\ell(c, f(\hat{z}_0))$, we compute a guided change Δz_0 in clean data space as

$$\Delta z_0 = \arg \min_{\Delta} \ell(c, f(\hat{z}_0 + \Delta)). \quad (5.3.2.1)$$

Empirically, we solve Equation (5.3.2.1) with m -step gradient descent, where we use $\Delta = 0$ as a starting point. Since $\hat{z}_0 + \Delta z_0$ minimizes $\ell(c, f(z))$ directly, Δz_0 is the change in clean data space that best enforces the constraint. Then, we translate Δz_0 back to the noisy data space of z_t by calculating the *guided denoising prediction* $\tilde{\epsilon}$ that satisfies

$$z_t = \sqrt{\alpha_t}(\hat{z}_0 + \Delta z_0) + \sqrt{1 - \alpha_t}\tilde{\epsilon}. \quad (5.3.2.2)$$



Figure 5.2: Illustration of the Universal Stepwise Refinement (USR) in face-guided generation. The left-most image is the guidance face, while the adjacent images are generated with USR using $k = 1$ and $k = 2$, respectively. A modest increment in USR steps from 1 (the trivial value) to 2 significantly enhances both guidance adherence and image fidelity.

Reusing Equation (5.2.1.3), we can rewrite $\tilde{\epsilon}$ as an augmentation to the original denoising prediction

$\epsilon_{\theta}(z_t, t)$ by

$$\tilde{\epsilon} = \epsilon_{\theta}(z_t, t) - \sqrt{\alpha_t/(1 - \alpha_t)}\Delta z_0. \quad (5.3.2.3)$$

Comparing to forward guidance, backward guidance (as Equation (5.3.2.3)) produces an optimized direction for the generated image to match the given prompt, and hence prioritizes enforcing the constraint. Furthermore, calculation of a gradient step for Equation (5.3.2.1) is computationally cheaper than forward guidance (Equation (5.3.1.4)), and we can therefore afford to solve Equation (5.3.2.1) with multiple gradient steps, further improving the match with the given prompt.

We note that the names “forward” and “backward” are used analogously to the forward and backward Euler methods.

5.3.3 Universal Stepwise Refinement

Upon applying our universal guidance to standard generation pipelines, we observe that in some instances, the images generated exhibit artifacts or unusual behaviors that differentiate

them from natural images. Similar findings have been reported in [122, 124], where linear guidance functions were explored. Although we attempted to enhance the realism by adjusting the guidance strength $s(t)$, finding an optimal balance that both guarantees image realism and adherence to guidance constraints proved challenging, particularly with complex guidance functions. We hypothesize that the direction of guidance offered by our universal method might not always correlate with image realism, diverting the image from a natural image sampling trajectory.

Drawing motivation from [122, 124], we propose a more nuanced approach, termed **Universal Stepwise Refinement** (USR), to navigate these challenges. Specifically, after sampling $z_{t-1} = S(z_t, \hat{\epsilon}_t, t)$, we reintroduce Gaussian noise $\epsilon' \sim \mathcal{N}(0, I)$ to z_{t-1} , deriving z'_t as per the equation:

$$z'_t = \sqrt{\alpha_t/\alpha_{t-1}} \cdot z_{t-1} + \sqrt{1 - \alpha_t/\alpha_{t-1}} \cdot \epsilon'. \quad (5.3.3.1)$$

Equation 5.3.3.1 ensures z'_t retains the appropriate noise scale for input at time step t . Given a refinement step k , we repeat the Universal Stepwise Refinement k times before advancing with the sampling for step $t - 1$. This refined process enables the exploration of various regions of the data manifold at a consistent noise scale to reach a solution that aligns with both the guidance and image quality requisites. Our empirical assessments indicate that using USR mechanism on top of proper guidance strength $s(t)$ significantly enhanced the image realism while still maintaining the adherence to the prompt, as shown in Figure 6.1.

We summarize our universal guidance algorithm composed of forward universal guidance, backward universal guidance and universal stepwise refinement for a single sampling step in Algorithm 5. For simplicity, the algorithm assumes only one guidance function, but can be easily adapted to handle multiple pairs of (f, l) . Additionally, the objectives of the forward and

backward guidance do not have to be identical, allowing different ways to simultaneously utilize multiple guidance functions.

5.4 Experiments

In this section, we present results testing our proposed universal guidance algorithm against a wide variety of guidance functions. Specifically, we experiment with Stable Diffusion [111], a diffusion model that is able to perform text-conditional generation by accepting text prompt as additional input, and experiment with a purely unconditional diffusion model trained on ImageNet [127], where we use pre-trained model provided by OpenAI [95]. We note that Stable Diffusion, while being a text-conditional generative model, can also perform unconditional image generation by simply using an empty string as the text prompt. We first present the experiment on Stable Diffusion for different guidance functions in Section 5.4.1, and present the results on ImageNet diffusion model in Section 5.4.2.

5.4.1 Results for Stable Diffusion

In this section, we present the results of guided image generation using Stable Diffusion as the foundation model. The guidance functions we experiment with include a segmentation network, a face recognition network, an object detection network and style guidance with CLIP feature extractor [121]. For experiments on Stable Diffusion, we discover that applying forward guidance already produce high-quality images that match the given prompt, and hence set $m = 0$. To perform forward guidance on Stable Diffusion, we forward the predicted clean latent variable computed by Equation (5.2.1.3) through the image decoder of Stable Diffusion to obtain

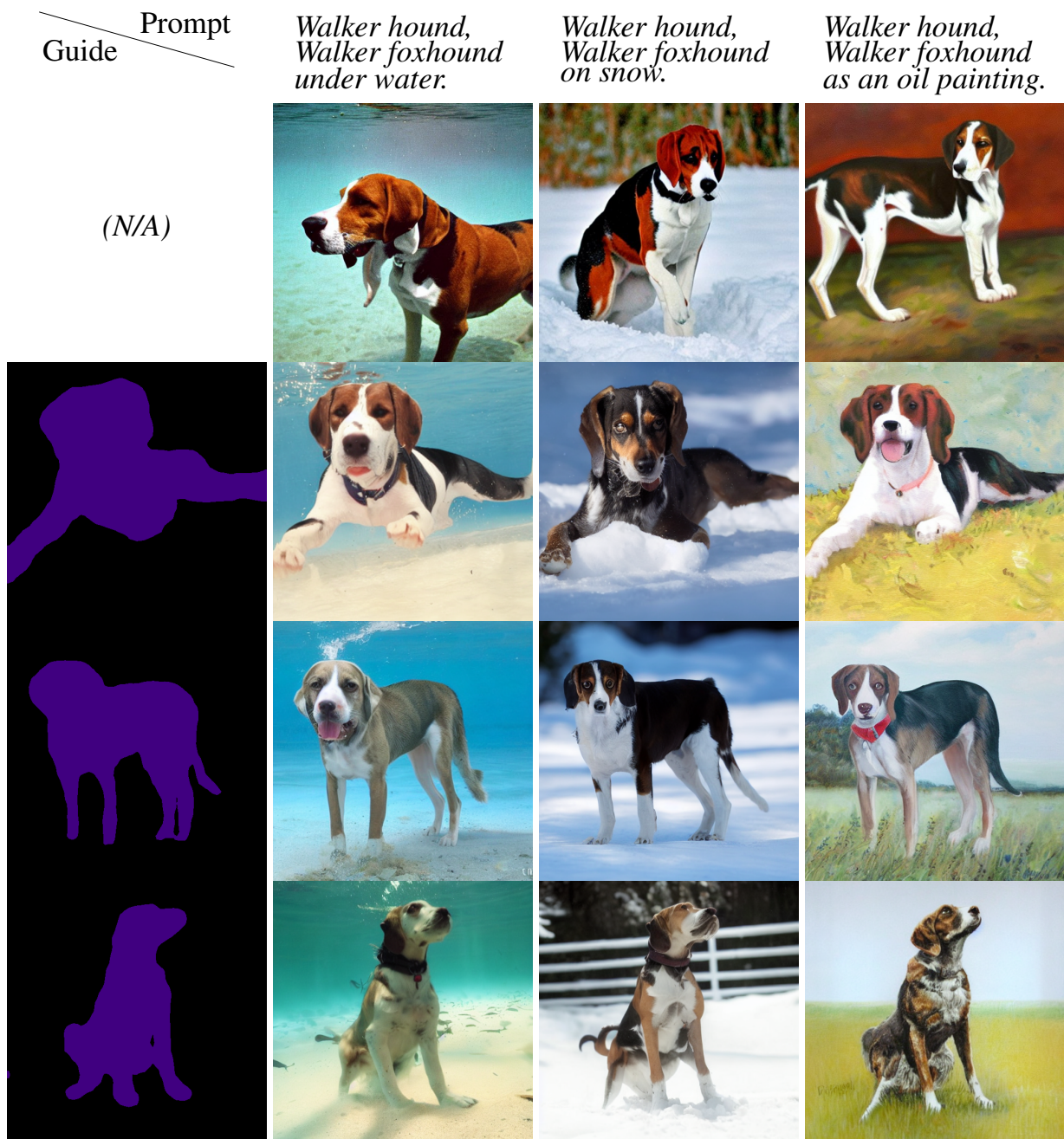


Figure 5.3: In addition to matching the text prompts (above each column), these images are guided by an image segmentation pipeline. Each column contains examples of images generated to match the prompt and the segmentation map in the left-most column. The top-most row contains examples generated without guidance.



Figure 5.4: Each column contains examples of images generated to match the prompt and the identity of the images in the left-most column. The top-most row contains examples generated without guidance.

predicted clean images. We discuss the qualitative results and implementation details for each guidance function in its corresponding subsection. We summarize our quantitative evaluation in Table 5.1, where we evaluate how well the generated images match the external guidance with task-specific metrics, and include the similarity of CLIP embedding between text prompts and images generated with and without guidance. For all tasks, the minimum difference between CLIP similarities indicates that our algorithm performs guidance without sacrificing the effectiveness of the underlying diffusion model.

Segmentation Map Guidance. To guide image generation using a segmentation map as prompt, we use a MobileNetV3-Large [128] with a segmentation head, and a publicly available pre-trained model in PyTorch [129]. We use standard per-pixel cross-entropy loss between a given prompt and the predicted segmentation of generated images as our loss function ℓ .

Task	Metric	Value
Segmentation	mIoU	0.898
	CLIP similarity	(0.247) 0.249
Face	Face similarity	0.642
	CLIP similarity	(0.287) 0.234
Object detection	mAP@50	0.634
	CLIP similarity	(0.263) 0.246

Table 5.1: Quantitative analysis of different guidance applied on Stable Diffusion. The reference value in parenthesis is obtained with no external guidance.

In our experiment, we combine segmentation maps that depict objects of different shapes with new text prompts. We use the text prompt as a fixed additional input to Stable Diffusion to perform text-conditional sampling, and guide the text-conditional generated images to match the given segmentation maps. Qualitative results are presented in Figure 5.3. From Figure 5.3, we see that the generated images show a clear separation between object and background that matches the given segmentation map nearly

perfectly. The generated object and background also each match their descriptive text (i.e. dog breed and environment description). Furthermore, the generated images are overall highly realistic. In Table 5.1, we evaluate mIoU between the ground truth segmentation map and the predicted segmentation of generated images to assess the match with constraint.

Face Recognition Guidance. For guiding generation to include a specific person’s likeness, we propose combining face detection and facial recognition modules into one guidance function. This setup produces a facial attribute embedding from an image of a face. We use multi-task cascaded convolutional networks (MTCNN) [130] for face detection, and we use facenet [131] for facial recognition. The guidance function f then crops out the detected face and outputs a facial attribute embedding as a prompt and we use an l_1 -loss between embeddings as the loss function ℓ .

