

©Copyright 2026

Caleb Zheng

Linearization of Neural Networks as a Dual Framework for Stability and Sparsity

Caleb Zheng

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2026

Reading Committee:

Eli Shlizerman, Chair

Linda Shapiro

Payman Arabshahi

Program Authorized to Offer Degree:
Electrical & Computer Engineering

University of Washington

Abstract

Linearization of Neural Networks as a Dual Framework for Stability and Sparsity

Caleb Zheng

Chair of the Supervisory Committee:
Professor Eli Shlizerman
Department of Electrical & Computer Engineering

Neural networks are powerful but difficult to reason about. Understanding why a model is stable, why it is trained effectively, or how it is being compressed without performance degradation requires tools that go beyond inspecting weights or monitoring loss curves. Linearization, which characterizes how a nonlinear system responds to small perturbations, offers a principled foundation for these questions. While a well known tool in fields such as control and dynamical systems, its potential as a unifying design and optimization language for neural networks has remained largely unexplored. This dissertation develops linearization as a dual framework for networks of neurons, addressing two complementary challenges: 1) Stability and hyperparameter sensitivity of Recurrent Neural Networks (RNNs), and 2) Performance recovery of compressed generative diffusion models.

My thesis introduces the following contributions to the above goals. For RNNs, I use linearization as a design target. By shaping the eigenvalue spectrum of the recurrent Jacobian at initialization, I proposed to obtain networks whose dynamics remain stable across a broad range of regimes, making FORCE-based training robust to network configuration rather than confined to a narrow edge-of-chaos band. Turning from design to analysis, I then show that the Lyapunov Exponent spectrum, a signature of a network’s long-run dynamical behavior derived from its linearization, encodes rich information about model performance. I introduce a representation learning approach that maps these spectra into a compact latent space, enabling reliable cross-architecture performance prediction from early

in training and substantially reducing the cost of hyperparameter search. Building on this, I further leverage the Lyapunov Spectrum distance as a more reliable proxy than loss-based metrics for guiding automated search over pruned RNN variants, consistently identifying better-performing configurations at lower computational cost.

For diffusion models, I demonstrate that Jacobian matching, aligning the input-output derivatives between dense teacher and compressed student, provides an effective finetuning signal that standard training objectives do not exploit. Since naïve Jacobian matching is incompatible with diffusion training’s inherent noise injection, I develop a second-order extension inspired by finite-time Lyapunov exponents that resolves this incompatibility and recovers generation quality after pruning. Together, these contributions establish linearization as a versatile theoretical language for both analysis and deployment of neural networks.

TABLE OF CONTENTS

	Page
List of Figures	iii
Chapter 1: Introduction	1
1.1 Motivation	1
1.2 Summary of Research Contributions	3
1.3 Thesis Outline	6
Chapter 2: Preliminaries	7
2.1 Recurrent Neural Networks	7
2.2 FORCE Learning	10
2.3 Transformers and Attention	11
2.4 Diffusion Generative Models	12
2.5 Dynamical Systems and Lyapunov Theory	18
2.6 Knowledge Distillation	22
2.7 Network Pruning	25
Chapter 3: Related Work	28
3.1 RNN Stability and Initialization	28
3.2 Hyperparameter Search	30
3.3 Representation Learning	32
3.4 Generative Modeling and Diffusion Models	33
3.5 Neural Networks as Dynamical Systems	36
3.6 Knowledge Distillation	39
3.7 Network Pruning	41
Chapter 4: R-FORCE: Robust Training of Random Recurrent Neural Networks through Jacobian Eigenvalue Shaping	43
4.1 Motivation	43
4.2 Method: R-FORCE Initialization	44
4.3 Experiments	50

Chapter 5:	AeLLE: Lyapunov-guided Representation of Recurrent Neural Network Performance	55
5.1	Motivation	55
5.2	Method	56
5.3	Experiments	59
Chapter 6:	Hyperpruning: Efficient Search through Pruned Variants of Recurrent Neural Networks Leveraging Lyapunov Spectrum	66
6.1	Motivation	67
6.2	Method	68
6.3	Experiments	71
Chapter 7:	2ndMatch: Finetuning Pruned Diffusion Models via Second-Order Jacobian Matching	77
7.1	Motivation	77
7.2	Method	79
7.3	Experiments	82
Chapter 8:	Conclusion	88
8.1	Contributions	88
8.2	Synthesis	89
8.3	Limitations	90
8.4	Future Directions	91
Appendix A:	Appendix	122
A.1	Derivation of the DDPM training objective	122
A.2	AeLLE	125
A.3	Hyperpruning	130
A.4	2ndMatch	135

LIST OF FIGURES

Figure Number		Page
1.1	Overview of the four roles linearization plays across the contributions of this dissertation. Role I (Design Target): the input Jacobian is shaped at initialization to promote stable training dynamics in recurrent networks [CZ 1]. Role II (Predictive Features): Lyapunov Exponents serve as compact representations for predicting RNN performance [CZ 2]. Role III (Dynamical Compass): the Lyapunov Spectrum distance between a dense model and its pruned variants guides hyperparameter search [CZ 3]. Role IV (Optimization Objective): second-order Jacobian matching inspired by FTLE [CZ 4] supervises the finetuning of pruned diffusion models.	4
2.1	Unfolding a recurrent neural network through time. The same set of parameters is reused at each timestep to update the hidden state, and unfolding the recurrence reveals the sequential chain of transformations applied to produce the output.	8
2.2	Self-attention and transformer layer. Left: Self-attention mechanism. Each input token is projected into query, key, and value vectors. Query-key similarities determine attention weights, which produce weighted mixtures of the value vectors. Right: A transformer block consisting of multi-head attention followed by a feedforward network, each wrapped with residual connections and layer normalization.	13
2.3	Forward and reverse diffusion. Data is gradually perturbed into noise in the forward process, and the learned reverse process reconstructs the sample through a multi-step denoising trajectory, making diffusion sampling computationally expensive. The forward and reverse illustrations use two CC0 images from Wikimedia Commons, “Siberian Husky (Unsplash).jpg” and “Siberian-husky-1291343 1920.jpg,” respectively. The intermediate noisy images were synthetically generated by the author.	16
2.4	Local sensitivity in a dynamical system. Three initially close points (A, B, C) evolve under a time-varying flow field. Although their starting positions differ only slightly, the resulting trajectories (A', B', C') diverge rapidly, especially near separatrices where the flow exhibits strong deformation. Such divergence motivates the use of Lyapunov-based measures to quantify stability and sensitivity [199].	23

2.5	Jacobian matching is equivalent to output matching with input noise. With limited calibration points, many student functions (pink) fit the teacher (blue) at the observed locations but diverge elsewhere (top two rows). Matching outputs at noise-perturbed inputs (Solution 1) enforces consistency in a neighborhood around each point; directly matching input Jacobians (Solution 2) achieves the same effect, since the two objectives are equivalent to first order in the noise magnitude.	25
2.6	Unstructured, neuron-level, and filter-level structured pruning. Unstructured pruning removes individual weights (left), neuron pruning removes whole neurons in fully connected layers (middle), and filter pruning removes entire convolutional filters (right). Structured sparsity forms dense, hardware-friendly subnetworks that enable speedups in CNNs and diffusion models.	26
3.1	Comparison of generative modeling paradigms. GANs rely on adversarial training between a generator and a discriminator. VAEs optimize a variational lower bound on the data likelihood. Diffusion models gradually corrupt data with noise and learn to reverse the denoising process, achieving superior quality and training stability at the cost of slower inference.	34
4.1	R-FORCE Initialization Properties. (A) Spectral distribution of M_R . (B) Q-Q plot of M_R entries (blue) versus a standard Gaussian (red). (C) Evolution of $\lambda_{J(t)}$ during pre-training (top), training (middle), and testing (bottom), shown relative to the stability boundary (dashed red). Zoomed-in views highlight the right boundary. All plots use $g = 1.5$	45
4.2	Generation of body joints movements with Rank1-RRNN. (A) Rank1-RRNN with 66 readout units for the generation of joint movements time series. (B) Errors of Transfer-FORCE (orange), FORCE (blue), Full-FORCE (cyan), and R-FORCE (green) in generating the evolution of the deep squat movement. Each dot corresponds to a readout unit result. The red cross denotes the readout unit with the largest error (worst case) for each method. (C) The evolution of the readout unit in the worst case with the ground-truth target (red).	48
4.3	Spiking neural network with Rank1-RRNN. (Left) MAE of FORCE (blue) and R-FORCE(green) in generating the spectrum of birdsong. Each dot corresponds to a readout unit, and black bars are 99% confidence interval. (Middle) Comparison between FORCE (blue) and R-FORCE(green) of a randomly selected band over flat and peak regions against ground truth (red). (Right) Spectrum comparison among FORCE (top row), R-FORCE(middle row) prediction, and ground truth (bottom row). The right column is the zoom-in plot of the selected area (red rectangle) in the Left column.	52

5.1	Clustering across network architecture: AeLLE for networks trained on SMNIST Task. <i>Left:</i> In this task, the network predicts the digit shown in a given image. Higher losses correspond to lower confidence in the correct digit label. <i>Center:</i> AeLLE representation shows that networks of similar error are clustered together. When classifying these networks by high accuracy (green) and low accuracy (red), high-accuracy networks, regardless of architecture, are consistently located farther to the right (more positive PC1) than low-accuracy networks. <i>Right:</i> Networks of the same type often clustered together in AeLLE representation, with some overlap between similar architectures (such as coRNN and ASRNN). For each network architecture (except vanilla RNN), there are variants with high and low accuracy that are separated along the first PC dimension.	58
5.2	Clustering by performance across network size: AeLLE for LSTMs trained on CharRNN. <i>Left:</i> The CharRNN task predicts the next character in a sequence; the heatmaps show the predicted character distribution for models with different test errors, where larger errors correspond to lower confidence in the true next character. <i>Center:</i> AeLLE embedding (first two principal components) colored by accuracy. High-accuracy networks (green) cluster to the left of the median PC ₁ boundary (black line) and low-accuracy networks (red) to the right; the stacked histogram below shows the same separation. <i>Right:</i> The same embedding colored by network size (64, 128, 256, 512). The sizes are broadly mixed throughout the space, with larger networks covering a wider range but still overlapping smaller ones, so network size alone does not separate performance.	60
5.3	AeLLE on Sequential MNIST. <i>Left:</i> Example SMNIST inputs (row-wise sequences) and corresponding prediction heatmaps for models with different test errors. <i>Middle:</i> AeLLE embedding (first two principal components) colored by accuracy; high- and low-accuracy networks separate along PC ₁ (black line). The bottom panel shows stacked histograms along PC ₁ for the two groups. <i>Right:</i> Same embedding colored by architecture (LSTM, GRU, RNN, ASRNN, coRNN, LipschitzRNN, NoisyRNN, LEM), with stacked histograms highlighting how different model families populate the AeLLE space.	61
6.1	(A) Lyapunov Spectrum (LS) curves of the dense network (black) and two pruned variants—Pruned 1 (green) and Pruned 2 (red)—evaluated at pre-training and at epochs 1, 3, and 5 (left to right). The dense network serves as a reference trajectory in LS space. Arrows indicate the deviation magnitude from the dense model. (B) Temporal evolution of the L_2 distance between each pruned variant and the dense network in LS space throughout training. (C) Perplexity curves of the two pruned variants over training epochs.	66

6.2	LS-based Hyperpruning: starts with a set of candidates (blue) and a pre-trained dense reference model (black), LSH executes candidate removal and generation. Candidate Removal: Each candidate is trained for E epochs and projected to the LS space, along with the reference. Candidates with further distance (light blue) are removed based on their distance to the reference. Candidate Generation: New candidates (brown) are adaptively generated based on remaining candidates and inserted into the candidate set. After several iterations of removal and generation, the remaining candidates in the set are extensively trained, and the best one (green) is picked as the final selection.	70
6.3	(Left) Pruning Ratio vs. Perplexity: Comparison among LSH (blue), TPE/loss-based (brown), and selfish-RNN (gray) on different pruning ratios. (Right) Achieved Perplexity vs Time Budget: Comparison between LSH (blue) and loss-based full training search (brown) on required time budget (hours) to reach a target perplexity.	74
7.1	Examples of improvement in preserving fidelity and consistency of output images compared to baseline (BK-SDM) on pruning Stable Diffusion (SD) 1.4, Dense teacher, to 33%.	77
7.2	Sketch illustration of the difference between pruned Baseline (orange) and our 2ndM (purple). Both Baseline and 2ndM match the dense teacher (black) on the training samples (green). While 2ndM constrains the expansion/contraction behavior of nearby points (red and blue), the Baseline does not have such constraints. As a result, the baseline accumulates drift along the denoising trajectory, leading to deviations at $t = T$. In contrast, 2ndM preserves stable, teacher-aligned trajectories, resulting in higher-fidelity generation.	79
7.3	Per-class rFID and qualitative comparison between Dense, 2ndMatch, and baseline finetuning (DP) on ImageNet 256×256	84
A.1	Visual comparison of ImageNet samples from dense models (top), 2ndM (middle) and baseline Diff-Pruning (bottom).	155
A.2	Visual comparison of LSUN-Bedroom and LSUN-Church samples from dense models (top), 2ndM (middle) and baseline Diff-Pruning (bottom).	157
A.3	Visual comparison of MSCOCO samples from dense models (top), 2ndM (middle) and baseline Diff-Pruning (bottom).	158
A.4	Visual comparison of CIFAR-10 samples from dense models (top), 2ndM (middle) and baseline Diff-Pruning (bottom).	158
A.5	Visual comparison of CelebA samples from dense models (top), 2ndM (middle) and baseline Diff-Pruning (bottom).	159

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere gratitude to my advisor, Professor Eli Shlizerman, for his guidance, support, and encouragement throughout my Ph.D. journey. His insight, patience, and high standards have deeply shaped my development as a researcher. I am especially thankful for the freedom he gave me to explore different research directions, as well as for his steady guidance during moments of challenge and uncertainty. I would also like to thank my dissertation committee members, Professor Linda Shapiro, Professor Payman Arabshahi, and Professor Sergey Rabotyagov, for their time, thoughtful feedback, and valuable suggestions. Their questions and comments helped me strengthen this dissertation and encouraged me to think more carefully about the broader significance of my work. I am grateful to the mentors, collaborators, labmates, and friends I have met at the University of Washington. Their support, conversations, and encouragement have made my graduate study more meaningful and have helped shape both my research and personal growth. I would also like to acknowledge the many mentors, colleagues, and collaborators I have had the privilege to work with during my internships and research projects. Their perspectives, feedback, and encouragement broadened my understanding of research beyond the university and inspired me to think more deeply about how my work can contribute to real-world problems.

I am thankful to my friends back in China, Bo Chen, Daowei Jiang, and Bin Zhao, for their lasting friendship and encouragement. Despite the distance and time difference, their support and companionship have remained an important part of my life. I would also like to thank all of my friends from the 100s apartment over the years of my Ph.D. The friendship, conversations, meals, laughter, and everyday companionship we shared helped make Seattle feel like home. I am grateful for the many ordinary yet precious moments we spent together, which brought warmth and joy to this long journey. I am especially grateful

to my church mentors, Bo Zheng and Hartie Chang, for their mentorship, encouragement, and care throughout different stages of my Ph.D. Their guidance has been meaningful to me academically, personally, and spiritually.

Most importantly, I would like to thank my parents for their unconditional love, sacrifice, and support. Their encouragement and support made it possible for me to come to the United States and pursue my studies. Throughout many years of research, growth, and life far from home, their love has been a constant source of strength. Words cannot fully express my gratitude for all they have done for me.

Finally, I give thanks to God, who has sustained me with strength, wisdom, and grace throughout this journey. This dissertation is not only the result of years of research, but also a testimony to the people, love, and faithfulness that have carried me along the way.

Chapter 1

INTRODUCTION

1.1 *Motivation*

Deep neural networks have transformed artificial intelligence by learning rich hierarchical representations directly from data, achieving state-of-the-art performance across a wide range of tasks, including image recognition [45, 85], natural language processing [1, 26, 175, 176], and generative modeling [92, 183, 208, 210]. Their expressive power stems from composing many layers of nonlinear transformations, enabling them to capture complex patterns that handcrafted features cannot. However, this same complexity makes neural networks expensive to configure and deploy.

For Recurrent Neural Networks (RNNs), which reuse the same recurrent weights at each step, the spectral regime fixed at initialization strongly shapes whether the hidden-state dynamics vanish, explode, or remain well-conditioned. In approaches like FORCE learning, where the recurrent weights remain fixed after initialization, the chosen spectral regime governs the dynamics throughout training, so an initialization that is too quiescent or too chaotic cannot be corrected by learning at all [212]. Choosing a stable initialization is therefore a prerequisite for stable and robust training of an RNN. Even with a well-conditioned initialization, however, it is only one of many hyperparameters that must be selected. Other hyperparameters, such as architectures, hidden sizes, learning rates, and regularization schemes, together span a large configuration space. Candidate in this space must be evaluated by running a full training procedure, a process that scales poorly as models grow in dimensions. The difficulty does not end once a satisfactory model has been identified and trained. Both RNNs and diffusion models reuse the same network across many steps. An RNN is applied across timesteps of the sequence it processes, and a diffusion model is applied across the steps of its denoising trajectory. Therefore, the cost of network inference is required to be applied over and over again with each step. For diffusion

models, which rerun a heavy denoising backbone tens or hundreds of times to produce a single sample, this is most acute. As a result, this makes them expensive to deploy, and thus shrinking the model holds the potential to compound savings across the whole trajectory. Network pruning, which removes redundant structure, is among the most effective tools for model shrinking [61, 79, 132]. However, pruning raises two challenges of its own. The first is that finding a good pruned model is non-trivial. Most pruned configurations lead to performance degradation significantly. Identifying a strong configuration means searching a large space of candidates whose quality can ordinarily be judged only by training each to completion, precisely what one would like to avoid. The second is that the performance loss due to pruning must be recovered. This is especially difficult for diffusion models whose iterative denoising compounds the small per-step errors introduced by pruning into large distortions in the final sample. This dissertation argues that linearization offers a principled dual framework that addresses these challenges. From designing a stable initialization and predicting which configurations will train well to searching for good pruned models and recovering the most degraded performance, linearization and associated methods I have introduced provide tools that reveal network’s behavioral properties without requiring full training and thus supply direct optimization signals to guide training and compression.

Linearization studies how a nonlinear system responds to small local perturbations. In neural networks, it is captured by the *input Jacobian*, the derivative of a network’s output with respect to its input. In an autonomous system that does not require input (e.g., RNN), the output of each step is simply the next hidden state. In that case, the input Jacobian becomes the derivative between consecutive hidden states. The eigenvalues of this Jacobian indicate whether a perturbation grows, locally decays, or is preserved. The more general case is a non-autonomous system, e.g., RNN driven by an input sequence. For such networks, the effective update of the system changes at every step, and therefore no single Jacobian captures its longtime behavior. Composing input Jacobians along the input-driven trajectory and measuring their asymptotic growth rate yields the Lyapunov Exponents (LE), which characterize the stability of the network’s dynamics [161, 239]. A similar situation is in diffusion models. The same network is applied at every step but conditioned on the denoising timestep as an extra input, so its effective transformation varies across a fixed,

finite trajectory even though the network itself does not change. Applying the same idea over this finite horizon led me to consider the Finite-Time Lyapunov Exponents (FTLE), which measure the local rate of trajectory expansion or contraction over that window [199]. Together, these quantities make linearization a natural and unified lens for studying neural network behavior across different architectures and tasks.

1.2 Summary of Research Contributions

The contributions of this dissertation are organized around four roles that linearization plays, illustrated in Figure 1.1: design target for stable initialization, source of predictive features, compass for pruning search, and optimization objective for compression. I detail each role below.

The first role is as an *initialization design target*. Standard random initialization places most Random Recurrent Neural Networks (RRNNs) in quiescent or chaotic regimes, where existing training approaches such as FORCE succeed within this typically narrow edge-of-chaos regime [212]. I propose R-FORCE (Robust-FORCE), which aims to establish an optimal initialization that places the network in this regime. R-FORCE achieves such initialization by analyzing the eigenvalue spectrum of the recurrent Jacobian and identifying structural patterns associated with successful training dynamics. Based on this analysis, R-FORCE designs the initialization of the recurrent connectivity matrix according to four generating principles that ensure the Jacobian eigenvalues remain within the stability region across a broad range of dynamic regimes. Integrating this initialization with FORCE learning yields a training procedure that is robust to diverse network configurations and consistently outperforms standard FORCE learning on sequential signal generation tasks [CZ 1].

The second role is as a source of *predictive features*. Rather than designing the Jacobian, I ask what information can be extracted from it. For RNNs, the Lyapunov Exponents are derived by composing input Jacobians across time steps. They encode rich information about the network’s dynamical behavior. A practical challenge is that LE spectra characteristics vary in length across architectures since networks differ in hidden size, making direct cross-architecture comparison difficult. I introduce AeLLE (Autoencoder Latent Lyapunov

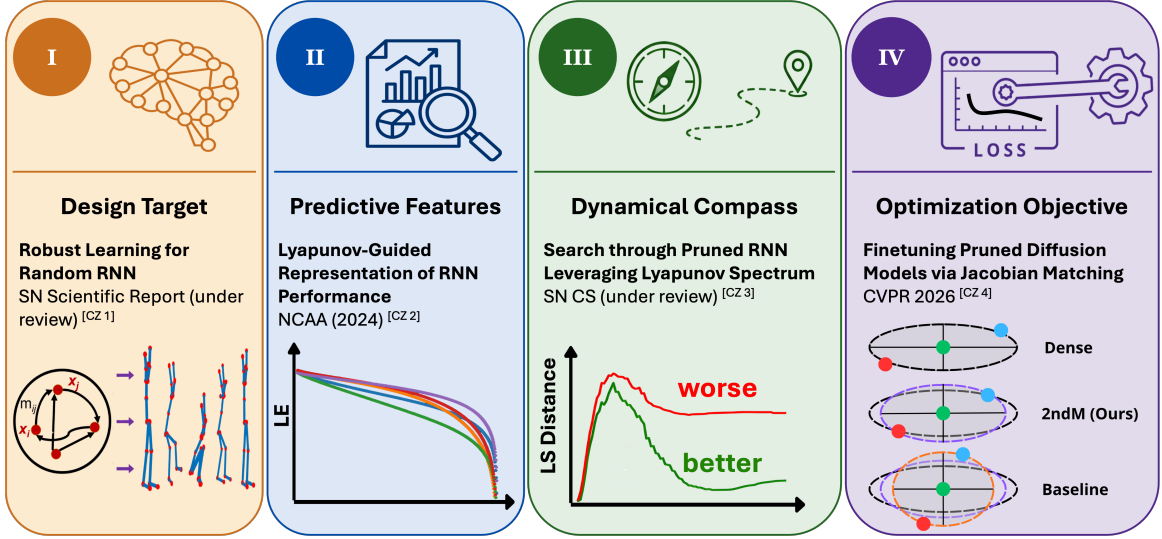


Figure 1.1: Overview of the four roles linearization plays across the contributions of this dissertation. Role I (Design Target): the input Jacobian is shaped at initialization to promote stable training dynamics in recurrent networks [CZ 1]. Role II (Predictive Features): Lyapunov Exponents serve as compact representations for predicting RNN performance [CZ 2]. Role III (Dynamical Compass): the Lyapunov Spectrum distance between a dense model and its pruned variants guides hyperparameter search [CZ 3]. Role IV (Optimization Objective): second-order Jacobian matching inspired by FTLE [CZ 4] supervises the finetuning of pruned diffusion models.

Exponent), a representation learning framework that embeds LE spectra of varying dimensions into a fixed-size latent space via an autoencoder. The latent representation generalizes across diverse RNN families and is available from the very beginning of training. I show that AeLLE-based features outperform both direct LE features and loss-based metrics as early predictors of final model performance, substantially reducing the cost of hyperparameter and architecture selection [CZ 2].

The third role is as a *dynamical compass* for search. The Lyapunov Spectrum (LS) distance between a dense model and a pruned variant measures how much pruning has altered the network’s dynamical fingerprint. I show that this distance is a more reliable

early proxy for final performance than loss-based metrics. This is since the LS distance reflects structural changes to the network’s dynamics that the loss has not yet had time to reveal. I introduce Hyperpruning (LSH), which reframes the search over pruned RNN configurations as a hyperparameter optimization problem and integrates the LS distance as a low-cost evaluation criterion within automated search. LSH identifies better-performing pruned configurations at an order-of-magnitude lower computational cost than standard methods and, in some cases, finds pruned variants that surpass the dense model [CZ 3].

The fourth role is as an *optimization objective* within the diffusion model compression pipeline. For pruned diffusion models, aligning Jacobian-based quantities between a dense teacher and a compressed student provides a supervision signal that the standard denoising training objective cannot supply. I therefore develop 2ndMatch, a second-order Jacobian matching method inspired by FTLE. Since naïve first-order Jacobian matching is redundant with the standard denoising objective, 2ndMatch instead aligns the second-order sensitivity of the pruned student with that of the dense teacher, providing a complementary supervision signal that recovers generation quality across U-Net and Transformer architectures [CZ 4].

The key contributions of this dissertation are summarized as follows:

- I propose R-FORCE, an initialization framework for Random RNNs that designs the recurrent connectivity matrix so that the Jacobian eigenvalues remain within the stability region, making FORCE learning robust across diverse dynamic regimes [CZ 1].
- I introduce AeLLE, an autoencoder-based representation of RNN Lyapunov Exponent spectra that predicts model performance across architectures from the very beginning of training, enabling low-cost hyperparameter selection [CZ 2].
- I propose Hyperpruning (LSH), a Lyapunov Spectrum-guided hyperparameter search algorithm for pruned RNNs that outperforms loss-based search strategies in both result quality and computational efficiency [CZ 3].
- I develop 2ndMatch, a second-order Jacobian matching objective for finetuning pruned

diffusion models, inspired by FTLE, which consistently improves generation quality across U-Net and Transformer architectures [CZ 4].

1.3 Thesis Outline

The remainder of this dissertation is organized as follows. Chapter 2 introduces the theoretical background underlying the contributions, including dynamical systems and Lyapunov theory, recurrent neural networks, diffusion models, network pruning, and Jacobian-based knowledge transfer. Chapter 3 surveys related work on RNN stability and hyperparameter optimization, diffusion model compression, and Jacobian-based training methods. Chapter 4 presents R-FORCE, the Jacobian eigenvalue shaping framework for robust training of random recurrent neural networks [CZ 1]. Chapter 5 presents AeLLE, the Lyapunov Exponent-based representation learning framework for predicting RNN performance across architectures [CZ 2]. Chapter 6 introduces Hyperpruning and the LSH algorithm for Lyapunov Spectrum-guided search over pruned RNN variants [CZ 3]. Chapter 7 develops 2nd-Match, the FTLE-inspired second-order Jacobian matching approach for finetuning pruned diffusion models [CZ 4]. Chapter 8 concludes the dissertation, summarizing the contributions and discussing directions for future work.

Chapter 2

PRELIMINARIES

In this chapter, we cover the foundational concepts underlying this dissertation. We introduce neural network models (RNNs, transformers, and diffusion models) and training methods (FORCE learning), tools from dynamical systems theory and linearization (Lyapunov Exponents and Finite-Time Lyapunov Exponents), and methods relevant to model compression (knowledge distillation and network pruning).

2.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) model temporal sequences by maintaining a hidden state h_t that evolves according to

$$h_{t+1} = f_{\theta}(h_t, x_t), \quad (2.1)$$

where f_{θ} is the recurrent update function parameterized by learnable weights θ . Different RNN variants have different forms of f_{θ} . Gated architectures such as LSTMs [95] and GRUs [39] incorporate gates and memory cells, while the simple *vanilla RNN* [188] apply a single affine transformation followed by a nonlinearity

$$h_{t+1} = \phi(W_h h_t + W_x x_t + b_h) \quad (2.2)$$

with W_h the recurrent weight matrix, W_x the input projection, and $\phi(\cdot)$ a nonlinear activation. The output at each timestep is typically

$$y_t = W_y h_t + b_y. \quad (2.3)$$

Figure 2.1 visualizes this computation. Each timestep reuses the same parameters to update the hidden state and produce an output, forming a system where the same transformation is applied repeatedly over time steps. Since the transformation is repeatedly applied,

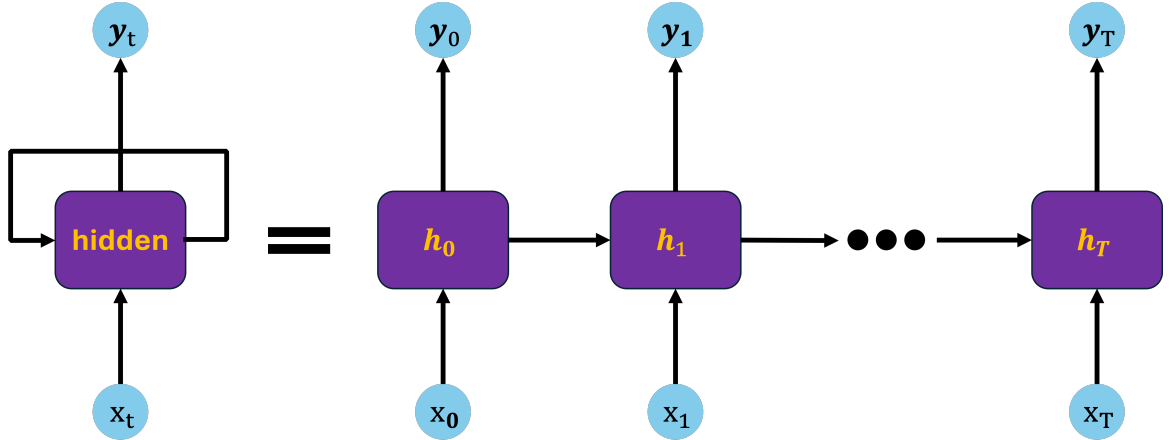


Figure 2.1: **Unfolding a recurrent neural network through time.** The same set of parameters is reused at each timestep to update the hidden state, and unfolding the recurrence reveals the sequential chain of transformations applied to produce the output.

the behavior of the hidden state is governed by the Jacobian of f_θ . Small perturbations may shrink, grow, or remain marginally stable as they propagate through time. This sensitivity to the recurrent dynamics makes RNNs expressive but also difficult to train such that improper initialization or parameter choices can push the system into vanishing-, exploding-, or edge-of-chaos regimes.

Understanding these dynamical properties provides a principled foundation for designing stable initializations and improving the efficiency of RNN training.

2.1.1 Spectral Properties at Initialization

The dynamical regime of RNN at initialization is determined almost entirely by the spectrum of its recurrent weight matrix. For a vanilla RNN with update $h_{t+1} = \phi(a_t)$, where $a_t = W_h h_t + W_x x_t + b_h$ is the pre-activation, the recurrent Jacobian that governs how a perturbation propagates between consecutive hidden states is $J_t = \text{diag}(\phi'(a_t)) W_h$. Near the origin, where the activation is close to linear and $\phi'(a_t) \approx 1$, this Jacobian reduces to W_h itself. In such a case, the eigenvalues of the recurrent matrix govern whether small perturbations locally grow or decay and link the spectral structure of W_h directly to the

stability of the hidden-state dynamics introduced above.

When the recurrent connectivity is initialized randomly, this spectrum takes a strikingly regular form. Consider an $N \times N$ matrix W_h whose entries are drawn independently from a zero-mean distribution with variance g^2/N . As $N \rightarrow \infty$, the *circular law* states that the eigenvalues of W_h become asymptotically uniformly distributed over a disk of radius g centered at the origin of the complex plane,

$$\{\lambda_i(W_h)\} \xrightarrow{N \rightarrow \infty} \text{Uniform}(\{z \in \mathbb{C} : |z| \leq g\}). \quad (2.4)$$

This result, first established for the complex Gaussian (Ginibre) ensemble [64, 65] and later shown to hold universally for arbitrary i.i.d. entries with finite variance [217], makes the scaling parameter g a single, interpretable control knob: it is precisely the spectral radius of W_h , the magnitude of its largest eigenvalue.

The spectral radius separates qualitatively distinct dynamical regimes. When $g < 1$, all eigenvalues of W_h lie strictly inside the unit circle, so the linearization at the origin contracts every perturbation and the network relaxes to a quiescent fixed point. When $g > 1$, the origin becomes unstable, and activity grows until the saturating nonlinearity bounds it, producing the rich, irregular activity that Sompolinsky et al. [205] identified as a transition to chaos in randomly connected networks. The linear stability boundary, therefore, sits at $g \approx 1$, the so-called *edge of chaos*, where the network is most expressive while remaining bounded. The precise onset of sustained chaos, however, is shifted to a larger gain: because $|\phi'(a_t)| \leq 1$ for saturating activations such as $\tanh$, the activation derivative reduces the effective gain of the Jacobian J_t along a trajectory, so whether perturbations ultimately expand or contract is set not by the raw spectral radius of W_h but by the Lyapunov spectrum of the full Jacobian product $J_t \cdots J_1$, the central object introduced in Section 2.5 and analyzed throughout this dissertation. The same spectral viewpoint underlies classical results on the stability of large random systems [144], and structured connectivity, such as sparse or excitatory-inhibitory matrices, deforms the circular law in predictable ways that further shape the accessible dynamics [178].

This spectral picture is the foundation for the initialization method developed in Chapter 4: rather than accepting the uniform disk that random initialization produces, R-FORCE

reshapes the eigenvalue distribution of the recurrent matrix so that its dynamics remain stable and trainable across a far broader range of regimes than the narrow edge-of-chaos band.

2.2 FORCE Learning

Reservoir Computing. Reservoir computing is a framework for training recurrent networks in which the recurrent connectivity is randomly initialized and kept fixed throughout training [104, 140]. The fixed recurrent network, called *reservoir*, is driven by an input signal and produces high-dimensional nonlinear responses. Only the output weights that read from the reservoir are trained. The reservoir weights remain fixed and are not updated. Due to this setup, the training reduces to a linear regression problem, making it computationally efficient. A key assumption is that a randomly initialized reservoir with sufficiently rich dynamics can, in principle, support the computation of any desired output through the right choice of readout weights.

FORCE Learning. FORCE learning (First-Order Reduced and Controlled Error) [212] extends reservoir computing by updating the output weights online using recursive least squares (RLS), allowing the network to learn time-varying target signals. Consider a recurrent network with reservoir state $r(t) \in \mathbb{R}^N$ and scalar output $z(t) = \mathbf{w}^\top r(t)$, where $\mathbf{w}$ is the readout weight vector. At each timestep, FORCE computes the instantaneous error between the output and the target signal $f(t)$,

$$e(t) = z(t) - f(t), \quad (2.5)$$

and updates both the running inverse correlation matrix $P(t) \in \mathbb{R}^{N \times N}$ and the readout weights via the RLS rule:

$$P(t) = P(t-1) - \frac{P(t-1) r(t) r(t)^\top P(t-1)}{1 + r(t)^\top P(t-1) r(t)}, \quad (2.6)$$

$$\mathbf{w}(t) = \mathbf{w}(t-1) - e(t) P(t) r(t). \quad (2.7)$$

The matrix $P(t)$ approximates the running inverse of the reservoir activity covariance, so the weight update effectively normalizes the learning signal by the correlation structure of

the reservoir. This one-step update follows from the Sherman–Morrison identity applied to the running least squares objective and avoids the need to invert a matrix at each step.

Edge-of-Chaos Limitation. The success of FORCE learning is sensitive to the spectral radius g of the recurrent weight matrix W_h . When $g \ll 1$, the reservoir is quiescent: activity decays rapidly, dynamics are too impoverished to support complex computations, and FORCE fails to converge. When $g \gg 1$, the reservoir is chaotic: activity diverges exponentially, and the RLS updates cannot keep pace with the instability. FORCE learning succeeds reliably only in a narrow regime near $g \approx 1$, the so-called *edge of chaos*, where the reservoir exhibits rich but bounded dynamics. This sensitivity to initialization motivates R-FORCE, which designs the recurrent connectivity so that the Jacobian eigenvalues remain within the stability region across a much broader range of dynamic regimes.

2.3 Transformers and Attention

Transformers model sequential data using self-attention, which allows each token to directly interact with all others in the sequence [223]. The left side of Figure 2.2 illustrates the core computation. Given an input sequence $X \in \mathbb{R}^{T \times d_{\text{model}}}$, where T is the sequence length and d_{model} is the embedding dimension, the model first projects each token into queries, keys, and values via

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V, \quad (2.8)$$

where W_Q , W_K , and W_V are learned projection matrices. Self-attention then computes pairwise similarity scores $QK^\top$, converts them into attention weights using a softmax, and forms weighted mixtures of the value vectors

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V. \quad (2.9)$$

This operation enables the model to contextualize each token by aggregating information from all positions in the sequence.

Multi-Head Attention and Transformer Layers. To capture multiple interaction patterns simultaneously, transformers apply several attention heads in parallel

$$\text{MHA}(X) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W_O, \quad (2.10)$$

where W_O is an output projection. The right panel of Figure 2.2 shows how multi-head attention is integrated into a full transformer layer which consists of multi-head self-attention (MHA), a positionwise feedforward network (FFN), residual connections and layer normalization. Formally, for hidden representation $H^{(\ell)}$ at layer ℓ :

$$H^{(\ell+1/2)} = H^{(\ell)} + \text{MHA}\left(\text{LN}(H^{(\ell)})\right), \quad (2.11)$$

$$H^{(\ell+1)} = H^{(\ell+1/2)} + \text{FFN}\left(\text{LN}(H^{(\ell+1/2)})\right). \quad (2.12)$$

From NLP to Vision. While transformers originally were proposed for machine translation [223], they have become the dominant architecture across many domains. In computer vision, the Vision Transformer (ViT) [51] demonstrated that patch-based self-attention achieves strong image classification performance without any convolutional structure. Subsequent work has extended transformers to object detection [28], segmentation [252], and video understanding [20]. Of particular relevance to this dissertation, transformer-based architectures have been adapted as backbones for diffusion models. Further developments in the Diffusion Transformer (DiT) [169] provide an alternative to the U-Net denoising network with a transformer and achieve state-of-the-art image generation quality, making it one of the architectures targeted by the compression method studied in Chapter 7.

2.4 Diffusion Generative Models

Diffusion generative models are closely related to score-based generative modeling [210]. A central idea in these models is to characterize the data distribution $p_{\text{data}}(x)$ through its *score function*, defined as the gradient of the log-density with respect to the data

$$s(x) = \nabla_x \log p_{\text{data}}(x), \quad (2.13)$$

rather than by its density directly. The score points in the direction of increasing data density at any given point x and can be used to generate samples via *Langevin dynamics* [234].