We explore different combinations of face guidance and text prompts. Similarly to the segmentation case, we use the text prompt as a fixed additional conditioning to Stable Diffusion and guide this text-conditional trajectory with our algorithm so that the face in the generated image looks similar to the face prompt. In Figure 5.4, we clearly see that the facial characteristics of a given face prompt are reproduced almost perfectly on the generated images. The descriptive text of either background, material, or style is also realized correctly and blends nicely with the generated faces. We again summarize our quantitative evaluation in Table 5.1. We evaluate the similarity between facial attributes of ground-truth identity and the generated faces. In general, two faces are considered to be from the same person if the similarity is over 0.5, and our algorithm can effectively guide the generated face to meet the criteria.

Object Location Guidance For Stable Diffusion, we also present the results of guided image



Figure 5.5: In addition to text prompts (above each column), these images are guided by an object detector. Each column contains examples of images generated to match the prompt and the bounding boxes used for guidance.

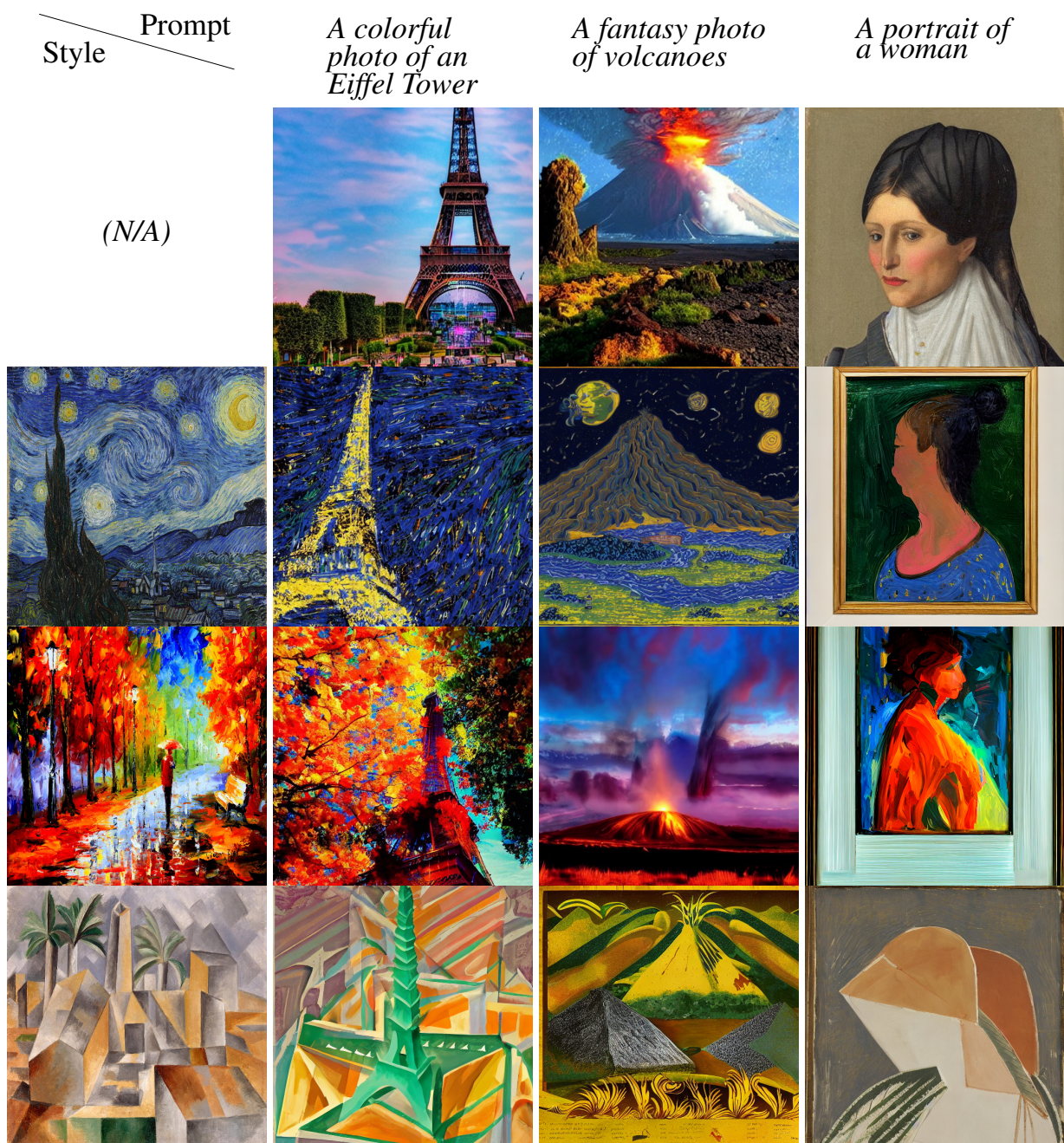


Figure 5.6: In addition to text prompts, these images are guided by a style image. Each column contains examples of images generated to match the text prompt and the style used for guidance.

generation with an object detection network. For this experiment, we use Faster-RCNN [132] with Resnet-50-FPN backbone [133], a publicly available pre-trained model in Pytorch, as our object detector. We use bounding boxes with class labels as our object location prompt. We construct a loss function ℓ by the sum of three individual losses, namely (1) anchor classification loss, (2) bounding box regression loss and (3) region label classification loss.

We again experiment with different combinations of text prompt and object location prompt, and similarly use the text prompt as a fixed conditioning to Stable Diffusion. Using our proposed guidance algorithm, we perform guided image generation that generates and matches the objects presented in the text prompt to the given object locations. The results are presented in Figure 5.5. We observe from Figure 5.5 that objects in the descriptive text all appear in the designated location with the appropriate size indicated by the given bounding boxes. Each location is filled with appropriate, high-quality generations that align with varied image content prompts, ranging from “beach” to “oil painting”. In Table 5.1, we use mAP@50 to measure how well the generated images satisfy the constraint.

Style Guidance Finally, we conclude our experiments on Stable Diffusion by guiding the image generation based on a reference style given by a style image. To achieve so, we capture the reference style from the style image by the image feature extractor from CLIP, and use the resulting image embedding as prompts. The loss function calculates the negative cosine similarity between the embedding of generated images and the embedding of the style image. Similar to previous experiments, we control the content using text input as additional conditioning to the Stable Diffusion model. We experiment with combinations of different style images and different text prompts, and present the results in Figure 5.6. From Figure 5.6, we can see that the generated

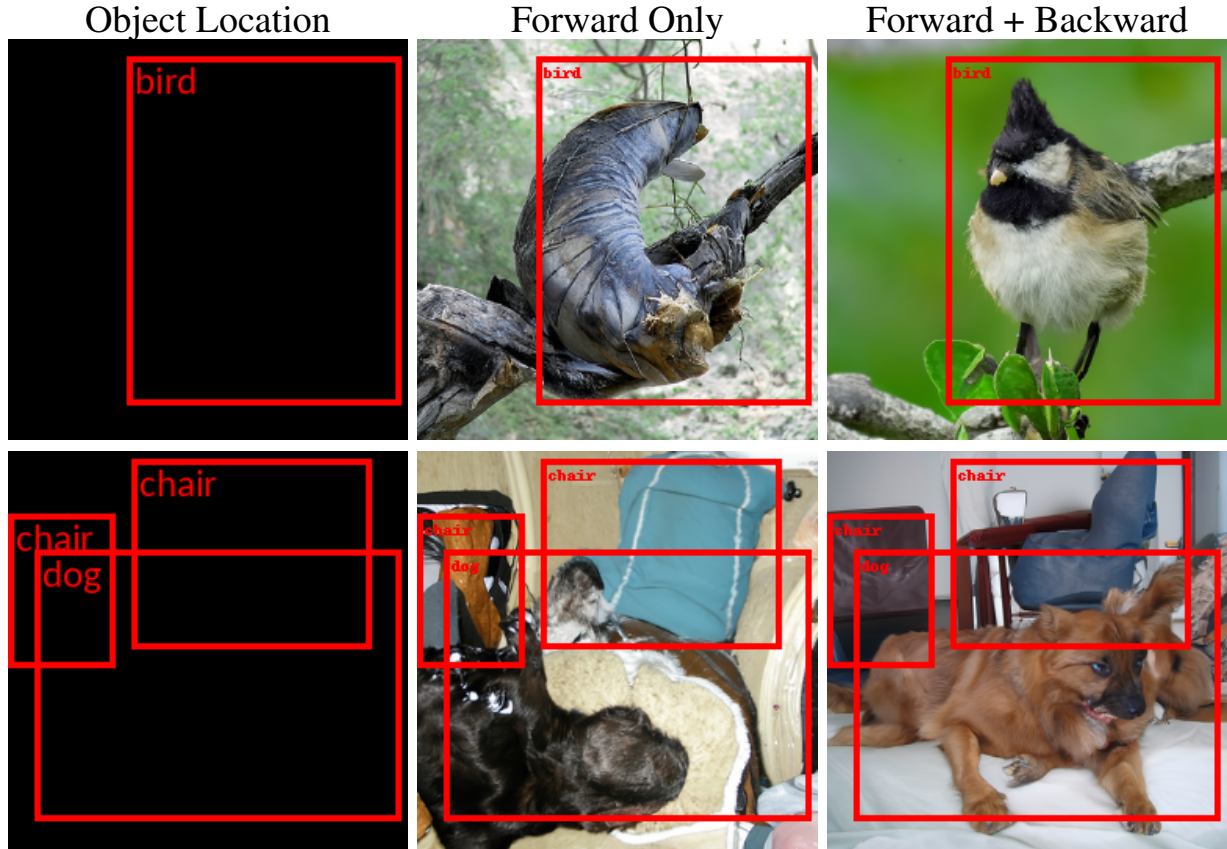


Figure 5.7: Generation guided by object detection with the unconditional ImageNet model. While both forward and backward guidance produces realistic images with the desired objects in the designated locations, forward guidance alone produces the wrong objects or the wrong locations/sizes.

images contain contents that match the given text prompts, while exhibiting style that matches the given style images.

5.4.2 Results for ImageNet Diffusion

In this section, we present results for guided image generation using an unconditional diffusion model trained on ImageNet. We experiment with object location guidance and a hybrid guided image generation task which we term segmentation-guided inpainting. We will discuss results of each guidance separately.

Object Location Guidance.

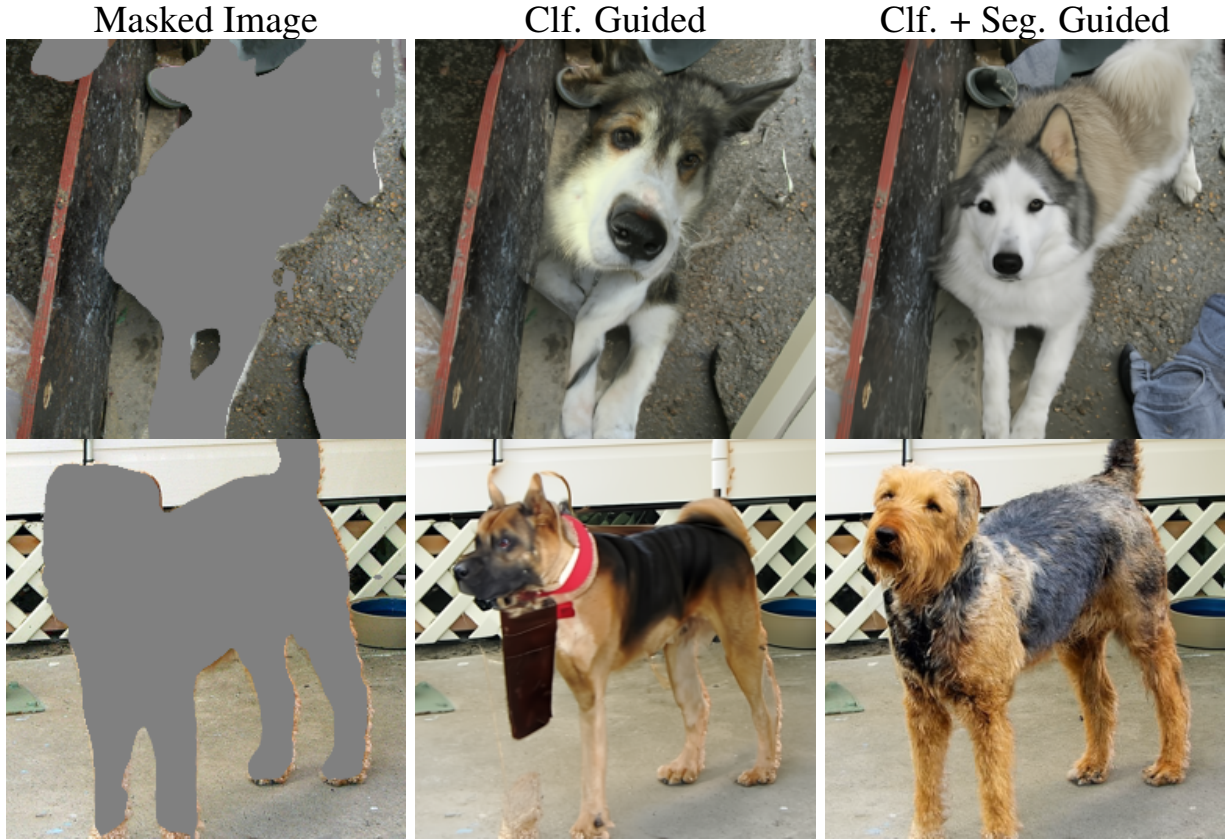


Figure 5.8: Our guidance algorithm can incorporate feedback from multiple guidance functions. Left to right: The inpainting prompt; The classifier-guided inpainting output; Images generated with both classifier and segmentation guidance, where realistic dogs are generated exactly on the mask.

Similar to object location guidance for Stable Diffusion, we also use the same network architecture and the same pre-trained model as our object detection network, and construct an identical loss function ℓ for our guidance algorithm. However, unlike Stable

Object Location	Fwd. Only	Fwd. + Bkd.
Bounding box 1	0.39	0.90
Bounding box 2	0.18	0.36

Table 5.2: Quantitative analysis of forward guidance only versus combination of forward and backward guidance for object detection guidance on ImageNet with bounding boxes in Figure 5.7. The metric is mAP@50.

Diffusion, object locations are the only prompts available for guided image generation. We experiment with different object location prompts using either (1) only forward universal guidance and (2) both forward and backward universal guidance. We observe from Figure 5.7

that applying both forward and backward guidance generates images that are realistic and the objects match the prompt nicely. On the other hand, while images generated using only forward guidance remain realistic, they feature objects with mismatching categories and locations. The observation is further backed by quantitative evaluation presented in Table 5.2. The evaluation is based on mAP@50 between the ground truth object locations and the predicted bounding boxes of generated images, and clearly shows that the combination of forward and backward guidance leads to a much better match with the constraint. The results demonstrate the effectiveness of our universal guidance algorithm, and also validate the necessity of our backward guidance.

Segmentation-Guided Inpainting. In this experiment, we aim to explore the ability of our algorithm to handle multiple guidance functions. We perform guided image generation with combined guidance from an inpainting mask, a classifier and a segmentation network. We first generate images with masked regions as the prompt for inpainting. We then pick an object class c as the prompt for classification and generate a segmentation mask where the masked regions are considered foreground objects of the same class c . We use ℓ_2 loss on the non-masked region as the loss function for inpainting, and set the corresponding $s(t) = 0$, or equivalently only use backward guidance for inpainting. We use the same segmentation network as described in Section 5.4.1. For classification guidance, we use the classifier that accepts noisy input [95], and perform the original classifier guidance as Equation (5.3.1.1) instead of our forward guidance. The results in Figure 5.8 show that when using both inpainting and classifier as guidance, our algorithm generates realistic images that both match the inpainting prompt and are classified correctly to the given object class. Adding in segmentation guidance, our algorithm further improves by making a near-perfect match to both the segmentation map and the inpainting prompt

while maintaining realism, demonstrating that our algorithm effectively combines feedback from multiple guidance functions.

Part V

AI Reasoning + Generation

Chapter 6: DualCode: Discrete Embeddings for Unified Image Generation and Reasoning

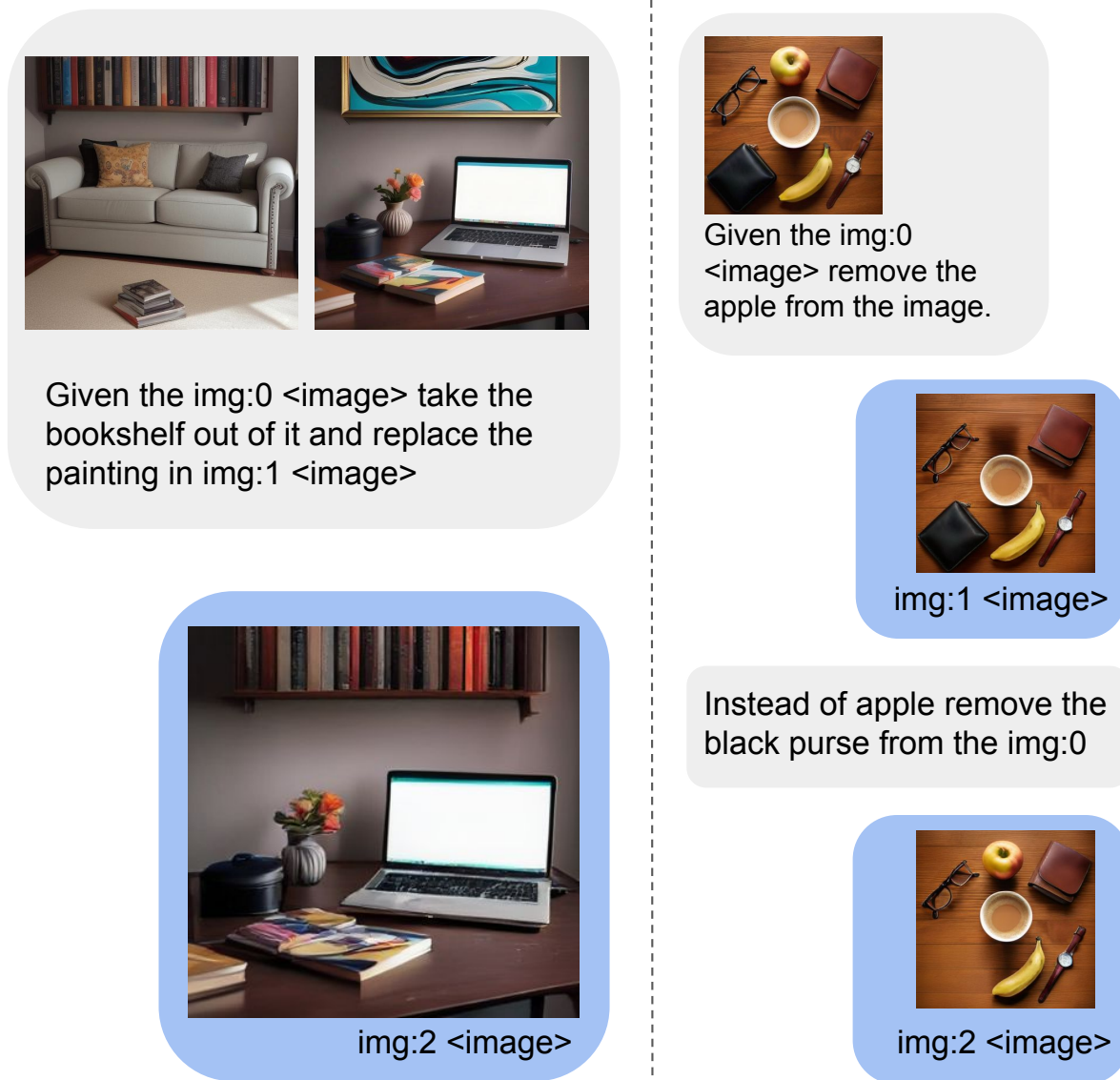


Figure 6.1: Demonstrating the capability of DualCode embeddings for image editing. On the **left**, it transfers a bookshelf from one image to another, replacing the painting. In multi-turn editing (**right**), when asked to remove the apple, it performs the task correctly. Furthermore, when asked to remove the black purse instead, the autoregressive nature of the model allows it to retain the history of inputs, remembering the removed apple and thus only removing the black purse.

6.1 Introduction

Multimodal Large Language Models (MLLMs) [134, 135, 136, 137, 138, 139, 140, 141, 142, 143] are capable of generating both text and images and hence need to support complex tasks such as image reasoning and high-precision image editing. These models can thus describe images through text generation and perform detailed modifications based on editing instructions. While there are multiple ways to design such models, adopting a decoder-only transformer architecture, similar to large language models (LLMs), has proven highly effective. To align with this architecture, vision embeddings must be represented as sequences of embeddings, ensuring compatibility with next-token prediction tasks and enabling a unified approach for text and image generation.

Current vision-language models (VLMs) [136, 137, 142] often use CLIP[144, 145, 146] embeddings, which are optimized for semantic understanding and language alignment. These continuous embeddings can afford to be lossy because they prioritize semantic comprehension over preserving detailed visual information, making them suitable for reasoning tasks. However, high-precision image editing demands embeddings that retain detailed visual information for accurate reconstruction and enable token-based processing. Models like Chameleon [143] use VQGAN [147] embeddings to meet this requirement, as they provide discrete, lossless representations that maintain intricate visual details essential for editing. While VQGAN embeddings excel in visual fidelity, they lack the semantic alignment needed for understanding complex instructions. Conversely, CLIP embeddings align well with language but sacrifice detailed visual information.

SEED-X [140] addresses the need for lossless embeddings by training an autoregressive model to produce continuous embeddings, which condition diffusion models alongside the input

image for high-precision image editing. However, SEED-X [140] is not a decoder-only model and, thus, lacks the scalability of LLM-based architectures. Furthermore, its reliance on manual source image selection at decoding leads to challenges in multi-image and iterative editing, as it risks forgetting previous inputs. To achieve a truly scalable, decoder-only architecture that preserves every detail of past image inputs, a unified embedding is required—one that combines the high-fidelity reconstruction of VQGAN with the semantic alignment strengths of CLIP, making it both adaptable and efficient for complex tasks.

Training a vision encoder from scratch that blends these properties with language alignment would require significant computational resources and massive datasets. To address this challenge, we introduce **DualCode**, a simple, lightweight framework that extends existing VQGAN embeddings by incorporating semantic tokens derived from models such as CLIP [145] and DINOv2 [148]. This approach bridges the gap between detailed visual reconstruction and semantic understanding without the need for costly retraining.