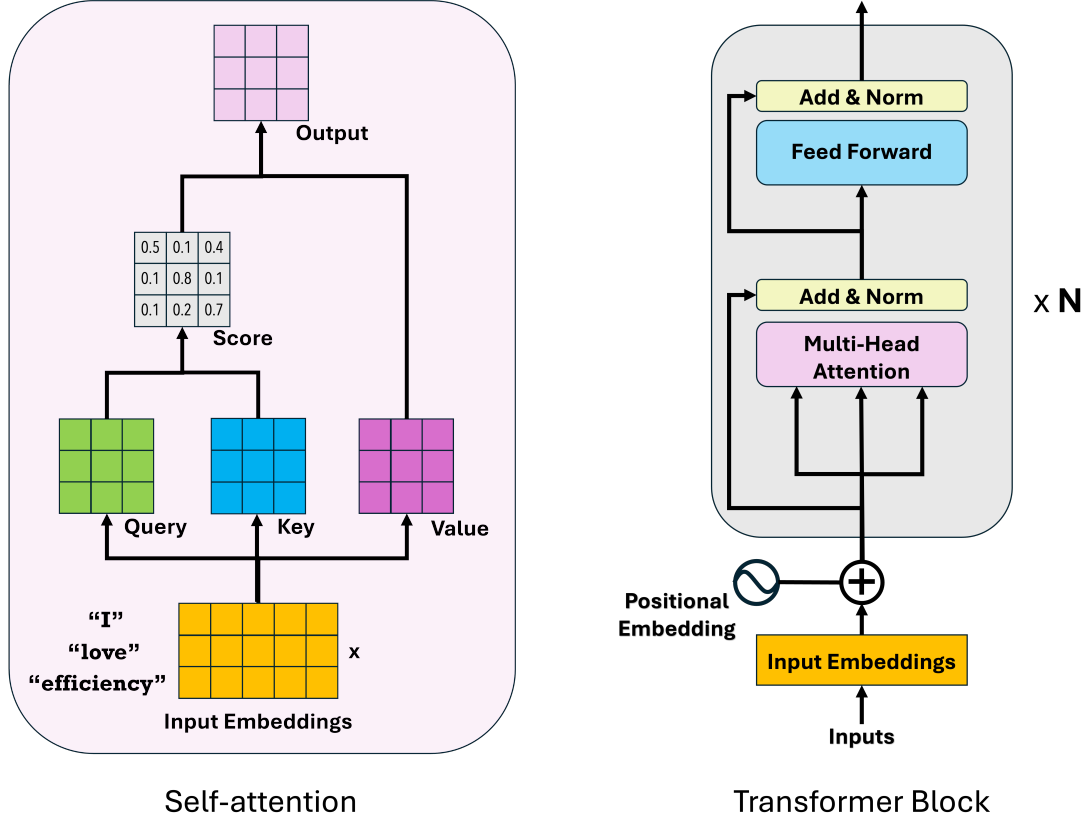


Figure 2.2: **Self-attention and transformer layer.** **Left:** Self-attention mechanism. Each input token is projected into query, key, and value vectors. Query–key similarities determine attention weights, which produce weighted mixtures of the value vectors. **Right:** A transformer block consisting of multi-head attention followed by a feedforward network, each wrapped with residual connections and layer normalization.

Starting from random noise $x_0 \sim \mathcal{N}(0, I)$, iteration of

$$x_{k+1} = x_k + \frac{\delta}{2} s(x_k) + \sqrt{\delta} z_k, \quad z_k \sim \mathcal{N}(0, I), \quad (2.14)$$

converges in distribution to $p_{\text{data}}(x)$ as $\delta \rightarrow 0$ and $k \rightarrow \infty$. A central challenge is that $s(x) = \nabla_x \log p_{\text{data}}(x)$ is unknown since we only have access to samples from p_{data} , not the density itself.

Score Matching. Score matching [99] estimates the score directly from data by minimizing the expected squared difference between the model score $s_\theta(x)$ and the true score

$$\mathcal{L}_{\text{SM}}(\theta) = \mathbb{E}_{x \sim p_{\text{data}}} \left[\|s_\theta(x) - \nabla_x \log p_{\text{data}}(x)\|^2 \right]. \quad (2.15)$$

While the true score is unknown, Hyvärinen shows that this objective is equivalent (up to a constant) to

$$\mathcal{L}_{\text{SM}}(\theta) = \mathbb{E}_x \left[\text{tr}(\nabla_x s_\theta(x)) + \frac{1}{2} \|s_\theta(x)\|^2 \right], \quad (2.16)$$

which can be minimized from the samples alone. Computing the trace of the Jacobian $\nabla_x s_\theta(x)$ requires a backpropagation pass for each output dimension, making it prohibitively expensive for high-dimensional data. Two scalable alternatives address this bottleneck. Sliced Score Matching (SSM) [209] approximates the trace using random projection. For a random vector $v \sim \mathcal{N}(0, I)$, the identity $\mathbb{E}_v[v^\top \nabla_x s_\theta(x) v] = \text{tr}(\nabla_x s_\theta(x))$ allows replacement of the full trace with a single vector-Jacobian product, reducing cost to $O(1)$ passes regardless of dimension. The other alternative is Denoising Score Matching, which sidesteps the Jacobian entirely and is more widely adopted in practice due to its simplicity and efficiency.

Denoising Score Matching. Denoising score matching (DSM) [224] avoids Jacobian computation by perturbing the data with Gaussian noise. Given a noise level σ , consider the perturbed distribution $q_\sigma(\tilde{x} | x) = \mathcal{N}(\tilde{x}; x, \sigma^2 I)$. The score of this perturbed distribution is tractable:

$$\nabla_{\tilde{x}} \log q_\sigma(\tilde{x} | x) = -\frac{\tilde{x} - x}{\sigma^2} = -\frac{\epsilon}{\sigma}, \quad \epsilon = \frac{\tilde{x} - x}{\sigma}. \quad (2.17)$$

DSM trains a network $s_\theta(\tilde{x}, \sigma)$ to match this tractable score:

$$\mathcal{L}_{\text{DSM}}(\theta) = \mathbb{E}_{x \sim p_{\text{data}}, \tilde{x} \sim q_\sigma(\cdot | x)} \left[\left\| s_\theta(\tilde{x}, \sigma) + \frac{\tilde{x} - x}{\sigma^2} \right\|^2 \right], \quad (2.18)$$

which is equivalent to training the network to predict the normalized noise ϵ/σ from the corrupted sample $\tilde{x}$.

NCSN: Multi-Scale Score Matching. A single noise level σ is insufficient. Small σ gives an accurate score near the data manifold but insufficient coverage of low-density

regions, while large σ covers the full space but yields an inaccurate score. Noise Conditional Score Networks (NCSN) [208] address this by training a single network $s_\theta(x, \sigma)$ across a geometric sequence of noise levels $\sigma_1 > \sigma_2 > \dots > \sigma_L$ with a weighted objective

$$\mathcal{L}_{\text{NCSN}}(\theta) = \frac{1}{L} \sum_{i=1}^L \lambda(\sigma_i) \mathbb{E}_{x, \tilde{x}} \left[\left\| s_\theta(\tilde{x}, \sigma_i) + \frac{\tilde{x} - x}{\sigma_i^2} \right\|^2 \right], \quad (2.19)$$

where the weighting $\lambda(\sigma_i) = \sigma_i^2$ balances the magnitude of the score across scales. Generation then uses annealed Langevin dynamics in which Langevin steps are run at each noise level from large to small. This gradually refines the sample toward the data distribution.

Such a denoising framework naturally corresponds to the structure of diffusion models. In the forward process, clean data x_0 is progressively perturbed into near-Gaussian noise through a sequence of noise-injection steps. A diffusion model learns the reverse transitions, producing an iterative denoising trajectory

$$x_{t-1} = g_\theta(x_t, t) \quad (2.20)$$

gradually reconstructing the data distribution. Figure 2.3 illustrates this forward–reverse structure.

Example: DDPM Reverse Transition. In Denoising Diffusion Probabilistic Models (DDPM) [92], the forward process is defined as

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, \quad \epsilon \sim \mathcal{N}(0, I), \quad (2.21)$$

with α_t a per-step variance schedule and $\bar{\alpha}_t = \prod_{i=1}^t \alpha_i$ its cumulative product. The model learns to predict the injected noise via a neural network $\epsilon_\theta(x_t, t)$, enabling a closed-form Gaussian reverse step

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + \sigma_t z, \quad z \sim \mathcal{N}(0, I), \quad (2.22)$$

where σ_t controls the stochasticity of the reverse transition. A useful property of this forward process is that x_t can be sampled directly from x_0 in closed form without simulating all intermediate steps, since the Gaussian noise accumulates analytically. This makes training efficient: rather than running the full Markov chain, each training step samples a random timestep t , corrupts a clean example to obtain x_t in one shot, and trains the network to predict the added noise ϵ .

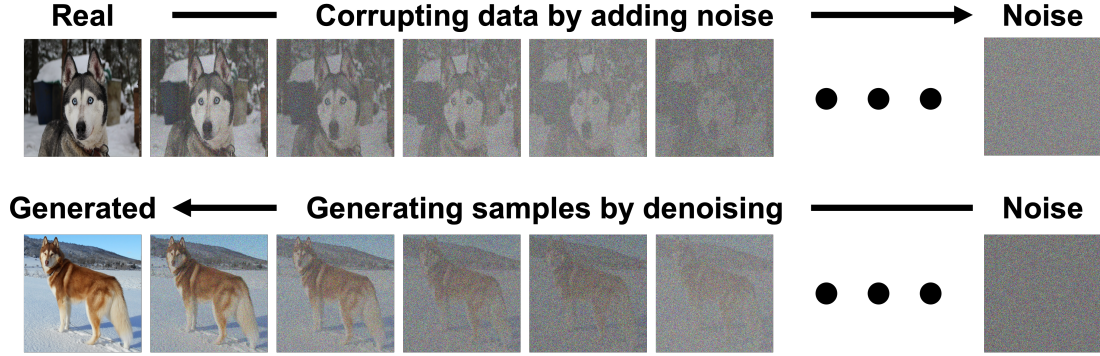


Figure 2.3: **Forward and reverse diffusion.** Data is gradually perturbed into noise in the forward process, and the learned reverse process reconstructs the sample through a multi-step denoising trajectory, making diffusion sampling computationally expensive. The forward and reverse illustrations use two CC0 images from Wikimedia Commons, “Siberian Husky (Unsplash).jpg” and “Siberian-husky-1291343 1920.jpg,” respectively. The intermediate noisy images were synthetically generated by the author.

Network Architecture. The noise-prediction network $\epsilon_{\theta}(x_t, t)$ must process inputs at many noise levels, so it is conditioned on the timestep t via a time embedding (typically a sinusoidal encoding passed through a small MLP). The dominant backbone architecture is the U-Net [185], which uses an encoder-decoder structure with skip connections between matched resolution levels, allowing the network to combine coarse semantic features with fine spatial details during denoising. More recently, the Diffusion Transformer (DiT) [169] replaces the U-Net with a transformer that operates on image patches, injecting the timestep and class conditioning through adaptive layer normalization. Both the U-Net and DiT backbones are targeted by 2ndMatch, which is designed to be architecture-agnostic.

Modern Diffusion Models. Modern diffusion models build on the basic DDPM framework, incorporating several advances to achieve practical, high-quality generation. Rather than operating in pixel space, Latent Diffusion Models (LDMs) [183] run the diffusion process in a compact latent space: a pretrained VAE encoder $\mathcal{E}$ first compresses an image x into a low-dimensional representation $z_0 = \mathcal{E}(x) \in \mathbb{R}^{d_z}$ with $d_z \ll d_x$, and the VAE decoder

$\mathcal{D}$ reconstructs the final image as $\hat{x} = \mathcal{D}(z_0)$ after denoising. For text-to-image generation, a pretrained text encoder τ_ϕ (typically CLIP [173]) maps a prompt y to a sequence of embeddings $c = \tau_\phi(y) \in \mathbb{R}^{T_c \times d_c}$, which are injected into the denoising network via cross-attention to produce a conditioning-aware noise prediction $\epsilon_\theta(z_t, t, c)$. To further strengthen prompt alignment at inference time, *classifier-free guidance* (CFG) [94] interpolates between conditional and unconditional prediction

$$\tilde{\epsilon}_\theta(z_t, t, c) = \epsilon_\theta(z_t, t, \emptyset) + w(\epsilon_\theta(z_t, t, c) - \epsilon_\theta(z_t, t, \emptyset)), \quad (2.23)$$

where $\emptyset$ denotes null conditioning and $w > 1$ is a guidance scale that trades sample diversity for prompt fidelity.

Inference Cost and Compression. Diffusion models consistently outperform single-pass generative models such as VAEs [113] and GANs [68] on image fidelity, diversity, and text alignment, making them the dominant paradigm for high-quality image synthesis. This quality advantage comes at a cost: generating a single sample requires iterating the denoising network for 50 to 100 or more steps, making inference substantially slower than single-pass alternatives. Reducing this cost through network pruning is practically important, but challenging: since the same network is applied at every denoising step, per-step errors accumulate along the trajectory, so a pruned model that appears close to the dense model at each step may still produce significantly distorted final samples. Quality recovery after pruning is therefore non-trivial, and motivates the method developed in Chapter 7.

Evaluation Metrics. Three metrics are commonly used to evaluate generative model outputs. *Inception Score* (IS) [193] evaluates generated images using a pretrained Inception-v3 classifier by measuring two properties simultaneously. Each generated image should be confidently classifiable (low entropy of $p(y | x)$), and the overall set of generated images should cover diverse categories (high entropy of the marginal $p(y) = \mathbb{E}_x[p(y | x)]$). IS is defined as

$$\text{IS} = \exp(\mathbb{E}_x[D_{\text{KL}}(p(y | x) \| p(y))]) , \quad (2.24)$$

where higher IS indicates better quality and diversity. However, IS does not compare generated images to real images, making it insensitive to mode dropping. *Fréchet Inception Distance* (FID) [88] addresses this by measuring the statistical distance between the distributions of real and generated images in Inception-v3 feature space. Both distributions are modeled as Gaussians with mean μ and covariance Σ , and FID is computed as

$$\text{FID} = \|\mu_r - \mu_g\|^2 + \text{Tr}\left(\Sigma_r + \Sigma_g - 2(\Sigma_r \Sigma_g)^{1/2}\right), \quad (2.25)$$

where subscripts r and g denote real and generated sets, respectively. A lower FID indicates that the generated distribution is closer to the real one, and it has become the standard metric for unconditional and class-conditional generation. For text-to-image generation, *CLIP score* [87] measures the semantic alignment between a generated image and its text prompt as the cosine similarity between their CLIP [173] embeddings, with higher values indicating better prompt fidelity. These metrics are used throughout the experimental evaluations in this dissertation; see Appendix A.1 for the full mathematical derivation of the DDPM training objective.

2.5 Dynamical Systems and Lyapunov Theory

The eigenvalues of the recurrent matrix (Section 2.1.1) characterize single-step stability in the linear regime at initialization. The tools in this section extend the eigenvalues to characterize the asymptotic, input-driven behavior of the full nonlinear system. Dynamical systems theory provides a mathematical framework for understanding how a state evolves over time. In discrete time, a system is written as

$$x_{t+1} = f(x_t, t), \quad (2.26)$$

and in continuous time as

$$\frac{dx(t)}{dt} = f(x(t), t), \quad (2.27)$$

where f specifies the evolution rule. Both RNNs and diffusion models are discrete-time systems. RNNs update their hidden state step by step over the input sequence, and diffusion models iterate the denoising network over a fixed number of timesteps. The discrete-time formulation, therefore, applies directly to both.

A central distinction is whether the update rule depends explicitly on time. In a *time-invariant* system, the same transformation is applied at every step,

$$x_{t+1} = f(x_t), \quad (2.28)$$

and long-term behaviors (convergence to equilibria, oscillations, or chaotic attractors) arise entirely from the structure of f . RNNs are a canonical example: the recurrent update $h_{t+1} = f_\theta(h_t, x_t)$ uses the same parameters θ at every timestep, and the temporal input x_t acts as an external driving signal rather than changing the dynamics itself. In contrast, a *time-dependent* system evolves according to a rule that changes with time, $x_{t+1} = f(x_t, t)$. Diffusion models fall into this category: the denoising network $\epsilon_\theta(x_t, t)$ is explicitly conditioned on the timestep t , so the effective transformation applied to the state changes at every step along the denoising trajectory. Since the dynamics evolve with t , stability must often be evaluated over finite windows rather than asymptotically, which motivates the Finite-Time Lyapunov Exponents introduced later in this section.

To study sensitivity to small perturbations, one examines the *input Jacobian* of the dynamics,

$$J_t = \frac{\partial f(x_t, t)}{\partial x_t}, \quad (2.29)$$

which linearizes the evolution of an infinitesimal deviation δx_t

$$\delta x_{t+1} \approx J_t \delta x_t. \quad (2.30)$$

Eigenvalues of J_t indicate whether perturbations locally expand, contract, or remain neutral around a given state. For an RNN, $J_t = \partial f_\theta / \partial h_t$ has the same functional form at every step as the parameters do not depend on t , while its value still varies as the state and input evolve. For a diffusion model, $J_t = \partial \epsilon_\theta(x_t, t) / \partial x_t$ depends explicitly on the timestep t and the transformation itself changes at every denoising step.

Lyapunov Exponents (LE). For time-invariant systems such as RNNs, one can characterize the long-run sensitivity by composing the input Jacobians across all timesteps and measuring their asymptotic growth rate. The Lyapunov exponent in a direction v_0 is defined

as

$$\lambda = \lim_{T \rightarrow \infty} \frac{1}{T} \log \|J_{T-1} \cdots J_1 J_0 v_0\|, \quad (2.31)$$

where v_0 is an initial perturbation direction. With a generic random v_0 , this converges almost surely to the largest exponent λ_1 , since the most expanding direction dominates exponentially. As shown in Figure 2.4, a positive exponent signifies exponential divergence and chaotic behavior, while a negative exponent indicates contraction toward stable modes.

This definition has two practical limitations. First, directly computing the product $J_{T-1} \cdots J_0$ causes numerical overflow for large T , and it only reveals the largest exponent rather than the full spectrum. Both issues are resolved by the QR algorithm, which periodically re-orthogonalizes the basis by decomposing

$$J_t Q_t = Q_{t+1} R_{t+1}, \quad (2.32)$$

where the columns of Q_t track the most expanding directions without overflow, and the diagonal entries of R_t record the instantaneous expansion factor in each direction after removing contributions from directions already accounted for. The k th Lyapunov exponent is then recovered as the average log expansion in the k th direction

$$\lambda_k = \frac{1}{T} \sum_{t=1}^T \log r_t^{(k)}, \quad (2.33)$$

with $r_t^{(k)}$ the k th diagonal entry of R_t . The collection $\{\lambda_1, \dots, \lambda_n\}$ is the *Lyapunov spectrum*, providing a multidimensional fingerprint of the RNN's dynamical regime. All-negative exponents indicate a stable, contracting system. Largest exponent near zero indicates the edge-of-chaos regime, and positive exponents indicate chaotic, diverging dynamics.

Random Dynamical Systems and Input-Driven RNNs. Lyapunov exponents defined above are properties of a long product of Jacobians,

$$T_t = J_{t-1} J_{t-2} \cdots J_0, \quad (2.34)$$

and for them to be well defined this product must possess a stable asymptotic growth rate. For a measure-preserving (autonomous) system, Oseledets' multiplicative ergodic theorem

guarantees that the limit

$$\Lambda = \lim_{t \rightarrow \infty} \left(T_t^\top T_t \right)^{\frac{1}{2t}} \quad (2.35)$$

exists and the Lyapunov exponents are the logarithms of its eigenvalues, $\lambda_k = \log \mu_k(\Lambda)$, which take the same value for almost every initial condition and perturbation direction within an ergodic component [162, 187].

An input-driven RNN, however, is not autonomous in the usual sense. While the recurrent map is time-invariant with its parameters fixed, each Jacobian

$$J_t = \text{diag}(\phi'(a_t)) W_h, \quad a_t = W_h h_t + W_x x_t + b_h, \quad (2.36)$$

depends on the pre-activation a_t and therefore on the external input x_t . Thus, the product $T_t = T_t(x_0, \dots, x_{t-1})$ itself is a function of the input sequence and changes from one input to the next. The appropriate framework is then the framework of *random dynamical systems* theory. When the inputs are drawn from a stationary, ergodic process, $x_t \sim \mu_x$, the driven network forms a random dynamical system, and the multiplicative ergodic theorem extends to this setting [187, 63]. In particular, the limit

$$\lambda_k = \lim_{t \rightarrow \infty} \frac{1}{t} \log \sigma_k(T_t) \quad (2.37)$$

exists for μ_x -almost every input realization and takes the same value regardless of which realization is drawn, where $\sigma_k(T_t)$ denotes the k th singular value of T_t . The Lyapunov spectrum is therefore an intrinsic property of the network together with its input distribution, not an artifact of any single input sequence, which is what makes it a reproducible signature of a trained RNN. In practice, it is computed by evolving the QR recursion above along one long input-driven trajectory and averaging the per-step log expansions, $\lambda_k = \frac{1}{T} \sum_{t=1}^T \log r_t^{(k)}$ [227].

Finite-Time Lyapunov Exponents (FTLE). For time-dependent systems such as diffusion models, the Lyapunov exponent definition does not apply for two reasons: 1) The denoising trajectory has a fixed finite length, so the limit $T \rightarrow \infty$ cannot be taken. 2) Even within that fixed window, the dynamics change at every step as the timestep t advances.

Thus, there is no single repeated transformation whose asymptotic behavior can be characterized. Finite-Time Lyapunov Exponents (FTLE) instead quantify sensitivity over a finite time window, making them a natural tool for analyzing the diffusion denoising chain. Given a starting point $x(t_0)$ and a time horizon t_1 , the flow map

$$\Phi_{t_0}^{t_0+t_1}(x(t_0)) = x(t_0) + \int_{t_0}^{t_0+t_1} f(x(\tau), \tau) d\tau \quad (2.38)$$

describes the state after evolving for time t_1 . For a nearby point $x'(t_0) = x(t_0) + v(t_0)$, a first-order Taylor expansion yields

$$v(t_0 + t_1) \approx \frac{d\Phi_{t_0}^{t_0+t_1}}{dx} v(t_0), \quad (2.39)$$

so the perturbation is stretched by the Jacobian of the flow map. The associated finite-time Cauchy–Green deformation tensor is

$$\mathcal{C} = J^\top J, \quad J = \frac{d\Phi_{t_0}^{t_0+t_1}}{dx}, \quad (2.40)$$

and the FTLE is defined as the maximum rate of stretching over the interval:

$$\Lambda_{t_0}^{t_1}(x) = \frac{1}{t_1} \ln \sqrt{\lambda_{\max}(\mathcal{C})}. \quad (2.41)$$

Applied to diffusion models, FTLE measure how sensitive the final denoised output is to perturbations injected at an intermediate denoising step, capturing the compounding effect of the time-varying network across the trajectory. This perspective directly inspires the second-order Jacobian matching objective developed in Chapter 7.

2.6 Knowledge Distillation

Knowledge distillation (KD) [90] is a training strategy in which a smaller *student* model learns to mimic the behavior of a larger pretrained *teacher* model. Rather than training with ground-truth labels, the student uses a richer supervision encoded in the teacher’s output distributions and internal representations. Such an approach enables comparable performance but with lower computational cost.

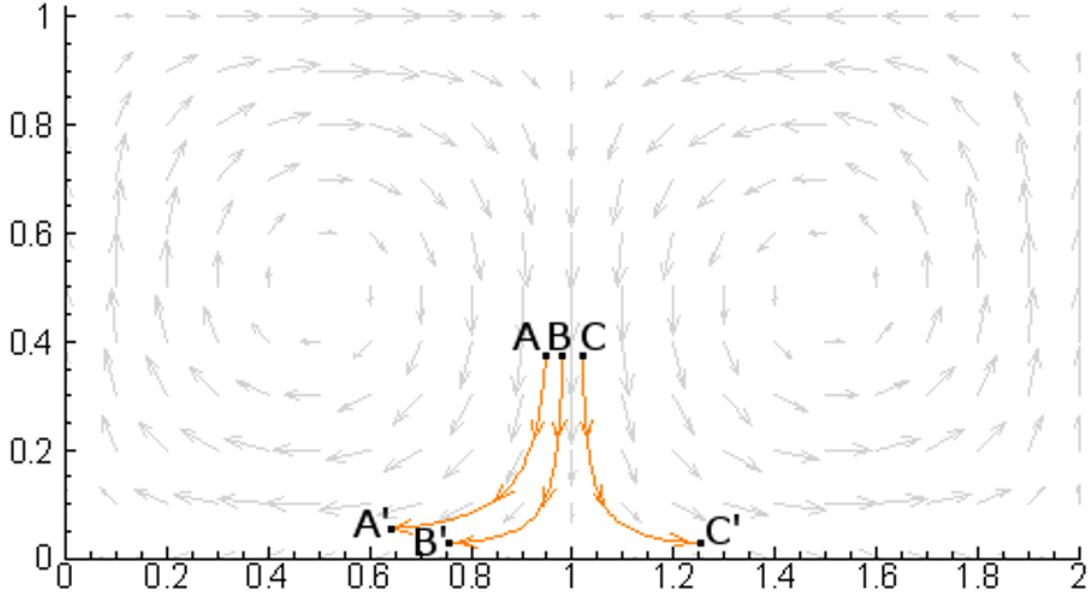


Figure 2.4: **Local sensitivity in a dynamical system.** Three initially close points (A, B, C) evolve under a time-varying flow field. Although their starting positions differ only slightly, the resulting trajectories (A', B', C') diverge rapidly, especially near separatrices where the flow exhibits strong deformation. Such divergence motivates the use of Lyapunov-based measures to quantify stability and sensitivity [199].

Standard Knowledge Distillation. In the original formulation, the student is trained to match the teacher’s softened output probabilities (soft labels) by minimizing the KL divergence between their distributions at a common temperature τ :

$$\mathcal{L}_{\text{KD}} = \tau^2 D_{\text{KL}}\left(\sigma\left(\frac{z^t}{\tau}\right) \parallel \sigma\left(\frac{z^s}{\tau}\right)\right), \quad (2.42)$$

where z^t and z^s are the teacher’s and student’s logits and σ denotes the softmax. Raising the temperature τ softens both distributions, exposing the relative similarity structure among classes that a hard argmax would discard. The final training loss is typically a weighted combination of this distillation term and the task-specific supervised loss on ground-truth labels (e.g., cross-entropy for classification).

Feature-Level Distillation. Standard KD constrains only the final output layer. Feature-level distillation [184] extends supervision to intermediate activations, providing a denser signal that captures how the teacher structures its internal representations at different levels of abstraction. Let $h_\ell^t(x) \in \mathbb{R}^{d_\ell}$ and $h_\ell^s(x) \in \mathbb{R}^{d'_\ell}$ denote the teacher’s and student’s activations at layer ℓ . Since teacher and student may differ in width, a learned linear adapter $A_\ell \in \mathbb{R}^{d_\ell \times d'_\ell}$ aligns the student’s representation to the teacher’s dimension, and the feature distillation loss is

$$\mathcal{L}_{\text{feat}} = \sum_{\ell} \|h_\ell^t(x) - A_\ell h_\ell^s(x)\|^2. \quad (2.43)$$

Minimizing this loss across multiple layers encourages the student to not only reproduce the teacher’s final predictions but also to develop similar intermediate representations.

Derivative-Level Distillation. Both output matching and feature matching constrain the student only at observed data points, leaving it free to deviate arbitrarily in their neighborhoods. Sobolev training [42] addresses this by augmenting the distillation objective with terms that match the teacher’s derivatives up to order K :

$$\mathcal{L}_{\text{Sob}} = \|f^t(x) - f^s(x)\|^2 + \lambda_1 \|J^t(x) - J^s(x)\|_F^2 + \lambda_2 \|H^t(x) - H^s(x)\|_F^2 + \dots, \quad (2.44)$$

where $J = \partial f / \partial x$ is the Jacobian and $H = \partial^2 f / \partial x^2$ is the Hessian. Matching k -th order derivatives constrains the student to agree with the teacher in a local neighborhood of each data point rather than only at the point itself, providing richer supervision per sample and making the approach especially effective when calibration data is limited. In practice, the most commonly used term is the first-order one, Jacobian matching ($J^t(x) \approx J^s(x)$), which aligns the input-output sensitivity of teacher and student. Figure 2.5 illustrates why this is valuable: with only a small number of calibration points, output matching is underdetermined, and many student functions interpolate the data while diverging elsewhere. Jacobian matching resolves this by constraining the slopes at each point (Solution 2 in the figure). An equivalent approach is to match outputs at noise-perturbed inputs (Solution 1): expanding around a clean input x with perturbation $\delta \sim \mathcal{N}(0, \sigma^2 I)$ and taking the expectation gives

$$\mathbb{E}_\delta \left[\|f^t(x + \delta) - f^s(x + \delta)\|^2 \right] \approx \|f^t(x) - f^s(x)\|^2 + \sigma^2 \|J^t(x) - J^s(x)\|_F^2, \quad (2.45)$$

so the two solutions in Figure 2.5 are equivalent to first order in σ .

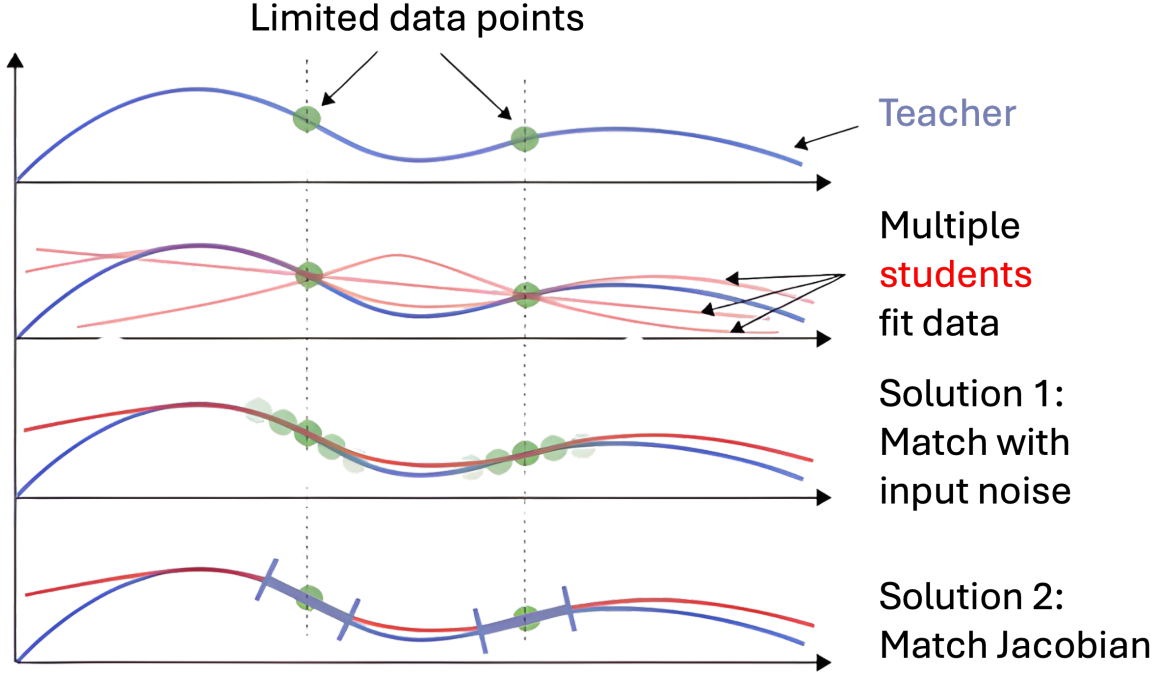


Figure 2.5: **Jacobian matching is equivalent to output matching with input noise.** With limited calibration points, many student functions (pink) fit the teacher (blue) at the observed locations but diverge elsewhere (top two rows). Matching outputs at noise-perturbed inputs (Solution 1) enforces consistency in a neighborhood around each point; directly matching input Jacobians (Solution 2) achieves the same effect, since the two objectives are equivalent to first order in the noise magnitude.

2.7 Network Pruning

Network pruning reduces inference cost and memory footprint by removing redundant parameters from a trained model. Formally, given a pretrained network f_θ with p structural units and a calibration dataset $\mathcal{D}$, pruning seeks a binary mask $m \in \{0, 1\}^p$ that solves

$$\min_{m \in \{0, 1\}^p} \mathcal{L}(f_{\theta \odot m}; \mathcal{D}) \quad \text{subject to} \quad \|m\|_0 \leq k, \quad (2.46)$$

where $\theta \odot m$ denotes the parameters with pruned units zeroed out, and k is the target sparsity budget. This is a combinatorial problem over an exponentially large search space,

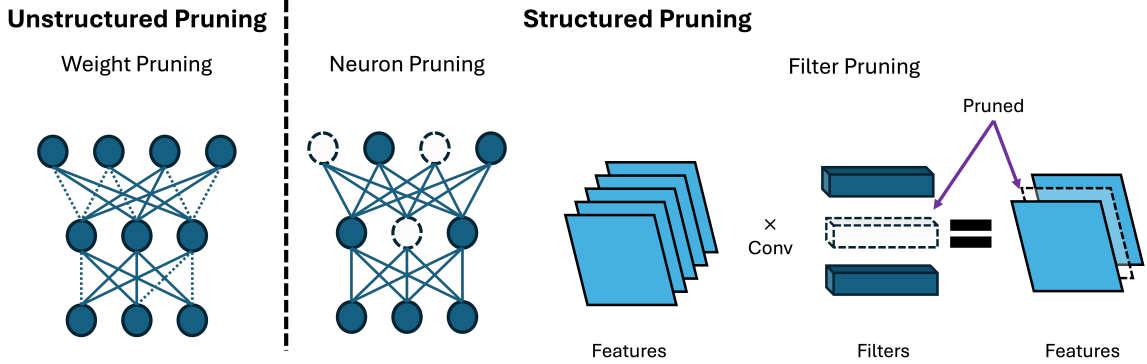


Figure 2.6: Unstructured, neuron-level, and filter-level structured pruning. Unstructured pruning removes individual weights (left), neuron pruning removes whole neurons in fully connected layers (middle), and filter pruning removes entire convolutional filters (right). Structured sparsity forms dense, hardware-friendly subnetworks that enable speedups in CNNs and diffusion models.

and practical methods approximate it through different pruning criteria or relaxations.

Unstructured and Structured Pruning. Pruning methods differ in what they remove. *Unstructured pruning* zeros out individual weights wherever they are least important, producing sparse weight tensors [80]. High sparsity ratios are achievable with minimal quality loss, but irregular sparsity patterns are not efficiently supported by standard hardware accelerators, which are optimized for dense matrix operations. *Structured pruning* removes entire structural units such as convolutional filters, attention heads, or transformer layers, so the pruned model remains dense and executes efficiently without specialized sparse kernels [122]. Because the methods studied in this dissertation target large-scale image generation, where inference latency on standard hardware is the primary bottleneck, the focus is on structured pruning.

Pruning Criteria. The choice of criterion determines which units are removed. Magnitude-based criteria [80, 122] remove units whose weights have the smallest norm, relying on the assumption that small weights contribute little to the output. Regularization can reinforce

this assumption during training: an ℓ_1 penalty $\lambda\|w\|_1$ drives individual weights exactly to zero [134], while ℓ_2 regularization $\lambda\|w\|_2^2$ (weight decay) shrinks weights toward zero without zeroing them. For structured pruning, group sparsity regularization $\lambda\sum_g\|w_g\|_2$ encourages entire filters or attention heads to vanish simultaneously [235], making the subsequent pruning step more reliable. More principled criteria [152] estimate the effect of removing unit i by applying a Taylor expansion to the loss change:

$$\Delta\mathcal{L}_i = \mathcal{L}(\theta) - \mathcal{L}(\theta; w_i = 0) \approx -g_i w_i + \frac{w_i^2}{2} H_{ii}, \quad (2.47)$$

where $g_i = \partial\mathcal{L}/\partial w_i$ and $H_{ii} = \partial^2\mathcal{L}/\partial w_i^2$ are the gradient and Hessian diagonal at weight w_i . Keeping only the first-order term gives the criterion $|\Delta\mathcal{L}_i| \approx |g_i w_i|$; including the second-order term gives $\Delta\mathcal{L}_i \approx g_i^2/(2H_{ii})$, which is the basis of the classical Optimal Brain Damage [120] approach.

Chapter 3

RELATED WORK

3.1 *RNN Stability and Initialization*

Recurrent neural networks have been a foundational architecture for modeling sequential data, achieving strong results in language modeling [15, 149], machine translation [214, 8], speech recognition [71, 72], music generation [70, 182], and time series prediction [89, 195] before the rise of transformers. Their core appeal is the ability to maintain and update a hidden state across arbitrarily long input sequences, making them natural models for temporal dependencies. RNNs are trained by backpropagation through time (BPTT) [236], which unrolls the recurrence and propagates gradients backward through each timestep. Because this backward pass repeatedly multiplies by the recurrent weight matrix, gradients tend to shrink or grow exponentially with the sequence length, the well-known vanishing and exploding gradient problem that makes long-range dependencies difficult to learn [16, 167]. Gated variants such as Long Short-Term Memory (LSTM) [95] and Gated Recurrent Units (GRU) [39] were introduced specifically to address this difficulty, using learned gate mechanisms to control information flow and mitigate these gradient pathologies. Despite these advances, training RNNs reliably remains challenging, and a key source of difficulty is initialization: the choice of initial weights determines whether the recurrent dynamics are well-conditioned from the outset, and poor initialization can prevent learning even when the architecture and optimizer are otherwise appropriate.

In deep feedforward networks, initialization research established that weights should be scaled to balance signal variance across layers. Xavier initialization [66] derived the appropriate scale for saturating activations such as tanh and sigmoid, while He initialization [84] extended the analysis to ReLU networks. Orthogonal initialization [196] and the related concept of dynamical isometry [170] further showed that preserving singular values of the input-output Jacobian near one is a stronger condition that accelerates learning in deep

linear and nonlinear networks. These principles share a common assumption: each weight matrix is applied once per forward pass, so the analysis reduces to ensuring layer-to-layer variance is balanced.

This assumption breaks down entirely for recurrent networks. In an RNN, the recurrent weight matrix W_h is applied at every timestep, so a forward pass of length T involves the product W_h^T . Even a slight deviation of the spectral radius from one causes this product to either explode or vanish exponentially with T , regardless of how carefully the initial variance was calibrated [16, 167]. Pascanu et al. [167] made this gap explicit by showing that the gradient of the loss with respect to early hidden states involves a product of T Jacobians, and that the conditions for stable gradient flow in RNNs are therefore fundamentally stricter than those for feedforward networks: the spectral radius of the recurrent Jacobian must remain near one, not just at initialization but throughout training. Standard feed-forward initialization heuristics provide no such guarantee, motivating the spectral control and dynamical stability methods described below.