To this end, our contributions are as follows:

- We introduce **DualCode**, a simple and lightweight framework that constructs unified embeddings by augmenting pre-trained VQGAN [149] embeddings with semantic tokens from CLIP [145] and DINOv2 [148], effectively combining high visual fidelity with deep semantic understanding.
- We demonstrate the superior performance of **DualCode** compared to standalone VQGAN [149] embeddings in controlled settings, keeping variables such as pretrained LLMs, model sizes, data, and compute constant.
- We propose an approach to augment instruction-tuning datasets for image reasoning and

editing using GroundingDino for generating weakly labeled data from image-caption pairs.

- Finally, we show **DualCode**’s effectiveness against popular autoregressive models like SEED-X [140] and Chameleon [143], proving its robustness and versatility across diverse tasks.

6.2 Background

MLLMs [134, 135, 136, 137, 138, 139, 140, 141, 142, 143] typically use an autoregressive backbone where transformers are trained to generate both text and image outputs. While text is commonly processed with well-established discrete representations, the optimal approach for handling image data remains an active area of research. Continuous and discrete embedding strategies each offer distinct advantages and unique challenges.

In Continuous embedding input setups [136, 149, 150], images are encoded into continuous embeddings via a visual encoder, allowing these embeddings to be interwoven with text tokens for joint autoregressive modeling. The leverage pre-trained models like CLIP [145] to capture high-level semantic information, facilitating complex image reasoning tasks. Works like [136, 137, 151] explore strategies to better align image and text embeddings in a shared space to improve multimodal coherence. LLaVA [136, 137] adopts a straightforward approach, by training a projection matrix to align CLIP embeddings with text while Brave [151] combines multiple vision embeddings using MEQ-former. Thus given the vision embeddings $\mathbf{X}_v$ and its corresponding caption $\mathbf{X}_c$, θ , is optimized by minimizing the cross-entropy loss:

$$L_{\text{align}} = - \sum_{i=1}^L \log p_{\theta}(c_i | \mathbf{X}_v, \mathbf{X}_{c < i}) \quad (6.2.0.1)$$

where c_i represents the i -th token in the caption $\mathbf{X}_c$.

When it comes to generating the image, models are typically trained to regress on target continuous embeddings [149, 150]. However, these embeddings are inherently lossy thus compromising finer image features and limiting their effectiveness for high-precision image editing. This limitation is addressed by SEED-X [140], which introduces a ‘*source image*’ during the decoding stage. The source image, typically the last generated one in automated workflows or provided by the user, enables SEED-X [140] to preserve details that may be lost in CLIP encoding. This approach balances high-level semantic coherence with essential image details, enhancing continuity in image editing tasks. Nevertheless, in multi-turn image editing, this design choice continues to lose information from previous images, as it relies on a single, most recent image to preserve details.

In a discretized representation setup, there is a unified training objective, allowing for smoother integration of image and text within an autoregressive framework and making them easier to train and scale, using decoder-only architectures. These discrete embeddings [152, 153, 154, 155, 156] can be lossy as well, which are then passed through a diffusion model as well, to generate the final image. While models like Chameleon [143] and Emu3 [141] employ discrete, lossless embeddings derived from VQGANs to retain detailed visual information for image reasoning and high-precision image editing.

VQGANs [144, 149, 157] offer a powerful approach to construct discrete, lossless image embeddings. Typically, it consists of an image encoder, decoder, and quantization module. The loss function can be written as:

$$L_{VQ} = L_{\text{recon}} + L_{\text{adv}} + L_{\text{quant}} \quad (6.2.0.2)$$

where $L_{\text{recon}}(x, \hat{x})$ represents the reconstruction loss between the input image x and its reconstruction, L_{adv} denotes the adversarial loss, and $L_{\text{quant}} = \|z_e(x) - \text{sg}(z_q)\|^2 + \beta \|z_q - \text{sg}(z_e(x))\|^2$ is the quantization loss. Here, $z_e(x)$ is the encoder output, z_q the quantized latent vector, and $\text{sg}(\cdot)$ is the stop-gradient operator, which restricts updates to backpropagate.

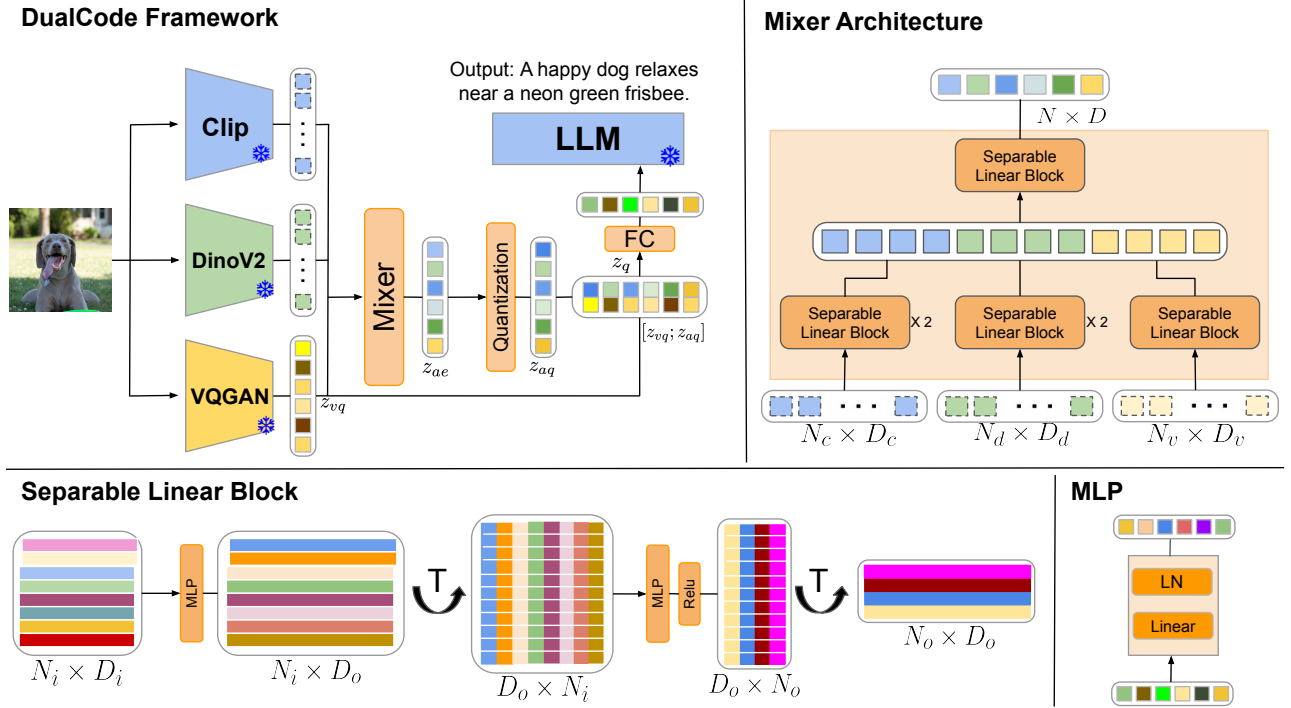


Figure 6.2: **Overview of the Unified Vision Embedding Framework:** **Top Left:** The DualCode framework integrates features from CLIP, DinoV2, and VQGAN through a Mixer architecture, generating unified embeddings of size $N_v \times D$. A quantization module processes these embeddings to produce discrete outputs, which are then concatenated with the original VQGAN embeddings and projected into the LLM input space via fully connected layers. **Top Right:** The Mixer architecture combines input embeddings from different sources (e.g., CLIP: $N_c \times D_c$, DinoV2: $N_d \times D_d$, VQGAN: $N_v \times D_v$) into a cohesive embedding of size $N \times D$. **Bottom Left:** The Mixer architecture uses Separable Linear Blocks, each comprising two MLPs. The first MLP projects the input from $N_i \times D_i$ to $N_i \times D_o$, followed by a transpose operation. The second MLP transforms the transposed input from $D_o \times N_i$ to $D_o \times N_o$ before transposing it back to $N_o \times D_o$, ensuring adaptable and consistent embedding transformation. **Bottom Right:** Each MLP block within the Separable Linear Block includes a linear layer followed by layer normalization, facilitating stability and efficient processing.

6.3 DualCode

Decoder-only models can benefit from discrete vision embeddings, enabling more efficient training through large-scale pretraining with a cross-entropy-based loss, similar to LLMs. Current models [141, 143] often rely on VQGAN embeddings for their high fidelity in image reconstruction, which is essential for high-precision image editing. However, while these embeddings retain detailed visual information, they underperform in vision-language reasoning tasks compared to models with other pre-trained embeddings, like CLIP [145], due to a lack of semantic alignment. This gap highlights the need for discrete embeddings that not only preserve visual fidelity but also achieve language alignment, supporting both precise image editing and comprehensive image reasoning. This can be achieved by modifying the VQGAN loss function (6.2.0.2) by adding the alignment loss from (6.2.0.1). Thus, formally the unified loss function L_{ULF} as:

$$L_{\text{ULF}} = L_{\text{recon}} + L_{\text{adv}} + L_{\text{quant}} + L_{\text{align}} \quad (6.3.0.1)$$

where the image embedding used in L_{align} are z_q . Developing such embeddings from scratch would require vast resources, with datasets surpassing 100 million image-caption pairs, posing a significant computational challenge. To address this issue we propose **DualCode**, where instead of training a new model from scratch, we enhance the discrete embeddings by incorporating semantic features from models such as CLIP [145] and DINOv2 [148] in conjunction to using VQGAN [149] embeddings. This approach results in a **unified vision embedding** that retains VQGAN’s high-fidelity visual details while embedding the semantic richness necessary for robust image understanding and editing tasks making it a lightweight and computationally

efficient framework.

We construct DualCode as shown in Figure 6.2, where we extend the existing VQGAN codebook, which has a length of C_v , by adding a new set of language-aligned discrete embeddings z_{aq} derived from a new codebook of length C_a constructed from different vision embeddings. These two embeddings from two different codebooks are concatenated in parallel, resulting in a combined codebook of effective length $C_v \times C_a$. This structure splits the discrete embedding z_q into two components: z_{vq} , derived from a frozen VQGAN, and z_{aq} , trained for language alignment. The unified loss function L_{ULF} can thus be expressed as:

$$L_{\text{ULF}} = L_{\text{recon}} + L_{\text{adv}} + L_{\text{quant}}^{vq} + L_{\text{quant}}^{aq} + L_{\text{align}} \quad (6.3.0.2)$$

Here, L_{recon} and L_{adv} correspond to the frozen VQGAN discrete embeddings z_{vq} and are already optimized, ensuring high-quality image reconstruction. The quantization loss L_{quant} is split into L_{quant}^{vq} for z_{vq} , which preserves visual fidelity, and L_{quant}^{aq} for z_{aq} , which is minimized jointly with the alignment loss L_{align} to ensure semantic consistency with text. The alignment loss operates on the concatenated embedding $z_q = [z_{vq}; z_{aq}]$, optimizing only z_{aq} while keeping z_{vq} frozen. This design allows DualCode to achieve discrete, lossless, and semantically enriched embeddings efficiently, making it suitable for tasks requiring detailed image reconstruction and language alignment.

To construct z_{aq} , as seen in Figure 6.2, the Mixer architecture is used to combine multiple vision embeddings of varying lengths and dimensions into a consistent output size of $N_o \times D_o$. However, directly training a linear layer that projects inputs from $N_i \times D_i$ to $N_o \times D_o$ or a transformer-like architecture would require a large parameter space, making it computationally

expensive. To address this, we employ **Separable Linear Blocks** within the Mixer to break the transformation into two sequential linear operations, significantly reducing computational costs. Each Separable Linear Block first projects the input from $N_i \times D_i$ to $N_i \times D_o$, adjusting only the feature dimension. The intermediate result is then transposed to $D_o \times N_i$, allowing the second linear layer to project it to $D_o \times N_o$, modifying the sequence length while keeping the feature dimension consistent. Finally, the output is transposed back to $N_o \times D_o$ as desired.

Once the Mixer processes these unified embeddings, z_a is quantized into discrete representations z_{aq} using the alignment codebook of length C_a , which is then projected into the input space of the LLM to minimize L_{align} using a projection matrix. This unified embedding, constructed with approximately 35M additional parameters for $D = 16$ and $C_a = 8192$, leverages existing image features to create a lightweight and easily trainable framework. tures, making the framework lightweight and easy to train.

6.4 Experiments

In this section, we evaluate DualCode’s performance on image reasoning and image editing tasks, comparing it to models that rely solely on VQGAN embeddings. To isolate DualCode’s impact, we perform experiments in a controlled research environment, maintaining other variables such as the pre-trained LLM backbone, model sizes, training data, compute resources, evaluation data, and evaluation strategies constant. This controlled setup enables us to assess DualCode’s unique contributions to semantic alignment and image fidelity, independent of external factors.

6.4.1 Training Methodology

Our training process consists of two key stages: **Stage I** focuses on building DualCode, and as discussed in Section 6.3, our design choice requires training only the language-aligned embeddings z_{aq} . Similar to Stage I training in [136, 137], this stage optimizes z_{aq} and the projection matrix W , which is applied to the concatenated embedding $z_q = [z_{vq}; z_{aq}]$ to create continuous embeddings in the LLM input space by minimizing the entropy of generating image captions. This stage, therefore, establishes discrete representations along with their corresponding continuous embeddings. For a fair comparison with models that use only VQGAN embeddings, we also train continuous embeddings derived solely from VQGAN’s discrete tokens, ensuring both models operate under similar compute and data conditions. This stage provides the foundation for language-driven image reasoning and editing. The pre-trained VQGAN model used in our experiments is from LlamaGen [149], which performs 16x downsampling, and the CLIP model is CLIP-ViT-L-336px as in [136]. All experiments are conducted with an image resolution of 336x336 pixels.

In **Stage II**, we build on the language-aligned embeddings from Stage I to enable image reasoning and editing. In Stage I, z_q was optimized while the LLM remained frozen; in this stage, however, we reverse this setup by freezing z_q and unfreezing the LLM, allowing it to be fully trained. This stage aims to train two multi-modal models—one using DualCode embeddings and the other using only VQGAN embeddings—to enable a fair comparison between the two embedding approaches.

To handle multi-modal input in Stage II, we use continuous embeddings obtained through the projection matrix W and add additional trainable embeddings to ensure no loss of visual

information. For the model using only VQGAN embeddings, we represent the image as a sequence of discrete tokens $z_q = [z_{vq}]$ along with their corresponding one-hot encoding i_v . The continuous embedding is defined as $e_c = W_z \cdot z_q + W_i \cdot i_v$, where $W_i \cdot i_v$ is implemented as a PyTorch embedding layer. With VQGAN’s codebook size of 16,384 unique tokens [149] and an embedding dimension of 4096, resulting in 67.2M parameters.

For the model trained with DualCode embeddings, this approach becomes computationally intensive, as the combined codebook size is $C_v \times C_a$. Setting $C_a = 8192$ results in approximately 134M unique tokens, which would require an estimated 552B parameters if each token had unique trainable embeddings. To optimize, we treat z_{vq} and z_{aq} separately in the concatenated embedding $z_q = [z_{vq}; z_{aq}]$, resulting in $e_c = W_z \cdot [z_{vq}; z_{aq}] + W_{iv} \cdot i_v + W_{ia} \cdot i_a$. The additional parameter count comes from the new embedding $W_{ia} \cdot i_a$, bringing the total to 100.74M parameters—an increase of 33M relative to the VQGAN-only model, which is very small given the 7B parameter LLM. Thus DualCode model uses three embedding layers: one for text and two for visual embeddings, while the VQGAN-only model requires only two embedding layers.

Similarly, on the **output side**, we retain the original language head and add three additional heads for the DualCode embeddings: one for VQGAN token prediction, one for image-aligned token prediction, and a signal head that indicates whether the output should be text or image, directing the LLM to the appropriate output head based on the task. For the model trained with only VQGAN embeddings, a total of three heads are required.

6.4.2 Training Data

In **Stage I**, we train DualCode by optimizing z_{aq} on a dataset of 10 million image-caption pairs. For a fair comparison with VQGAN embeddings [149], we also train a projection matrix on VQGAN embeddings to create continuous embeddings in the LLM input space, using the same data.

In **Stage II**, we train two models—one using DualCode embeddings and the other using only VQGAN embeddings—so that each can handle both image reasoning and editing tasks in a unified manner. Constructing a dataset for both tasks is challenging, as combining existing instruction-tuning datasets yields fewer than 5 million examples. To address this, we introduce an intermediate training phase using weak labels generated by GroundingDino [158], a tool that detects objects in images based on captions. This intermediate data provides information about object localization, spatial relationships, and basic image editing operations such as object-guided inpainting, removal, and color adjustments. We provide the full list of tasks used to generate weak labels in the Appendix. By generating weak labels without manual intervention, this phase allows us to expand the training set to 30 million examples, improving the model’s performance in both image reasoning and editing.

After training on weak labels, we further refine the model with high-quality instruction-tuning datasets that combine image editing and reasoning data. For image reasoning, we use images and questions from the LLaVa-Next dataset [136] and generate answers using Meta’s open-source Llama 3.2 (11B) model [159], as internal restrictions prevent us from using answers generated by OpenAI models. For image editing, we integrate datasets from SEED-X [140], instructPix2Pix [160], and MagicBrush [161] for single-image editing tasks.

For multi-image generation, we create synthetic datasets where variations x_1 and x_2 are derived from a base image x_0 . Specifically, we detect and select an object in x_0 , producing x_2 by inpainting the region after removing the object from x_0 , and creating x_1 by placing the object on an empty background with outpainting. The multi-image task then requires generating an image like x_0 but with the object from x_1 positioned as in x_2 . For multi-turn editing, we construct a challenging dataset that requires preserving past image inputs. For example, the model might be instructed to remove object A from x_0 to create x_1 , followed by an instruction to remove object B instead of object A, resulting in x_2 . This setup encourages the model to retain information about previous edits, as generating x_2 requires recalling details about the previously removed object A.

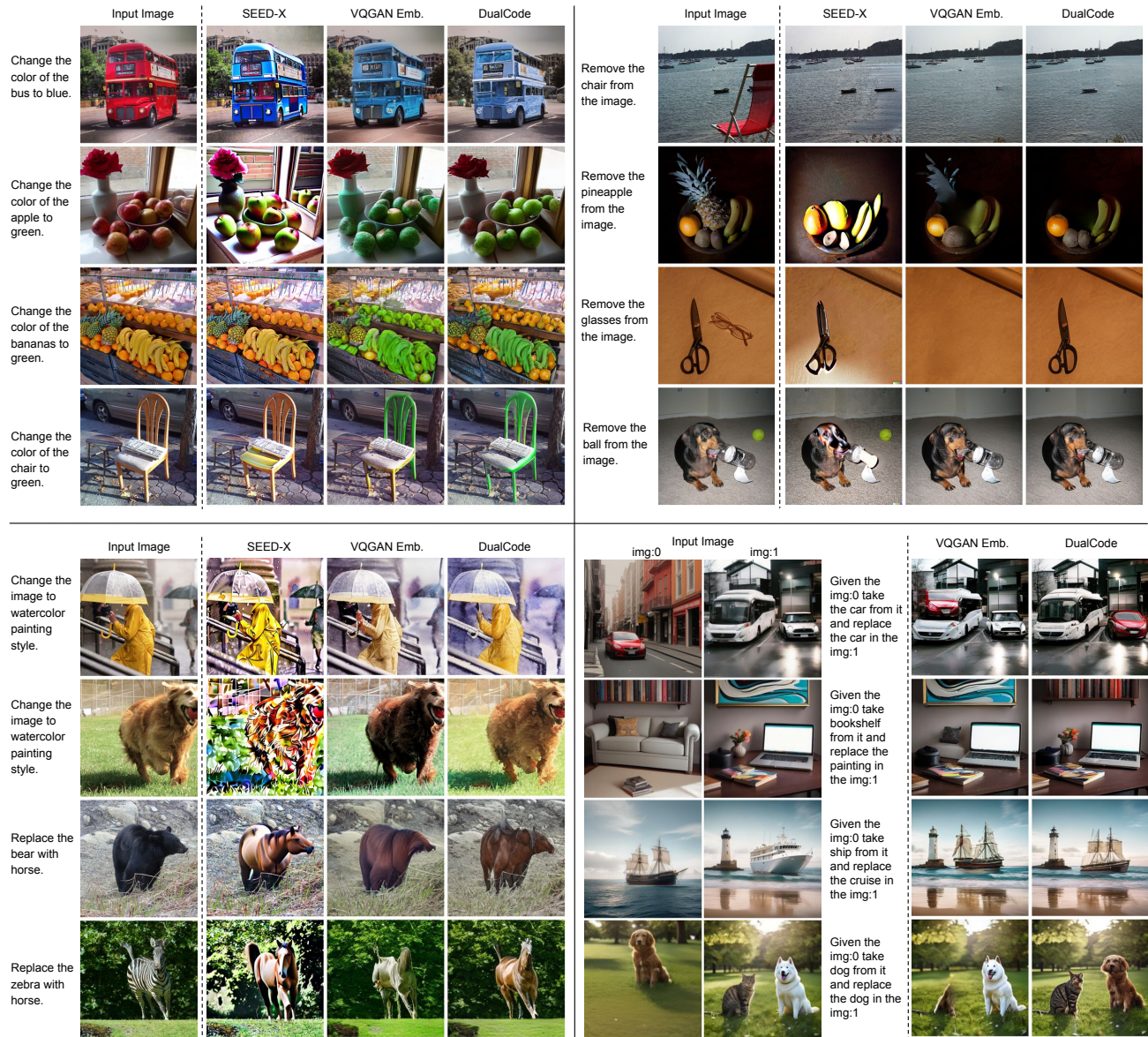


Figure 6.3: Comparison of DualCode, VQ-GAN, and SEED-X-Edit models across various tasks. **Top Left:** DualCode achieves precise color alteration, outperforming VQ-GAN in consistency and SEED-X in content preservation. **Top Right:** DualCode excels in object removal with clean inpainting, while VQ-GAN struggles with precision. **Bottom Left:** DualCode produces natural style transfer results and effective object replacements, though SEED-X outshines in the latter task due to its pretrained SDXL model. **Bottom Right:** DualCode demonstrates superior multi-image editing capabilities, surpassing VQ-GAN. SEED-X is excluded from multi-turn editing due to its single image-to-image training limitations.

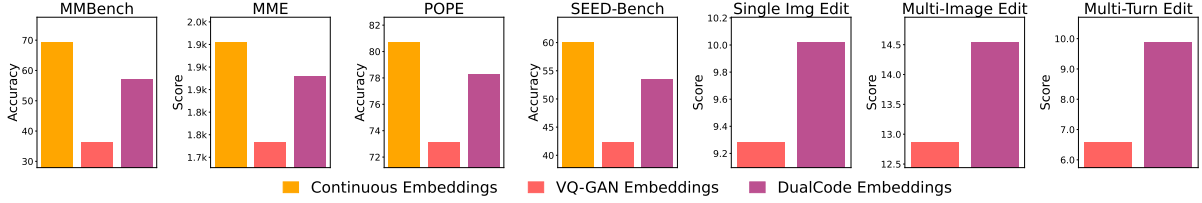


Figure 6.4: Performance of various vision embeddings: LLM aligned Continuous Embeddings, Discrete VQ-GAN embeddings, and DualCode Embeddings. For image reasoning tasks, performance is evaluated using four benchmarks: MME, MMBench, POPE, and MMStar. For image editing tasks, we evaluate the embeddings on three scenarios: Single Image-to-Image, Multi-Image, and Multi-Turn Image Editing. The editing capabilities are assessed using Gemini-1.5-Pro and GPT-4o-mini. Continuous embeddings are lossy and hence unsuitable for high-precision image editing.