The responses to this challenge fall into two broad categories. The first applies direct algorithmic fixes: gradient clipping and normalization [167, 36, 70] prevent the most damaging gradient explosions but leave the underlying spectral instability unaddressed. The second, more principled category treats spectral control as a design constraint. Orthogonal and unitary parameterizations constrain the recurrent weight matrix so that its eigenvalues lie on the unit circle, ensuring that the recurrent Jacobian is norm-preserving at every step [5, 238, 147, 86, 229, 141, 118, 148]. The spectral distribution in the complex plane has been identified as a direct indicator of stability: networks whose Jacobian eigenvalues cluster near the unit circle exhibit well-conditioned long-horizon dynamics [75, 117]. A complementary structural approach designs the recurrent update to correspond to a stable numerical discretization of an ordinary differential equation: antisymmetric RNNs [30] enforce purely imaginary eigenvalues by construction, and coupled oscillatory architectures [189] draw on Hamiltonian structure to obtain conservative dynamics that neither grow nor decay.

A distinct line of work sidesteps recurrent training instability by fixing the recurrent connectivity entirely. In reservoir computing, a randomly initialized recurrent network drives a learned linear readout, reducing training to a convex regression problem [104, 140].

FORCE learning extends this framework with online recursive least-squares updates on the readout, enabling the network to track complex time-varying targets [212]. Despite its efficiency, FORCE succeeds reliably only in a narrow dynamical regime near the edge of chaos, where the spectral radius of the recurrent matrix is close to one, and the reservoir exhibits rich but bounded activity [212, 47]; small deviations from this regime cause learning to diverge or stall. R-FORCE addresses this sensitivity by actively shaping the Jacobian eigenvalue distribution to maintain stable, expressive dynamics across a much broader range of initializations.

3.2 Hyperparameter Search

The preceding section showed that initialization, viewed through a dynamical systems lens, can be designed with principled theoretical guarantees: conditions such as spectral radius constraints and dynamical isometry provide clear criteria for well-conditioned recurrent dynamics. Initialization is, however, just one of many design choices that must be specified before training, and it is the exception rather than the rule in having such theoretical foundations. Learning rate schedules, regularization strengths, batch sizes, optimizer settings, and architectural dimensions such as layer width and depth are equally consequential but far less amenable to principled design: their effect on performance is task-dependent, non-monotonic, and difficult to predict without empirical evaluation. When all of these choices are considered jointly, the space of possible configurations is combinatorially large, and the theoretical tools used to analyze initialization do not carry over. Hyperparameter optimization (HPO) addresses this by treating configuration selection as a search problem, with the goal of identifying settings that maximize validation performance under a fixed computational budget [18, 202, 97, 24]. As models scale and training runs grow more expensive, the cost of HPO itself increasingly dominates overall compute, making the efficiency of the search procedure a first-order concern [24, 177].

The simplest proposal strategies, grid search and random search, evaluate configurations drawn from a predefined grid or uniformly at random; random search has been shown to offer better coverage in high-dimensional spaces for the same number of evaluations [18]. Bayesian optimization (BO) improves sample efficiency by fitting a probabilistic surrogate

model to the history of evaluations and selecting new candidates through an acquisition function that balances exploration and exploitation [202, 97, 203]. Tree-structured Parzen Estimators (TPE) and their adaptive extensions are widely integrated into practical HPO and AutoML frameworks [17, 219, 53]. Evolutionary and population-based methods jointly optimize hyperparameters and model weights across a population of candidates; Population-Based Training (PBT) has become prominent in reinforcement learning and large-scale supervised settings [103, 52, 44].

A key bottleneck shared by all these approaches is that each configuration must be evaluated by training a model, which is expensive. Multi-fidelity HPO mitigates this by exploiting cheap low-fidelity approximations (fewer training epochs, subsampled data, or reduced model size) to eliminate unpromising candidates early and reserve full-budget evaluations for the most promising ones [23, 123, 58]. Successive Halving and Hyperband formalize this as a bandit-style resource allocation problem [123], and BOHB combines this scheduling with Bayesian optimization [58]. Learning curve extrapolation methods go further by predicting final performance directly from early training trajectories, enabling termination decisions without even completing a low-fidelity run [50, 114]. Recent extensions address multi-objective and resource-aware settings that balance accuracy against runtime, memory, or energy costs [153], as well as meta-learning strategies that warm-start the search using knowledge transferred from related tasks [177, 98].

Proxy-based and zero-cost methods take the efficiency argument further by predicting model performance from properties computed at or near initialization, sidestepping training entirely. In the neural architecture search (NAS) literature, zero-cost proxies such as the gradient signal (SNIP [121]), the gradient-Hessian product (GraSP [231]), the path-norm (SynFlow [216]), and kernel-based separability scores [35] score architectures from a single forward-backward pass. For recurrent architectures, however, existing proxies are limited because they treat the network as a static function and ignore the temporal structure of RNN dynamics. AeLLE and Hyperpruning address this gap by using Lyapunov spectra as structured performance proxies that capture the dynamical regime of the network from early in training, providing a principled and computationally efficient basis for hyperparameter and architecture selection.

3.3 Representation Learning

Representation learning seeks to map raw, high-dimensional observations into lower-dimensional features that expose the underlying explanatory factors of the data, making them easier to compare and to reuse across downstream tasks [14]. Classical methods approach this with fixed, largely unsupervised criteria. Principal component analysis (PCA) projects data onto the orthogonal directions of greatest variance, yielding the optimal linear embedding in a least-squares sense [106], and kernel PCA extends this to nonlinear structure by solving the same eigenproblem in a reproducing-kernel feature space [197]. Manifold learning methods instead assume the data lie near a low-dimensional manifold and seek coordinates that preserve its local geometry, including Isomap, which preserves geodesic distances [218], locally linear embedding, which reconstructs each point from its neighbors [186], Laplacian eigenmaps, which preserve neighborhood structure through the graph Laplacian [12], and t-SNE, which matches a probabilistic notion of neighborhood similarity for visualization [221]. These approaches are powerful but typically produce a fixed embedding tied to a specific dataset and distance metric, and most do not provide an explicit encoder that generalizes to unseen inputs.

Deep representation learning replaces these fixed criteria with mappings learned by neural networks, which capture richer nonlinear structure and yield an explicit, reusable encoder. Autoencoders are the canonical unsupervised instance: an encoder compresses an input into a latent code and a decoder reconstructs it, so that minimizing reconstruction error forces the latent space to retain the input’s salient structure [91]. Denoising and variational variants regularize this latent space, the former by reconstructing clean inputs from corrupted ones [225] and the latter by imposing a probabilistic prior that yields a smooth, generative latent geometry [113, 181].

A latent space is most useful when its geometry reflects a downstream quantity of interest. Metric and contrastive learning shape embeddings so that distances encode semantic similarity [76, 34], and supervised or semi-supervised autoencoders augment the reconstruction objective with a predictive term, making the latent code simultaneously faithful to the input and informative about a target label. A related line of work learns latent

representations of dynamical systems themselves, embedding complex temporal behavior into low-dimensional coordinates in which the dynamics become simpler to analyze [156]. AeLLE builds on these ideas. It trains an autoencoder on Lyapunov spectra of varying lengths to produce a fixed-size latent embedding whose geometry correlates with model accuracy across architectures, converting a heterogeneous collection of spectra into a single comparable, performance-predictive representation.

3.4 *Generative Modeling and Diffusion Models*

Generating realistic images, audio, and other high-dimensional data has been a longstanding goal of machine learning. Early deep generative models approached this through two dominant frameworks. Variational Autoencoders (VAEs) [113, 181] cast generation as probabilistic inference, learning a latent space via an encoder-decoder architecture and training it with an evidence lower bound objective. While tractable and stable to train, VAEs frequently produce blurry or over-smoothed outputs because the Gaussian decoder assumption is too restrictive to capture sharp, multi-modal distributions [43]. Generative Adversarial Networks (GANs) [69, 174, 109, 110, 25] instead pit a generator against a discriminator in an adversarial game, achieving considerably sharper samples; however, GANs are notoriously difficult to train, prone to mode collapse, and sensitive to architectural and hyperparameter choices [4, 137, 146]. Diffusion models emerged as a third paradigm that sidesteps both issues by training a network to reverse a gradual noising process, achieving a stable training objective, improved mode coverage, and state-of-the-art image quality. Figure 3.1 illustrates the contrast among the three paradigms.

The diffusion paradigm, which overcame the limitations of VAEs and GANs, developed over roughly a decade of converging work. The foundational idea was introduced by Sohl-Dickstein et al. [204], who framed generation as learning to reverse a fixed Markovian noising process using tools from non-equilibrium thermodynamics, but the framework was initially difficult to scale. Song and Ermon [208] independently developed score-based generative models, estimating gradients of the data log-density (the score function) and sampling via Langevin dynamics, and subsequently showed that training on a sequence of noise levels (NCSN) substantially improved generation quality [208]. The breakthrough in practical

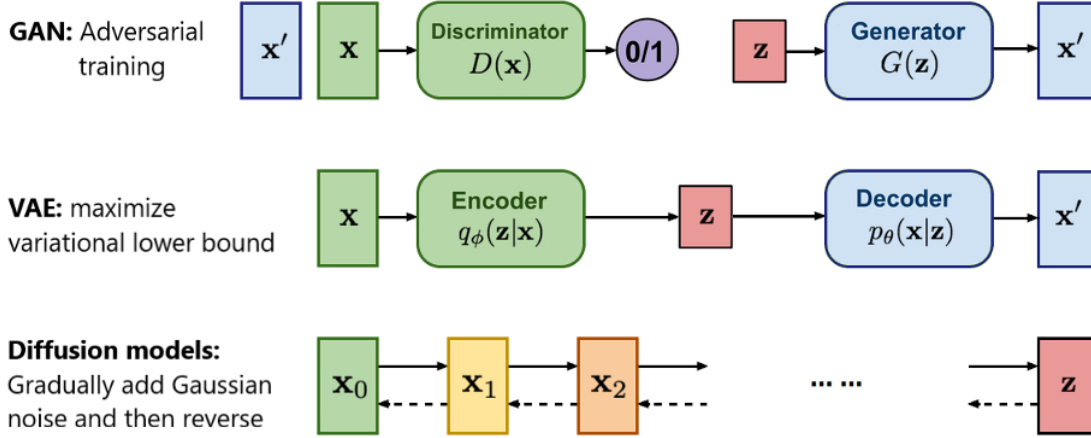


Figure 3.1: **Comparison of generative modeling paradigms.** GANs rely on adversarial training between a generator and a discriminator. VAEs optimize a variational lower bound on the data likelihood. Diffusion models gradually corrupt data with noise and learn to reverse the denoising process, achieving superior quality and training stability at the cost of slower inference.

image synthesis came with Denoising Diffusion Probabilistic Models (DDPM) [92], which established the noise-prediction training objective and produced high-quality unconditional samples. Improved DDPM [158] introduced learned variance schedules and cosine noise schedules, and Dhariwal and Nichol [48] demonstrated with an improved U-Net backbone and classifier guidance that diffusion models could surpass state-of-the-art GANs on image quality benchmarks. Song et al. [210] subsequently unified score-based and DDPM approaches within a common stochastic differential equation (SDE) framework, thereby establishing the theoretical foundations of the field.

Scaling diffusion to high-resolution and text-conditioned generation required both architectural and algorithmic advances. Latent Diffusion Models (LDMs) [183] moved the diffusion process into a compressed VAE latent space, dramatically reducing computational cost while preserving perceptual quality; this work became the foundation of Stable Diffusion,

the first widely deployed open-source text-to-image system. Concurrent large-scale text-to-image systems such as DALL-E 2 [179] and Imagen [191] demonstrated that conditioning on powerful text encoders (CLIP and T5, respectively) yields strong prompt alignment and photorealistic outputs, establishing text-to-image diffusion as a practically important capability. SDXL [171] extended the LDM framework with larger U-Net backbones, multi-aspect-ratio training, and a two-stage refinement process, setting a new quality bar for open-weight generation. On the architectural side, the Diffusion Transformer (DiT) [169] replaced the U-Net entirely with a Vision Transformer operating on latent patches, demonstrating that the favorable scaling laws of transformers apply to diffusion and achieving state-of-the-art class-conditional generation. Subsequent transformer-based architectures have extended this design space with timestep-conditioned attention mechanisms [83], spatially adaptive token density [105, 3], and hybrid autoregressive-diffusion formulations [249].

Despite their quality advantages, diffusion models are substantially slower to sample from than VAEs or GANs because generation requires iterating a large denoising backbone for many steps, each of which involves a full forward pass. A major research direction addresses this by reducing the number of denoising steps. DDIM [206] was among the first to observe that the Markovian sampling process can be recast as a deterministic ODE, enabling high-quality synthesis in tens of steps rather than hundreds. Higher-order numerical solvers such as DPM-Solver [136], DEIS [247], and EDM [108] further reduce the number of steps by more accurately discretizing the reverse-diffusion ODE. Schedule optimization methods [232] search for noise schedules that concentrate denoising steps in the most informative regions of the trajectory. Distillation-based samplers take a more aggressive approach by training a student model to reproduce the multi-step teacher’s output in far fewer steps: progressive distillation [194] halves the required steps iteratively, while consistency models [207, 138] train the network to produce the same denoised output regardless of which timestep it is queried at, enabling one- or few-step generation. A related reformulation, flow matching [129, 133], learns straight-line probability flows between noise and data distributions, further simplifying the trajectory and reducing the number of function evaluations needed. Complementary to these step-reduction methods, feature-caching approaches exploit the high similarity of intermediate activations between

adjacent denoising steps. DeepCache [139] caches and reuses high-level U-Net features across steps, applying the full network only at selected key frames. Further work has extended caching to cross-attention outputs, which stabilize early in the denoising trajectory and need not be recomputed at every step, as well as to transformer-based backbones and learnable cache schedules [237, 131, 2]. A related strategy maintains a model zoo or cascade of denoisers of varying capacity: coarser or lighter models handle low-frequency, easy-to-denoise steps, while the full model is reserved for high-frequency, detail-critical steps [10, 93]. Speculative decoding methods borrow the draft-then-verify paradigm from autoregressive language models, using a small draft denoiser to propose candidate trajectories that are then selectively accepted or corrected by the large model, reducing the expected number of full-model evaluations [124, 37]. Adaptive computation methods similarly route each timestep through lightweight vs. full-capacity subnetworks based on estimated denoising difficulty [130, 164, 246].

3.5 Neural Networks as Dynamical Systems

Neural networks process inputs through sequences of nonlinear transformations, and when those transformations are recurrent or iterative, the qualitative behavior of the resulting dynamical system has direct consequences for both learning and generalization. Linearization is a central analytical tool in this context: by approximating the nonlinear dynamics with a linear map near a point or along a trajectory, one can apply eigenvalue and spectral theory to characterize stability, convergence, and sensitivity to perturbations. Local linearization around fixed points and slow points of a recurrent network, via the Jacobian of the hidden-state update, reveals the stability regime, bifurcation structure, and computational mechanisms of the network [213, 142, 67, 230]. Many RNN architecture designs are directly motivated by controlling this linearized dynamics: constraints on the spectral norm of the recurrent Jacobian prevent trajectories from exploding or vanishing, and the orthogonal, unitary, and Lipschitz RNN families can all be understood as imposing a well-conditioned linear approximation at every step [16, 167]. A continuous-time formulation makes this correspondence explicit: Neural ODEs parameterize the hidden state’s evolution directly as an ordinary differential equation whose vector field is a neural network [33], so

that sensitivity and stability are governed by the Jacobian of that field, and interpreting residual and recurrent networks as discretizations of such ODEs has motivated stability-constrained designs [75]. Spectral analysis of long Jacobian products captures global gradient stability but has limited applicability to practical architectures, because the spectra of such products reflect intricate long-term dependencies that simple eigenvalue criteria cannot fully capture. A more global linearization strategy applies Koopman operator theory to embed the full nonlinear RNN dynamics into a linear operator on an extended observable space, demonstrating that such representations capture the dominant dynamical modes and achieve comparable performance to their nonlinear counterparts [156]. The broader tradition of analyzing neural network dynamics through statistical physics traces to Sompolinsky et al. [205], who showed that randomly connected networks undergo a phase transition from ordered to chaotic dynamics as the weight distribution gain increases, with the boundary (the edge of chaos) marking the regime of maximal computational capacity. Bertschinger and Natschläger [22] reinforced this picture in the reservoir computing setting. Universality analyses of RNN fixed points further suggest that architecturally distinct networks can be organized into similarity classes that exhibit similar dynamics when optimally trained, thereby motivating the development of tools to characterize and compare RNN variants within a unified framework without relying on architecture-specific approximations.

Among the dynamical tools that meet this need, Lyapunov exponents provide a natural trajectory-level generalization of local linearization. Rather than examining the Jacobian at a single fixed point, they track how infinitesimal perturbations grow or shrink along entire orbits, yielding a global characterization that is not tied to any particular reference point or equilibrium. Lyapunov exponents (LEs) quantify this asymptotic rate at which nearby trajectories in state space diverge or converge, and the full collection, the Lyapunov spectrum, provides a multi-dimensional characterization of the dynamical regime. Individual spectral features carry distinct dynamical interpretations: the maximal LE determines linear stability; positive exponents signal chaotic dynamics, with larger magnitudes indicating stronger chaos; an entirely negative spectrum signals convergence to a fixed-point attractor; zero exponents correspond to limit cycles or quasiperiodic orbits; the mean exponent reflects the rate of phase-space volume contraction, analogous to Kolmogorov-Sinai entropy;

and the spectral variance captures heterogeneity in stability across directions, reflecting the conditioning of long Jacobian products [172, 205]. Oseledets’ multiplicative ergodic theorem guarantees that the spectrum is well-defined and trajectory-invariant within an ergodic component, and random dynamical systems theory extends this guarantee to non-autonomous systems such as RNNs driven by stationary input sequences, ensuring that the spectrum captures intrinsic network properties rather than artifacts of a particular input realization [63]. In the context of deep feedforward networks, Poole et al. [172] used LE-based analysis to study how the input-space geometry evolves across layers, linking vanishing and exploding gradients to mean-field signal propagation theory. For recurrent networks, the Lyapunov spectrum has been used to characterize stability [75], measure the effective memory capacity of a reservoir [104], serve as a diagnostic for the edge-of-chaos regime [21], and analyze how information propagates through RNN hidden states via dynamical systems tools [227]. The QR-decomposition algorithm for computing the full spectrum [63] makes these analyses tractable for networks of practical size. While individual spectral features have been shown to correlate with network performance, the selection of relevant features remains task-dependent and difficult to determine a priori. AeLLE addresses this by learning a compact autoencoder representation of the full spectrum that enables cross-architecture performance prediction, and Hyperpruning uses the Lyapunov spectrum distance as a proxy to guide the pruning search. These analyses are well-suited to stationary recurrent systems, but rest on the assumption that the governing dynamics are time-invariant, and that trajectories can be run long enough for asymptotic behavior to emerge. This assumption fails for systems whose equations of motion change explicitly at every step.

For such time-varying or finite-horizon systems, standard Lyapunov exponents carry two important limitations: 1) They assume time-invariance and rely on analytically known dynamics, whereas systems such as diffusion models are explicitly time-dependent and defined only through learned parameters; 2) The asymptotic behavior they characterize may not converge within the finite horizon of practical interest. Finite-Time Lyapunov Exponents (FTLE) address both limitations by measuring the maximum local expansion or contraction of the flow map over a prescribed time window, characterized through the singular values of the flow map’s Jacobian [199]. Originally developed in fluid dynamics to identify coherent

structures and transport barriers in geophysical flows [77, 199] and subsequently adopted in biomedical modeling to analyze cardiovascular flows and tissue deformation, FTLE have since been applied to characterize sensitivity in sequential machine learning systems. Computing FTLE for high-dimensional networks is expensive because it requires evaluating and tracking the full Jacobians across the entire trajectory. Scalable approximations based on random projections make computation tractable by estimating the leading singular vectors using a small number of vector-Jacobian products without forming the full Jacobian matrix. The diffusion denoising chain is a natural setting for FTLE analysis: the denoising network changes at every step due to its explicit timestep conditioning, making global LEs inapplicable, while FTLE over the trajectory capture how per-step perturbations compound into trajectory-level deviations. The second-order matching objective developed in 2ndMatch is directly inspired by this FTLE perspective, aligning the sensitivity patterns of the pruned and dense denoising networks through projected second-order Jacobian matching.

3.6 *Knowledge Distillation*

Knowledge distillation is one of several paradigms for reusing the knowledge captured in a trained network. Transfer learning reuses a model pretrained on a large source task, as an initialization or a fixed feature extractor, to accelerate learning on a related target task [163, 241]. Continual or incremental learning extends a single model to a sequence of tasks without revisiting earlier data, where the central obstacle is catastrophic forgetting; distillation is a standard remedy, constraining the updated model to reproduce the predictions of its earlier self [126, 253, 180]. These paradigms share a common mechanism, transferring the functional behavior of one network to another, and differ mainly in purpose: adapting to a new task, retaining old ones, or, in the teacher-student compression setting that is the focus of this section, reproducing a large model’s behavior in a smaller one.

In this compression setting, the student is trained to reproduce the outputs or internal representations of a larger teacher without matching its architecture, receiving supervision from the teacher rather than from ground-truth labels alone. The foundational approach introduced by Hinton et al. [90] trains the student to minimize the KL divergence between its softened predictions and the teacher’s, transferring the relative similarity struc-

ture among classes that hard labels discard. Feature-level distillation extends supervision to intermediate activations, training the student to produce hidden representations similar to the teacher’s at selected layers [184]. Subsequent work has proposed matching attention maps [243], relational structures [165], and flow-of-solution procedures [168]. All of these methods constrain the student at discrete training points but leave it free to deviate in the neighborhood of each point.

Sobolev training [42] addresses this limitation by augmenting the distillation objective with derivative-matching terms, enforcing agreement between teacher and student not just at observed points but in a local neighborhood around each. Matching higher-order derivatives provides supervision over an expanding neighborhood around each training example, making the approach especially valuable under limited data. First-order Jacobian matching is equivalent to output matching at noise-perturbed inputs, connecting derivative-level supervision to standard data augmentation practice. Computing the full input Jacobian is expensive, requiring one backward pass per output dimension; random-projection estimators reduce this to a small number of vector-Jacobian products [211].

Applying these distillation objectives to the diffusion model compression introduces challenges specific to the multi-step denoising process. LAPTOP-Diff, for instance, combines output- and feature-level distillation after one-shot structured pruning [245]. These standard objectives match teacher and student at each individual timestep but do not account for the compounding of per-step errors along the trajectory. Because the denoising chain is a time-varying dynamical system, small per-step deviations can accumulate into large degradation at the final sample, even when the per-step outputs are close. First-order Jacobian matching between teacher and student is also insufficient in this setting: as the standard denoising training objective already aligns the first-order input response through the noise prediction loss, naive first-order matching adds no new signal. 2ndMatch addresses this by aligning the second-order sensitivity of the denoising chain using a formulation inspired by FTLE.

3.7 *Network Pruning*

The growing computational demands of modern neural networks, spanning recurrent architectures for sequential modeling [95, 39] and large generative models for image synthesis [183, 171, 169], have made model compression a practical necessity for deployment [38, 79]. Model compression encompasses several complementary strategies. Knowledge distillation trains a compact student to reproduce the behavior of a larger teacher, as covered in Section 3.6. Quantization reduces the numerical precision of weights and activations from floating point to low-bit integer representations, cutting both memory footprint and arithmetic cost [101, 96, 155]. For diffusion models, quantization requires timestep-aware calibration because the noise statistics encountered at each denoising step differ substantially [200, 125, 250, 32]. Low-rank decomposition factorizes weight matrices or tensors into products of smaller factors, reducing parameter count while preserving the linear transformation up to approximation error [46, 192]. Among these strategies, network pruning improves efficiency by identifying and removing parameters or structural units that contribute the least to the model’s output, reducing both parameter count and inference cost without fundamentally altering the architecture. Unstructured pruning zeros out individual weights [80, 79], achieving high sparsity ratios with minimal quality loss but producing irregular sparsity patterns that are inefficient on standard hardware. A middle ground is semi-structured (m:n) sparsity, which requires that exactly m out of every n consecutive weights be non-zero; the 2:4 pattern in particular is natively accelerated by NVIDIA Ampere Sparse Tensor Cores, offering hardware speedups with less accuracy loss than fully unstructured sparsity [150, 254]. Structured pruning removes entire filters, attention heads, or transformer blocks [122], yielding dense subnetworks that execute efficiently without specialized sparse kernels. For large-scale generative models where inference latency on standard hardware is the primary bottleneck, structured pruning is the preferred approach.

Traditional pruning pipelines train a dense network, apply an importance criterion, and fine-tune to recover accuracy. Magnitude-based criteria remove units with the smallest weight norms, relying on the assumption that small weights contribute little to the output [80, 122]. Regularization during training can reinforce this assumption by driving

unimportant weights toward zero through ℓ_1 sparsity [134] or group sparsity [235]. More principled criteria apply Taylor expansions to estimate the effect of removing each unit on the training loss; first-order criteria such as SNIP [121] and Taylor importance [152] use gradient-weight products, while second-order methods such as Optimal Brain Damage [120] and Optimal Brain Surgeon [82] incorporate Hessian information. Dynamic sparse training methods (SET [151], RigL [57]) maintain fixed sparsity throughout training via periodic prune-and-regrow cycles, avoiding the separate training and pruning phases.

Structured pruning of recurrent networks presents challenges beyond the feedforward case because the recurrent Jacobian depends on the learned dynamics, and removing units can destabilize the temporal behavior even when the per-step output change is small. Selfish-RNN demonstrated that non-uniform sparsity allocation across gating components yields large performance gains under high sparsity [132]. Lottery-ticket-style methods have been extended to RNNs, showing that sparse subnetworks of LSTMs can match the performance of their dense counterparts when initialized from the same seed [62, 154]. Hyperpruning addresses the search problem over pruned RNN variants by using the Lyapunov spectrum distance as a proxy to identify better-performing configurations without requiring full re-training of each candidate.

Pruning diffusion backbones has attracted growing attention as a practical path to reducing inference cost without the limitations of step-reduction methods alone. Diff-Pruning applied Taylor-based importance scoring to remove weight groups [61], and BK-SDM compressed Stable Diffusion by removing residual and attention blocks and retraining the smaller backbone with distillation [112]. LD-Pruner measured block importance by how convolutional and attention layers affect latent statistics relative to the dense model [29]. LAPTOP-Diff proposed a one-shot layer-level criterion based on output loss differences, followed by output- and feature-level distillation fine-tuning [245]. EcoDiff introduced a differentiable mask optimized across the full sampling trajectory [248]. DiP-GO formulated pruning as a subnetwork search using a lightweight controller to assign block-level importance scores [255]. OBS-Diff adapts Optimal Brain Surgeon to diffusion architectures with a timestep-aware Hessian [256]. Comprehensive surveys underscore that respecting model-specific temporal dynamics during compression remains an open challenge [38].

Chapter 4

R-FORCE: ROBUST TRAINING OF RANDOM RECURRENT NEURAL NETWORKS THROUGH JACOBIAN EIGENVALUE SHAPING

4.1 Motivation

Recurrent neural networks (RNNs) remain fundamental tools for modeling temporal and sequential structure in data. Among them, Random RNN (RRNNs) are particularly appealing due to their architectural simplicity and computational efficiency. Low-rank variants such as Rank1- and RankN-RRNNs further capture biologically inspired motifs: a recurrent population generates rich dynamics, while a low-dimensional readout aggregates activity to produce task-relevant outputs. Despite these advantages, training RRNNs through gradient descent remains highly unstable due to vanishing and exploding gradients. FORCE learning [212] offers an alternative: a recursive least-squares update with feedback control that enables RRNNs to perform *target-learning*, autonomously generating a prescribed multi-dimensional sequence for thousands of timesteps with no external driving input. FORCE learning is particularly effective in low-rank architectures, where a low-dimensional readout feeds its output back to modulate the recurrent population. This efficiency carries a hidden cost: in a Rank1-RRNN, only the readout weights are trainable, so the recurrent reservoir is fixed at its initial values, and the entire burden of producing well-conditioned, trainable dynamics falls on how that reservoir is initialized.

However, FORCE learning is notoriously sensitive to the initialization of the recurrent connectivity matrix. RRNNs typically operate in either a quiescent regime—where activity rapidly decays—or a chaotic regime—where small perturbations amplify unpredictably [205]. Both regimes are effectively non-trainable: FORCE can neither inject lasting activity into a dissipative reservoir nor rein in a chaotic one. Only a narrow band of the global scaling parameter $g \in [1.3, 1.5]$, yields stable dynamics near the *edge of chaos*, where FORCE learning reliably succeeds. Even within this band, up to 15% of trials fail due to

instability, divergence, or degradation during testing. Full-FORCE [47] mitigates failures by introducing an auxiliary stabilizing network, but this substantially increases computational cost and still does not eliminate instability.

The underlying fragility points toward the spectral properties of the recurrent dynamics. The Jacobian of the RRNN governs how perturbations evolve through time, providing a principled lens for analyzing stability [75, 117]. Eigenvalues outside the numerical stability region lead to divergence; eigenvalues deep inside it cause accelerated decay and loss of memory. Thus, successful training and long-horizon generation require that the Jacobian spectrum remain near the stability boundary—neither vanishing nor exploding. While architectures such as orthogonal, unitary, or antisymmetric RNNs [5, 86, 30] enforce stable spectra through carefully constructed initializations, these designs target gradient-based learning and do not directly apply to FORCE learning, where only readout weights are updated.

These observations motivate the need for a connectivity initialization tailored specifically to FORCE learning—one that directly shapes the Jacobian spectrum to remain near the stability boundary throughout training and generation. By analyzing how FORCE interacts with RRNN spectral structure, I derive principles for constructing connectivity matrices that maintain stability across a much broader range of scaling parameters g . This leads to *Robust FORCE (R-FORCE)*, a spectral initialization scheme that stabilizes RRNN dynamics, reduces sensitivity to hyperparameters, and significantly expands the effective operating regime of FORCE learning beyond the narrow edge-of-chaos window.

4.2 Method: R-FORCE Initialization

A Rank1-RRNN is a firing-rate model whose neuronal state $\mathbf{x}(t) \in \mathbb{R}^N$ evolves as

$$\tau \frac{dx_i}{dt} = -x_i + g \sum_{j=1}^N M_{ij} r_j + (\mathbf{w}_F)_i z, \quad (4.1)$$

with rates $\mathbf{r}(t) = \tanh(\mathbf{x}(t))$ and scalar readout $z(t) = \mathbf{w}(t)^T \mathbf{r}(t)$, where τ is the time constant, g scales the fixed recurrent matrix M , the feedback vector $\mathbf{w}_F$ is fixed, and only the readout $\mathbf{w}(t)$ is updated during training. To quantify how an early perturbation

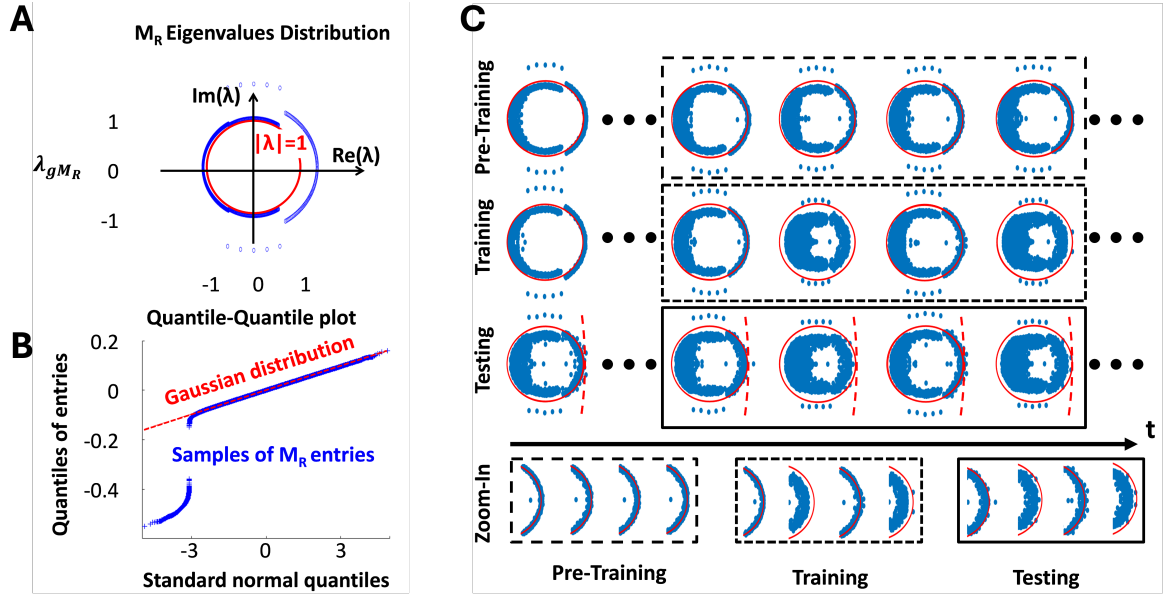


Figure 4.1: **R-FORCE Initialization Properties.** (A) Spectral distribution of M_R . (B) Q-Q plot of M_R entries (blue) versus a standard Gaussian (red). (C) Evolution of $\lambda_{J(t)}$ during pre-training (top), training (middle), and testing (bottom), shown relative to the stability boundary (dashed red). Zoomed-in views highlight the right boundary. All plots use $g = 1.5$.

propagates through this evolution, I linearize the current state $\mathbf{x}(t)$ about the initial state $\mathbf{x}(0)$, yielding the Jacobian

$$J(t) = -\mathcal{I} + (gM + \mathbf{w}_F \mathbf{w}(t)^T) \times \text{Diag}(\mathbf{1} - \tanh^2(\mathbf{x}(t))), \quad (4.2)$$

which is therefore a key indicator of training stability. I evolve Equation 4.1 with the forward-Euler method, whose numerical stability region is a disk in the complex plane centered at $-\tau$ with radius τ . Eigenvalues of $J(t)$ that fall inside this disk correspond to contracting modes that dissipate information, whereas eigenvalues outside it grow and destabilize the trajectory; reliable target-learning therefore requires the spectrum of $J(t)$ to remain near the boundary of this region throughout the generation horizon, neither vanishing nor diverging.

Four components contribute to $J(t)$: the readout $\mathbf{w}(t)$, state $\mathbf{x}(t)$, feedback vector $\mathbf{w}_F$,

and the recurrent matrix M . The first three only introduce rank-1 perturbations [27], leaving the eigenvalue structure of M as the primary determinant of Jacobian stability. Empirically, I observe that the spectrum of $J(t)$ closely tracks that of M across pre-training, training, and testing. This motivates the design of an initialization for M whose spectral structure naturally induces stable Jacobian dynamics.

To achieve this, I introduce *R-FORCE*, a principled initialization in which the recurrent matrix is constructed as $M_R = V_R D_R V_R^T$, where both the eigenvectors V_R and eigenvalues D_R are engineered to maintain stability throughout training.

Eigenvector construction. To ensure real-valued connectivity, V_R must contain conjugate eigenvector pairs [222]. I obtain such an orthogonal basis by sampling a random matrix $M_N \sim \mathcal{N}(0, 1)$, forming an antisymmetric matrix $M_A = M_N - M_N^T$, and taking its eigendecomposition. This yields an orthogonal matrix with the required conjugate-pair structure.

Eigenvalue design. The central idea of R-FORCE is to shape the eigenvalues λ_{M_R} so that the induced Jacobian spectrum remains near the boundary of the numerical stability region—avoiding both rapid decay and explosive divergence. Through extensive empirical exploration, I identify four principles for a stable spectrum:

1. *Circular structure*: eigenvalues lie on circles whose radii control uniform contraction or expansion.
2. *Multiple radii*: distributing eigenvalues across several radii avoids the narrow stability range produced by a single circle.
3. *Non-uniform density*: outer radii, being more unstable, receive fewer eigenvalues; inner radii receive proportionally more.
4. *Segmented arcs*: placing eigenvalues only on selected angular sectors prevents drift into unstable or overly dissipative quadrants.

These principles emerge from a progression of increasingly capable distributions, each addressing the failure mode of the previous one. A single circle of radius g reproduces the behavior of standard FORCE, controlling the error only in the narrow band $g \in [1.2, 1.4]$. Spreading eigenvalues across several concentric circles, with the largest and smallest radii chosen to span the full range of g , extends robustness to small g but not to large g ; mak-

ing the radii non-uniform recovers performance for large g , though at the expense of small g . Weighting the circles with an eigenvalue density that decreases with radius, so that high-magnitude (more readily unstable) eigenvalues are rare, tightens the confidence interval across all g . Finally, restricting eigenvalues to angular arcs rather than full circles, and shifting those arcs with g , keeps the rightmost component of the spectrum within the empirically optimal band $[1.2, 1.4]$, which I find to be the single most important factor for stable training. Combining all four principles yields the R-FORCE distribution.

The resulting spectrum (Fig. 4.1.A) consists of four arcs with radii

$$[r_1, r_2, r_3, r_4] = [0.7, 0.72, 0.9, 1.2] \times g.$$

Eigenvalue densities are defined inversely proportional to radius:

$$l_i = \frac{g^2}{|r_i - 1.15|}, \quad d_i = Q \cdot \frac{l_i}{\sum_{j=1}^k l_j},$$

where d_i is the fraction of eigenvalues assigned to the i -th arc, and (Q, k) is adjusted based on the largest radius to prevent instability at high g . Angular sectors (θ_i) are chosen so that the rightmost portion of the spectrum consistently lies in the empirically optimal region $[1.2, 1.4]$. A fixed-arc baseline confirms that this adaptive angular placement is essential for robustness.

Connectivity construction. Given V_R and D_R , I construct the recurrent matrix via $M_R = V_R D_R V_R^T$. Because this procedure combines random antisymmetric seeds with a controlled spectral structure, it generates many distinct yet reliably stable initializations.

Resulting spectral and dynamical properties. Figure 4.1 summarizes the resulting behavior for $g = 1.5$. Panel A shows the structured arc-shaped eigenvalue distribution of M_R . Panel B presents a Q-Q comparison of matrix entries against a standard Gaussian: most entries align closely with a zero-mean normal distribution, with mild heavy-tail deviations introduced by the spectral constraints.

Panel C visualizes how the Jacobian eigenvalues evolve throughout pre-training, training, and testing. Prior to training, eigenvalues remain tightly clustered near the stability

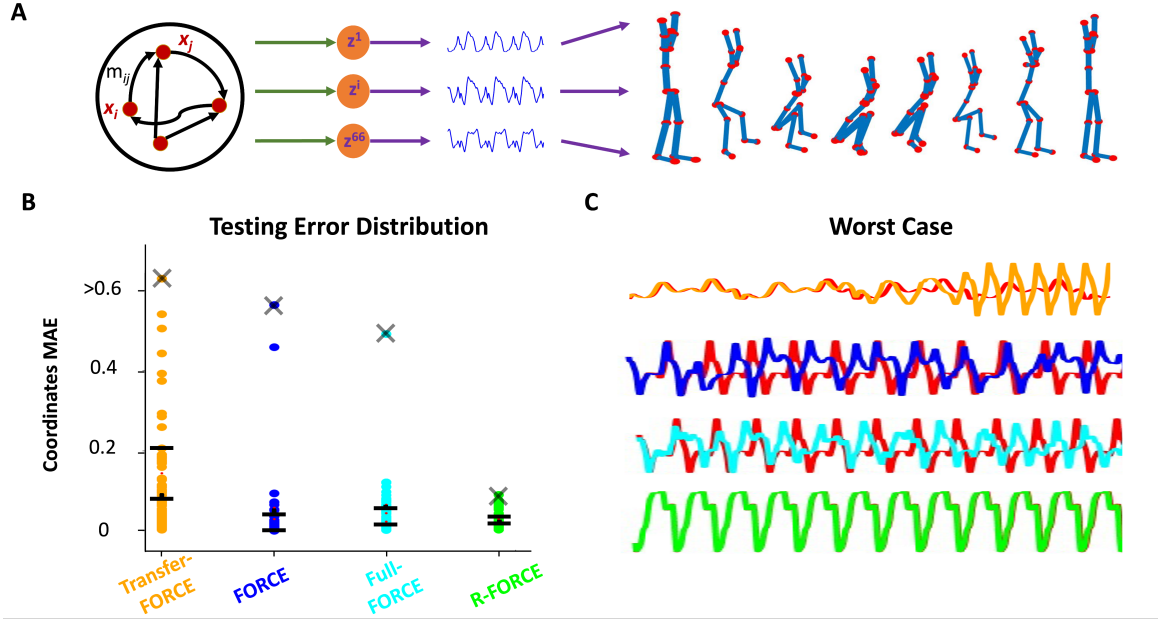


Figure 4.2: **Generation of body joints movements with Rank1-RRNN.** (A) Rank1-RRNN with 66 readout units for the generation of joint movements time series. (B) Errors of Transfer-FORCE (orange), FORCE (blue), Full-FORCE (cyan), and R-FORCE (green) in generating the evolution of the deep squat movement. Each dot corresponds to a readout unit result. The red cross denotes the readout unit with the largest error (worst case) for each method. (C) The evolution of the readout unit in the worst case with the ground-truth target (red).

boundary, indicating a trainable regime. During training, eigenvalues near the right boundary undergo controlled oscillatory motion, reflecting the network’s ability to encode periodic or time-evolving structure. This behavior persists during testing, where oscillatory modes remain stable and bounded. These observations confirm that R-FORCE places the network in a regime that is simultaneously expressive, numerically stable, and capable of long-horizon sequence generation.

Table 4.1: **Target-learning MAE ($\times 10^{-2}$) for five body-movement tasks under three dynamical regimes.** Each entry shows mean $\pm$ CI. RFORCE achieves the lowest error across all settings.

Method	Deep Squat	Arm Scaption	Leg Raise	Hurdle Step	Arm Raise
Optimal Regime ($g = 1.5$)					
Transfer-FORCE (400)	27 $\pm$ 7.6	34 $\pm$ 10	27 $\pm$ 10	29 $\pm$ 11	29 $\pm$ 8.3
Full-FORCE (400)	8.5 $\pm$ 3.2	16 $\pm$ 6.4	5.7 $\pm$ 1.6	14 $\pm$ 6.8	11 $\pm$ 5.0
FORCE (400)	6.9 $\pm$ 2.6	12 $\pm$ 5.8	4.0 $\pm$ 1.9	3.5 $\pm$ 2.4	5.0 $\pm$ 2.7
R-FORCE (400)	3.3$\pm$1.4	4.1$\pm$1.7	1.9$\pm$0.7	3.3$\pm$1.7	3.8$\pm$2.3
Transfer-FORCE (1000)	14 $\pm$ 6.4	19 $\pm$ 15	9.2 $\pm$ 7.0	14 $\pm$ 11	9.6 $\pm$ 8.6
Full-FORCE (1000)	4.4 $\pm$ 2.1	4.5 $\pm$ 2.3	3.0 $\pm$ 0.8	3.8 $\pm$ 1.0	4.0 $\pm$ 0.9
FORCE (1000)	2.9 $\pm$ 2.9	5.2 $\pm$ 3.7	2.1 $\pm$ 1.2	3.8 $\pm$ 2.7	2.1 $\pm$ 1.5
R-FORCE (1000)	1.0$\pm$0.3	1.9$\pm$0.9	0.9$\pm$0.8	1.9$\pm$4.9	1.5$\pm$1.2
Quiescent Regime ($g = 1.0$)					
Transfer-FORCE (1000)	69 $\pm$ 9.3	58 $\pm$ 9.7	32 $\pm$ 8.0	57 $\pm$ 26	48 $\pm$ 8.1
Full-FORCE (1000)	88 $\pm$ 8.7	40 $\pm$ 11	78 $\pm$ 7.4	76 $\pm$ 17	82 $\pm$ 5.5
FORCE (1000)	81 $\pm$ 4.7	73 $\pm$ 4.5	63 $\pm$ 8.2	80 $\pm$ 6.5	72 $\pm$ 4.2
R-FORCE (1000)	2.4$\pm$0.6	2.6$\pm$1.6	1.3$\pm$0.3	5.7$\pm$4.3	2.2$\pm$1.0
Chaotic Regime ($g = 1.8$)					
Transfer-FORCE (1000)	109 $\pm$ 17	99 $\pm$ 14	116 $\pm$ 16	104 $\pm$ 15	102 $\pm$ 13
Full-FORCE (1000)	86 $\pm$ 29	75 $\pm$ 14	47 $\pm$ 7.5	76 $\pm$ 17	72 $\pm$ 13
FORCE (1000)	63 $\pm$ 9.5	49 $\pm$ 11	21 $\pm$ 7.7	42 $\pm$ 9.0	42 $\pm$ 10
R-FORCE (1000)	7.9$\pm$3.9	4.5$\pm$2.2	2.9$\pm$2.1	8.7$\pm$5.8	4.4$\pm$2.0

4.3 Experiments

I evaluate R-FORCE in three settings of increasing complexity: synthetic target-learning across a range of dynamical regimes, multi-dimensional human motion data, and biologically inspired spiking neural networks. The goal is to assess stability, long-horizon accuracy, and robustness to failure modes that commonly arise in FORCE-based training. All models share the same architectures and hyperparameters, and results are averaged across multiple random seeds to improve reliability.

Target-Learning across Dynamical Regimes I first validate that R-FORCE broadens the operating regime of FORCE learning on the canonical target-learning task. Using a Rank1-RRNN with $N = 1000$ neurons, I train the network to autonomously generate three one-dimensional targets of increasing difficulty: a multi-periodic superposition of sine waves, a rapidly changing (discontinuous) signal, and a noisy target, while sweeping the scaling parameter over $g \in [1, 2]$. FORCE and Transfer-FORCE [215] control the error only near the edge-of-chaos value $g_{opt} = 1.5$ and diverge elsewhere, whereas R-FORCE keeps the error low across the entire range of g . Table 4.2 reports the multi-periodic case: averaged over $g \in [1, 2]$, R-FORCE reduces the mean absolute error by roughly an order of magnitude relative to both baselines, and it remains the most accurate method even at each baseline’s own optimal g .

I observe the same qualitative behavior on the RankN-RRNN configuration, in which the feedback loop is internal to the reservoir rather than routed through the readout, making the network closer to a standard RNN. As summarized in Table 4.3, FORCE is effective only on the narrow interval $g \in [1.3, 1.5]$ and Full-FORCE on the slightly wider $g \in [1.1, 1.5]$, whereas R-FORCE controls the error for all tested g and attains a lower error and tighter confidence interval than either baseline at their optimal g . Notably, R-FORCE on RankN-RRNN slightly outperforms its Rank1-RRNN counterpart, indicating that the additional trainable recurrent weights are put to good use once the initialization keeps the dynamics stable.

Table 4.2: **Rank1-RRNN target-learning (multi-periodic target)**. MAE and 99% CI ($\times 10^{-2}$) for FORCE, Transfer-FORCE, and R-FORCE at three values of g and averaged over $g \in [1, 2]$.

Method	$g = 1.0$	$g = 1.5$	$g = 1.8$	$g \in [1.0, 2.0]$
FORCE	82 $\pm$ 10	12 $\pm$ 11	89 $\pm$ 7.8	56 $\pm$ 11
Transfer-FORCE	86 $\pm$ 8.6	5.5 $\pm$ 9.0	92 $\pm$ 29	63 $\pm$ 15
R-FORCE	7.5$\pm$5.3	3.3$\pm$1.9	4.4$\pm$5.4	3.8$\pm$1.6

Table 4.3: **RankN-RRNN target-learning (multi-periodic target)**. MAE and 99% CI ($\times 10^{-2}$) for Full-FORCE, FORCE, and R-FORCE at three values of g and averaged over $g \in [1, 2]$.

Method	$g = 1.0$	$g = 1.5$	$g = 1.8$	$g \in [1.0, 2.0]$
Full-FORCE	55 $\pm$ 46	9.2 $\pm$ 15	77 $\pm$ 5.9	41 $\pm$ 8.8
FORCE	42 $\pm$ 5.6	5.3 $\pm$ 9.3	52 $\pm$ 6.0	32 $\pm$ 6.2
R-FORCE	9.7$\pm$12	2.1$\pm$1.6	1.6$\pm$0.9	2.7$\pm$1.5

Multi-Dimensional Targets Modeling I first test R-FORCE on the UI-PRMD physical rehabilitation dataset, which contains 66-dimensional human joint trajectories[220]. A Rank1-RRNN is trained to autonomously generate the full temporal sequence for each movement, with one readout unit corresponding to each joint coordinate (Fig. 4.2A).

Figure 4.2B reports the error distribution for the deep-squat movement across all readout dimensions. R-FORCE achieves the lowest median MAE and the tightest dispersion among all compared methods. Importantly, the worst-case error (marked by the red cross) is significantly reduced, indicating improved robustness in multi-dimensional prediction settings. Transfer-FORCE exhibits large variability and frequent extreme errors, while FORCE and Full-FORCE also display occasional failures. By contrast, R-FORCE consistently avoids

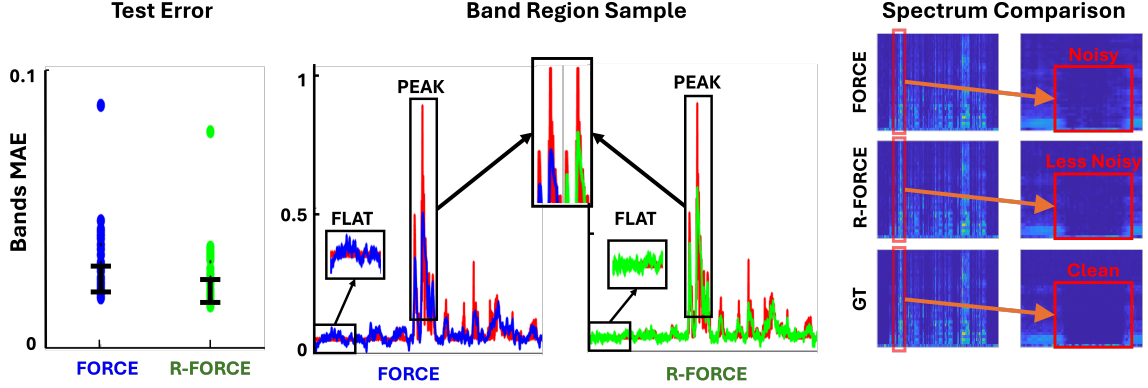


Figure 4.3: **Spiking neural network with Rank1-RRNN.** (Left) MAE of FORCE (blue) and R-FORCE(green) in generating the spectrum of birdsong. Each dot corresponds to a readout unit, and black bars are 99% confidence interval. (Middle) Comparison between FORCE (blue) and R-FORCE(green) of a randomly selected band over flat and peak regions against ground truth (red). (Right) Spectrum comparison among FORCE (top row), R-FORCE(middle row) prediction, and ground truth (bottom row). The right column is the zoom-in plot of the selected area (red rectangle) in the Left column.

these pathological cases and maintains uniform accuracy across joints.

Temporal behavior in the worst-case dimension is shown in Fig. 4.2C. Transfer-FORCE and FORCE suffer from drifting, oscillatory deviations, or late-stage divergence, whereas R-FORCE remains closely aligned with the target trajectory throughout the entire sequence.

This advantage holds across all five movements, both network sizes, and the three dynamical regimes summarized in Table 4.1: R-FORCE achieves the lowest error in every setting, improving over the baselines by roughly an order of magnitude in the non-optimal regimes ($g = 1.0, 1.8$). The benefit is also practical. The smaller Rank1-RRNN-400 trained with R-FORCE matches or exceeds the larger Rank1-RRNN-1000 trained with FORCE or Full-FORCE, which translates into a substantial speed-up: for the deep-squat movement, R-FORCE-400 trains in roughly 8 minutes, compared with about 50 and 90 minutes for FORCE-1000 and Full-FORCE-1000, respectively.

Spiking Neural Networks Modeling To evaluate generalizability beyond firing-rate networks, I incorporate R-FORCE into spiking neural networks, a more biologically realistic model of neuronal dynamics, following the FORCE-based spiking setup of [159]. I consider two standard spiking neuron models. In the Theta model [56], each neuron is described by a phase variable θ_i that evolves as

$$\dot{\theta}_i = (1 - \cos \theta_i) + \pi^2(1 + \cos \theta_i) I, \quad (4.3)$$

where I is the input current and a spike is emitted each time θ_i crosses π . The Izhikevich model [100] captures a richer range of firing patterns through a membrane potential v_i coupled to an adaptation current u_i :

$$C\dot{v}_i = k(v_i - v_r)(v_i - v_t) - u_i + I, \quad (4.4)$$

$$\dot{u}_i = a(b(v_i - v_r) - u_i), \quad (4.5)$$

where C is the membrane capacitance, v_r and v_t are the resting and threshold potentials, a is the reciprocal of the adaptation time constant, k controls the action-potential half-width, and b controls the resonance properties of the neuron.

I first apply R-FORCE to a Theta network generating a simple 1 Hz sine wave, and it achieves a lower error (0.122) than FORCE (0.193), about 30% lower. I then turn to the more demanding task of reproducing a 33-band birdsong spectrogram with an Izhikevich network, rather than the raw audio waveform. Following [41], the network idles for 5 s (1 repetition), trains for 85 s (17 repetitions), and is tested for 10 s (2 repetitions). This task involves rapid temporal variations and sharp spectral peaks, making it challenging for classical FORCE learning.

As shown in Fig. 4.3 (left), R-FORCE achieves a substantially lower mean error (0.0254 ± 0.0055) and tighter 99% confidence intervals than FORCE (0.0308 ± 0.0063) across all read-out dimensions, reflecting FORCE’s higher variance and outliers under spiking dynamics.

Figure 4.3 (middle) compares generated outputs for a randomly selected frequency band. FORCE struggles in both low-energy (flat) regions—where it introduces oscillatory noise—and high-energy (peak) regions, where it frequently produces large overshoots.

R-FORCE suppresses these artifacts and tracks the ground-truth signal with much higher fidelity across both regimes.

The complete spectrogram comparison in Fig. 4.3 (right) highlights these differences more clearly. FORCE outputs contain diffuse noise and structural irregularities across multiple bands, especially visible in the zoomed-in regions. R-FORCE yields significantly cleaner spectrograms that more accurately reproduce the fine temporal–spectral structure of the target.

Chapter 5

AELLE: LYAPUNOV-GUIDED REPRESENTATION OF RECURRENT NEURAL NETWORK PERFORMANCE

5.1 Motivation

Recurrent neural networks (RNNs) are a standard tool for modeling sequential data—from speech and text to motion capture, music, and general multivariate time series. Yet even when two networks share essentially the same hyperparameters (architecture, initialization, optimization settings), they can perform very differently once deployed on different datasets or tasks. This variability is hard to predict because RNNs are not just parameterized function approximators, but nonlinear, high-dimensional, non-autonomous dynamical systems with time-varying parameters: both global hyperparameters and trainable connectivity weights evolve during learning and interact with the input statistics. As a consequence, it is difficult to explain *why* a given configuration works well for one task but fails on another, or to systematically select good configurations a priori.

Architectural variants of RNNs have been introduced to improve stability and robustness, ranging from classical gated models such as LSTMs [95] and GRUs [39, 40] to more recent designs that impose dynamical constraints, including Anti-Symmetric RNNs (AS-RNN) [31], Orthogonal RNNs (ORNN) [229], Coupled Oscillatory RNNs (coRNN) [189], Lipschitz RNNs [55], and others. While these architectures often yield improved robustness, they represent isolated points in a vast design landscape. Even within a fixed architecture, performance can vary widely as initialization, optimization settings, architectural width, or input statistics are changed. Empirically, networks that appear nearly identical in their hyperparameters can achieve markedly different accuracies, illustrating the difficulty of predicting performance from architectural choices alone.

From a dynamical-systems perspective, such variability is expected: RNNs evolve according to nonlinear, high-dimensional, non-autonomous dynamics whose trajectories are

highly sensitive to both global hyperparameters and learned recurrent weights. A classical tool for characterizing these dynamics is the Lyapunov spectrum, which quantifies the expansion and contraction rates of perturbations along different directions of the state space [187, 162]. Recent work has adapted Lyapunov exponent (LE) computation to modern RNNs [54], and examples in [226] suggest that certain spectral features correlate with robustness and accuracy. However, simple statistics such as maximum or mean LE exhibit weak and inconsistent correlations, indicating that the informative structure of the LE spectrum is more subtle than can be captured by a few handcrafted features.

These observations motivate the development of *AeLLE*, a data-driven method that learns the dynamical features of the LE spectrum most predictive of RNN performance. AeLLE employs an autoencoder to compress each full LE spectrum into a low-dimensional latent representation that isolates critical dynamical signatures. A subsequent linear embedding of this latent space—denoted AeLLE—acts as an effective classifier: it reliably separates high- and low-accuracy networks across architectures, initialization scales, and network widths, using only the LE spectrum as input. Notably, such separation does not emerge when standard embeddings (e.g., PCA) are applied directly to the raw LE spectra, confirming that the autoencoder step is essential for extracting performance-relevant structure.

Moreover, AeLLE identifies these features early in training, long before the network has converged. This suggests that the latent organization of the Lyapunov spectrum reflects aspects of trainability and dynamical organization established at initialization and that evolve only modestly during optimization. Together, these results show that AeLLE provides a principled and general mechanism for interpreting RNN dynamics, comparing architectures, and predicting final accuracy based solely on dynamical-system characteristics.

5.2 Method

The AeLLE framework consists of three stages: (1) computing the Lyapunov Exponent (LE) spectrum of a given RNN configuration, (2) learning a compact latent representation of the spectrum using an autoencoder, and (3) embedding the latent variables into a low-dimensional space for classification.

Computation of Lyapunov Exponents To characterize the dynamics of each RNN, we compute its Lyapunov Exponent (LE) spectrum using the QR-based algorithm of Benettin et al. [13, 49], following implementations for RNNs in [54, 226]. For a fixed task, each LE spectrum is estimated over batches of validation sequences drawn from a consistent input distribution. Hidden states are initialized to zero, and an orthonormal basis of perturbation vectors is initialized as $\mathbf{Q}_0 = I$.

At each timestep, we compute the Jacobian of the hidden state transition,

$$[\mathbf{J}_t]_{ij} = \frac{\partial h_t^j}{\partial h_{t-1}^i},$$

which captures the local sensitivity of the dynamics as the network processes input x_t . Perturbation evolution is tracked using a QR decomposition,

$$\mathbf{Q}_{t+1}, \mathbf{R}_{t+1} = QR(\mathbf{J}_t \mathbf{Q}_t).$$

The diagonal entries of $\mathbf{R}_t$ provide instantaneous expansion factors. The k -th Lyapunov exponent for a sequence of length T is

$$\lambda_k = \frac{1}{T} \sum_{t=1}^T \log(r_t^k).$$

For each network configuration, spectra are averaged over n input sequences. To compare networks of different sizes, all spectra are interpolated to a fixed dimensionality, preserving spectral shape while normalizing for network size. Full Jacobian expressions for each architecture are provided in the Supplementary Materials A.2.

Autoencoder for LE Spectra To extract features of the LE spectrum that correlate with task performance, we train an autoencoder composed of an encoder ϕ , a decoder ψ , and an auxiliary linear predictor ξ (Fig. 5.1). Given an input LE spectrum Z , the model produces both a reconstruction and an accuracy prediction:

$$\hat{Z} = (\psi \circ \phi)(Z), \quad \hat{T} = (\xi \circ \phi)(Z).$$

Training minimizes a combined reconstruction–prediction objective,

$$L = \|Z - \hat{Z}\|^2 + \alpha \|T - \hat{T}\|^2,$$

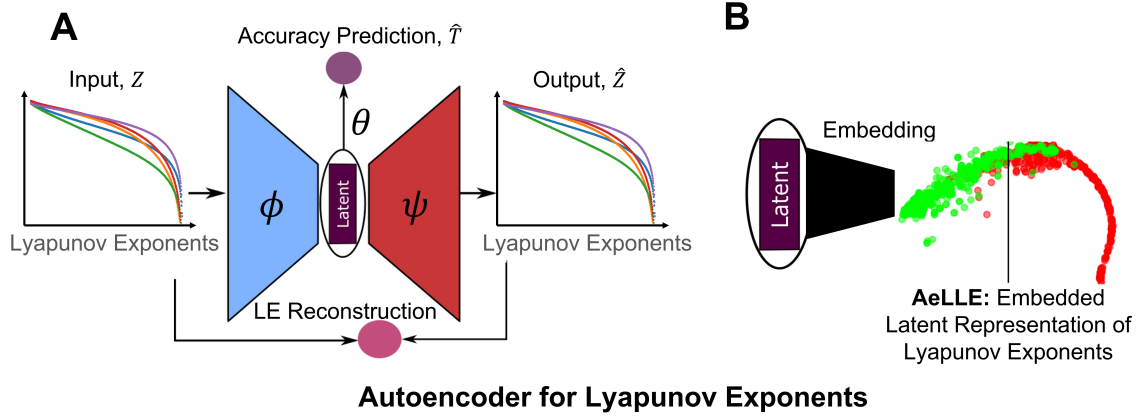


Figure 5.1: **Clustering across network architecture: AeLLE for networks trained on SMNIST Task.** *Left:* In this task, the network predicts the digit shown in a given image. Higher losses correspond to lower confidence in the correct digit label. *Center:* AeLLE representation shows that networks of similar error are clustered together. When classifying these networks by high accuracy (green) and low accuracy (red), high-accuracy networks, regardless of architecture, are consistently located farther to the right (more positive PC1) than low-accuracy networks. *Right:* Networks of the same type often clustered together in AeLLE representation, with some overlap between similar architectures (such as coRNN and ASRNN). For each network architecture (except vanilla RNN), there are variants with high and low accuracy that are separated along the first PC dimension.

where α controls the weight of prediction relative to reconstruction. α gradually increases during training so that the model first learns to reconstruct spectra faithfully and later emphasizes predictive structure. Too little emphasis on prediction yields latent variables unrelated to performance; too much emphasis destabilizes reconstruction. The staged weighting reliably produces latent representations that encode accuracy-relevant dynamical information.

Embedding of Latent Representations After training, the autoencoder’s latent variables encode the essential dynamical features of the LE spectrum that correlate with per-

formance. To organize these latent variables into a space suitable for classification across architectures and hyperparameter settings, we apply Principal Component Analysis (PCA) to the latent vectors. Projection onto the first two or three principal components yields a low-dimensional embedding in which high-accuracy and low-accuracy networks naturally form separable clusters (Fig. 5.1B). This simple linear embedding outperforms direct PCA on the raw LE spectra, indicating that the autoencoder has isolated a low-dimensional manifold of performance-relevant dynamical features. As a result, AeLLE supports straightforward classification using linear boundaries and does not require more complex nonlinear embeddings such as t-SNE or UMAP.

5.3 Experiments

We evaluate AeLLE on two sequence-modeling benchmarks that stress different sources of variation: character prediction (CharRNN) with LSTMs of varying size, which probes generality across *network size*, and Sequential MNIST (SMNIST) over a range of recurrent architectures, which probes generality across *network architecture*. Across both, we ask (i) whether the AeLLE embedding separates high- and low-accuracy models, and (ii) whether a classifier trained on AeLLE can predict final performance early in training.

Character Prediction Across Network Sizes

The character-prediction task (CharRNN) takes a sequence of characters as input and predicts the next character, which is a standard benchmark for recurrent models. We train LSTM networks on the English translation of Tolstoy’s *War and Peace*, following the setup of [107]: each of the 82 unique characters is mapped to an index, and the text is split into disjoint sequences of length $l = 101$, where the first 100 characters form the input and the final character is the target. The loss is the cross-entropy between the predicted and true character.

The two hyperparameters with the largest effect on accuracy are the network size (number of hidden units) and the weight initialization. We generate 1,200 variants, varying the hidden size over $\{64, 128, 256, 512\}$ and sampling initial weights from a symmetric uniform distribution whose half-width p ranges over $[0.04, 0.4]$, and split them 80%/20% into AeLLE

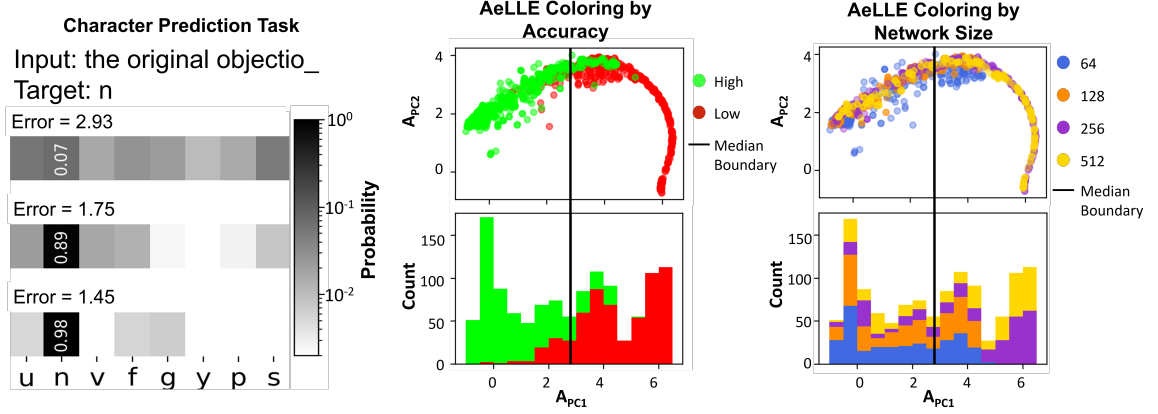


Figure 5.2: **Clustering by performance across network size: AeLLE for LSTMs trained on CharRNN.** *Left:* The CharRNN task predicts the next character in a sequence; the heatmaps show the predicted character distribution for models with different test errors, where larger errors correspond to lower confidence in the true next character. *Center:* AeLLE embedding (first two principal components) colored by accuracy. High-accuracy networks (green) cluster to the left of the median PC_1 boundary (black line) and low-accuracy networks (red) to the right; the stacked histogram below shows the same separation. *Right:* The same embedding colored by network size (64, 128, 256, 512). The sizes are broadly mixed throughout the space, with larger networks covering a wider range but still overlapping smaller ones, so network size alone does not separate performance.

training and validation sets. As before, we project each LE spectrum onto the first two principal components of the trained latent space and label networks with loss below the median (< 1.75) as high-accuracy and the remainder as low-accuracy.

AeLLE again organizes the variants by accuracy (Fig. 5.2): high- and low-accuracy networks fall on opposite sides of a median threshold along PC_1 , with the high-accuracy networks concentrated to the left of the boundary. Coloring the same embedding by network size instead shows the sizes spread broadly and overlapping throughout the space, with larger networks merely covering a wider range. AeLLE therefore recovers a performance-aligned geometry that network size alone does not provide. Quantitatively, threshold classifiers built

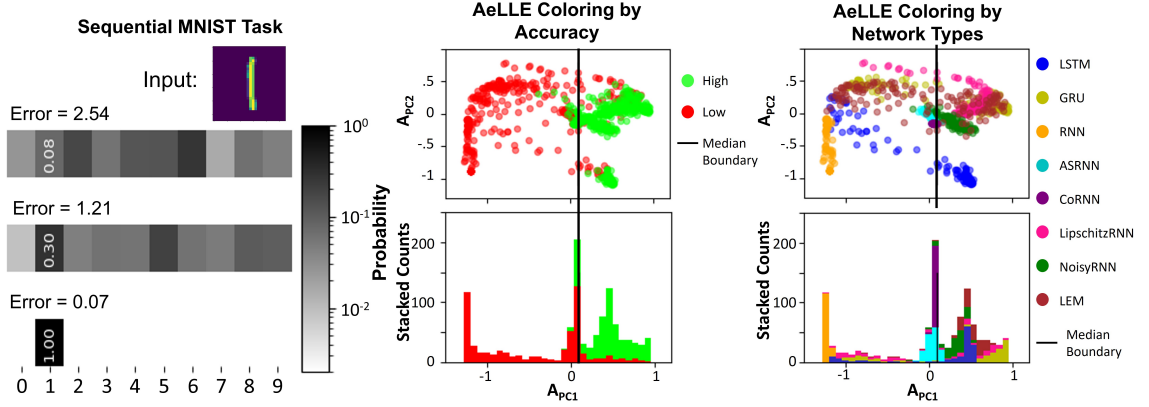


Figure 5.3: **AeLLE on Sequential MNIST.** *Left:* Example SMNIST inputs (row-wise sequences) and corresponding prediction heatmaps for models with different test errors. *Middle:* AeLLE embedding (first two principal components) colored by accuracy; high- and low-accuracy networks separate along PC_1 (black line). The bottom panel shows stacked histograms along PC_1 for the two groups. *Right:* Same embedding colored by architecture (LSTM, GRU, RNN, ASRNN, coRNN, LipschitzRNN, NoisyRNN, LEM), with stacked histograms highlighting how different model families populate the AeLLE space.

on the mean LE, the maximum LE, and the first principal component of the raw spectrum reach F1 scores of 0.834, 0.859, and 0.860, respectively, all already strong on this task; the AeLLE classifier improves on each, reaching 0.877.

Sequential MNIST Across RNN Architectures

In SMNIST [119, 9], each 28×28 image is unrolled row-wise into a sequence of grayscale pixel values. At each timestep, the network receives one row, and after the final timestep, it outputs a digit label. The loss is the cross-entropy between the predicted and true one-hot label.

We train eight recurrent architectures on this task: LSTM, GRU, vanilla RNN, Anti-Symmetric RNN (ASRNN) [31], Coupled Oscillatory RNN (coRNN) [189], Lipschitz RNN [55], NoisyRNN [127], and Long Expressive Memory (LEM) [190]. For each architecture, we sam-

ple 200 hyperparameter variants with a hidden size of 64, train for 10 epochs, and compute the LE spectrum of the final model, yielding 1,600 trained networks. We split them into 70% for AeLLE training, 10% for validation, and 20% for testing.

Figure 5.3 summarizes these results. The left panel shows example sequential predictions with different test errors. The middle panel visualizes the AeLLE embedding (the first two principal components) colored by accuracy: networks with loss below the median are labeled high-accuracy (green), and the rest low-accuracy (red). A simple threshold on the first principal component (PC_1 ; black vertical line) cleanly splits the two groups; the corresponding stacked histograms (bottom) show that high-accuracy models concentrate on the right side of the decision boundary. The right panel colors the same embedding by architecture. Architectures with stronger inductive biases for stable dynamics (e.g., GRU, LipschitzRNN, LEM, NoisyRNN) tend to occupy the high-accuracy region, whereas many vanilla RNN variants cluster in the low-accuracy half. Interestingly, both ASRNN and coRNN form compact clusters near the median boundary, reflecting their tightly controlled dynamics: when hyperparameters are favorable, they achieve competitive accuracy, but their operating region is narrow. Overall, AeLLE produces a geometry where models of very different architectures cluster by performance rather than by their raw hyperparameters.

For comparison, we also build simple classifiers based directly on LE statistics. Using only the mean LE, the maximum LE, or the first principal component of the raw LE spectrum yields F1 scores of 0.609, 0.566, and 0.608, respectively, on the test set. In contrast, the AeLLE classifier defined by a median threshold on PC_1 achieves an F1 score of 0.859. This gap shows that performance-relevant features of the dynamics are not captured by standard LE summaries alone and that the autoencoder-based representation is essential.

Early Prediction with a Pre-trained AeLLE Classifier

We next evaluate whether AeLLE can forecast a network’s final accuracy early in training. Using the same pool of SMNIST models, we compute LE spectra at several checkpoints—before training (0%), and after 10%, 20%, and 50% of the full 10-epoch training schedule. At each checkpoint, we project the spectra through a *fixed* AeLLE model that was trained solely

Table 5.1: **Unified comparison of AeLLE and baseline classifiers.** Each cell shows **F1 / Recall / Precision** for predicting final RNN accuracy at different training stages. Best F1 at each stage is shown in **bold**.

Method	0%	10%	20%	50%
Loss	– / – / –	0.35 / 0.17 / 0.95	0.66 / 0.46 / 0.98	0.86 / 0.78 / 0.97
LE-Mean	0.71 / 0.99 / 0.56	0.73 / 0.98 / 0.58	0.73 / 0.98 / 0.59	0.73 / 0.96 / 0.59
LE-Max	0.72 / 0.65 / 0.90	0.74 / 0.61 / 0.93	0.73 / 0.59 / 0.94	0.72 / 0.59 / 0.94
LE-PCA	0.76 / 1.00 / 0.56	0.74 / 0.97 / 0.60	0.74 / 0.97 / 0.60	0.74 / 0.95 / 0.60
AeLLE	0.85 / 0.94 / 0.77	0.89 / 0.92 / 0.87	0.90 / 0.92 / 0.87	0.86 / 0.89 / 0.84

on fully trained networks, and we classify variants using the same median PC_1 threshold determined on a validation split.

Table 5.1 summarizes the performance of AeLLE at each checkpoint and compares it against several baselines: loss-based classification, LE mean, LE max, and PCA applied directly to the raw LE spectrum. Each entry reports the corresponding F1, recall, and precision values.

Already at initialization (0% training), the pre-trained AeLLE classifier achieves an F1 score of 0.85, substantially outperforming all baselines. After only one epoch (10% of training), AeLLE reaches an F1 score of 0.89 and maintains a high level of accuracy (F1 between 0.86 and 0.90) through 50% training. In contrast, the loss-based classifier has excellent precision but extremely poor recall in the early stages, incorrectly discarding many networks that eventually achieve high accuracy. Although loss becomes a competitive predictor by 50% training, this requires half of the total training budget, whereas AeLLE provides reliable predictions from the outset.

Across all checkpoints, AeLLE consistently achieves the highest F1 score among all LE-based classifiers. Neither simple statistics of the spectrum (LE mean, LE max) nor PCA on the raw spectrum produces comparable predictive performance. These results indicate that AeLLE extracts stability-related dynamical features that emerge early and remain robust

throughout training, enabling accurate early prediction without requiring partially trained performance signals.

Contribution of the AeLLE Representation

Finally, we investigate whether the performance gains of AeLLE arise from the latent representation itself or whether they could be matched by applying more flexible classifiers directly to Lyapunov spectra. To evaluate this, we compare several combinations of representations and classifiers: (i) linear regression applied to the raw LE spectrum, (ii) classifiers operating on PCA-reduced LE spectra, and (iii) the same classifiers applied to the AeLLE latent representation.

Table 5.2 reports F1 scores for each method when trained on checkpoints from 0–50% of training and evaluated on a pooled test set. Linear models trained on raw LEs or their PCA projections achieve F1 scores in the range 0.62–0.76. In contrast, classifiers operating on the AeLLE latent space consistently reach F1 scores between 0.85 and 0.90 across all training stages, regardless of whether classification is performed using a simple PC_1 threshold or a linear regression model. On average, this represents an improvement of roughly 0.12 over the best LE- or PCA-based baselines. This gap indicates that the autoencoder is extracting dynamical structure from the LE spectrum that is not recoverable through direct linear processing of the raw data.

To further probe the structure of the AeLLE latent space, we assess how many principal components of the latent representation are needed for effective classification. Table 5.3 summarizes F1 scores obtained using the median threshold over 1, 2, 4, and 10 principal components. Even using only the first principal component yields strong performance (F1 0.81 at 0% training and 0.84 at 50%). Incorporating a handful of components (4–10 PCs) improves F1 to 0.89–0.91, suggesting that the dynamical features relevant for distinguishing high- and low-performing networks lie on a compact, low-dimensional manifold. Together, these results reinforce the conclusion that AeLLE provides both meaningful compression of the LE spectrum and an intrinsically aligned representation of network performance.

Table 5.2: **Comparison of F1 scores across classifiers and representations.** Rows correspond to different representations of the Lyapunov Exponents (raw, PCA, or AeLLE) combined with either a PC1 threshold classifier or a linear regression classifier. Columns report F1 scores at different training stages. The best score for each training duration is shown in **bold**.

Representation & Classifier	0%	10%	20%	30%	40%	50%	Test Set
LE (Raw) + Linear Regression	0.62	0.61	0.61	0.62	0.64	0.61	0.62
LE (PCA) + PC1	0.72	0.74	0.74	0.75	0.75	0.74	0.74
LE (PCA) + Linear Regression	0.73	0.77	0.78	0.78	0.77	0.75	0.76
AeLLE (PCA) + PC1	0.85	0.89	0.89	0.90	0.89	0.86	0.88
AeLLE (PCA) + Linear Regression	0.86	0.89	0.89	0.90	0.89	0.87	0.88

Table 5.3: **Effect of PCA dimensionality on AeLLE classification.** F1 scores of the pre-trained AeLLE classifier using the median threshold over different numbers of principal components (PCs). The full latent dimension is 32.