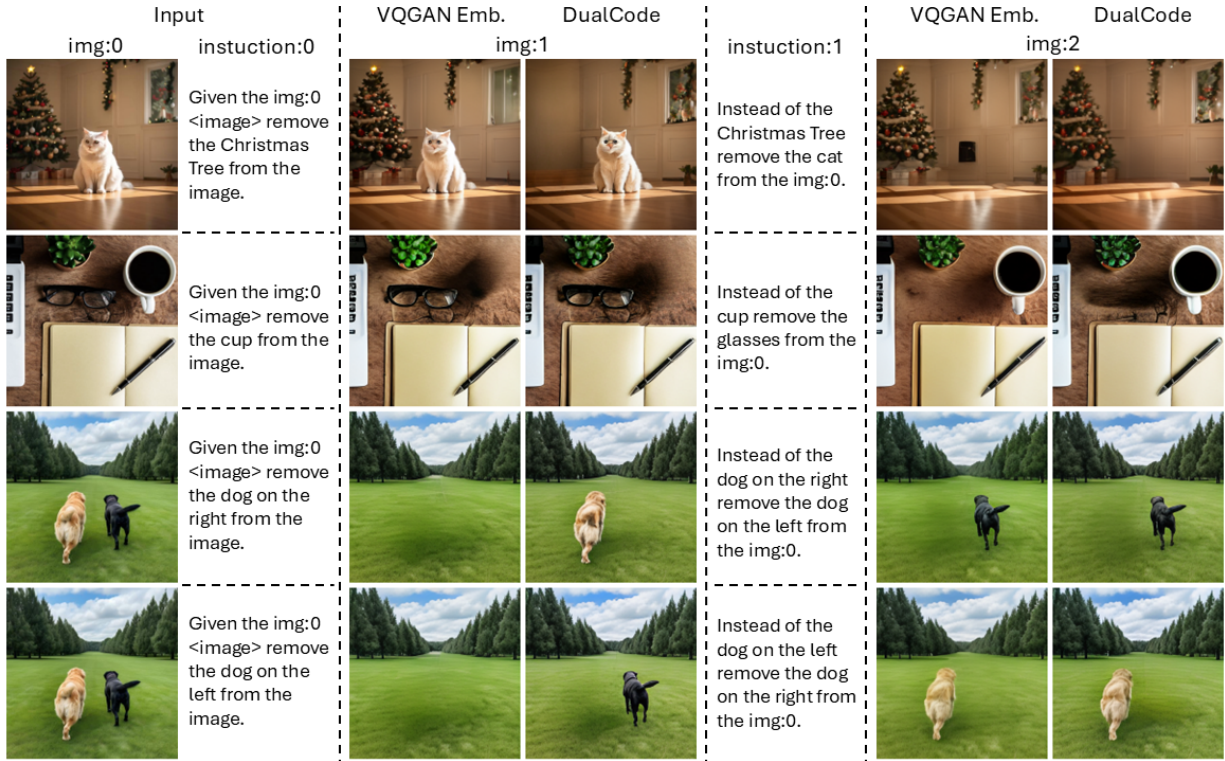


Figure 6.5: Demonstrating multi-turn image editing with end-to-end autoregressive models trained on VQGAN and DualCode embeddings. Unlike traditional models, which modify only the last image and ‘forget’ previous edits, an end-to-end autoregressive model retains image history and attends to prior edits. In this example, the model first removes object A (e.g., the Christmas tree, cup, or dog), then removes object B instead of A, all while referencing the original image. DualCode outperforms VQGAN, which sometimes misses the target object or removes unintended elements, showing superior performance in multi-turn image editing.

6.5 Results

In this section, we evaluate and present results for both image editing and reasoning capabilities. For image reasoning, we utilize four benchmarks: MMBench [162], MME [163], POPE [164], and SEED-Bench [165], which assess the embeddings’ capacity to understand and reason about visual content. The image editing tasks include Single Image-to-Image Editing, Multi-Image Editing, and Multi-Turn Editing. These tasks are evaluated using Gemini-1.5-Pro [166] and GPT-4o-mini [167] APIs, which provide scores between 1 and 10, following a methodology similar to [168, 169]. The sum of both scores is reported. For Single Image-to-Image Editing, we use the dataset from [168], focusing particularly on images in categories such as Color Alteration, Object Removal, Style Alteration, and Object Replacement. For the multi-turn and multi-image editing tasks, testing data is carefully generated locally similar to training data as discussed in 6.4.2 due to the absence of standard evaluation benchmarks.

We initially demonstrate the impact of training with weakly labeled data, combined with high-quality image editing and reasoning instruction data, as shown in the first two rows of Tables 6.1 and 6.2. Quantitative results show that the models trained with DualCode embeddings improve significantly when weakly labeled data is included, highlighting the effectiveness of such data. Thus, in subsequent experiments, we present results for models trained using weakly labeled data.

To assess the influence of vision embeddings on image reasoning and editing performance, we compare three types of vision embeddings: Continuous Embeddings (based on CLIP [145] and DINOv2 [148] features), VQ-GAN Embeddings [149], and our proposed DualCode. To minimize the influence of external factors, we ensure consistency in model size, data, and compute

Model	MMB	MME	POPE	SEED-B
DualCode (No WLT)	<u>63.0</u>	1583.93	65.9	51.85
DualCode	57.0	1880.57	<u>78.3</u>	<u>53.57</u>
SEED-X-I	77.4	1858.0	85.13	66.8
SEED-X-Edit	25.7	605.3	50.00	25.3
Chameleon-7B	27.30	524.26	70.45	27.09

Table 6.1: Comparison of DualCode with other models across four image reasoning tasks. Results are shown for DualCode with and without weakly labeled training, alongside SEED-X-I and SEED-X-Edit optimized for reasoning and editing, and Chameleon-7B. Best results are **bold**, second best are underlined.

Model	Color	Replace	Style	Removal	Avg.
DualCode (No WLT)	5.66	3.55	12.70	7.70	7.60
DualCode	10.08	<u>6.93</u>	14.42	8.89	10.06
VQ-GAN	<u>9.24</u>	5.13	<u>13.89</u>	<u>8.80</u>	<u>9.26</u>
SEED-X-Edit	7.36	7.21	12.32	7.07	8.49

Table 6.2: Performance comparison of models across image editing tasks: Color Alteration, Object Replacement, Style Alteration, and Object Removal. DualCode is evaluated in two versions: with and without weakly labeled data. Comparisons include VQ-GAN and SEED-X-Edit. Scores are generated using Gemini-1.5-Pro and GPT-4o-mini. Best results are **bold**, second best are underlined.

resources while training separate models for VQ-GAN embeddings and DualCode embeddings.

Each of these models is trained on both image reasoning and editing tasks as discussed in Section 6.4.2. For Continuous Embeddings, we train only on image reasoning data, as they are unsuitable for high-precision image editing due to their lossy nature.

As shown in Figure 6.4, Continuous Embeddings perform the best in image reasoning across the benchmarks, consistently outperforming both VQ-GAN and DualCode embeddings. While DualCode embeddings are slightly behind Continuous Embeddings, they still perform competitively and show a significant improvement over VQ-GAN embeddings, demonstrating the benefits of language-aligned discrete vision embeddings. Furthermore, even for image editing tasks, DualCode embeddings consistently outperform VQ-GAN embeddings.

We also compare the performance of our model trained with DualCode embeddings against

other autoregressive models designed for both image reasoning and editing. Table 6.1 shows a comparison with SEED-X [140] and Chameleon-7B [143]. DualCode, a unified model for both tasks, significantly outperforms Chameleon-7B and achieves performance comparable to SEED-X-I [140], which is specifically optimized for image reasoning. In Table 6.2, our trained models achieve higher scores compared to SEED-X-Edit [140] across various categories in single image-to-image tasks.

The qualitative results are presented in Figure 6.3 and Figure 6.5. In Figure 6.3, we compare the image editing capabilities of models that were trained with VQGAN and DualCode embeddings in a controlled research environment. We find that, in most cases, DualCode outperforms models trained with VQGAN embeddings. This is largely due to the ability of the DualCode embeddings to accurately identify object boundaries, resulting in high-precision image editing, whether it involves single image-to-image or multi-image editing. We also compare the performance of SEED-X [140] for single image-to-image editing and observe that DualCode achieves better results in all categories except when asked to replace objects, which aligns with our quantitative results. In Figure 6.5, it is evident that DualCode embeddings also deliver superior performance in multi-turn image editing compared to the VQGAN embeddings. The open-source model SEED-X-Edit [140] is not trained for multi-turn editing, so a direct comparison was not possible.

Note: Models with lossy image embeddings require a source image during decoding to retain information that would otherwise be lost, as outlined in SEED-X [140]. In complex editing tasks, selecting this source image is challenging and often requires manual intervention. Automating this process by always using the latest generated image fails in multi-turn edits that require references to earlier images. For example, in Figure 6.5, if object A is removed in the first step, a second instruction to “remove object B instead of A” lacks spatial details about A,

highlighting the limitations of lossy embeddings in such scenarios. Consequently, even a well-trained model may struggle with these tasks due to the inherent limitations of this design choice.

6.6 Conclusion

Large Language Models (LLMs) have demonstrated impressive capabilities in text reasoning and editing, and building on this foundation, multi-modal LLMs hold great promise. However, while text can easily be converted into discrete sequences without losing details, achieving the same for images is challenging, especially when finer details must be preserved. VQGAN offers a straightforward approach for obtaining discrete lossless representations, but they fall short in reasoning tasks compared to CLIP. In this paper, we do not aim state-of-the-art MLLM, rather, we investigate whether VQGAN embeddings are truly the best choice for discrete lossless image representations. We hypothesize a better performance can be achieved if embeddings are semantically aligned as well. While training such models from scratch with alignment loss could address this, it is resource-intensive. To tackle this, we propose DualCode, built on existing VQGAN and semantically aligned embeddings. They are easy to train and significantly outperform VQGAN in reasoning and editing tasks, paving the way toward building discrete, lossless, and semantically aligned image embeddings for MLLMs.

Bibliography

- [1] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Hui Wu, Marc Ph Stoecklin, Heqing Huang, and Ian Molloy. Protecting intellectual property of deep neural networks with watermarking. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 159–172, 2018.
- [2] Dario Amodei and Danny Hernandez. Ai and compute. *Heruntergeladen von <https://blog.openai.com/aiand-compute>*, 2018.
- [3] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.
- [4] Chuan Li. *OpenAI’s GPT-3 Language Model: A Technical Overview*, 2020. URL <https://lambdalabs.com/blog/demystifying-gpt-3/#1>.
- [5] Frank Hartung and Martin Kutter. Multimedia watermarking techniques. *Proceedings of the IEEE*, 87(7):1079–1107, 1999.
- [6] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin’ichi Satoh. Embedding watermarks into deep neural networks. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval*, pages 269–277, 2017.
- [7] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. Turning your weakness into a strength: Watermarking deep neural networks by backdooring. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1615–1631, 2018.
- [8] Masoumeh Shafieinejad, Jiaqi Wang, Nils Lukas, Xinda Li, and Florian Kerschbaum. On the robustness of the backdoor-based watermarking in deep neural networks. *arXiv preprint arXiv:1906.07745*, 2019.
- [9] William Aiken, Hyounghick Kim, and Simon Woo. Neural network laundering: Removing black-box backdoor watermarks from deep neural networks. *arXiv preprint arXiv:2004.11368*, 2020.