Training %	PC1	PC2	PC4	PC10
0%	0.81	0.81	0.84	0.89
50%	0.84	0.84	0.87	0.91

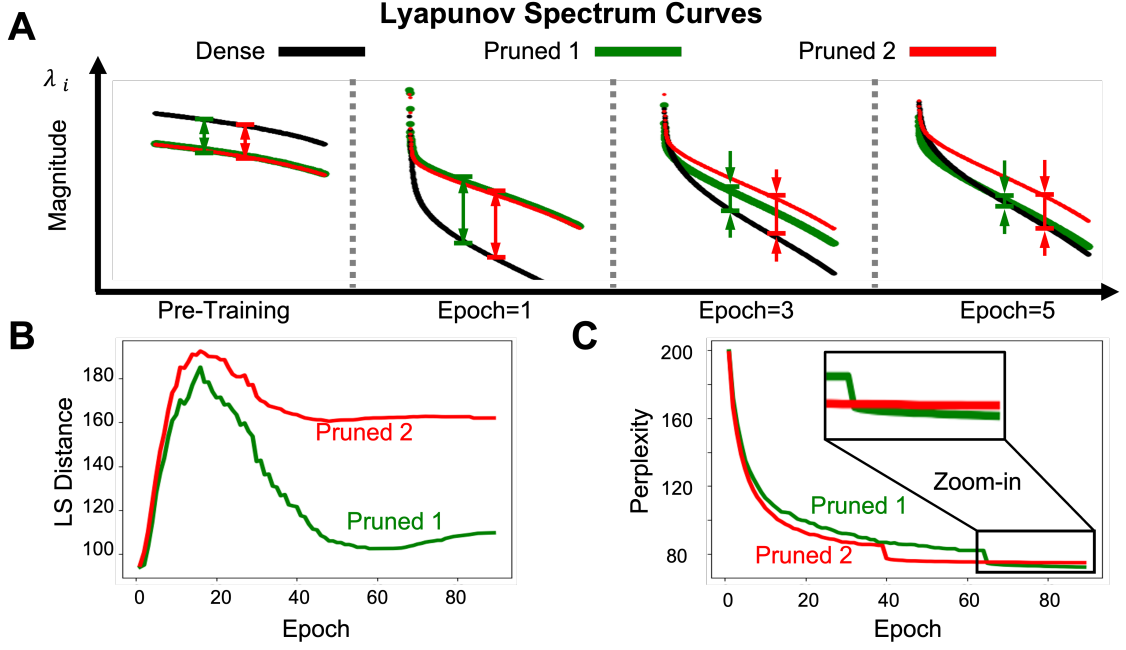


Figure 6.1: (A) Lyapunov Spectrum (LS) curves of the dense network (black) and two pruned variants—Pruned 1 (green) and Pruned 2 (red)—evaluated at pre-training and at epochs 1, 3, and 5 (left to right). The dense network serves as a reference trajectory in LS space. Arrows indicate the deviation magnitude from the dense model. (B) Temporal evolution of the L_2 distance between each pruned variant and the dense network in LS space throughout training. (C) Perplexity curves of the two pruned variants over training epochs.

Chapter 6

HYPERPRUNING: EFFICIENT SEARCH THROUGH PRUNED VARIANTS OF RECURRENT NEURAL NETWORKS LEVERAGING

LYAPUNOV SPECTRUM

6.1 *Motivation*

Recurrent neural networks (RNNs) remain essential tools for modeling sequential data in domains such as natural language processing, speech, and video understanding. Despite their strong performance, modern RNN architectures—such as LSTMs, RHNs, and other gated or deep recurrent models—are expensive to train and deploy. Their computational cost scales linearly with sequence length and quadratically with hidden size, making them difficult to use in large-scale training or on resource-limited platforms.

Model compression has therefore become an indispensable component of RNN deployment. Among existing techniques, network pruning is particularly attractive because it can simultaneously reduce parameter count, memory footprint, and inference latency. However, conventional dense-to-sparse pruning provides limited benefit during training: models are trained densely for most of their optimization trajectory, keeping training cost comparable to that of the dense model. More recent Dynamic Sparse Training (DST) approaches attempt to reduce this cost by training with a fixed sparsity budget from the start, but they depend heavily on pruning controls (e.g., growth rules, removal criteria, redistribution schedules). These controls behave differently across architectures, datasets, and sparsity levels, meaning that no universal pruning strategy exists. As a result, choosing an effective pruning configuration becomes a hyperparameter optimization problem—one in which both the pruning methodology and its internal controls must be selected. We refer to this challenge as hyperpruning.

Hyperpruning is uniquely difficult because each candidate configuration requires hours or days of training to fully evaluate, and early training signals (e.g., initial loss or perplexity) are not reliable indicators of eventual accuracy. As shown in Fig. 6.1, two pruning configurations may exhibit nearly identical early loss curves yet diverge significantly after full training. This makes classical search strategies inefficient, as they lack a principled notion of distance between pruning configurations or between a pruned model and its dense reference.

Recent work suggests that sequence models, including RNNs, can be fruitfully analyzed as dynamical systems, whose behavior is characterized by the Lyapunov Spectrum (LS)—a measure of how hidden-state trajectories expand or contract over time. Prior results show that LS correlates with trainability and final accuracy in RNNs, and that autoencoder-based embeddings of LS can reveal low-dimensional structure aligned with model quality. Surprisingly, this structure persists even under pruning: in Fig. 6.1, the LS of a pruned model remains consistently closer to that of the dense model when its final performance is better, even though its early perplexity is higher for more than 60 epochs. In other words, the Lyapunov geometry of a pruned model provides a more stable and reliable indicator of its eventual performance than traditional loss-based metrics.

These observations motivate the development of a pruning-oriented hyperparameter search framework that leverages LS-based geometric information. If LS can reveal, early in training, how closely a pruned model follows the dynamical behavior of its dense counterpart, then LS could serve as a powerful predictive distance metric—capable of guiding hyperpruning decisions at a fraction of the cost of full training.

Based on this insight, our goal is to design a principled hyperpruning algorithm that leverages the LS geometry to evaluate and select pruning configurations efficiently. By anchoring pruning decisions to dynamical similarity rather than noisy early loss, we aim to dramatically reduce the computational load of iterative pruning searches while retaining or surpassing the performance of dense baselines.

6.2 Method

This section describes the computation of the Lyapunov Spectrum (LS) and the proposed LS-based Hyperpruning (LSH) algorithm. As illustrated in Fig. 6.2, LSH begins with an initial pool of pruned network configurations and iteratively removes non-promising candidates based on their LS distance to a dense reference model. New candidates are then introduced through adaptive sampling. After several cycles of removal and generation, the remaining configurations are fully trained, and the best-performing pruned model is selected. The LS-based distance metric can also be incorporated into standard hyperparameter optimization algorithms.

Lyapunov Spectrum Computation

Lyapunov Exponents (LEs) quantify the contraction or expansion rate of nearby trajectories in a dynamical system. For two infinitesimally close initial states separated by ϵ , their distance after T steps evolves approximately as $\epsilon e^{\lambda T}$, where λ is the corresponding LE. An RNN has one LE per hidden-state dimension. Although RNNs are non-autonomous due to their dependence on input sequences, LS can still be defined by appealing to random dynamical systems theory when inputs are drawn from a stationary distribution.

Following the procedure in [228], LS is computed by tracking the evolution of hidden states together with the deformation of an orthogonal basis. Let $\mathcal{X}$ be a batch of K validation sequences of length T ,

$$\mathcal{X} = \left\{ \mathbf{x}^k = (x^{1,k}, \dots, x^{T,k}) \in \mathbb{R}^d \right\}_{k=1}^K.$$

We initialize the hidden state and orthogonal basis as

$$h^0 = 0, \quad \mathcal{B}^0 = I.$$

For each timestep t , the hidden state evolves according to

$$h^{t,k} = f(h^{t-1,k}, x^{t,k}),$$

where f denotes the RNN update. The orthogonal basis $\mathcal{B}^{t,k}$ is evolved via

$$\mathcal{B}^{t,k}, R^{t,k} = qr\left(J^{t,k} \mathcal{B}^{t-1,k}\right),$$

where $J^{t,k}$ is the Jacobian between hidden states at consecutive time steps, and QR decomposition yields the instantaneous stretching factors. Full Jacobian expressions for all architectures used in this work appear in the Supplementary Material.

Let $r_i^{t,k}$ be the i -th diagonal entry of $R^{t,k}$. The i -th Lyapunov exponent is computed by averaging over samples and time,

$$\lambda_i = \frac{1}{KT} \sum_{k=1}^K \sum_{t=1}^T \log r_i^{t,k}. \quad (6.1)$$

The LS is denoted by $\Lambda = \{\lambda_i\}_{i=1}^N$, where N is the hidden-dimensionality of the network.

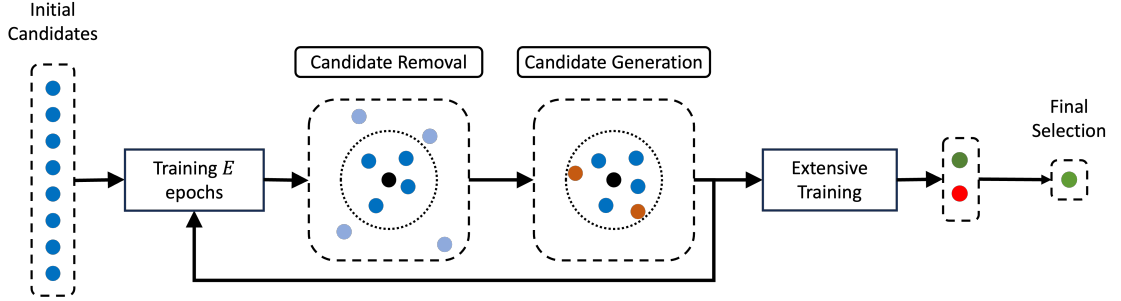


Figure 6.2: LS-based Hyperpruning: starts with a set of candidates (blue) and a pre-trained dense reference model (black), LSH executes candidate removal and generation. **Candidate Removal:** Each candidate is trained for E epochs and projected to the LS space, along with the reference. Candidates with further distance (light blue) are removed based on their distance to the reference. **Candidate Generation:** New candidates (brown) are adaptively generated based on remaining candidates and inserted into the candidate set. After several iterations of removal and generation, the remaining candidates in the set are extensively trained, and the best one (green) is picked as the final selection.

LS-Based Hyperpruning (LSH)

The goal of LSH is to efficiently identify a high-performing pruned RNN configuration. Let $\mathcal{P}$ denote a pool of candidate pruned configurations, where each candidate $c^i \in \mathcal{P}$ specifies a pruning method together with its associated hyperparameters. A fully trained dense network $\hat{c}$ serves as a reference model, and its Lyapunov Spectrum $\hat{\Lambda}$ provides the target against which pruned variants are evaluated. The LSH procedure alternates between two key operations—LS-based removal and adaptive generation—followed by a final selection stage.

In the LS-based removal step, each candidate c^i is trained for E epochs. After epoch j , we compute its Lyapunov Spectrum Λ_j^i using Eq. 6.1. The sequence of LS values accumulated during this brief training phase is denoted

$$\Lambda^i = \{\Lambda_0^i, \Lambda_1^i, \dots, \Lambda_E^i\}.$$

To compare candidates with the dense reference, both Λ^i and $\hat{\Lambda}$ are projected into a low-dimensional embedding space. Following prior work, we use Principal Component Analysis (PCA) to construct a 2D embedding, termed *LS Space*, in which the Euclidean distance between the most recent spectrum Λ_{-1}^i and $\hat{\Lambda}$ quantifies the similarity between a pruned candidate and the dense model. Candidates with the largest distances are discarded; specifically, among n candidates, the farthest $n/2$ are removed. Although this step efficiently filters out weak configurations, it does not guarantee that the optimal-pruned model resides within the initial pool.

To compensate for this possibility, LSH includes an adaptive generation mechanism that expands the search space during the pruning process. After removal, new candidates are generated based on the surviving configurations using Bayesian optimization techniques such as TPE or ATPE. These methods are adapted to produce $n/4$ new candidates whose hyperparameters are sampled according to the structure inferred from successful configurations. Because the algorithm removes $n/2$ candidates but adds only $n/4$, the pool size gradually decreases, automatically concentrating computation on more promising regions of the search space.

After m cycles of LS-based removal and adaptive generation, the remaining candidates in $\mathcal{P}$ undergo full training until convergence. The configuration that achieves the best validation performance is chosen as the final pruned network. This combination of early LS-based screening, targeted candidate generation, and final full training allows LSH to explore a broad hyperparameter space while drastically reducing the overall computational cost of hyperpruning.

6.3 Experiments

In this section, we validate the performance and the generality of the proposed LS-based Hyperpruning (LSH) method across various RNN architectures and datasets. We evaluate stacked LSTM [244] and RHN [257] models on the Penn TreeBank Dataset [143], and AWD-LSTM-MoS [240] on the WikiText-2 Dataset [145] for language modeling tasks. To ensure comparability, we adopt the same experimental setup as in prior work [132] for consistency. We also examine the robustness of LSH by conducting experiments on stacked

Table 6.1: The testing perplexity for language modeling task on Penn TreeBank Dataset with stacked LSTM and RHN, and on WikiText-2 Dataset with AWD-LSTM-MoS. DSR, RigL, SNFS, SET, ISS, GMP, Dense, and Selfish-RNN data are from [132].

Datasets	PTB				Wiki-2	
Models	Stacked-LSTM		RHN		AWD-LSTM-MoS	
	# Param	PPL	# Param	PPL	# Param	PPL
Dense	66.0M	78.6	23.5M	65.4	-	-
Dense (NT-ASGD)	66.0M	72.4	23.5M	61.8	35.0M	63.3
	Sparsity = 0.67		Sparsity = 0.53		Sparsity = 0.55	
DSR (Adam)		88.2		63.2		67.0
RigL (Adam)		85.6		64.4		68.5
SNFS (Adam)		86.3		71.1		76.2
SET (Adam)		85.5		61.1		69.6
ISS	21.8 M	78.7	11.1 M	65.4	15.6 M	-
RigL (SNT)		75.9		62.5		65.2
GMP (SNT)		74.8		64.0		-
Selfish-RNN (SNT)		71.7		60.4		63.0
LSH (Ours)		69.9		59.0		62.2

LSTM under different pruning ratios. Furthermore, we measure the time efficiency of LSH in comparison to traditional loss-based full training search, reporting the time required to achieve a given target accuracy. Finally, we evaluate LSH under five different HPO algorithms to demonstrate its effectiveness as an early-estimation criterion compared to loss-based distance.

Comparison with SOTA Pruning methods

We evaluate the effectiveness of LSH by comparing its final perplexity with that of dense models and a broad set of state-of-the-art pruning methods. Experiments are conducted on three standard benchmarks: stacked LSTM and RHN on the Penn Treebank (PTB) dataset, and AWD-LSTM-MoS on the WikiText-2 dataset, using pruning ratios of 0.67, 0.53, and 0.55, respectively. LSH performs hyperpruning over both methodological and non-methodological hyperparameters. Methodological hyperparameters include sparse initialization, death mode, and redistribution mode. These control (1) the initial sparsity pattern, (2) how weights are removed, and (3) how surviving weights are reallocated during growth. We consider 2 initialization modes, 4 death modes, and 3 redistribution modes, yielding 24 methodological variants. For non-methodological hyperparameters, we search over the death rate—the proportion of active weights removed at each pruning step—selected from a continuous range $[0.4, 0.9]$. Across all architectures, LSH consistently discovers non-trivial and architecture-specific hyperparameter configurations. For example, LSH selects a death rate of 0.58 for stacked LSTM, markedly different from the empirical value of 0.8 used by Selfish-RNN [132]. The initialization choices also differ across models: the stacked LSTM favors uniform initialization, whereas AWD-LSTM-MoS achieves the best performance with ER initialization. These results highlight the importance of principled hyperparameter selection rather than relying on heuristics.

Table 6.1 summarizes performance across all baselines. LSH achieves the lowest perplexity across all benchmarks, surpassing both dense and sparse state-of-the-art models. Relative to the previous best model (Selfish-RNN), LSH reduces perplexity from 71.7 to 69.9 on stacked LSTM, from 60.4 to 59.0 on RHN, and from 63.0 to 62.2 on AWD-LSTM-MoS. Importantly, these improvements arise solely from improved hyperparameter search—no architectural modifications or additional training tricks are used—demonstrating the value of LS-guided hyperpruning as an effective selection mechanism.

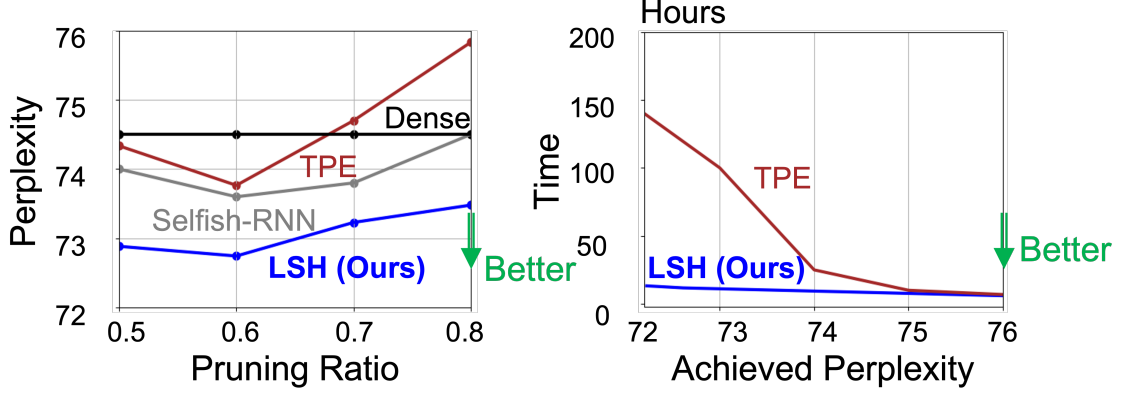


Figure 6.3: (Left) Pruning Ratio vs. Perplexity: Comparison among LSH (blue), TPE/loss-based (brown), and selfish-RNN (gray) on different pruning ratios. (Right) Achieved Perplexity vs Time Budget: Comparison between LSH (blue) and loss-based full training search (brown) on required time budget (hours) to reach a target perplexity.

Robustness Across Pruning Ratios

To evaluate robustness, we examine the performance of LSH under varying pruning ratios, ranging from moderate (0.5) to high (0.8) in increments of 0.1. Each experiment uses the stacked LSTM model with a fixed removal/insertion schedule ($E = 3$) and a candidate set size of 20. TPE is used as the underlying optimization algorithm.

As illustrated in Fig. 6.3-Left, LSH (in blue) consistently outperformed TPE/loss-based (in brown) and Selfish-RNN (in gray) methods across all pruning ratios. LSH also consistently achieved higher accuracy than the dense networks across all pruning ratios, showcasing its robustness.

Time Efficiency and Search Speed

To estimate the computational efficiency of LSH, we compared the time required for LSH to reach a specific target perplexity with that of the baseline loss-based full training search method, in which candidates are fully trained sequentially until one candidate reaches the target perplexity. We used stacked LSTM with a fixed pruning ratio of 0.67 for these

Table 6.2: **LSH vs. loss-based distance under different HPO algorithms.** Lower perplexity is better. Stacked LSTM on PTB with sparsity 0.67. LSH variants (bottom block) consistently outperform their loss-based counterparts.

Method	Low Budget	Moderate Budget	High Budget
Hyperband [123]	72.2 ± 0.4	72.4 ± 0.2	72.4 ± 0.4
Hyperband + LSH	71.6 ± 1.6	70.3 ± 2.0	70.7 ± 0.8
BOHB [58]	72.4 ± 0.5	72.5 ± 0.2	72.7 ± 0.4
BOHB + LSH	71.3 ± 0.9	71.5 ± 1.2	70.1 ± 0.5
TPE [17]	72.3 ± 0.8	72.4 ± 1.0	72.7 ± 1.1
TPE + LSH	70.9 ± 3.0	70.9 ± 2.6	69.9 ± 2.6
ATPE [19]	72.0 ± 1.9	72.7 ± 0.1	72.7 ± 0.1
ATPE + LSH	70.9 ± 0.4	69.9 ± 0.3	69.9 ± 0.3
GS	72.5	72.5	72.5
GS + LSH	70.8	70.8	70.8

experiments. Fig. 6.3-Right shows the time (y-axis) required to find a configuration with a particular target perplexity (x-axis).

Our observations indicated that LSH outperforms the loss-based approach in terms of computational efficiency across all perplexities. Specifically, for finding the optimal configuration (validation perplexity of 72), LSH took approximately 15 hours, whereas the loss-based approach required approximately 150 hours (> 6 days). This $10\times$ improvement in efficiency demonstrates LSH’s advantage in rapidly identifying high-quality configurations.

Performance Comparison: LS-Based vs. Loss-Based Hyperparameter Selection

To evaluate whether Lyapunov Spectrum (LS) distance provides a more informative early signal than loss, we compare LSH against standard loss-based selection across several hyperparameter optimization (HPO) algorithms: Grid Search (GS), TPE [17], ATPE [19], Hyperband [123], and BOHB [58]. All experiments use the stacked LSTM on PTB with a pruning ratio of 0.67, and results are reported under three resource budgets corresponding to different initial candidate pool sizes (Low, Moderate, High). Each configuration is evaluated three times, and we report mean perplexity with 95% confidence intervals.

Table 6.2 groups the results into two blocks: (1) conventional loss-based distance (top), and (2) LS-based distance used in LSH (bottom). Across all budgets and all HPO algorithms, the LS-based versions consistently achieve lower perplexity than their loss-based counterparts. This confirms that LS distance captures more predictive information about a candidate’s eventual quality early in training.

Notably, the advantage of LSH becomes more pronounced as the available budget increases. Under high-budget settings, LSH-ATPE and LSH-TPE achieve the best overall perplexities (both 69.9), outperforming all loss-based methods, which remain at or above 72.4. Even simple methods such as LSH-GS benefit from LS distance, reducing perplexity from 72.5 (loss-based GS) to 70.8. These trends suggest that LS-based distance enables HPO algorithms to more effectively identify promising regions of the hyperparameter space, improving both selection quality and stability.

Chapter 7

2NDMATCH: FINETUNING PRUNED DIFFUSION MODELS VIA SECOND-ORDER JACOBIAN MATCHING

7.1 Motivation

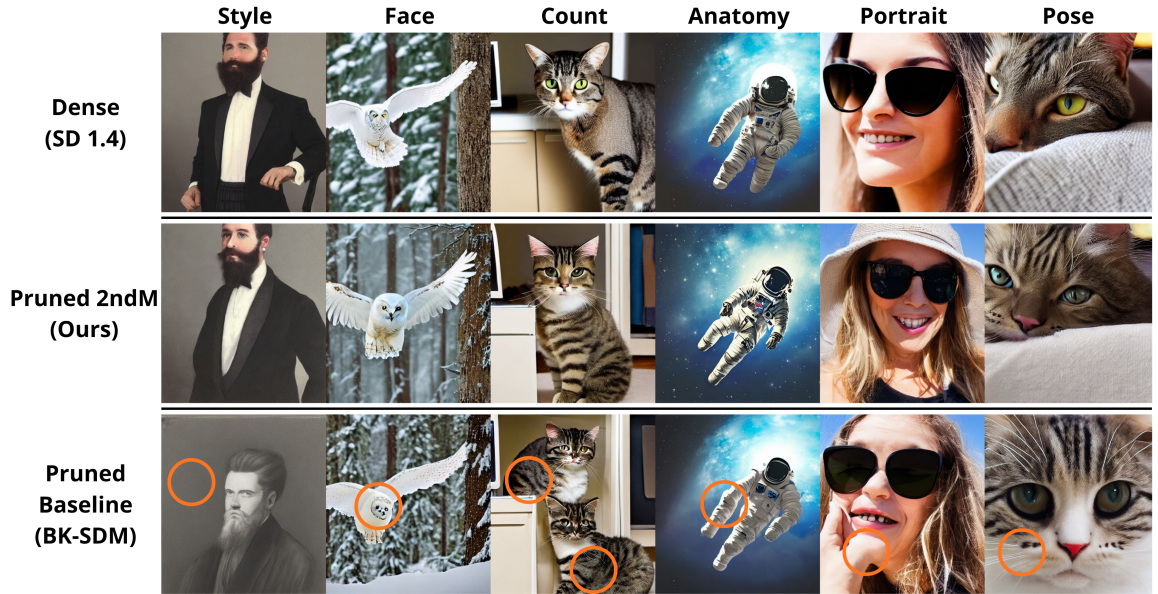


Figure 7.1: Examples of improvement in preserving fidelity and consistency of output images compared to baseline (BK-SDM) on pruning Stable Diffusion (SD) 1.4, Dense teacher, to 33%.

Diffusion models have become the leading approach for high-quality generative modeling across images, videos, and audio [92, 191, 166]. Their main drawback is computational cost: sampling requires hundreds of iterative denoising steps, each of which invokes a large U-Net or Transformer backbone [183, 169]. This makes real-time inference difficult, especially for deployment on resource-limited systems.

Pruning offers a promising path toward computationally efficient diffusion models, and recent work has adapted pruning strategies to diffusion architectures through structured sparsity, importance scoring, and trajectory-aware pruning rules [61, 29, 248, 59, 255]. However, even after pruning and finetuning, pruned models consistently lag behind dense models in fidelity [255]. A central reason is that the finetuning objective is inherited from dense training: denoising score matching (DSM), which supervises only the noise prediction at each timestep.

While DSM is effective for dense models, it is insufficient for pruned models. Due to reduced capacity, pruned models exhibit heightened sensitivity to small perturbations, and this sensitivity accumulates along the denoising trajectory. As illustrated in Fig. 7.2, even if a pruned model matches the dense teacher on the training samples at each timestep, small discrepancies compound through time, causing gradual divergence from the teacher’s trajectory and ultimately degrading sample quality.

Although the dense teacher remains available during finetuning, existing distillation approaches—whether output-level or feature-level [111]—do not capture how the teacher responds to perturbations. First-order Jacobian matching [42, 211] addresses local sensitivity but is much less meaningful for diffusion models: because inputs are already noise-perturbed, DSM implicitly provides first-order supervision. Moreover, first-order Jacobians only characterize instantaneous behavior at a single timestep and do not reflect how perturbations propagate across the full denoising sequence.

This motivates a shift in perspective: rather than supervising only outputs or one-step gradients, the finetuning objective should guide the pruned model to preserve the *temporal evolution of perturbations* observed in the dense teacher. The theory of Finite-Time Lyapunov Exponents (FTLE) provides a principled way to quantify how small perturbations expand or contract over finite time horizons, and it is directly related to trajectory stability in discrete-time dynamical systems. However, computing FTLE exactly requires tracking high-dimensional Jacobians across many timesteps and is computationally infeasible for modern diffusion models.

Motivated by these insights, this project aims to develop a scalable finetuning objective that approximates FTLE-style temporal sensitivity through efficient second-order projec-

tions. The overarching goal is to reduce trajectory drift, stabilize the denoising process, and close the quality gap between pruned and dense diffusion models—enabling more deployable and reliable sparse diffusion backbones.

7.2 Method

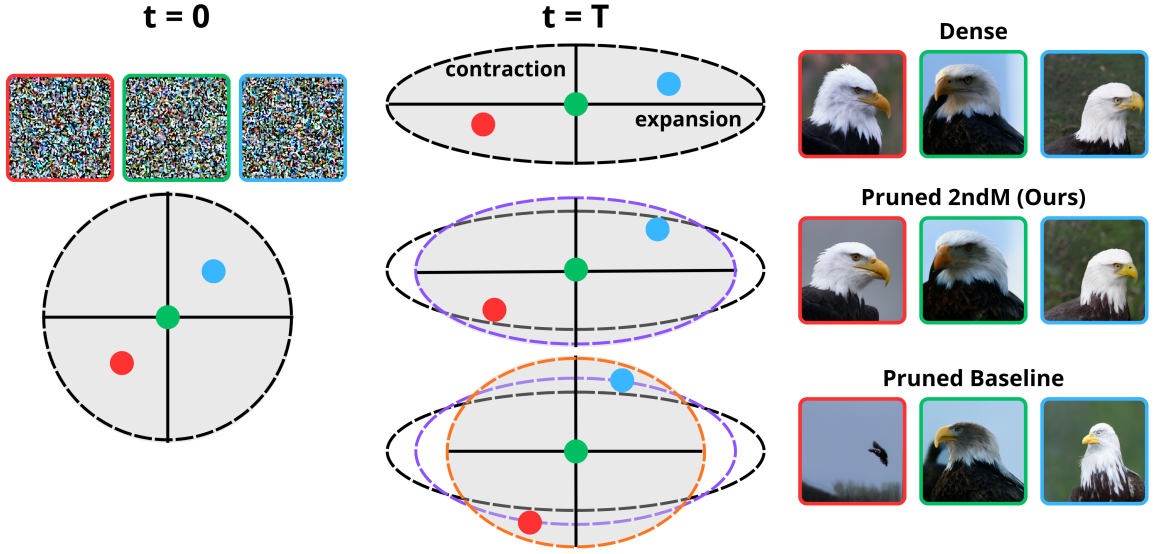


Figure 7.2: Sketch illustration of the difference between pruned Baseline (orange) and our **2ndM** (purple). Both Baseline and 2ndM match the dense teacher (black) on the training samples (green). While 2ndM constrains the expansion/contraction behavior of nearby points (red and blue), the Baseline does not have such constraints. As a result, the baseline accumulates drift along the denoising trajectory, leading to deviations at $t = T$. In contrast, 2ndM preserves stable, teacher-aligned trajectories, resulting in higher-fidelity generation.

Given a pruned diffusion model s and its dense teacher $s_{\mathcal{D}}$, the objective of this project is to design a finetuning framework that reduces the quality gap introduced by pruning. Let x denote a clean data sample, $\tilde{x}$ a noised version at diffusion timestep t , and ϵ the Gaussian noise added during the forward diffusion process. Our goal is to learn parameters θ of the pruned model $s(\cdot; \theta)$ such that its behavior matches that of the dense teacher $s_{\mathcal{D}}(\cdot; \theta_{\mathcal{D}})$ both locally (per timestep) and temporally (across the full denoising trajectory).

To accomplish this, we introduce a hybrid objective combining three complementary losses: (i) noise prediction, (ii) output-level knowledge distillation, and (iii) a second-order Jacobian matching objective inspired by finite-time sensitivity analysis. The combined objective is:

$$\mathcal{L}_{\text{total}} = \lambda_{\text{NP}}\mathcal{L}_{\text{NP}} + \lambda_{\text{KD}}\mathcal{L}_{\text{KD}} + \lambda_{\text{Jac}}\mathcal{L}_{\text{2nd-Jac}}, \quad (7.1)$$

where $\lambda_{\text{NP}}, \lambda_{\text{KD}}, \lambda_{\text{Jac}}$ are scalar weighting coefficients.

Noise Prediction (NP)

In DDPMs [92], training is formulated as predicting the added Gaussian noise ϵ from the noised input $\tilde{x}$ at timestep t sampled from a variance schedule. This yields the standard noise prediction objective:

$$\mathcal{L}_{\text{NP}} = \mathbb{E}_{\tilde{x}, t} \|s(\tilde{x}, t) - \epsilon\|_2^2.$$

This loss remains essential but is often insufficient for pruned models, whose reduced capacity can cause unstable or biased predictions early in finetuning.

Knowledge Distillation (KD)

To stabilize the early stages of finetuning, we incorporate output-level knowledge distillation from the dense teacher model $s_{\mathcal{D}}$:

$$\mathcal{L}_{\text{KD}} = \mathbb{E}_{\tilde{x}, t} \|s(\tilde{x}, t) - s_{\mathcal{D}}(\tilde{x}, t)\|_2^2.$$

While NP and KD offer strong supervision at each individual timestep, they treat timesteps independently. Thus, they neglect how perturbations propagate over the multi-step denoising trajectory—an issue particularly severe for pruned models (see Fig. 7.2).

Why First-Order Jacobian Matching is Insufficient

A natural extension is to align the Jacobians

$$J = \nabla_x s(x, t), \quad J_{\mathcal{D}} = \nabla_x s_{\mathcal{D}}(x, t),$$

which describe the local sensitivity of each model to input perturbations. However, in diffusion models, the inputs $\tilde{x}$ are already perturbed by Gaussian noise, and output matching implicitly enforces first-order Jacobian similarity.

Let $\zeta \sim \mathcal{N}(0, \sigma^2 I)$ be a small perturbation. Then a Taylor expansion gives:

$$(s(x + \zeta) - s_{\mathcal{D}}(x + \zeta))^2 = (s(x) - s_{\mathcal{D}}(x))^2 + \sigma^2 \|J - J_{\mathcal{D}}\|_2^2 + \mathcal{O}(\sigma^4),$$

showing that first-order Jacobian alignment provides limited benefit in practice while increasing computation. Furthermore, first-order Jacobians capture only *instantaneous* sensitivity, rather than the long-term accumulation of perturbations across timesteps.

Second-Order Jacobian Matching (2ndM)

To capture *temporal* perturbation growth, we draw inspiration from Finite-Time Lyapunov Exponents (FTLE), which measure how small perturbations expand or contract over finite horizons in dynamical systems. Let $J^\top J$ denote the local expansion metric for model s , characterizing how perturbations evolve at a single timestep. Directly matching $J^\top J$ is infeasible due to the high dimensionality of x (e.g., $> 10^5$).

We therefore adopt a scalable approximation by projecting J along random directions:

$$v \sim \mathcal{N}(0, I), \quad \hat{v} = v / \|v\|.$$

The projected second-order sensitivity is:

$$\|J\hat{v}\|_2^2 = \hat{v}^\top J^\top J \hat{v},$$

providing a tractable surrogate for comparing $J^\top J$ between pruned and dense models. This motivates the second-order Jacobian matching loss:

$$\mathcal{L}_{\text{2nd-Jac}} = \mathbb{E}_{x,t,v} [\|J\hat{v}\|_2^2 - \|J_{\mathcal{D}}\hat{v}\|_2^2].$$

Importantly, the term $J\hat{v}$ can be computed efficiently via vector–Jacobian products (VJPs), which modern autodiff frameworks (e.g., PyTorch) provide at little overhead. Thus, 2ndM is scalable to large U-Nets and Transformers.

Table 7.1: **Ablation of loss components on CIFAR-10.** Columns show the cumulative addition of components up to that column, except that *2nd JM is added on top of NP+KD only and does not include 1st JM.*

Loss	NP	NP + KD	NP + KD + 1st JM	NP + KD + 2nd JM (Ours)	Dense
FID ↓	5.29	5.05	5.14	4.58	4.19
FTLE	0.413	0.418	—	0.408	0.405

Empirical Observation

Table 7.1 demonstrates the contribution of each component. Although NP+KD improves FID, it leaves the FTLE gap nearly unchanged, confirming that timestep-wise supervision alone does not align temporal sensitivity. Adding first-order Jacobian matching offers negligible benefit, consistent with its theoretical redundancy. In contrast, second-order Jacobian matching significantly improves both generative quality ($5.05 \rightarrow \mathbf{4.58}$) and temporal stability ($0.418 \rightarrow \mathbf{0.408}$), indicating that 2ndM better captures how perturbations propagate along the denoising trajectory.

7.3 Experiments

We evaluate **2ndMatch** on both U-Net- and Transformer-based diffusion models. For U-Net architectures, we use pretrained DDPM [92], LDM-4, and Stable Diffusion v1.4 (SDM) [183]; for Transformer architectures we use U-ViT [11]. We follow standard evaluation protocols and report Fréchet Inception Distance (FID) [88], relative FID (rFID) [251], spatial FID (sFID) [157], Inception Score (IS) [193], Precision/Recall [116], and CLIP score [87], computed over 50K generated samples unless stated otherwise. Efficiency is measured by parameter count, MACs, speedup, and throughput (images/s) on a single NVIDIA A100 GPU. Datasets include CIFAR-10 [115], CelebA [135], LSUN-Church/Bedroom [242], ImageNet [45], and MS-COCO [128]. We adopt Diff-Pruning [61] as the main pruning baseline for DDPM, LDM-4, and U-ViT, and BK-SDM for SDM v1.4, following their training

and sampling settings [61, 111, 11].

Table 7.2: Quantitative comparison of methods applied to UNet-based diffusion models on LSUN-Church (256×256), LSUN-Bedroom (256×256), and ImageNet (256×256)

LSUN-Church (256×256)										
Method	#Params ↓	MACs ↓	Speed ↑	Imgs/s ↑	rFID ↓	sFID ↓	FID ↓	IS ↑	Prec. ↑	Rec. ↑
DDPM	113.7 M	248.7 G	1.0×	0.35/s	-	3.61	10.58	-	61.01	39.27
Diff-Pruning	63.2 M	138.8 G	1.34×	0.47/s	4.09	5.04	13.90	-	50.43	41.14
Deep Cache	113.7 M	156.0 G	1.48×	0.47/s	-	-	13.68	-	-	-
2ndM (Ours)	63.2 M	138.8 G	1.48×	0.47/s	2.08	4.13	11.25	-	54.36	38.51
LSUN-Bedroom (256×256)										
Method	#Params ↓	MACs ↓	Speed ↑	Imgs/s ↑	rFID ↓	sFID ↓	FID ↓	IS ↑	Prec. ↑	Rec. ↑
DDPM	113.7 M	248.7 G	1.0×	0.35/s	-	3.81	6.62	-	52.75	41.67
Diff-Pruning	63.2 M	138.8 G	1.48×	0.47/s	7.62	5.84	17.90	-	36.41	39.98
Deep Cache	113.7 M	156.0 G	1.48×	0.47/s	-	-	9.49	-	-	-
2ndM (Ours)	63.2 M	138.8 G	1.48×	0.47/s	2.16	4.53	9.68	-	44.15	39.18
ImageNet (256×256)										
Method	#Params ↓	MACs ↓	Speed ↑	Imgs/s ↑	rFID ↓	sFID ↓	FID ↓	IS ↑	Prec. ↑	Rec. ↑
LDM-4	400.9 M	99.8 G	1.0×	0.26/s	-	3.52	3.6	265.46	73.54	46.17
Diff-Pruning	175.8 M	43.2 G	1.7×	0.44/s	9.28	6.48	10.23	131.85	60.63	47.53
2ndM (Ours)	175.8 M	43.2 G	1.7×	0.44/s	4.11	4.41	5.68	168.40	68.56	48.92

U-Net-based Diffusion Models

Table 7.2 summarizes results on LSUN-Church, LSUN-Bedroom, and ImageNet at 256×256 resolution. Across all datasets, **2ndMatch** substantially improves generation quality over Diff-Pruning while preserving its computational advantages (same sparsity, parameters, MACs, and speed).

On ImageNet, 2ndMatch reduces FID from 10.23 to **5.68** and improves rFID from 9.28 to **4.11**, more than halving the discrepancy to the dense LDM-4 teacher. On LSUN-Church and LSUN-Bedroom, 2ndMatch consistently lowers FID, sFID, and rFID relative to Diff-Pruning (e.g., FID $13.90 \rightarrow$ **11.25** on LSUN-Church and $17.90 \rightarrow$ 9.68 on LSUN-Bedroom),

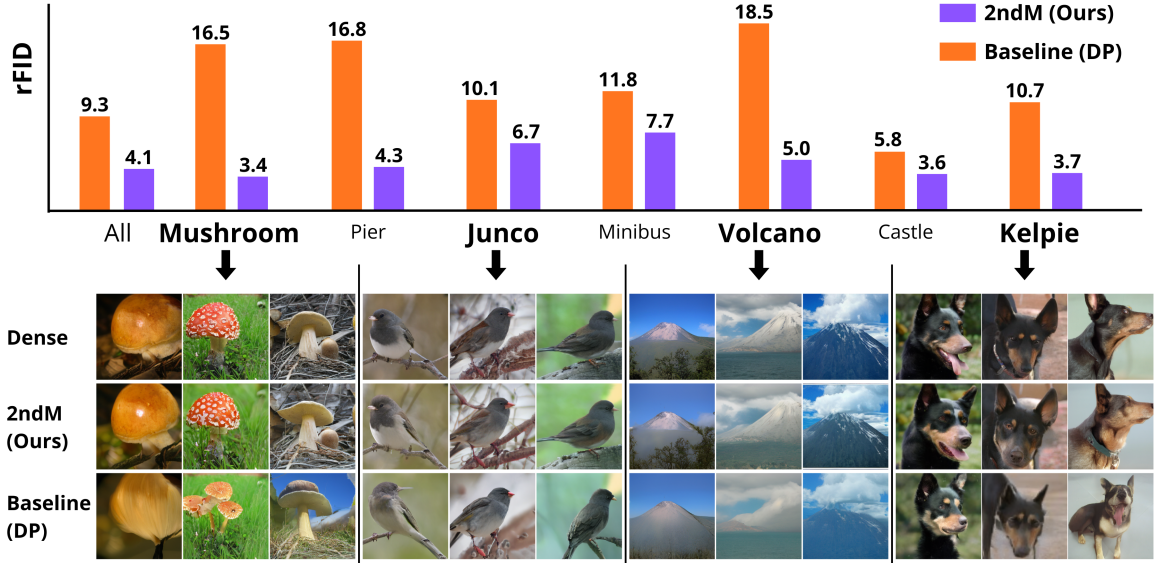


Figure 7.3: Per-class rFID and qualitative comparison between Dense, 2ndMatch, and baseline finetuning (DP) on ImageNet 256×256 .

while keeping identical compute. DeepCache achieves a slightly lower FID than 2ndMatch on LSUN-Bedroom, but at the cost of nearly $2\times$ the parameters and extra memory for feature caching. Figure 7.3 provides a per-class analysis on ImageNet. The top panel shows that rFID improves across all displayed classes when switching from Diff-Pruning to 2ndMatch, and the bottom panel illustrates that 2ndMatch more closely matches the dense model’s images in terms of global structure and fine details (e.g., for *mushroom*, *junco*, and *kelpie*).

We also apply 2ndMatch to BK-SDM–pruned Stable Diffusion v1.4 at different model sizes (Base/Small/Tiny). As shown in Table 7.3, 2ndMatch consistently improves FID and CLIP score under the same compute budget; for example, FID on the Base model improves from 15.76 to **13.84**, moving it much closer to the dense SDM baseline (13.05) without changing the architecture or sampling schedule.

Table 7.3: Generation quality on MS-COCO (256×256 , 30K samples) for BK-SDM-pruned Stable Diffusion 1.4 models of different sizes (Base / Small / Tiny), with and without 2ndM finetuning.

Method	#Param ↓	MACs ↓	Speed ↑	FID ↓	CLIP ↑
SDM-v1.4	1.04B	338.7G	1.0×	13.05	0.2958
Base	0.76B	223.8G	1.48×	15.76	0.2878
Base + 2ndM	0.76B	223.8G	1.48×	13.84	0.2900
Small	0.66B	217.7G	1.62×	16.98	0.2677
Small + 2ndM	0.66B	217.7G	1.62×	16.17	0.2682
Tiny	0.50B	205.0G	1.77×	17.12	0.2653
Tiny + 2ndM	0.50B	205.0G	1.77×	16.68	0.2636

Transformer-based Diffusion Models

Table 7.4 reports results on U-ViT [11] for CIFAR-10 (32×32) and CelebA (64×64). On CIFAR-10, 2ndMatch improves Diff-Pruning from FID 4.63 to **4.05** at the same sparsity (14.5M parameters, 3.7G MACs), providing a better quality-efficiency tradeoff than SparseDM (3:4), which is faster but significantly worse in FID (5.58). On CelebA, 2ndMatch further reduces FID from 3.57 (Diff-Pruning) to **3.35**, outperforming both SparseDM variants while matching Diff-Pruning’s compute. These results indicate that the proposed second-order Jacobian objective is architecture-agnostic and extends naturally to attention-based diffusion models.

Ablations and Robustness

We conduct extensive ablation studies to examine the robustness and generality of 2ndMatch across pruning ratios and pruning strategies.

Table 7.4: Quantitative comparison on Transformer-based diffusion models (U-ViT) for CIFAR-10 and CelebA.

Method	CIFAR-10 (32×32)				CelebA (64×64)			
	# Param	MACs	Speed	FID	# Param	MACs	Speed	FID
U-ViT	44M	11.3G	1.0×	3.11	44M	11.3G	1.0×	2.87
SparseDM (3:4)	11M	2.8G	4.0×	5.58	11M	2.8G	4.0×	5.05
Diff-Pruning	14.5M	3.7G	3.1×	4.63	14.5M	3.7G	3.1×	3.57
SparseDM (2:4)	22M	5.7G	2.0×	3.81	22M	5.7G	2.0×	3.52
2ndM (Ours)	14.5M	3.7G	3.1×	4.05	14.5M	3.7G	3.1×	3.35

Pruning ratio. Table 7.5 evaluates Diff-Pruning vs. 2ndMatch under varying pruning ratios on both U-Net and U-ViT. Across all sparsity levels, 2ndMatch improves FID, with gains increasing at higher pruning ratios (e.g., U-Net FID $9.33 \rightarrow \mathbf{7.10}$ at 70% pruning and $13.42 \rightarrow \mathbf{10.91}$ at 82% pruning). Similar trends hold for U-ViT, where 2ndMatch remains markedly better than Diff-Pruning even at 80% pruning (**5.01** vs. 6.68), indicating strong robustness to aggressive sparsification.

Pruning method. Finally, Table 7.6 compares several pruning strategies on CIFAR-10 at 44% sparsity, with and without 2ndMatch. In all cases, adding 2ndMatch on top of either baseline training or KD leads to the best FID (e.g., Diff-Pruning $5.29 \rightarrow 5.05$ with KD and further to **4.58** with 2ndMatch). Even simple random pruning benefits ($6.22 \rightarrow \mathbf{5.57}$), showing that the proposed objective is complementary to the choice of pruning criterion and can be plugged into a wide range of sparsification pipelines.

Table 7.5: Comparison of Diff-Pruning (DP) and 2ndM across pruning ratios on U-Net (left) and U-ViT (right).

U-Net							U-ViT						
PR	#Param	MACs	Speed	Imgs/s	DP	2ndM	PR	#Param	MACs	Speed	Imgs/s	DP	2ndM
dense	35.7M	6.1G	1.00×	31.1		4.19	dense	44M	11.3G	1.00×	0.54		3.11
0.56	14.3M	2.7G	1.46×	45.5	6.36	5.54	0.60	18.5M	4.7G	1.80×	0.97	3.83	3.75
0.70	8.6M	1.5G	2.19×	68.2	9.33	7.10	0.70	14.5M	3.7G	2.07×	1.12	4.63	4.05
0.82	5.1M	1.1G	2.58×	80.2	13.42	10.91	0.80	10.4M	2.6G	2.44×	1.32	6.68	5.01

Table 7.6: **FID scores on CIFAR-10 (44% pruning)** for different pruning methods with and without 2ndM. Lower is better.

FID ↓	Random	LAMP	Magnitude	Taylor	Diff-Pruning
Baseline	6.22	6.09	5.74	5.71	5.29
KD	5.88	5.57	5.53	5.39	5.05
2ndM	5.57	5.30	5.25	5.17	4.58

Chapter 8

CONCLUSION

This dissertation has demonstrated that linearization provides a principled dual framework for studying and improving neural networks, addressing the challenges of reliable training and efficient compression. Linearization through the input Jacobian gives rise to complementary tools for these challenges. The Lyapunov spectrum and its finite-time analog (FTLE) characterize how network dynamics respond to perturbations over time. Together, these tools inform design, analysis, and compression of neural networks across settings ranging from biological sequence generation to large-scale text-to-image synthesis.

8.1 Contributions

Role I: Design Target [CZ 1]. R-FORCE treats the spectral structure of the recurrent Jacobian as a design target for initialization. By introducing four generating principles for the recurrent connectivity matrix that keep Jacobian eigenvalues within the stability region, R-FORCE makes FORCE learning robust across a broad range of dynamic regimes. R-FORCE expands the operating range of FORCE far beyond the narrow edge-of-chaos band required by standard initialization and consistently improves generation quality on sequential signal tasks.

Role II: Predictive Features [CZ 2]. AeLLE (Autoencoder Latent Lyapunov Exponent) uses the Lyapunov Exponent spectrum computed from the input Jacobian as a performance-predictive feature. An autoencoder compresses LE spectra of variable length into a fixed-dimensional latent space that generalizes across RNN architectures and is available from the very beginning of training. AeLLE features outperform both raw LE statistics and loss-based metrics as early predictors of final model performance after training, enabling inexpensive architecture and hyperparameter selection.

Role III: Dynamical Compass [CZ 3]. Hyperpruning (LSH) reframes the search

over pruned RNN configurations as a hyperparameter optimization problem and introduces the Lyapunov Spectrum distance between a dense model and its pruned variants as the evaluation criterion. Since this distance reflects structural changes to the network’s dynamics before the loss has had time to reveal them, it is a more reliable early proxy for post-training performance than any loss-based metric. LSH identifies better-performing pruned configurations at an order-of-magnitude lower computational cost than conventional methods.

Role IV: Optimization Objective [CZ 4]. The final contribution in this thesis uses Jacobian-based quantities as a direct training signal for compressing diffusion models. 2ndMatch, inspired by FTLE, introduces a second-order Jacobian matching objective that aligns the input-space perturbation sensitivity of pruned and dense models throughout the denoising trajectory; since first-order Jacobian matching is effectively redundant with the standard denoising objective, the second-order term provides a genuinely complementary signal that consistently closes the quality gap across U-Net and Transformer architectures.

8.2 Synthesis

While the four contributions span a variety of architectures (random and gated RNNs, spiking networks, and U-Net and Transformer diffusion backbones) and different goals (initialization, performance prediction, pruning search, and finetuning), they are all expressions of a single object: the input Jacobian and the Lyapunov spectrum derived from it. R-FORCE shapes this spectrum at initialization; AeLLE reads it as a predictive feature; Hyperpruning measures distances within it; and 2ndMatch matches its finite-time, second-order behavior along a trajectory. Taken together, the dissertation shows that the Jacobian and its spectrum are not merely tools for after-the-fact analysis but can, in turn, serve as a design target, a predictive feature, a metric, and a training signal.

This versatility is what unifies the dual challenges of stability and sparsity. Across the contributions, linearization is used in one of two complementary modes: either to reveal networks’ behavioral properties *without full training*, as in R-FORCE’s initialization design and the early performance prediction of AeLLE and Hyperpruning, or to supply a *direct optimization signal*, as in 2ndMatch’s finetuning objective. Two lessons recur across these modes. First, the networks’ dynamical signature is established early, often at initialization,

and is a more reliable indicator of eventual quality than loss-based metrics. This makes design and search dramatically cheaper. Second, matching how a model responds to perturbations, rather than only its outputs, transfers behavior from one model to another more faithfully. This is why higher-order sensitivity matching succeeds where output matching alone does not. Linearization thus functions less as a single technique and more as a shared language in which problems of training and compression can be posed and solved.

8.3 *Limitations*

The methods developed here also share several limitations. The first is a computational limitation. Computing Lyapunov exponents requires evolving and re-orthogonalizing long products of Jacobians (in AeLLE and Hyperpruning), and the second-order objective of 2ndMatch requires vector–Jacobian products across the denoising trajectory. While scalable approximations such as QR-based estimation and random projection keep these tractable, they still impose overhead relative to purely loss-based methods, and the cost grows with sequence length and trajectory horizon.

A second set of limitations is methodological. Several components are derived empirically rather than from first principles. R-FORCE’s four generating principles and the staged loss schedule of AeLLE were identified through experimentation. AeLLE additionally requires training an autoencoder whose latent features are not directly interpretable. Hyperpruning’s Lyapunov-spectrum proxy still requires a few epochs of partial training and a dense reference model. 2ndMatch relies on a random-projection surrogate for finite-time sensitivity rather than exact FTLE. We showed the effectiveness of these approaches in practice, but it remains open why the particular choice in the approaches is optimal.

Finally, the scope of validation for our proposed methods has been limited, and the bounds of their regimes are unknown. Each contribution is demonstrated on a focused set of tasks and architectures, ranging from RNN-based signal generation and language modeling to diffusion-based image synthesis, and broader generality is shown only in part. More fundamentally, the entire framework rests on the input Jacobian; the complementary parameter-space view, captured by the Neural Tangent Kernel, is left for future explorations.

8.4 Future Directions

Several directions remain open. The Lyapunov-based analysis tools developed for RNNs have natural analogs for modern state-space models (SSMs) [74, 73] and diffusion model trajectories. Extending them to these architectures is a promising next step. A complementary direction concerns other ways to linearize a network. This dissertation works throughout with the *input Jacobian*, which describes how networks’ outputs respond to perturbations of their inputs and states. An additional Jacobian, *parameter Jacobian*, could be considered. Its outer product is the Neural Tangent Kernel (NTK) [102], which describes how the outputs respond to changes in the weights and governs the network evolution under gradient descent. Extending the linearization framework developed here to this parameter-space Jacobian, for example, to predict trainability, inform initialization, or transfer training dynamics from a dense teacher to a compressed student, would unify the input- and parameter-space views of linearization. On the compression side, the second-order Jacobian matching principle of 2ndMatch is not specific to pruning. Any setting that transfers behavioral properties from a dense teacher to a compressed student, including quantization-aware calibration [101] and distillation to lighter backbones [90], could benefit from the same FTLE-inspired sensitivity matching. Finally, integrating the search efficiency of Hyperpruning with the gradient-based precision of 2ndMatch, for example, by using Lyapunov-based proxies to identify promising pruned configurations before second-order finetuning, represents a natural synthesis of the contributions developed in this dissertation.

Publication

- [CZ 1] **Yang Zheng** and Eli Shlizerman. “R-FORCE: Robust Learning for Random Recurrent Neural Networks.” *Springer Nature Computer Science*, under review, 2026.
- [CZ 2] Ryan Vogt, **Yang Zheng**, and Eli Shlizerman. “Lyapunov-guided representation of recurrent neural network performance.” *Neural Computing and Applications*, 36(34):21211–21226, 2024.
- [CZ 3] **Caleb Zheng** and Eli Shlizerman. “Hyperpruning: Efficient Search through Pruned Variants of Recurrent Neural Networks Leveraging Lyapunov Spectrum.” *Springer Nature Computer Science*, under review, 2026.
- [CZ 4] **Caleb Zheng** and Eli Shlizerman. “2ndMatch: Finetuning Pruned Diffusion Models via Second-Order Jacobian Matching.” In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026.
- [CZ 5] **Yang Zheng** et al. “BI-MAML: Balanced Incremental Approach for Meta Learning.” *arXiv preprint arXiv:2006.07412*, 2020.
- [CZ 6] M. Chen, Z. Cui, X. Liu, J. Xiang, **Y. Zheng**, J. Li, and E. Shlizerman. “SAVVY: Spatial Awareness via Audio-Visual LLMs through Seeing and Hearing.” In *Advances in Neural Information Processing Systems*, 38:118999–119038, 2026.
- [CZ 7] **Caleb Zheng**, Jyotika Singh, Fang Tu, Weiyi Sun, Sujeeth Bharadwaj, Yassine Benajiba, Sujith Ravi, Eli Shlizerman, and Dan Roth. “DiffuMask: Diffusion Language Model for Token-Level Prompt Pruning.” *ACL*, under review, 2026.

BIBLIOGRAPHY

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Floren-
cia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anad-
kat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- [2] Shubham Agarwal, Subrata Mitra, Sarthak Chakraborty, Srikrishna Karanam, Koyel
Mukherjee, and Shiv Kumar Saini. Approximate caching for efficiently serving {Text-
to-Image} diffusion models. In *21st USENIX Symposium on Networked Systems De-
sign and Implementation (NSDI 24)*, pages 1173–1189, 2024.
- [3] Sotiris Anagnostidis, Gregor Bachmann, Yeongmin Kim, Jonas Kohler, Markos Geor-
gopoulos, Artsiom Sanakoyeu, Yuming Du, Albert Pumarola, Ali Thabet, and Edgar
Schönfeld. Flexidit: Your diffusion transformer can easily generate high-quality sam-
ples with less compute. In *Proceedings of the Computer Vision and Pattern Recogni-
tion Conference*, pages 28316–28326, 2025.
- [4] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative
adversarial networks. *arXiv preprint arXiv:1701.04862*, 2017.
- [5] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural
networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016.
- [6] H Babaee and TP Sapsis. A minimization principle for the description of modes asso-
ciated with finite-time instabilities. *Proceedings of the Royal Society A: Mathematical,
Physical and Engineering Sciences*, 472(2186):20150779, 2016.
- [7] Hessam Babaee, Mohamad Farazmand, George Haller, and Themistoklis P Sapsis.
Reduced-order description of transient instabilities and computation of finite-time
lyapunov exponents. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(6),
2017.

- [8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [9] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv e-prints*, pages arXiv–1803, 2018.
- [10] Yogesh Balaji, Seungjun Nah, Xun Huang, Arash Vahdat, Jiaming Song, Qinsheng Zhang, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, Bryan Catanzaro, Tero Karras, and Ming-Yu Liu. ediff-i: Text-to-image diffusion models with an ensemble of expert denoisers, 2023.
- [11] Fan Bao, Shen Nie, Kaiwen Xue, Yue Cao, Chongxuan Li, Hang Su, and Jun Zhu. All are worth words: A vit backbone for diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 22669–22679, 2023.
- [12] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, 15(6):1373–1396, 2003.
- [13] Giancarlo Benettin, Luigi Galgani, Antonio Giorgilli, and Jean-Marie Strelcyn. Lyapunov characteristic exponents for smooth dynamical systems and for Hamiltonian systems; a method for computing all of them. part 1: Theory. *Meccanica*, 15(1):9–20, 1980.
- [14] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.
- [15] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

- [16] Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.
- [17] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.
- [18] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.
- [19] James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, 2015.
- [20] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In *Icml*, 2021.
- [21] Nils Bertschinger, Thomas Natschläger, and Robert Legenstein. At the edge of chaos: Real-time computations and self-organized criticality in recurrent neural networks. In L. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2004.
- [22] Nils Bertschinger and Thomas Natschläger. Real-time computation at the edge of chaos in recurrent neural networks. *Neural Computation*, 16(7):1413–1436, 2004.
- [23] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, et al. Hyperparameter optimization: Foundations, algorithms, best practices and open challenges. *arXiv preprint arXiv:2107.05847*, 2021.
- [24] Bernd Bischl, Martin Binder, Michel Lang, Tobias Pielok, Jakob Richter, Stefan Coors, Janek Thomas, Theresa Ullmann, Marc Becker, Anne-Laure Boulesteix, et al. Hyperparameter optimization: Foundations, algorithms, best practices, and open

- challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 13(2):e1484, 2023.
- [25] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.
- [26] 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.
- [27] James R Bunch, Christopher P Nielsen, and Danny C Sorensen. Rank-one modification of the symmetric eigenproblem. *Numerische Mathematik*, 31(1):31–48, 1978.
- [28] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020.
- [29] Thibault Castells, Hyoungh-Kyu Song, Bo-Kyeong Kim, and Shinkook Choi. Ld-pruner: Efficient pruning of latent diffusion models using task-agnostic insights. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 821–830, 2024.
- [30] Bo Chang, Minmin Chen, Eldad Haber, and Ed H Chi. Antisymmetricrnn: A dynamical system view on recurrent neural networks. *arXiv preprint arXiv:1902.09689*, 2019.
- [31] Bo Chang, Minmin Chen, Eldad Haber, and Ed H Chi. AntisymmetricRNN: a dynamical system view on recurrent neural networks. In *International Conference on Learning Representations*, 2019.
- [32] Lei Chen, Yuan Meng, Chen Tang, Xinzhu Ma, Jingyan Jiang, Xin Wang, Zhi Wang, and Wenwu Zhu. Q-dit: Accurate post-training quantization for diffusion transformers. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 28306–28315, 2025.

- [33] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.
- [34] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PmLR, 2020.
- [35] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four {gpu} hours: A theoretically inspired perspective. In *International Conference on Learning Representations*, 2021.
- [36] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *arXiv preprint arXiv:1711.02257*, 2017.
- [37] Zigeng Chen, Xinyin Ma, Gongfan Fang, Zhenxiong Tan, and Xinchao Wang. Asyncdiff: Parallelizing diffusion models by asynchronous denoising. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.
- [38] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.
- [39] Kyunghyun Cho, Bart van Merriënboer, Çağlar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.
- [40] Junyoung Chung, Çağlar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.
- [41] SR Crandall and TA Nick. Neural population spiking activity during singing: adult and longitudinal developmental recordings in the zebra finch. *CRCNS. org* <http://dx.doi.org/10.6080/K0NP22C8>, 2014.

- [42] Wojciech M Czarnecki, Simon Osindero, Max Jaderberg, Grzegorz Swirszcz, and Razvan Pascanu. Sobolev training for neural networks. *Advances in neural information processing systems*, 30, 2017.
- [43] Bin Dai and David Wipf. Diagnosing and enhancing vae models. *arXiv preprint arXiv:1903.05789*, 2019.
- [44] Valentin Dalibard and Max Jaderberg. Faster improvement rate population based training. *arXiv preprint arXiv:2109.13800*, 2021.
- [45] 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.
- [46] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *Advances in neural information processing systems*, 27, 2014.
- [47] Brian DePasquale, Christopher J Cueva, Kanaka Rajan, LF Abbott, et al. full-force: A target-based method for training recurrent networks. *PloS one*, 13(2):e0191527, 2018.
- [48] Prafulla Dhariwal and Alexander Quinn Nichol. Diffusion models beat GANs on image synthesis. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.
- [49] L Dieci and E S Van Vleck. Computation of a few Lyapunov exponents for continuous and discrete dynamical systems. *Applied Numerical Mathematics*, 17(3):275–291, 1995.
- [50] Tobias Domhan, Jost Tobias Springenberg, Frank Hutter, et al. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, volume 15, pages 3460–8, 2015.

- [51] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- [52] Arkadiy Dushatskiy, Alexander Chebykin, Tanja Alderliesten, and Peter Bosman. Multi-objective population based training. In *International Conference on Machine Learning*, pages 8969–8989. PMLR, 2023.
- [53] Katharina Eggensperger, Matthias Feurer, Frank Hutter, James Bergstra, Jasper Snoek, Holger Hoos, Kevin Leyton-Brown, et al. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, 2013.
- [54] Rainer Engelken, Fred Wolf, and Larry F Abbott. Lyapunov spectra of chaotic recurrent neural networks. *arXiv preprint arXiv:2006.02427*, 2020.
- [55] N Benjamin Erichson, Omri Azencot, Alejandro Queiruga, Liam Hodgkinson, and Michael W Mahoney. Lipschitz recurrent neural networks. In *International Conference on Learning Representations*, 2020.
- [56] Bard Ermentrout. Ermentrout-kopell canonical model. *Scholarpedia*, 3(3):1398, 2008.
- [57] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pages 2943–2952. PMLR, 2020.
- [58] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International conference on machine learning*, pages 1437–1446. PMLR, 2018.
- [59] Gongfan Fang, Kunjun Li, Xinyin Ma, and Xinchao Wang. Tinyfusion: Diffusion transformers learned shallow. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 18144–18154, 2025.

- [60] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16091–16101, 2023.
- [61] Gongfan Fang, Xinyin Ma, and Xinchao Wang. Structural pruning for diffusion models. *arXiv preprint arXiv:2305.10924*, 2023.
- [62] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.
- [63] Karlheinz Geist, Ulrich Parlitz, and Werner Lauterborn. Comparison of different methods for computing lyapunov exponents. *Progress of theoretical physics*, 83(5):875–893, 1990.
- [64] Jean Ginibre. Statistical ensembles of complex, quaternion, and real matrices. *Journal of Mathematical Physics*, 6(3):440–449, 1965.
- [65] Vyacheslav L Girko. Circular law. *Theory of Probability & Its Applications*, 29(4):694–706, 1985.
- [66] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feed-forward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
- [67] Matthew D Golub and David Sussillo. Fixedpointfinder: A tensorflow toolbox for identifying and characterizing fixed points in recurrent neural networks. *Journal of open source software*, 3(31):1003, 2018.
- [68] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.
- [69] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley,

- Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.
- [70] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.
- [71] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376, 2006.
- [72] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.
- [73] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.
- [74] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.
- [75] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017.
- [76] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.
- [77] George Haller. Distinguished material surfaces and coherent structures in three-dimensional fluid flows. *Physica D: Nonlinear Phenomena*, 149(4):248–277, 2001.
- [78] George Haller and Guocheng Yuan. Lagrangian coherent structures and mixing in two-dimensional turbulence. *Physica D: Nonlinear Phenomena*, 147(3-4):352–370, 2000.

- [79] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.
- [80] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.
- [81] Philip Hartman. *Ordinary differential equations*. SIAM, 2002.
- [82] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992.
- [83] Ali Hatamizadeh, Jiaming Song, Guilin Liu, Jan Kautz, and Arash Vahdat. Dif-fit: Diffusion vision transformers for image generation. In *European Conference on Computer Vision*, pages 37–55. Springer, 2024.
- [84] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- [85] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [86] Kyle Helfrich, Devin Willmott, and Qiang Ye. Orthogonal recurrent neural networks with scaled cayley transform. *arXiv preprint arXiv:1707.09520*, 2017.
- [87] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. Clip-score: A reference-free evaluation metric for image captioning. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pages 7514–7528, 2021.
- [88] 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.
- [89] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. Recurrent neural networks for time series forecasting: Current status and future directions. *International Journal of Forecasting*, 37(1):388–427, 2021.
- [90] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
- [91] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.
- [92] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.
- [93] Jonathan Ho, Chitwan Saharia, William Chan, David J. Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation. *Journal of Machine Learning Research*, 23(47):1–33, 2022.
- [94] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.
- [95] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [96] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *journal of machine learning research*, 18(187):1–30, 2018.
- [97] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.
- [98] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

- [99] Aapo Hyvärinen and Peter Dayan. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(4), 2005.
- [100] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.
- [101] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.
- [102] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.
- [103] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.
- [104] Herbert Jaeger. The “echo state” approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148(34):13, 2001.
- [105] Weinan Jia, Mengqi Huang, Nan Chen, Lei Zhang, and Zhendong Mao. D²it: Dynamic diffusion transformer for accurate image generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 12860–12870, 2025.
- [106] I. T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer-Verlag, New York, 2002.
- [107] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv e-prints*, pages arXiv–1506, 2015.
- [108] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design

- space of diffusion-based generative models. *Advances in neural information processing systems*, 35:26565–26577, 2022.
- [109] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.
 - [110] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8110–8119, 2020.
 - [111] Bo-Kyeong Kim, Hyoungh-Kyu Song, Thibault Castells, and Shinkook Choi. On architectural compression of text-to-image diffusion models. *arXiv preprint arXiv:2305.15798*, 2023.
 - [112] Bo-Kyeong Kim, Hyoungh-Kyu Song, Thibault Castells, and Shinkook Choi. Bk-sdm: A lightweight, fast, and cheap version of stable diffusion. In *European Conference on Computer Vision*, pages 381–399. Springer, 2024.
 - [113] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
 - [114] Aaron Klein, Stefan Falkner, Jost Tobias Springenberg, and Frank Hutter. Learning curve prediction with bayesian neural networks. In *International conference on learning representations*, 2017.
 - [115] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. <https://www.cs.toronto.edu/kriz/learning-features-2009-TR.pdf>, 2009.
 - [116] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. *Advances in neural information processing systems*, 32, 2019.

- [117] Thomas Laurent and James von Brecht. A recurrent neural network without chaos. *arXiv preprint arXiv:1612.06212*, 2016.
- [118] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.
- [119] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [120] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.
- [121] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.
- [122] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.
- [123] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The journal of machine learning research*, 18(1):6765–6816, 2017.
- [124] Muyang Li, Tianle Cai, Jiaxin Cao, Qinsheng Zhang, Han Cai, Junjie Bai, Yangqing Jia, Ming-Yu Liu, Kai Li, and Song Han. Distrifusion: Distributed parallel inference for high-resolution diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.
- [125] Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17535–17545, 2023.

- [126] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.
- [127] Soon Hoe Lim, N Benjamin Erichson, Liam Hodgkinson, and Michael W Mahoney. Noisy recurrent neural networks. *Advances in Neural Information Processing Systems*, 34:5124–5137, 2021.
- [128] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.
- [129] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023.
- [130] Enshu Liu, Xuefei Ning, Zinan Lin, Huazhong Yang, and Yu Wang. Oms-dpm: Optimizing the model schedule for diffusion probabilistic models. In *International Conference on Machine Learning*, pages 21915–21936. PMLR, 2023.
- [131] Feng Liu, Shiwei Zhang, Xiaofeng Wang, Yujie Wei, Haonan Qiu, Yuzhong Zhao, Yingya Zhang, Qixiang Ye, and Fang Wan. Timestep embedding tells: It’s time to cache for video diffusion model. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 7353–7363, 2025.
- [132] Shiwei Liu, Decebal Constantin Mocanu, Yulong Pei, and Mykola Pechenizkiy. Selfish sparse rnn training. In *International Conference on Machine Learning*, pages 6893–6904. PMLR, 2021.
- [133] Xingchao Liu, Chengyue Gong, and qiang liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *The Eleventh International Conference on Learning Representations*, 2023.
- [134] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Pro-*

- ceedings of the IEEE international conference on computer vision*, pages 2736–2744, 2017.
- [135] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE international conference on computer vision*, pages 3730–3738, 2015.
- [136] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. *Advances in Neural Information Processing Systems*, 35:5775–5787, 2022.
- [137] Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are gans created equal? a large-scale study. *Advances in neural information processing systems*, 31, 2018.
- [138] Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference, 2023.
- [139] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15762–15772, 2024.
- [140] Wolfgang Maass, Thomas Natschläger, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.
- [141] Kehelwala DG Maduranga, Kyle E Helfrich, and Qiang Ye. Complex unitary recurrent neural networks using scaled cayley transform. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4528–4535, 2019.
- [142] Valerio Mante, David Sussillo, Krishna V Shenoy, and William T Newsome. Context-dependent computation by recurrent dynamics in prefrontal cortex. *nature*, 503(7474):78–84, 2013.

- [143] Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Using Large Corpora*, page 273, 1994.
- [144] Robert M May. Will a large complex system be stable? *Nature*, 238(5364):413–414, 1972.
- [145] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.
- [146] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *International conference on machine learning*, pages 3481–3490. PMLR, 2018.
- [147] Zakaria Mhammedi, Andrew Hellicar, Ashfaque Rahman, and James Bailey. Efficient orthogonal parametrisation of recurrent neural networks using householder reflections. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2401–2409. JMLR. org, 2017.
- [148] Tomas Mikolov, Armand Joulin, Sumit Chopra, Michael Mathieu, and Marc’Aurelio Ranzato. Learning longer memory in recurrent neural networks. *arXiv preprint arXiv:1412.7753*, 2014.
- [149] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5528–5531. IEEE, 2011.
- [150] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.
- [151] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.

- [152] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *International Conference on Learning Representations*, 2017.
- [153] Alejandro Morales-Hernández, Sebastian Rojas Gonzalez, Inneke Van Nieuwenhuijse, Ivo Couckuyt, Jeroen Jordens, Maarten Witters, and Bart Van Doninck. Bayesian multi-objective optimization of process design parameters in constrained settings with noise: an engineering design application. *Engineering with Computers*, 40(4):2497–2511, 2024.
- [154] Ari Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. *Advances in neural information processing systems*, 32, 2019.
- [155] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1325–1334, 2019.
- [156] Ilan Naiman and Omri Azencot. A Koopman approach to understanding sequence neural models, 2021.
- [157] Charlie Nash, Jacob Menick, Sander Dieleman, and Peter W Battaglia. Generating images with sparse representations. *arXiv preprint arXiv:2103.03841*, 2021.
- [158] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021.
- [159] Wilten Nicola and Claudia Clopath. Supervised learning in spiking neural networks with force training. *Nature communications*, 8(1):1–15, 2017.
- [160] Akira Okubo. Horizontal dispersion of floatable particles in the vicinity of velocity singularities such as convergences. In *Deep sea research and oceanographic abstracts*, volume 17, pages 445–454. Elsevier, 1970.

- [161] Valery Iustinovich Oseledec. A multiplicative ergodic theorem, lyapunov characteristic numbers for dynamical systems. *Transactions of the Moscow Mathematical Society*, 19:197–231, 1968.
- [162] V. Oseledets. Oseledets theorem. *Scholarpedia*, 3(1):1846, 2008. revision #142085.
- [163] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.
- [164] Zizheng Pan, Bohan Zhuang, De-An Huang, Weili Nie, Zhiding Yu, Chaowei Xiao, Jianfei Cai, and Anima Anandkumar. T-stitch: Accelerating sampling in pre-trained diffusion models with trajectory stitching. *arXiv preprint arXiv:2402.14167*, 2024.
- [165] Wonpyo Park, Dongju Kim, Yan Lu, and Minsu Cho. Relational knowledge distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.
- [166] Gaurav Parmar, Krishna Kumar Singh, Richard Zhang, Yijun Li, Jingwan Lu, and Jun-Yan Zhu. Zero-shot image-to-image translation. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–11, 2023.
- [167] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.
- [168] Nikolaos Passalis and Anastasios Tefas. Learning deep representations with probabilistic knowledge transfer. In *Proceedings of the European conference on computer vision (ECCV)*, pages 268–284, 2018.
- [169] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4195–4205, 2023.
- [170] Jeffrey Pennington, Samuel Schoenholz, and Surya Ganguli. Resurrecting the sigmoid

- in deep learning through dynamical isometry: theory and practice. *Advances in neural information processing systems*, 30, 2017.
- [171] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. In *International Conference on Learning Representations*, volume 2024, pages 1862–1874, 2024.
- [172] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.
- [173] 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.
- [174] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.
- [175] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.
- [176] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [177] Mohaimenul Azam Khan Raiaan, Sadman Sakib, Nur Mohammad Fahad, Abdullah Al Mamun, Md Anisur Rahman, Swakkhar Shatabda, and Md Saddam Hossain Mukta. A systematic review of hyperparameter optimization techniques in convolutional neural networks. *Decision Analytics Journal*, 11:100470, 2024.

- [178] Kanaka Rajan and LF Abbott. Eigenvalue spectra of random matrices for neural networks. *Physical review letters*, 97(18):188104, 2006.
- [179] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *ArXiv*, abs/2204.06125, 2022.
- [180] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.
- [181] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic back-propagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.
- [182] Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4364–4373. PMLR, 10–15 Jul 2018.
- [183] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10684–10695, 2022.
- [184] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.
- [185] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

- [186] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.
- [187] David Ruelle. Ergodic theory of differentiable dynamical systems. *Publications Mathématiques de l’Institut des Hautes Études Scientifiques*, 50(1):27–58, 1979.
- [188] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.
- [189] T. Konstantin Rusch and Siddhartha Mishra. Coupled oscillatory recurrent neural network (coRNN): An accurate and (gradient) stable architecture for learning long time dependencies. In *International Conference on Learning Representations*, 2021.
- [190] T. Konstantin Rusch, Siddhartha Mishra, N. Benjamin Erichson, and Michael W. Mahoney. Long expressive memory for sequence modeling, 2022.
- [191] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494, 2022.
- [192] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6655–6659. IEEE, 2013.
- [193] 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.
- [194] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv preprint arXiv:2202.00512*, 2022.

- [195] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.
- [196] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.
- [197] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.
- [198] Shawn C Shadden. Lagrangian coherent structures. *Transport and mixing in laminar flows: from microfluidics to oceanic currents*, pages 59–89, 2011.
- [199] Shawn C Shadden, Francois Lekien, and Jerrold E Marsden. Definition and properties of lagrangian coherent structures from finite-time lyapunov exponents in two-dimensional aperiodic flows. *Physica D: Nonlinear Phenomena*, 212(3-4):271–304, 2005.
- [200] Yuzhang Shang, Zhihang Yuan, Bin Xie, Bingzhe Wu, and Yan Yan. Post-training quantization on diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1972–1981, 2023.
- [201] Reza Shirkavand, Peiran Yu, Shangqian Gao, Gowthami Somepalli, Tom Goldstein, and Heng Huang. Efficient fine-tuning and concept suppression for pruned diffusion models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 18619–18629, 2025.
- [202] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.
- [203] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian opti-

- mization using deep neural networks. In *International conference on machine learning*, pages 2171–2180. PMLR, 2015.
- [204] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pages 2256–2265. pmlr, 2015.
- [205] Haim Sompolinsky, Andrea Crisanti, and Hans-Jurgen Sommers. Chaos in random neural networks. *Physical review letters*, 61(3):259, 1988.
- [206] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.
- [207] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 32211–32252. PMLR, 23–29 Jul 2023.
- [208] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.
- [209] Yang Song, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. Sliced score matching: A scalable approach to density and score estimation. In *Uncertainty in artificial intelligence*, pages 574–584. PMLR, 2020.
- [210] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.
- [211] Suraj Srinivas and François Fleuret. Knowledge transfer with jacobian matching. In *International Conference on Machine Learning*, pages 4723–4731. PMLR, 2018.
- [212] David Sussillo and Larry F Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–557, 2009.