- [10] Bitá Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. Deepsigns: A generic watermarking framework for ip protection of deep learning models. *arXiv preprint arXiv:1804.00750*, 2018.
- [11] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness, 2019.
- [12] Guy Katz, Derek A Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pages 443–452. Springer, 2019.
- [13] Ping-yeh Chiang, Michael J Curry, Ahmed Abdelkader, Aounon Kumar, John Dickerson, and Tom Goldstein. Detection as regression: Certified object detection by median smoothing. *arXiv preprint arXiv:2007.03730*, 2020.
- [14] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified adversarial robustness via randomized smoothing. *arXiv preprint arXiv:1902.02918*, 2019.
- [15] Yogesh Balaji, Tom Goldstein, and Judy Hoffman. Instance adaptive adversarial training: Improved accuracy tradeoffs in neural nets. *arXiv preprint arXiv:1910.08051*, 2019.
- [16] Sven Gowal, Krishnamurthy Dvijotham, Robert Stanforth, Rudy Bunel, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.
- [17] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Adversarial generative nets: Neural network attacks on state-of-the-art face recognition. *CoRR*, abs/1801.00349, 2018. URL <http://arxiv.org/abs/1801.00349>.
- [18] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.
- [19] Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. *Advances in Neural Information Processing Systems*, 34:6695–6706, 2021.
- [20] Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Arpit Bansal, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. Datasets for studying generalization from easy to hard examples, 2021.
- [21] Felix A Gers and E Schmidhuber. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, 2001.

- [22] Jürgen Schmidhuber, Daan Wierstra, Matteo Gagliolo, and Faustino Gomez. Training recurrent networks by evolino. *Neural computation*, 19(3):757–779, 2007.
- [23] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014.
- [24] Lukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.
- [25] Karlis Freivalds and Renars Liepins. Improving the neural gpu architecture for algorithm learning. *arXiv preprint arXiv:1702.08727*, 2017.
- [26] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.
- [27] Jürgen Schmidhuber. Self-delimiting neural networks. *arXiv preprint arXiv:1210.0118*, 2012.
- [28] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.
- [29] Cristobal Eyzaguirre and Alvaro 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.
- [30] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2018.
- [31] Ofir Press, Noah Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2021.
- [32] Marco Ciccone, Marco Gallieri, Jonathan Masci, Christian Osendorfer, and Faustino Gomez. Nais-net: Stable deep networks from non-autonomous differential equations. *Advances in Neural Information Processing Systems*, 31, 2018.
- [33] Kwwabena Nuamah. Deep algorithmic question answering: Towards a compositionally hybrid AI for algorithmic reasoning. *arXiv preprint arXiv:2109.08006*, 2021.
- [34] Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. In *32nd Conference on Neural Information Processing Systems*, pages 3368–2278. Neural Information Processing Systems Foundation, 2018.
- [35] Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021.
- [36] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

- [37] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.
- [38] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.
- [39] Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach*, volume 5.01. GMD-Forschungszentrum Informationstechnik Bonn, 2002.
- [40] Richard E Ladner and Michael J Fischer. Parallel prefix computation. *Journal of the ACM (JACM)*, 27(4):831–838, 1980.
- [41] John Loeber. #16: Notes on Arithmetic in GPT-4, February 2024. URL <https://loeber.substack.com/p/16-notes-on-arithmetic-in-gpt-4>.
- [42] Nayoung Lee, Kartik Sreenivasan, Jason D Lee, Kangwook Lee, and Dimitris Papailiopoulos. Teaching arithmetic to small transformers. *arXiv preprint arXiv:2307.03381*, 2023.
- [43] Ruochi Shen, Sébastien Bubeck, Ronen Eldan, Yin Tat Lee, Yuanzhi Li, and Yi Zhang. Positional description matters for transformers arithmetic. *arXiv preprint arXiv:2311.14737*, 2023.
- [44] Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Josh Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? a study in length generalization. *arXiv preprint arXiv:2310.16028*, 2023.
- [45] Yongchao Zhou, Uri Alon, Xinyun Chen, Xuezhi Wang, Rishabh Agarwal, and Denny Zhou. Transformers can achieve length generalization but not robustly. *arXiv preprint arXiv:2402.09371*, 2024.
- [46] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*, 2019.
- [47] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.
- [48] Siavash Golkar, Mariel Pettee, Michael Eickenberg, Alberto Bietti, Miles Cranmer, Geraud Krawezik, Francois Lanusse, Michael McCabe, Ruben Ohana, Liam Parker, et al. xval: A continuous number encoding for large language models. *arXiv preprint arXiv:2310.02989*, 2023.
- [49] Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jian, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D Hwang, et al. Faith and fate: Limits of transformers on compositionality. *arXiv preprint arXiv:2305.18654*, 2023.

- [50] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [51] OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023. URL <https://api.semanticscholar.org/CorpusID:257532815>.
- [52] Cem Anil, Yuhuai Wu, Anders Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. *Advances in Neural Information Processing Systems*, 35:38546–38556, 2022.
- [53] Samy Jelassi, Stéphane d’Ascoli, Carles Domingo-Enrich, Yuhuai Wu, Yuanzhi Li, and François Charton. Length generalization in arithmetic transformers. *arXiv preprint arXiv:2306.15400*, 2023.
- [54] Zhen Yang, Ming Ding, Qingsong Lv, Zhihuan Jiang, Zehai He, Yuyi Guo, Jinfeng Bai, and Jie Tang. Gpt can solve mathematical problems without a calculator. *arXiv preprint arXiv:2309.03241*, 2023.
- [55] Petar Veličković, Adrià Puigdomènech Badia, David Budden, Razvan Pascanu, Andrea Banino, Misha Dashevskiy, Raia Hadsell, and Charles Blundell. The clrs algorithmic reasoning benchmark. In *International Conference on Machine Learning*, pages 22084–22102. PMLR, 2022.
- [56] Gleb Rodionov and Liudmila Prokhorenkova. Discrete neural algorithmic reasoning. *arXiv preprint arXiv:2402.11628*, 2024.
- [57] Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive attention span in transformers. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 331–335, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1032. URL <https://aclanthology.org/P19-1032>.
- [58] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020.
- [59] Borja Ibarz, Vitaly Kurin, George Papamakarios, Kyriacos Nikiforou, Mehdi Bennani, Róbert Csordás, Andrew Joseph Dudzik, Matko Bošnjak, Alex Vitvitskyi, Yulia Rubanova, et al. A generalist neural algorithmic learner. In *Learning on graphs conference*, pages 2–1. PMLR, 2022.
- [60] Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. *Advances in Neural Information Processing Systems*, 34:6695–6706, 2021.

- [61] Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Logical extrapolation without overthinking. *Advances in Neural Information Processing Systems*, 35, 2022.
- [62] Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pages 11398–11442. PMLR, 2023.
- [63] Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. *arXiv preprint arXiv:2311.12424*, 2023.
- [64] Artur Back de Luca and Kimon Fountoulakis. Simulation of graph algorithms with looped transformers. *arXiv preprint arXiv:2402.01107*, 2024.
- [65] Sean McLeish, Avi Schwarzschild, and Tom Goldstein. Benchmarking chatgpt on algorithmic reasoning. *arXiv preprint arXiv:2404.03441*, 2024.
- [66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.
- [67] Ofir Press, Noah Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations*, 2022. URL <https://openreview.net/forum?id=R8sQPpGCv0>.
- [68] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.
- [69] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140): 1–67, 2020.
- [70] Ta-Chung Chi, Ting-Han Fan, Alexander Rudnicky, and Peter Ramadge. Dissecting transformer length extrapolation via the lens of receptive field analysis. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13522–13537, 2023.
- [71] Ta-Chung Chi, Ting-Han Fan, Peter Ramadge, and Alexander Rudnicky. Kerple: Kernelized relative positional embedding for length extrapolation. In *Advances in Neural Information Processing Systems*, 2022.
- [72] Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers. *arXiv preprint arXiv:2305.19466*, 2023.

- [73] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568: 127063, 2024.
- [74] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.
- [75] Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. *International Conference on Learning Representations*, 2024.
- [76] Shanda Li, Chong You, Guru Guruganesh, Joshua Ainslie, Santiago Ontanon, Manzil Zaheer, Sumit Sanghai, Yiming Yang, Sanjiv Kumar, and Srinadh Bhojanapalli. Functional interpolation for relative positions improves long context transformers. *arXiv preprint arXiv:2310.04418*, 2023.
- [77] Anian Ruoss, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Róbert Csordás, Mehdi Bennani, Shane Legg, and Joel Veness. Randomized positional encodings boost length generalization of transformers. *arXiv preprint arXiv:2305.16843*, 2023.
- [78] Jonas Geiping and Tom Goldstein. Cramming: Training a language model on a single gpu in one day. In *International Conference on Machine Learning*, pages 11117–11143. PMLR, 2023.
- [79] Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. Mega: moving average equipped gated attention. *arXiv preprint arXiv:2209.10655*, 2022.
- [80] Cem Anil, Ashwini Pople, Kaiqu Liang, Johannes Treutlein, Yuhuai Wu, Shaojie Bai, J Zico Kolter, and Roger B Grosse. Path independent equilibrium models can better exploit test-time computation. *Advances in Neural Information Processing Systems*, 35:7796–7809, 2022.
- [81] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.
- [82] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, volume 37 of *JMLR Workshop and Conference Proceedings*, 2015.
- [83] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019.
- [84] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 32, 2020.

- [85] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *International Conference on Learning Representations*, 2021.
- [86] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8162–8171, 2021.
- [87] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *International Conference on Learning Representations*, 2021.
- [88] Diederik P. Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. On density estimation with diffusion models. *Advances in Neural Information Processing Systems*, 34, 2021.
- [89] Zahra Kadkhodaie and Eero Simoncelli. Stochastic solutions for linear inverse problems using the prior implicit in a denoiser. *Advances in Neural Information Processing Systems*, 34, 2021.
- [90] Yaniv Romano, Michael Elad, and Peyman Milanfar. The little engine that could: Regularization by denoising (RED). *arXiv preprint arXiv:1611.02862*, 2016.
- [91] Christopher A. Metzler, Ali Mousavi, and Richard G. Baraniuk. Learned D-AMP: principled neural network based compressive image recovery. *Advances in Neural Information Processing Systems*, 30, 2017.
- [92] Jay Whang, Mauricio Delbracio, Hossein Talebi, Chitwan Saharia, Alexandros G. Dimakis, and Peyman Milanfar. Deblurring via stochastic refinement. *arXiv preprint arXiv:2112.02475*, 2021.
- [93] Bahjat Kawar, Gregory Vaksman, and Michael Elad. Stochastic image denoising by sampling from the posterior distribution. *International Conference on Computer Vision Workshops*, 2021.
- [94] Chitwan Saharia, Jonathan Ho, William Chan, Tim Salimans, David J. Fleet, and Mohammad Norouzi. Image super-resolution via iterative refinement. *arXiv preprint arXiv:2104.07636*, 2021.
- [95] Prafulla Dhariwal and Alexander Quinn Nichol. Diffusion models beat gans on image synthesis. volume 34, 2021.
- [96] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *arXiv preprint arXiv:2206.00364*, 2022.
- [97] Bahjat Kawar, Michael Elad, Stefano Ermon, and Jiaming Song. Denoising diffusion restoration models. *arXiv preprint arXiv:2201.11793*, 2022.
- [98] Severi Rissanen, Markus Heinonen, and Arno Solin. Generative modelling with inverse heat dissipation. *arXiv preprint arXiv:2206.13397*, 2022.