- [213] David Sussillo and Omri Barak. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural computation*, 25(3):626–649, 2013.
- [214] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.
- [215] Hiroto Tamura and Gouhei Tanaka. Transfer-rls method and transfer-force learning for simple and fast training of reservoir computing models. *Neural Networks*, 143:550–563, 2021.
- [216] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in Neural Information Processing Systems*, 33:6377–6389, 2020.
- [217] Terence TAO, Van VU, and Manjunath KRISHNAPUR. Random matrices: Universality of esds and the circular law. *Annals of probability*, 38(5):2023–2065, 2010.
- [218] Joshua B Tenenbaum, Vin de Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.
- [219] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013.
- [220] Aleksandar Vakanski, Hyung-pil Jun, David Paul, and Russell Baker. A data set of human body movements for physical rehabilitation exercises. *Data*, 3(1):2, 2018.
- [221] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.
- [222] Charles F Van Loan and Gene H Golub. *Matrix computations*. Johns Hopkins University Press, 1983.

- [223] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [224] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674, 2011.
- [225] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, Pierre-Antoine Manzagol, and Léon Bottou. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12), 2010.
- [226] Ryan Vogt, Maximilian Puelma Touzel, Eli Shlizerman, and Guillaume Lajoie. On Lyapunov exponents for RNNs: Understanding information propagation using dynamical systems tools. *Frontiers in Applied Mathematics and Statistics*, 8, 2022.
- [227] Ryan Vogt, Yang Zheng, and Eli Shlizerman. Lyapunov-guided embedding for hyperparameter selection in recurrent neural networks. *arXiv preprint arXiv:2204.04876*, 2022.
- [228] Ryan Vogt, Yang Zheng, and Eli Shlizerman. Lyapunov-guided representation of recurrent neural network performance. *Neural Computing and Applications*, pages 1–16, 2024.
- [229] Eugene Vorontsov, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal. On orthogonality and learning recurrent networks with long term dependencies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3570–3578. JMLR. org, 2017.
- [230] Saurabh Vyas, Matthew D Golub, David Sussillo, and Krishna V Shenoy. Computation through neural population dynamics. *Annual review of neuroscience*, 43(1):249–275, 2020.
- [231] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

- [232] Daniel Watson, Jonathan Ho, Mohammad Norouzi, and William Chan. Learning to efficiently sample from diffusion probabilistic models. *arXiv preprint arXiv:2106.03802*, 2021.
- [233] John Weiss. The dynamics of enstrophy transfer in two-dimensional hydrodynamics. *Physica D: Nonlinear Phenomena*, 48(2-3):273–294, 1991.
- [234] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.
- [235] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016.
- [236] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.
- [237] Felix Wimbauer, Bichen Wu, Edgar Schoenfeld, Xiaoliang Dai, Ji Hou, Zijian He, Artsiom Sanakoyeu, Peizhao Zhang, Sam Tsai, Jonas Kohler, et al. Cache me if you can: Accelerating diffusion models through block caching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6211–6220, 2024.
- [238] Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. In *Advances in neural information processing systems*, pages 4880–4888, 2016.
- [239] Alan Wolf, Jack B Swift, Harry L Swinney, and John A Vastano. Determining lyapunov exponents from a time series. *Physica D: nonlinear phenomena*, 16(3):285–317, 1985.
- [240] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. Breaking the softmax bottleneck: A high-rank rnn language model. *arXiv preprint arXiv:1711.03953*, 2017.

- [241] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014.
- [242] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.
- [243] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *International Conference on Learning Representations*, 2017.
- [244] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.
- [245] Dingkun Zhang, Sijia Li, Chen Chen, Qingsong Xie, and Haonan Lu. Laptop-diff: Layer pruning and normalized distillation for compressing diffusion models. *arXiv preprint arXiv:2404.11098*, 2024.
- [246] Linfeng Zhang and Kaisheng Ma. Accelerating diffusion models with one-to-many knowledge distillation. *arXiv preprint arXiv:2410.04191*, 2024.
- [247] Qinsheng Zhang and Yongxin Chen. Fast sampling of diffusion models with exponential integrator. *arXiv preprint arXiv:2204.13902*, 2022.
- [248] Yang Zhang, Er Jin, Yanfei Dong, Ashkan Khakzar, Philip Torr, Johannes Stegmaier, and Kenji Kawaguchi. Effortless efficiency: Low-cost pruning of diffusion models. *arXiv preprint arXiv:2412.02852*, 2024.
- [249] Kaifeng Zhao, Gen Li, and Siyu Tang. Dartcontrol: A diffusion-based autoregressive motion model for real-time text-driven motion control. *arXiv preprint arXiv:2410.05260*, 2024.
- [250] Maosen Zhao, Pengtao Chen, Chong Yu, Yan Wen, Xudong Tan, and Tao Chen. Pioneering 4-bit fp quantization for diffusion models: Mixup-sign quantization and

- timestep-aware fine-tuning. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 18134–18143, 2025.
- [251] Bowen Zheng and Tianming Yang. Diffusion models are innate one-step generators. *arXiv preprint arXiv:2405.20750*, 2024.
- [252] Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip HS Torr, et al. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6881–6890, 2021.
- [253] Yang Zheng, Jinlin Xiang, Kun Su, and Eli Shlizerman. Bi-maml: Balanced incremental approach for meta learning. *arXiv preprint arXiv:2006.07412*, 2020.
- [254] Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n:m fine-grained structured sparse neural networks from scratch. In *International Conference on Learning Representations*, 2021.
- [255] Haowei Zhu, Dehua Tang, Ji Liu, Mingjie Lu, Jintu Zheng, Jinzhang Peng, Dong Li, Yu Wang, Fan Jiang, Lu Tian, et al. Dip-go: A diffusion pruner via few-step gradient optimization. *Advances in Neural Information Processing Systems*, 37:92581–92604, 2024.
- [256] Junhan Zhu, Hesong Wang, Mingluo Su, Zefang Wang, and Huan Wang. Obs-diff: Accurate pruning for diffusion models in one-shot. *arXiv preprint arXiv:2510.06751*, 2025.
- [257] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. Recurrent highway networks. In *International conference on machine learning*, pages 4189–4198. PMLR, 2017.

Appendix A

APPENDIX

A.1 Derivation of the DDPM training objective

This appendix derives the training objective of Denoising Diffusion Probabilistic Models (DDPM) [92] from the variational lower bound (ELBO), showing how the complex probabilistic formulation reduces to the simple ϵ -prediction mean squared error used in practice.

A.1.1 Forward Process

The forward process defines a Markov chain that gradually corrupts a clean data sample $x_0 \sim q(x_0)$ into isotropic Gaussian noise over T steps:

$$q(x_{1:T} | x_0) = \prod_{t=1}^T q(x_t | x_{t-1}), \quad q(x_t | x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1 - \beta_t} x_{t-1}, \beta_t I\right), \quad (\text{A.1})$$

where $\{\beta_t\}_{t=1}^T$ is a fixed variance schedule with $0 < \beta_t < 1$. Defining $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$, the marginal at any timestep t is available in closed form:

$$q(x_t | x_0) = \mathcal{N}\left(x_t; \sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t)I\right). \quad (\text{A.2})$$

This follows by induction, using the fact that a convex combination of Gaussians is Gaussian. Equation (A.2) gives the reparameterization

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, \quad \epsilon \sim \mathcal{N}(0, I), \quad (\text{A.3})$$

which allows sampling x_t directly from x_0 without simulating intermediate steps, enabling efficient training.

A.1.2 Reverse Process and Variational Lower Bound

The reverse process defines a learned Markov chain that denoises from $x_T \sim \mathcal{N}(0, I)$ back to x_0 :

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^T p_\theta(x_{t-1} | x_t), \quad p_\theta(x_{t-1} | x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)). \quad (\text{A.4})$$

Maximizing the log-likelihood $\log p_\theta(x_0)$ is intractable, so we instead maximize the evidence lower bound (ELBO), obtained via Jensen's inequality:

$$\log p_\theta(x_0) \geq \mathbb{E}_q \left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T} | x_0)} \right] =: -\mathcal{L}. \quad (\text{A.5})$$

Expanding and rearranging the expectation using the Markov structure, $\mathcal{L}$ decomposes into interpretable terms:

$$\mathcal{L} = \underbrace{D_{\text{KL}}(q(x_T | x_0) \| p(x_T))}_{\mathcal{L}_T} + \sum_{t=2}^T \underbrace{D_{\text{KL}}(q(x_{t-1} | x_t, x_0) \| p_\theta(x_{t-1} | x_t))}_{\mathcal{L}_{t-1}} - \underbrace{\mathbb{E}_q[\log p_\theta(x_0 | x_1)]}_{\mathcal{L}_0}. \quad (\text{A.6})$$

The term $\mathcal{L}_T$ is a constant with respect to θ since q has no learnable parameters and $p(x_T) = \mathcal{N}(0, I)$ is fixed. The term $\mathcal{L}_0$ is a reconstruction term. The dominant terms are the denoising matching losses $\mathcal{L}_{t-1}$, which measure how well the learned reverse step $p_\theta(x_{t-1}|x_t)$ matches the true posterior $q(x_{t-1}|x_t, x_0)$.

A.1.3 Tractable Posterior

The posterior $q(x_{t-1} | x_t, x_0)$ is tractable because conditioning on x_0 makes the joint Gaussian. By Bayes' rule:

$$q(x_{t-1} | x_t, x_0) \propto q(x_t | x_{t-1}) q(x_{t-1} | x_0). \quad (\text{A.7})$$

Completing the square in the exponent (both factors are Gaussian) yields:

$$q(x_{t-1} | x_t, x_0) = \mathcal{N}(x_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t I), \quad (\text{A.8})$$

with

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}} \beta_t}{1 - \bar{\alpha}_t} x_0 + \frac{\sqrt{\bar{\alpha}_t} (1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} x_t, \quad (\text{A.9})$$

$$\tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t. \quad (\text{A.10})$$

Since both $p_\theta(x_{t-1}|x_t)$ and $q(x_{t-1}|x_t, x_0)$ are Gaussian, each KL term $\mathcal{L}_{t-1}$ reduces to a mean squared error between their means.

A.1.4 The ϵ -Prediction Objective

Substituting the reparameterization $x_0 = (x_t - \sqrt{1 - \bar{\alpha}_t} \epsilon) / \sqrt{\bar{\alpha}_t}$ into $\tilde{\mu}_t$ gives:

$$\tilde{\mu}_t(x_t, \epsilon) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon \right). \quad (\text{A.11})$$

The network $\mu_\theta(x_t, t)$ is therefore parameterized as:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right), \quad (\text{A.12})$$

so minimizing $\mathcal{L}_{t-1}$ is equivalent to minimizing $\|\epsilon - \epsilon_\theta(x_t, t)\|^2$. Ho et al. [92] further show that dropping the time-dependent weighting and averaging uniformly over t yields a simplified objective that trains better in practice:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t, x_0, \epsilon} \left[\left\| \epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t) \right\|^2 \right]. \quad (\text{A.13})$$

This is the standard DDPM training loss: sample a random timestep t , corrupt x_0 to x_t in one step, and train the network to predict the noise ϵ that was added.

A.2 AeLLE

A.2.1 Jacobians of RNN Architectures

In the calculation of the Lyapunov spectrum, we use the Jacobian $\frac{\partial h_t}{\partial h_{t-1}}$ (or equivalent) to perform the evolution of the tangent space at each time step. For each network, we compute the Jacobian expression so we can implement it directly, significantly improving computation speed compared to numerical derivative estimation. The expressions for all non-vanilla RNN networks are shown (Note: ASRNN has the same Jacobian as RNN but with weight constraints, so it is not shown either).

LSTM

$$y_* = W_* x_t + U_* h_{t-1} + b_*,$$

where $*$ is determined by the gate/state. For LSTM, there are 4 different gates/states (f , i , o , c), each with a corresponding y_* , W_* , U_* , b_* .

The derivatives of each of these gates/states with respect to the hidden states and the input are shown below. The final line shows the derivative of the hidden state with respect to the hidden state at the previous time step.

$$\begin{aligned} \frac{\partial f_t}{\partial h_{t-1}} &= [\sigma(y_f) \circ (1 - \sigma(y_f))]^T \circ U_f \\ \frac{\partial i_t}{\partial h_{t-1}} &= [\sigma(y_i) \circ (1 - \sigma(y_i))]^T \circ U_i \\ \frac{\partial o_t}{\partial h_{t-1}} &= [\sigma(y_o) \circ (1 - \sigma(y_o))]^T \circ U_o \\ \frac{\partial c_t}{\partial h_{t-1}} &= \frac{\partial f_t}{\partial h_{t-1}} \circ c_{t-1} + \frac{\partial i_t}{\partial h_{t-1}} \circ \tanh(y_c) + i_t \circ \text{sech}^2(y_c) \circ U_c \\ \frac{\partial h_t}{\partial h_{t-1}} &= \frac{\partial o_t}{\partial h_{t-1}} \circ \tanh(c_t) + o_t \circ \text{sech}^2(c_t) \circ \frac{\partial c_t}{\partial h_{t-1}} \end{aligned}$$

$$\begin{aligned}
\frac{\partial f_t}{\partial x_t} &= [\sigma(y_f) \circ (1 - \sigma(y_f))]^T \circ W_f \\
\frac{\partial i_t}{\partial x_t} &= [\sigma(y_i) \circ (1 - \sigma(y_i))]^T \circ W_i \\
\frac{\partial o_t}{\partial x_t} &= [\sigma(y_o) \circ (1 - \sigma(y_o))]^T \circ W_o \\
\frac{\partial c_t}{\partial x_t} &= \frac{\partial f_t}{\partial x_t} \circ c_{t-1} + \frac{\partial i_t}{\partial x_t} \circ \tanh(y_c) + i_t \circ \text{sech}^2(y_c) \circ W_c \\
\frac{\partial h_t}{\partial x_t} &= \frac{\partial o_t}{\partial x_t} \circ \tanh(c_t) + o_t \circ \text{sech}^2(c_t) \circ \frac{\partial c_t}{\partial x_t}
\end{aligned}$$

GRU

$$\begin{aligned}
y_{1*} &= W_* x + b_{i*} \\
y_{2*} &= U_* h_{t-1} + b_{h*}
\end{aligned}$$

The corresponding derivatives for the GRU are shown below.

$$\begin{aligned}
\frac{\partial r_t}{\partial h_{t-1}} &= [\sigma(y_{1r} + y_{2r}) \circ (1 - \sigma(y_{1r} + y_{2r}))]^T \circ U_r \\
\frac{\partial z_t}{\partial h_{t-1}} &= [\sigma(y_{1z} + y_{2z}) \circ (1 - \sigma(y_{1z} + y_{2z}))]^T \circ U_z \\
\frac{\partial n_t}{\partial h_{t-1}} &= \text{sech}^2(y_{1n} + r_t \circ y_{2n}) * \frac{\partial r_t}{\partial h_{t-1}} \circ y_{2n} + r_t \circ U_n \\
\frac{\partial h_t}{\partial h_{t-1}} &= -\frac{\partial z_t}{\partial h_{t-1}} \circ n_t + (1 - z_t) \circ \frac{\partial n_t}{\partial h_{t-1}} + \frac{\partial z_t}{\partial h_{t-1}} \circ h_{t-1} + z_t \circ I
\end{aligned}$$

CoRNN

$$\begin{aligned}
y_t &= y_{t-1} + z_t \Delta t \\
z_t &= z_{t-1} + (\tanh(W_y y_{t-1} + W_z z_{t-1} + V u_t + b) - \gamma y_{t-1} - \epsilon z_{t-1}) \Delta t
\end{aligned}$$

we need to calculate the derivative of y_t w.r.t y_{t-1} .

$$\begin{aligned}
\frac{dy_t}{dy_{t-1}} &= \frac{d(y_{t-1} + z_t \Delta t)}{dy_{t-1}} \\
&= \frac{d(y_{t-1} + (z_{t-1} + (\tanh(W_y y_{t-1} + W_z z_{t-1} + V u_t + b) - \gamma y_{t-1} - \epsilon z_{t-1}) \Delta t) \Delta t)}{dy_{t-1}} \\
&= \frac{dy_{t-1}}{dy_{t-1}} + \Delta t \frac{dz_{t-1}}{dy_{t-1}} + \Delta t^2 \frac{d(\tanh(W_y y_{t-1} + W_z z_{t-1} + V u_t + b))}{dy_{t-1}} \\
&\quad - \Delta t^2 \frac{d(\gamma y_{t-1} + \epsilon z_{t-1})}{dy_{t-1}} \\
&= I + 0 + \Delta t^2 W_y^T (1 - \tanh^2(W_y y_{t-1} + W_z z_{t-1} + V u_t + b)) - \Delta t^2 \gamma I \\
&= (1 - \Delta t^2 \gamma) I + \Delta t^2 W_y^T (1 - \tanh^2(W_y y_{t-1} + W_z z_{t-1} + V u_t + b))
\end{aligned}$$

Lipschitz RNN

$$\begin{cases} \dot{h} = A_{\beta_A, \gamma_A} h + \tanh(W_{\beta_W, \gamma_W} h + Ux + b) \\ y = Dh \end{cases}$$

where $A_{\beta, \gamma} \in \mathbb{R}^{N \times N}$, $W_{\beta, \gamma} \in \mathbb{R}^{N \times N}$,

$$\begin{cases} A_{\beta_A, \gamma_A} = (1 - \beta_A)(M_A + M_A^T) + \beta_A(M_A - M_A^T) - \gamma_A I \\ W_{\beta_W, \gamma_W} = (1 - \beta_W)(M_W + M_W^T) + \beta_W(M_W - M_W^T) - \gamma_W I, \end{cases}$$

where $\beta_A, \beta_W \in [0, 1]$, $\gamma_A, \gamma_W > 0$ are tunable parameters, and $M_A, M_W \in \mathbb{R}^{N \times N}$ are trainable matrices. $\dot{h}$ is the derivative with respect to time, i.e., $\dot{h} = \frac{\partial h(t)}{\partial t}$. Therefore, the Jacobian matrix of h_t is:

$$\begin{aligned}
\frac{\partial h_{t+1}}{\partial h_t} &= \frac{\partial(h_t + \Delta t \cdot \alpha A h_t + \Delta t \cdot \tanh(W h_t + U x_t + b))}{\partial h_t} \\
&= I + \Delta t \alpha A + \Delta t W \text{sech}^2(W h_t + U x_t + b)
\end{aligned}$$

, where α is default to 1 and Δt is the time interval.

Noisy RNN

$$\begin{aligned}\eta_{mulpi} &= k * \mathcal{N} \sim (0, 1) + (1 - k) \\ h_{update} &= \alpha hA + \tanh(h_t W + Ux_t + b) \\ h &= h + d \cdot \eta_{mulpi} \cdot h_{update} + noise\end{aligned}$$

where, k, α are scale constant, and $\mathcal{N} \sim (0, 1)$ is the normal distribution

$$\begin{cases} A = \beta(B - B^T) + (1 - \beta)(B + B^T) - \gamma_A I \\ W = \beta(C - C^T) + (1 - \beta)(C + C^T) - \gamma_W I \end{cases}$$

$$\begin{aligned}\frac{\partial h_{t+1}}{\partial h_t} &= \frac{\partial(h_t + d \cdot \eta_{mulpi} \cdot (\alpha h_t A + \tanh(h_t W + Ux_t + b)) + noise)}{\partial h_t} \\ &= I + d \cdot \eta_{mulpi} \cdot (\alpha A + W \cdot \text{sech}^2(h_t W + Ux_t + b))\end{aligned}$$

LEM

$$z_n = (1 - \Delta t_n) \odot z_{n-1} + \Delta t_n \odot \sigma(W_z y_{n-1} + V_z u_n + b_z) \quad (\text{A.14})$$

$$y_n = (1 - \bar{\Delta} t_n) \odot y_{n-1} + \bar{\Delta} t_n \odot \sigma(W_y z_n + V_y u_n + b_y) \quad (\text{A.15})$$

, where

$$\Delta t_n = \Delta t \hat{\sigma}(W_1 y_{n-1} + V_1 u_n + b_1)$$

$$\bar{\Delta} t_n = \Delta t \hat{\sigma}(W_2 y_{n-1} + V_2 u_n + b_2)$$

We substitute $\bar{\Delta} t_n$ into Eq.A.15 and get the following:

$$\begin{aligned}y_n &= (1 - \underbrace{\Delta t \hat{\sigma}(W_2 y_{n-1} + V_2 u_n + b_2)}_{\bar{\Delta} t_n}) \odot y_{n-1} + \\ &\quad \underbrace{\Delta t \hat{\sigma}(W_2 y_{n-1} + V_2 u_n + b_2)}_{\bar{\Delta} t_n} \odot (W_y z_n + V_y u_n + b_y)\end{aligned}$$

Therefore, the Jacobian is:

$$\begin{aligned}
\frac{\partial y_n}{\partial y_{n-1}} &= -\Delta t W_2 \hat{\sigma}'(W_2 y_{n-1} + V_2 u_n + b_2) \odot y_{n-1} \\
&\quad + (1 - \Delta t \hat{\sigma}(W_2 y_{n-1} + V_2 u_n + b_2)) \\
&\quad + \Delta t W_2 \hat{\sigma}'(W_2 y_{n-1} + V_2 u_n + b_2) \odot (W_y Z_n + V_y u_n + b_y) \\
&\quad + \Delta t \hat{\sigma}(W_2 y_{n-1} + V_2 u_n + b_2) \odot (W_y \frac{\partial Z_n}{\partial y_{n-1}})
\end{aligned}$$

where

$$\begin{aligned}
\frac{\partial Z_n}{\partial y_{n-1}} &= \frac{\partial(1 - \Delta t \hat{\sigma}(W_1 y_{n-1} + V_1 u_n + b_1) \odot Z_{n-1})}{\partial y_{n-1}} \\
&\quad + \frac{\partial(\Delta t \hat{\sigma}(W_1 y_{n-1} + V_1 u_n + b_1) \odot \sigma(W_Z y_{n-1} + V_Z u_n + b_Z))}{\partial y_{n-1}} \\
&= -\Delta t W_1 \hat{\sigma}'(W_1 y_{n-1} + V_1 u_n + b_1) \odot Z_{n-1} \\
&\quad + \Delta t W_1 \hat{\sigma}'(W_1 y_{n-1} + V_1 u_n + b_1) \odot \sigma(W_Z y_{n-1} + V_Z u_n + b_Z) \\
&\quad + \Delta t \hat{\sigma}(W_1 y_{n-1} + V_1 u_n + b_1) \odot W_Z \sigma'(W_Z y_{n-1} + V_Z u_n + b_Z)
\end{aligned}$$

A.3 Hyperpruning

A.3.1 Jacobian Matrix Derivation of RNN Architectures

Stacked LSTM

The LSTM is defined by the following equations:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$

$$c'_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot c'_t$$

$$h_t = o_t \cdot \sigma_c(c_t),$$

where σ_g and σ_c are element-wise sigmoid function and *tanh* function, respectively. For simplicity, we use the following notation when calculating the derivatives:

$$y_* = W_* x_t + U_* h_{t-1} + b_*,$$

where $*$ can be f, i, o, c , depending on the gate. The derivative of each of those gates/states with respect to the hidden states is shown below:

$$\begin{aligned}
\frac{\partial f_t}{\partial h_{t-1}} &= [\sigma_g(y_f)(1 - \sigma_g(y_f))]^T U_f \\
\frac{\partial i_t}{\partial h_{t-1}} &= [\sigma_g(y_i)(1 - \sigma_g(y_i))]^T U_i \\
\frac{\partial o_t}{\partial h_{t-1}} &= [\sigma_g(y_o)(1 - \sigma_g(y_o))]^T U_o \\
\frac{\partial c_t}{\partial h_{t-1}} &= \frac{\partial f_t}{\partial h_{t-1}} c_{t-1} + \frac{\partial i_t}{\partial h_{t-1}} \tanh(y_c) + i_t \operatorname{sech}^2(y_c) U_c \\
\frac{\partial h_t}{\partial h_{t-1}} &= \frac{\partial o_t}{\partial h_{t-1}} \tanh(c_t) + o_t \operatorname{sect}^2(c_t) \frac{c_t}{\partial h_{t-1}},
\end{aligned}$$

The derivative of each of those gates/states with respect to the inputs is shown below:

$$\begin{aligned}
\frac{\partial f_t}{\partial x_t} &= [\sigma_g(y_f)(1 - \sigma_g(y_f))]^T W_f \\
\frac{\partial i_t}{\partial x_t} &= [\sigma_g(y_i)(1 - \sigma_g(y_i))]^T W_i \\
\frac{\partial o_t}{\partial x_t} &= [\sigma_g(y_o)(1 - \sigma_g(y_o))]^T W_o \\
\frac{\partial c_t}{\partial x_t} &= \frac{\partial f_t}{\partial x_t} c_{t-1} + \frac{\partial i_t}{\partial x_t} \tanh(y_c) + i_t \operatorname{sech}^2(y_c) W_c \\
\frac{\partial h_t}{\partial x_t} &= \frac{\partial o_t}{\partial x_t} \tanh(c_t) + o_t \operatorname{sect}^2(c_t) \frac{\partial c_t}{\partial x_t},
\end{aligned}$$

RHN

An RHN layer with a recurrent depth of L is described by:

$$\begin{aligned}
h_l^{[t]} &= p_l^{[t]} \cdot e_l^{[t]} + h_{l-1}^{[t]} \cdot c_l^{[t]}, \\
l &\in [1, L],
\end{aligned}$$

where

$$p_l^{[t]} = \tanh(W_P x^{[t]} \mathbb{1}_{\{l=1\}} + R_{P_l} h_{l-1}^{[t]} + b_{P_l}) \quad (\text{A.16})$$

$$e_l^{[t]} = \sigma(W_E x^{[t]} \mathbb{1}_{\{l=1\}} + R_{E_l} h_{l-1}^{[t]} + b_{E_l}) \quad (\text{A.17})$$

$$c_l^{[t]} = \sigma(W_C x^{[t]} \mathbb{1}_{\{l=1\}} + R_{C_l} h_{l-1}^{[t]} + b_{C_l}) \quad (\text{A.18})$$

and $\mathbb{1}$ is the indicator function. The output of the RNH layer is the output of the L^{th} Highway layer, i.e., $y^{[t]} = h_L^{[t]}$. The derivative is shown below:

$$\frac{h_1^{[t]}}{h_1^{[t-1]}} = \frac{\partial h_1^{[t]}}{\partial h_L^{[t-1]}} \prod_{l=1}^{L-1} \frac{\partial h_{l+1}^{[t-1]}}{\partial h_l^{[t-1]}}$$

where

$$\frac{\partial h_{l+1}^{[t-1]}}{\partial h_l^{[t-1]}} = P_{l+1}'^{[t-1]} \text{diag}(e_{l+1}^{[t-1]}) + E_{l+1}'^{[t-1]} \text{diag}(p_{l+1}^{[t-1]}) + \text{diag}(c_{l+1}^{[t-1]}) + C_{l+1}'^{[t-1]} \text{diag}(h_l^{[t-1]}),$$

with

$$\begin{aligned} P_{l+1}'^{[t-1]} &= R_{P_{l+1}}^T \text{diag}[\tanh'(R_{P_{l+1}} h_l^{[t-1]} + b_{P_{l+1}})] \\ E_{l+1}'^{[t-1]} &= R_{E_{l+1}}^T \text{diag}[\tanh'(R_{E_{l+1}} h_l^{[t-1]} + b_{E_{l+1}})] \\ C_{l+1}'^{[t-1]} &= R_{C_{l+1}}^T \text{diag}[\tanh'(R_{C_{l+1}} h_l^{[t-1]} + b_{C_{l+1}})], \end{aligned}$$

and

$$\frac{\partial h_1^{[t]}}{\partial h_L^{[t-1]}} = P_1'^{[t]} \text{diag}(e_1^{[t]}) + E_1'^{[t]} \text{diag}(p_1^{[t]}) + \text{diag}(c_1^{[t]}) + C_1'^{[t]} \text{diag}(h_L^{[t-1]})$$

with

$$\begin{aligned} P_1'^{[t]} &= R_{P_1}^T \text{diag}[\tanh'(W_P x^{[t]} + R_{P_1} h_L^{[t-1]} + b_{P_1})] \\ E_1'^{[t]} &= R_{E_1}^T \text{diag}[\tanh'(W_E x^{[t]} + R_{E_1} h_L^{[t-1]} + b_{E_1})] \\ C_1'^{[t]} &= R_{C_1}^T \text{diag}[\tanh'(W_C x^{[t]} + R_{C_1} h_L^{[t-1]} + b_{C_1})] \end{aligned}$$

If it is coupled, $c = 1 - e$. The RHN becomes

$$h_l^{[t]} = p_l^{[t]} \cdot e_l^{[t]} + h_{l-1}^{[t]} \cdot (1 - e_l^{[t]})$$

A.3.2 Embedding Spaces

Our work employed PCA as the embedding function and L_2 distance as the distance metric. However, it is important to note that there are no special constraints on this particular embedding function and distance metric.

To thoroughly investigate the completeness of LSH, we conducted ablation studies across various embedding spaces and distance measures to comprehensively understand LSH and its performance in different scenarios.

We use the following equations to calculate the distance between the candidate network c^i in the candidate pool $\mathcal{P}$ and the reference dense network $\hat{c}$:

$$\begin{aligned} [\hat{v}, v_0^i, \dots, v_{-1}^i] &= \text{embedding}([\hat{\Lambda}, \Lambda^i]) \\ s &= \text{distance}(\hat{v}, v_{-1}^i), \end{aligned}$$

where Λ_j^i stands for the LS of the i -th candidate at the j -th epoch and Λ^i groups all LS of c^i till current epoch, i.e., $\Lambda^i \equiv \{\Lambda_j^i\}_{j=0}$. $\hat{\Lambda}$ denotes the LS of the reference network. The embedded LS of reference $\hat{c}$ and candidate c^i at the current epoch, i.e., $\hat{v}$ and v_{-1}^i , is used to calculate the distance.

We conducted an ablation study using three embedding spaces: PCA, original LS, and T-SNE, along with two distance metrics: L_2 and Cosine distance, which is defined below.

$$\begin{aligned} L_2(p, q) &= \sqrt{\sum_{i=1}^n (p_i - q_i)^2} \\ \text{Cos}(p, q) &= 1 - \frac{p \cdot q}{|p||q|} = 1 - \frac{\sum_{i=1}^n (p_i q_i)}{\sqrt{\sum_{i=1}^n (p_i)^2} \sqrt{\sum_{i=1}^n (q_i)^2}}, \end{aligned}$$

Table. A.1 presents the testing perplexities obtained by the selected network based on these various combinations of embedding functions and distance metrics. Each combination performs reasonably, demonstrating the effectiveness of using different embedding spaces and distance metrics within the LSH framework. Surprisingly, a similar performance was achieved even when using the original LS with the Cosine distance metric. This finding suggests that the LS is informative regarding the network’s performance even at the early stages of training. Among the different combinations, we observed that PCA with L2 distance, the configuration used in LSH, is preferable over other combinations. This led to the choice of the PCA embedding function combined with the L2 distance metric in our experiments.

Table A.1: Perplexities of the final model selected based on different combinations of embedding spaces and distance metrics.

Embedding	PCA		LS		T-SNE	
Distance	L2	Cos	L2	Cos	L2	Cos
Perplexity	69.9	72.99	72.56	70.3	70.5	72.6

A.4 2ndMatch

A.4.1 Mathematical Derivation

1st-order Jacobian Matching

In this section, we provide a detailed explanation of first-order Jacobian matching (JM) and its equivalence to matching noisy soft targets. The derivation follows the formulation introduced in [211], where JM is used as a regularization strategy to transfer fine-grained input–output sensitivity from a teacher model to a student model in the context of image classification.

We begin with a general function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$. The first-order Taylor expansion of f around a point $x \in \mathbb{R}^n$, within a small perturbation neighborhood $\{x + v : \|v\| < \epsilon\}$, yields

$$f(x + v) = f(x) + \nabla_x f(x)v + O(\epsilon^2). \quad (\text{A.19})$$

When f represents a neural network, this linear approximation describes how its outputs change under small input perturbations. Now consider the squared error loss between a teacher model $s_{\mathcal{D}}(x)$ and a student model $s(x)$:

$$\ell(s(x), s_{\mathcal{D}}(x)) = \|s(x) - s_{\mathcal{D}}(x)\|_2^2 \quad (\text{A.20})$$

$$= \sum_{i=1}^m (s^i(x) - s_{\mathcal{D}}^i(x))^2. \quad (\text{A.21})$$

To account for nearby perturbations, let $v = \sigma z$, where $z \in \mathbb{R}^n$ is a unit normal random variable, and $\sigma \in \mathbb{R}$ is a scaling factor. The expected loss under such perturbations becomes

$$\mathcal{L}_{\text{noisy}} = \mathbb{E}_v \left[\|s(x + v) - s_{\mathcal{D}}(x + v)\|_2^2 \right]. \quad (\text{A.22})$$

We apply the first-order Taylor expansion to both the teacher and the student

$$s_{\mathcal{D}}(x + v) = s_{\mathcal{D}}(x) + J_{\mathcal{D}}(x)v + \mathcal{O}(\sigma^2), \quad (\text{A.23})$$

$$s(x + v) = s(x) + J(x)v + \mathcal{O}(\sigma^2), \quad (\text{A.24})$$

where $J_{\mathcal{D}}, J \in \mathbb{R}^{m \times n}$ are the teacher’s and student’s Jacobians of the output with respect to the input, respectively. The loss becomes

$$\mathcal{L}_{\text{noisy}} = \mathbb{E}_v \left[\sum_{i=1}^m \left(s^i(x) + J^i(x)v - s_{\mathcal{D}}^i(x) - J_{\mathcal{D}}^i(x)v \right)^2 + \mathcal{O}(\sigma^4) \right] \quad (\text{A.25})$$

$$\begin{aligned}
&= \sum_{i=1}^m \left(s^i(x) - s_{\mathcal{D}}^i(x) \right)^2 + \mathbb{E}_v \left[\sum_{i=1}^m \left((J^i(x) - J_{\mathcal{D}}^i(x))v \right)^2 \right] \\
&+ \mathbb{E}_v \left[\sum_{i=1}^m 2 \left(s^i(x) - s_{\mathcal{D}}^i(x) \right) (J^i(x) - J_{\mathcal{D}}^i(x))v \right] + \mathcal{O}(\sigma^4).
\end{aligned} \tag{A.26}$$

The cross term in Eq. A.26 vanishes since $\mathbb{E}[v] = 0$ and the high order error $\mathcal{O}(\sigma^4)$ is ignored for small σ . For isotropic Gaussian noise $v = \sigma z$, we have $\mathbb{E}[vv^\top] = \sigma^2 I$, and thus

$$\mathcal{L}_{\text{noisy}} \approx \|s(x) - s_{\mathcal{D}}(x)\|_2^2 + \sigma^2 \|J(x) - J_{\mathcal{D}}(x)\|_F^2, \tag{A.27}$$

where $\|\cdot\|_F$ denotes the Frobenius norm.

This expansion reveals that matching the teacher with the student on noisy inputs naturally decomposes into two terms: (i) an *output-level matching loss* $\|s(x) - s_{\mathcal{D}}(x)\|_2^2$, and (ii) a *Jacobian matching loss* $\|J(x) - J_{\mathcal{D}}(x)\|_F^2$,

The Jacobian term penalizes discrepancies in local input-output sensitivity, encouraging the student to preserve the teacher’s behavior in a neighborhood around x . In this sense, first-order JM can be interpreted as the leading-order correction induced by training on noisy soft targets.

In diffusion models, the training inputs x_t are *inherently noisy*, as they are generated by a forward noising process from clean data x_0 . At each timestep t , the model is trained on corrupted samples of the form

$$x_t = \alpha_t x_0 + \sigma_t \epsilon, \quad \epsilon \sim \mathcal{N}(0, I). \tag{A.28}$$

So the standard denoising objective already matches teacher and student predictions over a broad distribution of perturbed inputs across timesteps. From the perspective of the derivation above, this repeated matching over noisy x_t implicitly *encourages* first-order sensitivity alignment, making an explicit first-order JM term potentially redundant while adding non-trivial computational cost (e.g., Jacobian–vector products). This motivates moving beyond first-order alignment to *second-order* Jacobian-based objectives that capture temporal sensitivity and long-range perturbation growth along the denoising trajectory.

To empirically verify whether explicit first-order JM provides non-redundant supervision beyond standard diffusion losses, we ablate first-order JM on both low-resolution (CIFAR-10) and high-resolution (ImageNet) settings. As summarized in Table A.2 (and Table 7.1

FID ↓	NP	NP+KD	+1st JM	+2ndM (Ours)	Dense
CIFAR-10	5.29	5.05	5.14	4.58	4.19
ImageNet	10.23	7.96	8.15	5.68	3.60

Table A.2: Loss component ablation on CIFAR-10 and ImageNet (LDM-4). Adding 1st-order JM does not improve FID over NP+KD, whereas 2ndM yields a large gain.

in the main manuscript), first-order JM does not improve over the NP+KD baseline and can slightly degrade performance. On CIFAR-10, FID changes from 5.05 (NP+KD) to 5.14 (+1st JM), whereas our second-order matching (2ndM) improves FID to 4.58. A similar trend holds on ImageNet (LDM-4): FID changes from 7.96 (NP+KD) to 8.15 (+1st JM), while 2ndM yields a substantial gain ($7.96 \rightarrow 5.68$), narrowing the gap to the dense teacher (FID 3.60). Overall, these results support our claim that first-order alignment is largely redundant in diffusion training, whereas second-order sensitivity matching provides complementary and more effective supervision.

FTLE

This section provides additional detail on the motivation and derivation of Finite Time Lyapunov Exponents (FTLE). The notation and examples are adapted from [198].

We consider a general time-dependent dynamical system defined by the Ordinary Differential Equation (ODE)

$$\frac{dx(t)}{dt} = f(x(t), t), \quad (\text{A.29})$$

where $x \in \mathbb{R}^n$ denotes the system state, and $t \in \mathbb{R}$ denotes time. The function $f : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}^n$ governs the evolution of the system and is assumed to be smooth.

Given an initial time t_0 and interval t_1 , we define the flow map $\Phi \in \mathbb{R}^n \rightarrow \mathbb{R}^n$, which

transports a point x_0 to its location at time $t_0 + t_1$, i.e., x_1 .

$$\Phi_{t_0}^{t_0+t_1} : x_0 \rightarrow x_1 \quad (\text{A.30})$$

$$= x_0 \rightarrow x_0 + \int_{t_0}^{t_0+t_1} f(x(\tau), \tau) d\tau. \quad (\text{A.31})$$

By standard existence and uniqueness results [81], this map satisfies

$$\Phi_{t_0}^{t_0}(x_0) = x_0, \quad (\text{A.32})$$

$$\Phi_{t_0}^{t_0+t_1+t_2}(x) = \Phi_{t_0+t_1}^{t_0+t_1+t_2}(\Phi_{t_0}^{t_0+t_1}(x)). \quad (\text{A.33})$$

While numerical integration of f yields particle trajectories, analyzing time-dependent systems directly can be challenging since studying the system's overall behavior requires integrating many trajectories, which is computationally expensive. FTLE offers a principled way to quantify the rate at which nearby trajectories diverge, revealing coherent structures and deformation patterns in the dynamics. FTLE is more representative of the true transport behavior in a time-dependent system than instantaneous separation criteria such as the Okubo-Weiss criterion [233, 160] since it reflects the average or accumulated deformation of the flow.

The FTLE, denoted $\Lambda_{t_0}^{t_0+t_1}(x)$, is a scalar value, characterizing how infinitesimal perturbations to a point $x \in \mathbb{R}^n$ grow under the flow map $\Phi_{t_0}^{t_0+t_1}$ over a finite time interval $[t_0, t_0 + t_1]$. Specifically, to analyze this behavior, we consider the evolution of two nearby trajectories. Let $x_0 \in \mathbb{R}^n$ be a reference point at time t_0 , and let $x'_0 = x_0 + v_0$ be a neighboring point, where v_0 is an infinitesimal perturbation. Since the flow is continuously dependent on initial conditions, the trajectory of x' will remain close to that of x , at least for a short time. After t_1 , the separation of x and x' becomes

$$v_1 = \Phi_{t_0}^{t_0+t_1}(x_0) - \Phi_{t_0}^{t_0+t_1}(x'_0). \quad (\text{A.34})$$

Applying the first-order Taylor expansion of the flow map $\Phi_{t_0}^{t_0+t_1}$ around x_0 , we obtain

$$v_1 = \frac{d\Phi_{t_0}^{t_0+t_1}(x_0)}{dx_0} v_0 + O(\|v_0\|^2). \quad (\text{A.35})$$

Since v_0 is infinitesimal, the higher-order term $O(\|v_0\|^2)$ can be neglected. Therefore, the norm of the perturbation after t_1 becomes

$$\|v_1\| \approx \left\| \frac{d\Phi_{t_0}^{t_0+t_1}(x_0)}{dx_0} v_0 \right\| \quad (\text{A.36})$$

$$= \sqrt{v_0^\top \mathcal{C} v_0}, \quad (\text{A.37})$$

where $\mathcal{C} \in \mathbb{R}^{n \times n}$ is the finite-time Cauchy-Green deformation tensor, defined as

$$\mathcal{C} = J^\top J, \quad (\text{A.38})$$

$$J = \frac{d\Phi_{t_0}^{t_0+t_1}(x_0)}{dx_0}. \quad (\text{A.39})$$

Here, $(\cdot)^\top$ denotes the matrix transpose operation. The matrix $\mathcal{C}$ captures the accumulated stretching and compression of infinitesimal vectors under the flow over the time interval t_1 . To quantify the maximum stretching, we align v_0 with the eigenvector corresponding to the largest eigenvalue $\lambda_{\max}(\mathcal{C})$,

$$\max_{v_0} \|v_1\| = \sqrt{\lambda_{\max}(\mathcal{C})} \|v_0\|. \quad (\text{A.40})$$

Consequently, the FTLE is defined as

$$\Lambda_{t_0}^{t_0+t_1}(x) = \frac{1}{t_1} \ln \sqrt{\lambda_{\max}(\mathcal{C})}, \quad (\text{A.41})$$

The FTLE quantifies the local exponential rate of trajectory separation and has been commonly used to identify coherent structures and transport barriers in dynamical systems [78].

FTLE for Discrete-Time Diffusion Denoising Dynamics

Diffusion sampling is naturally a *discrete-time* dynamical system: each denoising step maps a latent (or image) state at step k to the next state $k+1$. This section reformulates FTLE in that setting using the same flow-map and Cauchy-Green notation as in Sec. A.4.1.

Discrete denoising dynamics and flow map. Let a diffusion sampler define an iterative update

$$x_{k+1} = \Phi_k(x_k; \theta), \quad k = 0, 1, \dots, K-1, \quad (\text{A.42})$$

where $x_k \in \mathbb{R}^n$ is the state at step k (e.g., latent/image), θ are model parameters, and Φ_k denotes one denoising update (e.g., DDPM/DDIM/Euler step) at the corresponding noise level. For an initial step k_0 and horizon m , define the m -step flow map

$$\Phi_{k_0:k_0+m}(x_{k_0}; \theta) \Phi_{k_0+m-1}(\cdot; \theta) \circ \cdots \circ \Phi_{k_0}(\cdot; \theta)(x_{k_0}), \quad (\text{A.43})$$

which transports x_{k_0} to x_{k_0+m} . It satisfies the composition property

$$\Phi_{k_0:k_0}(x; \theta) = x, \quad (\text{A.44})$$

$$\Phi_{k_0:k_0+m_1+m_2}(x; \theta) = \Phi_{k_0+m_1:k_0+m_1+m_2}(\Phi_{k_0:k_0+m_1}(x; \theta); \theta). \quad (\text{A.45})$$

Consider two nearby initial states at step k_0 : $x'_{k_0} = x_{k_0} + v_{k_0}$ with an infinitesimal perturbation v_{k_0} . After m denoising steps,

$$v_{k_0+m} = \Phi_{k_0:k_0+m}(x_{k_0} + v_{k_0}; \theta) - \Phi_{k_0:k_0+m}(x_{k_0}; \theta). \quad (\text{A.46})$$

Linearizing the m -step flow map around x_{k_0} yields

$$v_{k_0+m} \approx J_{k_0:k_0+m}(x_{k_0}; \theta) v_{k_0}, \quad (\text{A.47})$$

where the m -step Jacobian is

$$\begin{aligned} J_{k_0:k_0+m}(x_{k_0}; \theta) &:= \frac{\partial \Phi_{k_0:k_0+m}(x_{k_0}; \theta)}{\partial x_{k_0}} \\ &= \prod_{i=0}^{m-1} J_{k_0+i}(x_{k_0+i}; \theta), \end{aligned} \quad (\text{A.48})$$

$$J_k(x_k; \theta) := \frac{\partial \Phi_k(x_k; \theta)}{\partial x_k}. \quad (\text{A.49})$$

The product in (A.48) follows from the chain rule, since the overall sensitivity from x_{k_0} to x_{k_0+m} accumulates multiplicatively across intermediate denoising steps.

The finite-time Cauchy–Green deformation for diffusion flow is defined as

$$C_{k_0}^{k_0+m}(x_{k_0}; \theta) := J_{k_0:k_0+m}(x_{k_0}; \theta)^\top J_{k_0:k_0+m}(x_{k_0}; \theta). \quad (\text{A.50})$$

Then the perturbation norm after m steps satisfies

$$\|v_{k_0+m}\| \approx \|J_{k_0:k_0+m} v_{k_0}\| = \sqrt{\langle v_{k_0}, \mathcal{C}_{k_0}^{k_0+m} v_{k_0} \rangle}. \quad (\text{A.51})$$

The maximal stretching over all unit perturbations is

$$\max_{\|v_{k_0}\|=1} \|v_{k_0+m}\| \approx \sqrt{\lambda_{\max}(\mathcal{C}_{k_0}^{k_0+m}(x_{k_0}; \theta))}. \quad (\text{A.52})$$

The discrete FTLE at state x_{k_0} over m denoising steps is defined as the average exponential separation rate per step:

$$\Lambda_{k_0}^{k_0+m}(x_{k_0}; \theta) := \frac{1}{m} \ln \sqrt{\lambda_{\max}(\mathcal{C}_{k_0}^{k_0+m}(x_{k_0}; \theta))}. \quad (\text{A.53})$$

Relationship between single-step and multi-step Jacobian mismatch. Directly matching the full multi-step tensor $\mathcal{C}_{k_0}^{k_0+m}$ is expensive, as it requires Jacobian products across m steps. In practice, we use a single-step matching by setting $m=1$, for which

$$\mathcal{C}_k^{k+1}(x_k; \theta) = J_k(x_k; \theta)^\top J_k(x_k; \theta), \quad (\text{A.54})$$

This motivates our focus on matching second-order sensitivity statistics derived from $J_k^\top J_k$ at individual denoising steps, which empirically captures much of the relevant local stretching behavior while remaining tractable.

To further justify the single-step matching, consider a teacher ($s_{\mathcal{D}}$) and a student (s) dynamics with Jacobians $\{J_{k+i}^{\mathcal{D}}\}$ and $\{J_{k+i}\}$, respectively. Using the product-difference identity, the m -step Jacobian mismatch can be decomposed as

$$J_{k:k+m} - J_{k:k+m}^{\mathcal{D}} = \sum_{j=0}^{m-1} \left(\prod_{i=j+1}^{m-1} J_{k+i} \right) (J_{k+j} - J_{k+j}^{\mathcal{D}}) \left(\prod_{i=0}^{j-1} J_{k+i}^{\mathcal{D}} \right). \quad (\text{A.55})$$

Taking an operator norm and applying $\|AB\| \leq \|A\| \|B\|$ yields

$$\begin{aligned} \|J_{k:k+m} - J_{k:k+m}^{\mathcal{D}}\| &\leq \sum_{j=0}^{m-1} \left(\left\| \prod_{i=j+1}^{m-1} J_{k+i} \right\| \cdot \left\| J_{k+j} - J_{k+j}^{\mathcal{D}} \right\| \cdot \left\| \prod_{i=0}^{j-1} J_{k+i}^{\mathcal{D}} \right\| \right). \end{aligned} \quad (\text{A.56})$$

The multi-step Jacobian mismatch is upper bounded by a weighted sum of per-step mismatches, with weights determined by intermediate Jacobian products. Consequently,

# Steps	GPU Memory ↓	Training Time ↓	FID ↓
1	34 GB	18 hr	4.58
2	80 GB ($\times 2.35$)	40 hr ($\times 2.2$)	4.62 (+0.04)
3	120 GB ($\times 3.53$)	60 hr ($\times 3.3$)	4.55 (-0.03)

Table A.3: Cost–accuracy trade-off of multi-step sensitivity matching on CIFAR-10. Multi-step variants achieve FID comparable to that of the single-step objective but require substantially more memory and training time.

some steps may be more influential than others for the accumulated multi-step sensitivity mismatch. A step-wise matching objective, by contrast, promotes local alignment across all steps more uniformly, rather than allowing the overall mismatch to be dominated by only a few influential steps. In this sense, step-wise matching imposes a stronger constraint than matching only the composed multi-step behavior.

Empirical cost–accuracy trade-off (CIFAR-10). We additionally tested explicit multi-step variants. As shown in Table A.3, multi-step matching achieves *comparable* FID to the single-step objective but with substantially higher compute and memory costs that scale roughly linearly with the number of matched steps. Given the strong cost scaling and limited gains, we adopt the step-wise matching in the main method.

Table A.4: **Key training and architectural hyperparameters for DDPM, LDM-4 (Cin256-v2), and Stable Diffusion 1.4.**

Dataset	Model	Res.	Batch	LR	Linear β	T	Channels
CIFAR-10	DDPM	32^2	128	2×10^{-4}	0.0001 $\rightarrow$ 0.02	1000	[128, 256, 256, 256]
LSUN	DDPM	256^2	64	2×10^{-5}	0.0001 $\rightarrow$ 0.02	1000	[128, 128, 256, 256, 512, 512]
ImageNet	LDM-4	256^2 (latent 64^2)	128	2.56×10^{-4}	0.0015 $\rightarrow$ 0.0195	1000	[192, 384, 576, 960]
MS-COCO	SD v1.4	512^2 (latent 64^2)	256	5×10^{-5}	0.00085 $\rightarrow$ 0.012	1000	[320, 640, 1280, 1280]

Table A.5: **Key training and architectural hyperparameters for U-ViT-Small on CIFAR-10 and CelebA.**

Hyperparameter	CIFAR-10	CelebA
Resolution	32^2	64^2
Batch	128	128
Learning rate	2×10^{-4}	2×10^{-4}
Warmup	2500	5000
T	1000	1000
Embed dim	512	512
Patch	2	4
Depth	12	12
Heads	8	8
MLP ratio	4	4

A.4.2 FTLE Computation for Diffusion Models

For the FTLE results reported in Table 2 of the main manuscript, we evaluate the temporal sensitivity of each model after finetuning is completed. All models—NP, NP+KD, +1st JM, +2ndM, and the dense teacher—are evaluated using the same fixed batch of 128 noise samples at CIFAR-10 resolution ($3 \times 32 \times 32$). These shared inputs ensure that differences in FTLE arise solely from differences in the models rather than the initial conditions.

We use a DDIM sampler with 100 denoising steps for generating trajectories. However, computing the full Jacobian at each of these steps is infeasible even for CIFAR-10-sized models: backpropagating through the UNet across many consecutive timesteps builds up large computation graphs and quickly exhausts GPU memory. Due to these constraints, we compute Jacobians only at the first 10 DDIM timesteps—the maximum number our hardware can support without running out of memory. Using early-stage timesteps is also meaningful, as these steps exhibit the highest sensitivity and contribute most strongly to perturbation growth.

While 2ndM relies on random-projection-based second-order Jacobian supervision dur-

ing training—a necessity because obtaining full Jacobians at every step of training would be prohibitively expensive—our FTLE analysis uses the entire Jacobian matrix. This distinction is important: while random projections provide a scalable approximation for learning, FTLE requires the exact deformation tensor $J_t^T J_t$, which can only be computed from the full Jacobian. Thus, the FTLE numbers in Table 2 reflect the true sensitivity of each model and are not affected by the random-projection approximation used during training.

A.4.3 *Experimental Setup Details*

Package version

We adopt Diff-Pruning [61] as our primary pruning method. Since Diff-Pruning relies on *torch-pruning* library [60], we use version 1.5.2 for all Transformer-based experiments. This version includes full support for structural pruning in Transformer architectures. For U-Net-based diffusion models, however, we use the earlier 1.2.5 release. The newer versions modify the pruning behavior of U-Net blocks, resulting in different parameter and MAC counts compared to the original Diff-Pruning implementation. To ensure strict comparability with prior work and to maintain consistency across baselines, we retain the original pruning setup for all U-Net experiments.

Hyperparameters

We provide detailed architectural and training hyperparameters for both U-Net-based and Transformer-based (U-ViT) diffusion models used in our experiments.

U-Net Models. For U-Net architectures, we follow widely adopted configurations from prior diffusion model literature [61, 92, 183, 111]. These settings are applied across CIFAR-10, LSUN, ImageNet, and MS-COCO. Table A.4 reports the key hyperparameters for DDPM, LDM-4 (CIN256-v2), and Stable Diffusion v1.4, including input resolution, batch size, learning rate, linear β schedule, total number of diffusion steps, and channel widths.

Transformer-Based Models. For Transformer-based diffusion architectures, we adopt the U-ViT design [11] and evaluate it on CIFAR-10 and CelebA. Table A.5 summarizes both training hyperparameters (optimizer, learning rate schedule, warmup, batch size, number of

Table A.6: Training cost comparison for UNet-based diffusion models on CIFAR-10, LSUN, ImageNet and MSCOCO.

Dataset	Method	Batch	Memory	Time/Iter	Steps	Time
CIFAR10	Baseline	128	8 G	0.16 s	100k	4.4 hr
	KD	128	16 G	0.30 s	100k	8.3 hr
	2ndM	128	34 G	0.65 s	100k	18.1 hr
LSUN	Baseline	64	8×19 G	0.52 s	500k	72.2 hr
	KD	64	8×37 G	1.01 s	500k	140.3 hr
	2ndM	64	8×72 G	2.82 s	500k	391.7 hr
ImageNet	Baseline (DP)	128	8×13 G	0.34 s	200k	19.3 hr
	KD	128	8×26 G	0.53 s	200k	29.4 hr
	2ndM	128	8×45 G	0.99 s	200k	56.1 hr
MSCOCO	Baseline (Base)	256	8×43 G	2.17s	50k	30.2 hr
	Base + KD	256	8×65 G	3.40s	50k	47.2 hr
	Base + 2ndM	256	8×77 G	7.06s	50k	98.1 hr
	Baseline (Small)	256	8×37 G	1.29s	50k	17.8 hr
	Small + KD	256	8×55 G	2.08s	50k	28.9 hr
	Small + 2ndM	256	8×72 G	6.76s	50k	94.0 hr
	Baseline (Tiny)	256	8×37 G	1.29s	50k	18.0 hr
	Tiny + KD	256	8×55 G	2.08s	50k	28.9 hr
	Tiny + 2ndM	256	8×72 G	6.76s	50k	94.0 hr

Table A.7: Training cost comparison for Transformer-based diffusion models on CIFAR-10 and CelebA

Dataset	Method	Batch	Memory	Time / Step	Steps	Time
CIFAR10	Baseline	128	11 G	0.13 s	100k	3.6 hr
	2ndM	128	47 G	0.67 s	100k	18.6 hr
CelebA	Baseline	128	11 G	0.43 s	100k	11.9 hr
	2ndM	128	47 G	0.67 s	100k	18.6 hr

diffusion steps) and architectural details (patch size, embedding dimension, network depth, number of attention heads, and MLP ratio). Following common practice, we train both datasets for 100k steps rather than the full 500k-step schedule used in the original U-ViT paper. This shorter schedule is sufficient because pruned models are initialized from a pretrained dense checkpoint and therefore converge substantially faster than models trained from scratch [201], while still enabling fair comparison across finetuning methods.

A.4.4 Computational Cost Analysis

We report the training cost of our proposed 2ndM finetuning framework compared to the Diff-Pruning baseline across both U-Net-based and Transformer-based diffusion models. Tables A.6 and A.7 summarize batch sizes, GPU memory usage, per-iteration latency, training steps, and total wall-clock training time under matched pruning settings.

For U-Net models (Table A.6), we observe that 2ndM introduces a $4\text{--}5\times$ increase in per-iteration training time due to the Jacobian-vector product (JVP) computations required by second-order matching. This overhead translates directly into longer training duration: on CIFAR-10 (pruning ratio 0.44), training time increases from 4.4 to 18.1 hours; on LSUN-Church (pruning ratio 0.30, trained on 8 A100 GPUs), it increases from 72.2 to 391.7 hours. A similar pattern appears on ImageNet and MS-COCO.

In addition to the higher compute cost, 2ndM incurs a substantial memory overhead. Memory usage increases from 8G to 34G on CIFAR-10 and from $8 \times 19\text{G}$ to $8 \times 72\text{G}$ on

LSUN-Church, largely due to storing intermediate activations for JVPs and maintaining parallel teacher–student computation graphs. This overhead scales primarily with model size rather than input resolution. For MS-COCO, although the input resolution is 512^2 , the diffusion model operates in a 64^2 latent space; the dominant factor is the significantly larger U-Net backbone, which makes Jacobian computation particularly memory intensive. Consequently, the per-GPU batch size must be reduced to 8, and we employ `accumulate_grad = 4` to match the baseline’s effective batch size of 32 per GPU. Gradient accumulation preserves training stability but increases wall-clock time proportionally, explaining the larger training-time gap between 2ndM and the baseline on MS-COCO.

Transformer-based models (Table A.7) show a slightly different trend. The U-ViT architecture used for both CIFAR-10 and CelebA is identical in size, so increasing the image resolution slows down both baseline and 2ndM proportionally. As a result, the relative time gap narrows compared to U-Net models: training on CIFAR-10 increases from 3.6 to 18.6 hours, whereas on CelebA it increases from 11.9 to 18.6 hours. The higher resolution of CelebA increases the baseline cost, reducing the relative overhead introduced by second-order JM.

Overall, the increased training time and memory footprint stem from computing the second-order JM loss. While **2ndM** provides consistent improvements in generative quality, its resource demands may limit applicability in constrained settings. Importantly, this additional cost arises *only during training*: **2ndM** does not modify the model architecture or sampling procedure and therefore introduces *no overhead at inference time*.

A.4.5 Effect of KD Strength

We examine whether performance gains are solely attributable to stronger knowledge distillation (KD) supervision by sweeping the KD weight λ_{KD} and comparing KD-only training against KD augmented with 2ndM. Table A.8 shows that for every tested λ_{KD} , adding 2ndM consistently improves FID. For instance, at $\lambda_{\text{KD}}=1.0$, FID improves from 5.03 (KD only) to 4.58 (KD+2ndM). These results indicate that the improvement is not merely due to increasing the KD loss weight; rather, 2ndM provides complementary supervision beyond

λ_{KD}	0.01	0.1	1.0	10.0
FID (KD only) ↓	5.06	5.15	5.03	5.24
FID (KD + 2ndM) ↓	4.61	4.69	4.58	4.67

Table A.8: Sweep over KD weight λ_{KD} (with $\lambda_{\text{NP}}=1$). Adding 2ndM consistently improves FID across all KD strengths.

KD.

A.4.6 Jacobian Loss Weights vs. FID

2ndM introduces a hybrid finetuning objective composed of three terms:

$$\mathcal{L}_{\text{total}} = \lambda_{\text{NP}}\mathcal{L}_{\text{NP}} + \lambda_{\text{KD}}\mathcal{L}_{\text{KD}} + \lambda_{\text{2nd-Jac}}\mathcal{L}_{\text{2nd-Jac}} \quad (\text{A.57})$$

The relative weight of each component is a tunable hyperparameter. Because exhaustive hyperparameter searches are computationally expensive for large models and high-resolution datasets, we conduct systematic weight sweeps only on CIFAR-10 (U-Net and U-ViT) and CelebA (U-ViT). For all remaining experiments—including LSUN, ImageNet, and MS-COCO—we fix $\lambda_{\text{kd}} = 1.0$ and $\lambda_{\text{2nd-Jac}} = 10^{-3}$, which we find to be robust across architectures and datasets.

We analyze how the Jacobian loss weight affects generative performance under different pruning ratios. Experiments are conducted on both U-Net and Transformer-based diffusion models across CIFAR-10 and CelebA.

U-Net

As shown in Table A.9, the Jacobian loss weight has a noticeable impact on FID. Since the Jacobian term is intended to act as a relatively small regularizer on top of the stronger output-level KD supervision, we focus on comparatively small weights for high pruning ratios. Empirically, moderate pruning (e.g., $\text{PR} = 0.44$) can benefit from a relatively larger Jacobian weight, yielding substantial gains in FID, whereas more aggressively pruned models

favor smaller weights. In these high-pruning regimes, a weaker second-order signal is sufficient to guide the student toward the teacher’s sensitivity while avoiding over-constraining a model with limited capacity. Overall, the U-Net results indicate that the Jacobian term should remain significantly smaller than the KD term, and that the preferred weight gradually decreases as the pruning ratio becomes more aggressive.

Table A.9: Jacobian matching loss weight vs. FID on CIFAR-10 for the UNet-based diffusion model. A weight of 0 corresponds to the Diff-Pruning baseline (no Jacobian supervision).

CIFAR-10 32×32						
PR	FID ($\downarrow$) for different weights					
Weight	0	1e-4	1e-3	1e-2	1e-1	1
0	4.19	–	–	–	–	–
0.44	5.29	–	–	4.66	4.58	4.93
0.56	6.49	–	5.85	5.54	5.66	–
0.70	8.85	7.35	7.09	7.16	–	–
0.82	13.42	11.04	10.91	11.08	–	–

U-ViT

We observe similar behavior for the U-ViT backbone on CIFAR-10 and CelebA (Tables A.10 and A.11). Across both datasets, small to moderate Jacobian weights consistently improve FID over the Diff-Pruning baseline, with the most pruned models again favoring the smallest weights. This pattern aligns with the U-Net experiments: when pruning is mild, a stronger Jacobian regularizer can be beneficial, but under heavy pruning, the optimal setting shifts toward smaller values that complement, rather than dominate, the KD objective. Taken together, these studies support the use of $\lambda_{2\text{nd-Jac}} = 10^{-3}$ as a practical default in our large-scale experiments, reflecting the role of the Jacobian term as a light but effective second-order correction on top of the primary KD and noise-prediction losses.

Table A.10: Jacobian matching loss weight vs. FID on CIFAR-10 for the Transformer-based diffusion model. A weight of 0 corresponds to the Diff-Pruning baseline (no Jacobian supervision).

CIFAR-10 32×32						
PR	FID ($\downarrow$) for different weights					
Weight	0	5e-4	1e-3	1e-2	5e-2	1e-1
0	3.11	–	–	–	–	–
0.5	3.53	–	3.56	3.51	–	4.47
0.7	4.63	–	4.13	4.05	5.01	6.04
0.8	6.68	5.07	4.98	5.22	–	–

A.4.7 Further Potential Improvement

Optimally Time-Dependent (OTD) Random Projection

Random projections are widely used to approximate Jacobian behavior in neural network training due to their computational efficiency [209]. However, standard random vectors treat all directions uniformly, without regard to their dynamical relevance. In diffusion models, certain directions—particularly those aligned with unstable modes of the Jacobian—can dominate local dynamics and introduce significant variance. These directions are often associated with large FTLE values, indicating strong local instability and high sensitivity to input perturbations.

Because diffusion models evolve over many iterative steps, the dominant unstable directions can vary over time. To capture this time-varying structure, we explore *Optimally Time-Dependent* (OTD) modes, originally developed for reduced-order modeling in dynamical systems [7, 6]. OTD evolves a low-dimensional basis that tracks transiently unstable directions along a trajectory. Given the Jacobian J , the OTD basis $V = [v_1, v_2, \dots, v_r] \in \mathbb{R}^{n \times r}$

Table A.11: Jacobian matching loss weight vs. FID on CelebA for the Transformer-based diffusion model. A weight of 0 corresponds to the Diff-Pruning baseline (no Jacobian supervision).

CelebA (64×64)						
PR	FID ($\downarrow$) for different weights					
Weight	0	1e-3	5e-3	1e-2	1e-1	5e-1
dense	2.87	–	–	–	–	–
0.5	3.25	–	–	3.23	3.21	–
0.7	3.57	3.32	3.33	3.30	3.36	3.46
0.8	4.61	4.10	4.01	4.11	–	–

Table A.12: **FID results under different OTD hyperparameter settings.** Each column corresponds to a combination of the integration step size (Δt), the number of OTD update steps (*step*), and the probability of applying OTD (*Prob*). The rightmost column reports the FID obtained without OTD, shown for reference.

Δt	0.01						0.1						-
step	1			2			1			2			-
Prob	0.2	0.5	1.0	0.2	0.5	1.0	0.2	0.5	1.0	0.2	0.5	1.0	-
FID	4.66	4.62	4.71	4.63	4.63	4.66	4.64	4.62	4.67	4.63	4.62	4.67	4.58

evolves according to

$$\frac{\partial V}{\partial t} = JV - VV^T JV. \quad (\text{A.58})$$

In our setting, we adopt the simplest case with a single dominant mode ($r = 1$) to control computational cost. We initialize the basis vector $v \in \mathbb{R}^n$ by sampling from a standard

normal distribution and normalizing to unit norm:

$$v_1 \sim N(0, I), \hat{v}_1 \leftarrow \frac{v_1}{\|v_1\|} \quad (\text{A.59})$$

We then evolve $\hat{v}_1$ over time using the Forward Euler method to numerically integrate the OTD via Eq. A.58.

$$\hat{v}_1 = \hat{v}_1 + \Delta t (J\hat{v}_1 - \hat{v}_1 \hat{v}_1^\top J\hat{v}_1), \quad (\text{A.60})$$

where J is the Jacobian at the current timestep and Δt is the integration step size. This update yields a time-adaptive direction $\hat{v}$ that tracks the most unstable local direction at each step.

We then replace the standard random projection used in our second-order JM with this OTD-enhanced direction,

$$\mathcal{L}_{\text{2nd-Jac}} = \|\hat{v}^\top J^\top J\hat{v} - \hat{v}^\top J_{\mathcal{D}}^\top J_{\mathcal{D}}\hat{v}\|_2^2, \quad (\text{A.61})$$

where J and $J_{\mathcal{D}}$ denote the Jacobians of the pruned and dense (teacher) models, respectively. By aligning projections with the most dynamically relevant direction, this loss provides supervision that is both time-adaptive and informed by the local stability properties of the denoising dynamics. Intuitively, the shared OTD direction encourages the pruned model to match not only the teacher’s outputs but also the teacher’s most sensitive response direction, which could, in principle, lead to faster convergence and improved generative quality.

Despite this theoretical motivation, our empirical results in Table A.12 show that OTD-based projections do not improve FID under our current training setup. Across a range of hyperparameter choices—including different integration step sizes (Δt), the number of OTD updates per training iteration (*step*), and the probability of applying OTD (*Prob*)—the resulting FID scores remain comparable to, or slightly worse than, our standard random-projection baseline. This suggests that while OTD effectively identifies transiently unstable modes of the Jacobian, this additional focus on dynamically dominant modes does not directly translate into perceptual gains in our experiments.

Nonetheless, we view this direction as promising. Replacing uniform random projections with more principled, dynamically informed projection schemes may become beneficial when

combined with more accurate integration methods, adaptive schedules over training, or multiple tracked modes ($r > 1$). We leave a more systematic exploration of such OTD-based and other structured projection strategies for future work.

A.4.8 Limitation

While our finetuning framework provides an effective way to inject Jacobian-based supervision into pruned diffusion models, several limitations remain, highlighting opportunities for future improvement.

Computational Overhead from Jacobian Calculations. Our method introduces additional training cost due to the need to compute Jacobian-vector products, see Table. A.6 and A.7. While we employ random projections to reduce the dimensionality of Jacobian matching, the additional backward passes still increase training time and GPU memory usage—especially in high-resolution settings or with large backbone models. This overhead may limit the method’s accessibility to groups with substantial computational resources and constrain its scalability to even larger diffusion architectures.

Sensitivity to the Jacobian Loss Weight. Performance depends on choosing an appropriate coefficient for Jacobian loss, especially across different pruning ratios (Tables A.9, A.10, A.11). While moderate pruning often benefits from stronger Jacobian supervision, higher pruning ratios tend to require smaller weights to avoid overregularization. Identifying an optimal weight may therefore require multiple training runs, adding additional tuning cost. In practice, we find that a default Jacobian weight of 10^{-3} works reliably across most datasets and architectures that we work with, but further study is needed to develop more adaptive or self-tuning strategies.

Dependence on Dense Teacher Quality. As with distillation-based finetuning, our approach assumes access to a strong and stable dense teacher. If the teacher is weak or exhibits training instabilities, the student is inherently bounded by the teacher and may inherit suboptimal behaviors or instability patterns through both output-level distillation

and Jacobian-based supervision. While our experiments use well performing dense teachers, a systematic study of distillation under weak or unstable teachers (and how to perform Jacobian supervision in such settings) is beyond the scope of this work.

Table A.13: Relative FID (rFID) for selected ImageNet classes, sorted by class index.

Class ID	Class Name	DP rFID ↓	Jac rFID ↓
22	bald eagle	12.30	7.14
207	golden retriever	14.15	5.70
292	tiger	20.59	3.01
388	giant panda	18.27	7.41
393	anemone fish	20.47	4.71
654	minibus	11.80	7.67
718	pier	16.75	4.23
976	promontory	18.65	8.56
979	valley	18.91	9.30

A.4.9 Additional Quantitative Results on ImageNet

To better understand class-level behavior on ImageNet, Table A.13 reports the relative FID for a selection of representative classes, sorted by class index. In all cases, our Jacobian-based finetuning yields substantially lower relative FID compared to Diff-Pruning, indicating closer alignment with the dense teacher model. The improvements span diverse semantic categories—including animals, vehicles, and natural scenes—suggesting that the benefit of Jacobian supervision generalizes across heterogeneous visual concepts.

A.4.10 Visualization of Generated Images

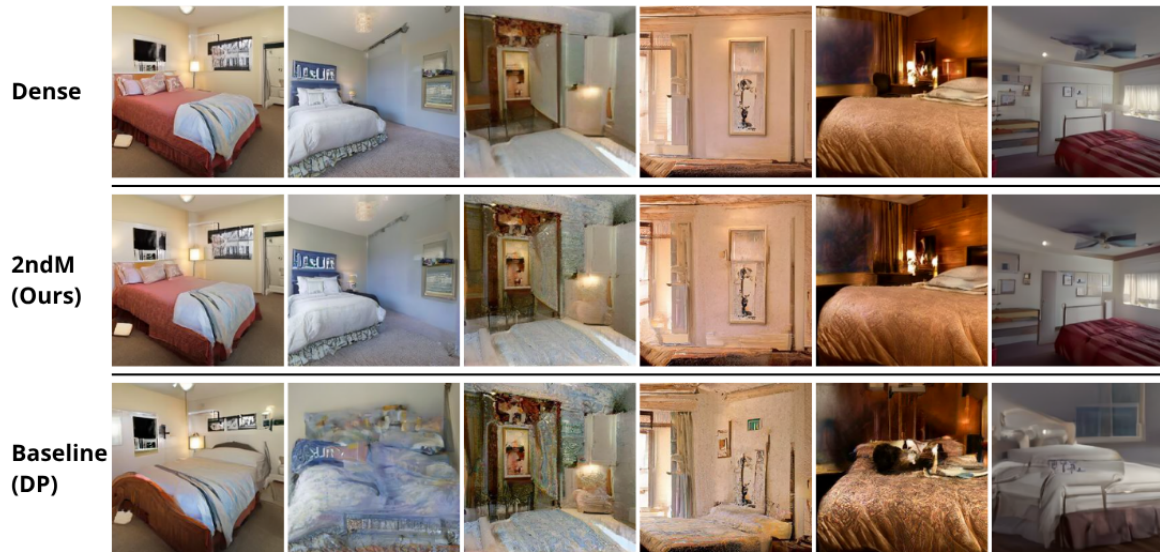
We provide additional qualitative comparisons between the baseline and **2ndM** across all datasets evaluated in this work. Representative samples for LSUN-Church and LSUN-



Figure A.1: Visual comparison of ImageNet samples from dense models (top), 2ndM (middle) and baseline Diff-Pruning (bottom).

Bedroom are shown in Figure A.2, while ImageNet, MS-COCO, CIFAR-10, and CelebA results are shown in Figures A.1, A.3, A.4, and A.5, respectively.

LSUN-Bedroom



LSUN-Church



Figure A.2: Visual comparison of LSUN-Bedroom and LSUN-Church samples from dense models (top), 2ndM (middle) and baseline Diff-Pruning (bottom).

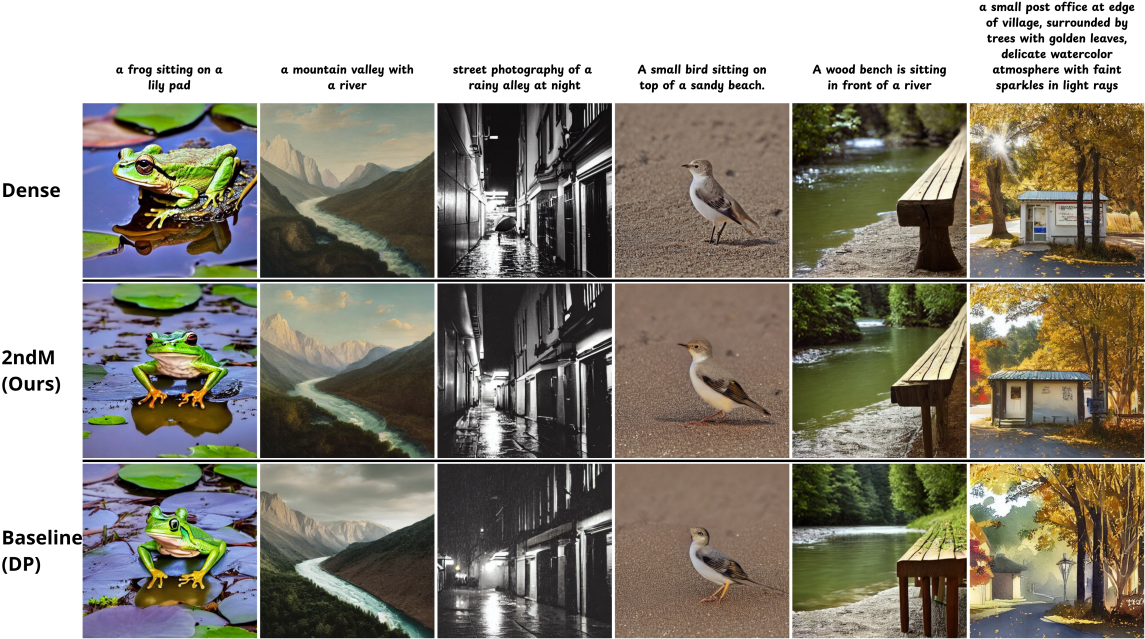


Figure A.3: Visual comparison of MSCOCO samples from dense models (top), 2ndM (middle) and baseline Diff-Pruning (bottom).

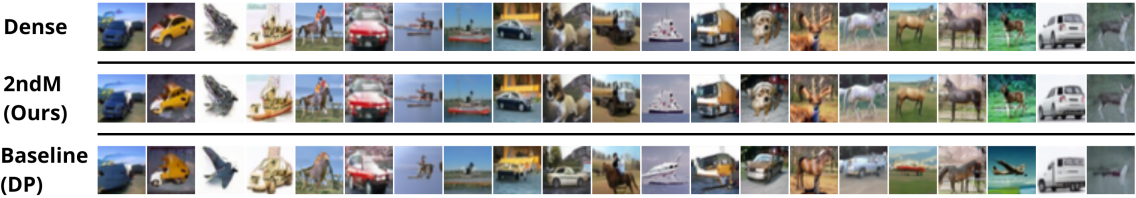


Figure A.4: Visual comparison of CIFAR-10 samples from dense models (top), 2ndM (middle) and baseline Diff-Pruning (bottom).

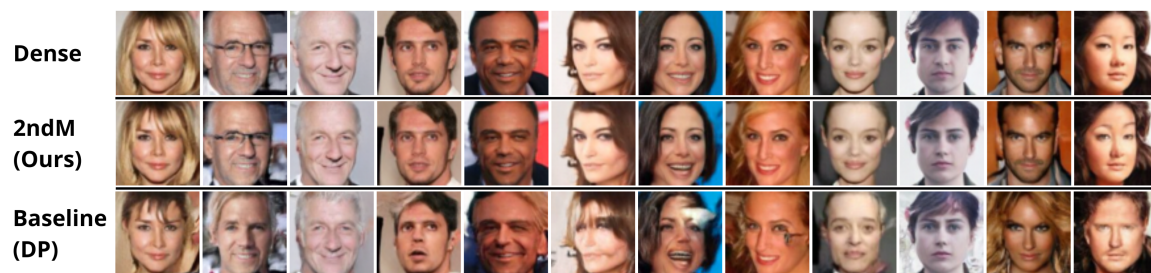


Figure A.5: Visual comparison of CelebA samples from dense models (top), 2ndM (middle) and baseline Diff-Pruning (bottom).