- [99] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in neural information processing systems*, 29, 2016.
- [100] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. Mode regularized generative adversarial networks. *arXiv preprint arXiv:1612.02136*, 2016.
- [101] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.
- [102] Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying mmd gans. *arXiv preprint arXiv:1801.01401*, 2018.
- [103] Shane Barratt and Rishi Sharma. A note on the inception score. *arXiv preprint arXiv:1801.01973*, 2018.
- [104] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.
- [105] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.
- [106] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.
- [107] Alexia Jolicoeur-Martineau, Rémi Piché-Taillefer, Ioannis Mitliagkas, and Remi Tachet des Combes. Adversarial score matching and improved sampling for image generation. *International Conference on Learning Representations*, 2021.
- [108] Jonathan Ho, Chitwan Saharia, William Chan, David J. Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation. *J. Mach. Learn. Res.*, 23, 2022.
- [109] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.
- [110] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.
- [111] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of CVPR*, 2022.
- [112] Arpit Bansal, Eitan Borgnia, Hong-Min Chu, Jie S Li, Hamid Kazemi, Furong Huang, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Cold diffusion: Inverting arbitrary image transforms without noise. *arXiv preprint arXiv:2208.09392*, 2022.

- [113] Zhifeng Kong, Wei Ping, Jiayi Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis. *arXiv preprint arXiv:2009.09761*, 2020.
- [114] Rongjie Huang, Max WY Lam, Jun Wang, Dan Su, Dong Yu, Yi Ren, and Zhou Zhao. Fastdiff: A fast conditional diffusion model for high-quality speech synthesis. *arXiv preprint arXiv:2204.09934*, 2022.
- [115] Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in Neural Information Processing Systems*, 34:17981–17993, 2021.
- [116] Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B Hashimoto. Diffusion-lm improves controllable text generation. *arXiv preprint arXiv:2205.14217*, 2022.
- [117] Jay Whang, Mauricio Delbracio, Hossein Talebi, Chitwan Saharia, Alexandros G Dimakis, and Peyman Milanfar. Deblurring via stochastic refinement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16293–16303, 2022.
- [118] Weilun Wang, Jianmin Bao, Wengang Zhou, Dongdong Chen, Dong Chen, Lu Yuan, and Houqiang Li. Semantic image synthesis via diffusion models. *arXiv preprint arXiv:2207.00050*, 2022.
- [119] Yuheng Li, Haotian Liu, Qingyang Wu, Fangzhou Mu, Jianwei Yang, Jianfeng Gao, Chunyuan Li, and Yong Jae Lee. Gligen: Open-set grounded text-to-image generation. *arXiv preprint arXiv:2301.07093*, 2023.
- [120] Lvmin Zhang and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. *arXiv preprint arXiv:2302.05543*, 2023.
- [121] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.
- [122] Yinhuai Wang, Jiwen Yu, and Jian Zhang. Zero-shot image restoration using denoising diffusion null-space model. *arXiv preprint arXiv:2212.00490*, 2022.
- [123] Hyungjin Chung, Jeongsol Kim, Michael T Mccann, Marc L Klasky, and Jong Chul Ye. Diffusion posterior sampling for general noisy inverse problems. *arXiv preprint arXiv:2209.14687*, 2022.
- [124] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11461–11471, 2022.

- [125] Hyungjin Chung, Byeongsu Sim, Dohoon Ryu, and Jong Chul Ye. Improving diffusion models for inverse problems using manifold constraints. *arXiv preprint arXiv:2206.00941*, 2022.
- [126] Alexandros Graikos, Nikolay Malkin, Nebojsa Jojic, and Dimitris Samaras. Diffusion models as plug-and-play priors. *arXiv preprint arXiv:2206.09012*, 2022.
- [127] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. IEEE, 2009.
- [128] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1314–1324, 2019.
- [129] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.
- [130] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE signal processing letters*, 23(10):1499–1503, 2016.
- [131] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.
- [132] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.
- [133] Yanghao Li, Saining Xie, Xinlei Chen, Piotr Dollar, Kaiming He, and Ross Girshick. Benchmarking detection transfer learning with vision transformers. *arXiv preprint arXiv:2111.11429*, 2021.
- [134] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International Conference on Machine Learning*, pages 12888–12900. PMLR, 2022.
- [135] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*, 2023.
- [136] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. *arXiv preprint arXiv:2310.03744*, 2023.

- [137] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *arXiv preprint arXiv:2304.08485*, 2023.
- [138] Zhiliang Peng, Wenhui Wang, Li Dong, Yaru Hao, Shaohan Huang, Shuming Ma, and Furu Wei. Kosmos-2: Grounding multimodal large language models to the world. *arXiv preprint arXiv:2306.14824*, 2023.
- [139] Pan Zhang, Xiaoyi Dong Bin Wang, Yuhang Cao, Chao Xu, Linke Ouyang, Zhiyuan Zhao, Shuangrui Ding, Songyang Zhang, Haodong Duan, Hang Yan, et al. Internlm-xcomposer: A vision-language large model for advanced text-image comprehension and composition. *arXiv preprint arXiv:2309.15112*, 2023.
- [140] Yuying Ge, Sijie Zhao, Jinguo Zhu, Yixiao Ge, Kun Yi, Lin Song, Chen Li, Xiaohan Ding, and Ying Shan. Seed-x: Multimodal models with unified multi-granularity comprehension and generation. *arXiv preprint arXiv:2404.14396*, 2024.
- [141] Xinlong Wang, Xiaosong Zhang, Zhengxiong Luo, Quan Sun, Yufeng Cui, Jinsheng Wang, Fan Zhang, Yuezhe Wang, Zhen Li, Qiyang Yu, et al. Emu3: Next-token prediction is all you need. *arXiv preprint arXiv:2409.18869*, 2024.
- [142] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. 2023.
- [143] Chameleon Team. Chameleon: Mixed-modal early-fusion foundation models. *arXiv preprint arXiv:2405.09818*, 2024.
- [144] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.
- [145] Mehdi Cherti, Romain Beaumont, Ross Wightman, Mitchell Wortsman, Gabriel Ilharco, Cade Gordon, Christoph Schuhmann, Ludwig Schmidt, and Jenia Jitsev. Reproducible scaling laws for contrastive language-image learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2818–2829, 2023.
- [146] Quan Sun, Yuxin Fang, Ledell Wu, Xinlong Wang, and Yue Cao. Eva-clip: Improved training techniques for clip at scale. *arXiv preprint arXiv:2303.15389*, 2023.
- [147] Oran Gafni, Adam Polyak, Oron Ashual, Shelly Sheynin, Devi Parikh, and Yaniv Taigman. Make-a-scene: Scene-based text-to-image generation with human priors. In *European Conference on Computer Vision*, pages 89–106. Springer, 2022.
- [148] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

- [149] Peize Sun, Yi Jiang, Shoufa Chen, Shilong Zhang, Bingyue Peng, Ping Luo, and Zehuan Yuan. Autoregressive model beats diffusion: Llama for scalable image generation. *arXiv preprint arXiv:2406.06525*, 2024.
- [150] Quan Sun, Qiyang Yu, Yufeng Cui, Fan Zhang, Xiaosong Zhang, Yueze Wang, Hongcheng Gao, Jingjing Liu, Tiejun Huang, and Xinlong Wang. Generative pretraining in multimodality. *arXiv preprint arXiv:2307.05222*, 2023.
- [151] Oğuzhan Fatih Kar, Alessio Tonioni, Petra Poklukar, Achin Kulshrestha, Amir Zamir, and Federico Tombari. Brave: Broadening the visual encoding of vision-language models. *arXiv preprint arXiv:2404.07204*, 2024.
- [152] Yuying Ge, Sijie Zhao, Ziyun Zeng, Yixiao Ge, Chen Li, Xintao Wang, and Ying Shan. Making llama see and draw with seed tokenizer. *arXiv preprint arXiv:2310.01218*, 2023.
- [153] Yuying Ge, Yixiao Ge, Ziyun Zeng, Xintao Wang, and Ying Shan. Planting a seed of vision in large language model. *arXiv preprint arXiv:2307.08041*, 2023.
- [154] Jiasen Lu, Christopher Clark, Sangho Lee, Zichen Zhang, Savya Khosla, Ryan Marten, Derek Hoiem, and Aniruddha Kembhavi. Unified-io 2: Scaling autoregressive multimodal models with vision, language, audio, and action. *arXiv preprint arXiv:2312.17172*, 2023.
- [155] Lili Yu, Bowen Shi, Ramakanth Pasunuru, Benjamin Muller, Olga Golovneva, Tianlu Wang, Arun Babu, Binh Tang, Brian Karrer, Shelly Sheynin, et al. Scaling autoregressive multi-modal models: Pretraining and instruction tuning. *arXiv preprint arXiv:2309.02591*, 2023.
- [156] Yang Jin, Kun Xu, Liwei Chen, Chao Liao, Jianchao Tan, Bin Chen, Chenyi Lei, An Liu, Chengru Song, Xiaoqiang Lei, et al. Unified language-vision pretraining with dynamic discrete visual tokenization. *arXiv preprint arXiv:2309.04669*, 2023.
- [157] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021.
- [158] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.
- [159] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- [160] Tim Brooks, Aleksander Holynski, and Alexei A Efros. Instructpix2pix: Learning to follow image editing instructions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18392–18402, 2023.

- [161] Kai Zhang, Lingbo Mo, Wenhui Chen, Huan Sun, and Yu Su. Magicbrush: A manually annotated dataset for instruction-guided image editing. *Advances in Neural Information Processing Systems*, 36, 2024.
- [162] Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. Mmbench: Is your multi-modal model an all-around player? In *European Conference on Computer Vision*, pages 216–233. Springer, 2025.
- [163] Shukang Yin, Chaoyou Fu, Sirui Zhao, Ke Li, Xing Sun, Tong Xu, and Enhong Chen. A survey on multimodal large language models. *arXiv preprint arXiv:2306.13549*, 2023.
- [164] Yifan Li, Yifan Du, Kun Zhou, Jinpeng Wang, Wayne Xin Zhao, and Ji-Rong Wen. Evaluating object hallucination in large vision-language models. *arXiv preprint arXiv:2305.10355*, 2023.
- [165] Bohao Li, Yuying Ge, Yixiao Ge, Guangzhi Wang, Rui Wang, Ruimao Zhang, and Ying Shan. Seed-bench: Benchmarking multimodal large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13299–13308, 2024.
- [166] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.
- [167] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [168] Yiwei Ma, Jiayi Ji, Ke Ye, Weihuang Lin, Zhibin Wang, Yonghan Zheng, Qiang Zhou, Xiaoshuai Sun, and Rongrong Ji. I2ebench: A comprehensive benchmark for instruction-based image editing. *arXiv preprint arXiv:2408.14180*, 2024.
- [169] Yi Huang, Jiancheng Huang, Yifan Liu, Mingfu Yan, Jiaxi Lv, Jianzhuang Liu, Wei Xiong, He Zhang, Shifeng Chen, and Liangliang Cao. Diffusion model-based image editing: A survey. *arXiv preprint arXiv:2402.17525*, 2024.