WHO SAID NEURAL NETWORKS AREN'T LINEAR?

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural networks are famously nonlinear. However, linearity is defined relative to a pair of vector spaces, $f{:}\mathcal{X}{\to}\mathcal{Y}$. Leveraging the algebraic concept of transport of structure, we propose a method to explicitly construct non-standard vector spaces where a neural network acts as a linear operator. ~~Is it possible to identify a pair of non-standard vector spaces for which a conventionally nonlinear function is, in fact, linear? This paper introduces a method that makes such vector spaces explicit by construction.~~ We find that if we sandwich a linear operator A between two invertible neural networks, $f(x) = g_y^{-1}(Ag_x(x))$, then the corresponding vector spaces $\mathcal{X}$ and $\mathcal{Y}$ are induced by newly defined addition and scaling actions derived from g_x and g_y. We term this kind of architecture a Linearizer. This framework makes the entire arsenal of linear algebra, including SVD, pseudo-inverse, orthogonal projection and more, applicable to nonlinear mappings. Furthermore, we show that the composition of two Linearizers that share a neural network is also a Linearizer. We leverage this property and demonstrate that training diffusion models using our architecture makes the hundreds of sampling steps collapse into a single step. We further utilize our framework to enforce idempotency (i.e. $f(f(x)) = f(x)$) on networks leading to a globally projective generative model and to demonstrate modular style transfer.

1 INTRODUCTION

Linearity occupies a privileged position in mathematics, physics, and engineering. Linear systems admit a rich and elegant theory: they can be decomposed through eigenvalue and singular value analysis, inverted or pseudo-inverted with well-understood stability guarantees, and manipulated compositionally without loss of structure. These properties are not only aesthetically pleasing, they underpin the computational efficiency of countless algorithms in signal processing, control theory, and scientific computing. Crucially, repeated application of linear operators simplifies rather than complicates: iteration reduces to powers of eigenvalues, continuous evolution is captured by the exponential of an operator, and composition preserves linearity (Strang, 2022a).

In contrast, non-linear systems, while more expressive, often defy such a structure. Iterating non-linear mappings can quickly lead to intractable dynamics; inversion may be ill-posed or undefined; and even simple compositional questions lack closed-form answers (Strogatz, 2024). Neural networks, the dominant modeling tool in modern machine learning, are famously nonlinear, placing their analysis and manipulation outside the reach of classical linear algebra (Hornik et al., 1989). As a result, tasks that are trivial in the linear setting, such as projecting onto a subspace, enforcing idempotency, or collapsing iterative procedures, become major challenges in the nonlinear regime that require engineered loss functions and optimization schemes.

~~This raises a natural and fundamental question: can we reinterpret conventionally nonlinear neural networks as linear operators in disguise?~~ This motivates our approach: applying the principle of conjugation (change of coordinates) to deep neural networks to induce a latent space where the mapping becomes linear. If so, we would gain access to the full arsenal of linear methods without sacrificing the expressive power of nonlinear function classes.

In this paper, we propose a framework that precisely achieves this goal. By embedding linear operators between two invertible neural networks, we induce new vector space structures under which the overall mapping is linear. We call such architectures *Linearizers*. This perspective not only offers a new lens on neural networks, but also enables powerful applications: collapsing hundreds

of diffusion sampling steps into one, enforcing structural properties such as idempotency, and more. In short, Linearizers provide a bridge between the expressive flexibility of nonlinear models and the analytical tractability of linear algebra.

2 Linearizer Framework

Linearity is not an absolute concept; it is a property defined relative to a pair of vector spaces, $f : \mathcal{X} \to \mathcal{Y}$. This begs the question: is it possible to identify a pair of spaces $\mathcal{X}$ and $\mathcal{Y}$, for which a function that is nonlinear over standard Euclidean space is, in fact, perfectly linear? To answer this, we must first deconstruct what defines a vector space. It is a structure comprised of a set of vectors (e.g. $\mathbb{R}^N$), a field of scalars (e.g., $\mathbb{R}$), and two fundamental operations: vector addition ($+$) and scalar multiplication ($\cdot$). We propose that while keeping the set of vectors and the field unchanged, we can redefine the operations in order to induce new vector spaces.

2.1 Definitions

We introduce a formalism we term as *Linearizer*, in which the relevant vector spaces are immediately identifiable by construction. The architecture we propose can be trained from scratch or by distillation from an existing model. Our approach is made practical by invertible neural networks Rezende & Mohamed (2015); Dinh et al. (2015; 2017). We build our model by wrapping a linear operator, a matrix A, between two such invertible networks, g_x and g_y:

Definition 1 (Linearizer)**.** *Let $\mathcal{X}, \mathcal{Y}$ be two spaces and $g_x : \mathcal{X} \to \mathcal{X}$, $g_y : \mathcal{Y} \to \mathcal{Y}$ be two corresponding invertible functions. Also, let $A : \mathcal{X} \to \mathcal{Y}$ be a linear operator. Then we define the Linearizer $\mathbb{L}_{\{g_x, g_y, A\}}$ as the following function $f : \mathcal{X} \to \mathcal{Y}$:*

$$f(x) = \mathbb{L}_{\{g_x, g_y, A\}}(x) = g_y^{-1}(A g_x(x)) \tag{1}$$

For this construction, we can define the corresponding vector spaces $\mathcal{X}$ and $\mathcal{Y}$, also shown in Figure 1:

Definition 2 (Induced Vector Space Operations)**.** *Let $g : V \to V$ be an invertible function. We define a new set of operations, $\oplus$ and $\odot$, for any two vectors $v_1, v_2 \in V$ and any scalar $a \in \mathbb{R}$:*

$$v_1 \oplus_g v_2 := g^{-1}(g(v_1) + g(v_2)) \tag{2}$$

$$a \odot_g v_1 := g^{-1}(a \cdot g(v_1)) \tag{3}$$

This definition follows the standard definition of transport of structure (or induced algebraic structure, Mac Lane & Birkhoff (1999)), where the operations $\oplus_g$ and $\odot_g$ are defined via the bijection g. We also note that only certain invertible maps g linearize a task; in practice, g_x, g_y, and the core A are *learned jointly* end-to-end[1] .

2.2 Linearity

The input space $\mathcal{X}$ is defined by operations $(\oplus_x, \odot_x)$ induced by g_x, and the output space $\mathcal{Y}$ by $(\oplus_y, \odot_y)$ induced by g_y.

Lemma 1. *$(V, \oplus_g, \odot_g)$ is a vector space over $\mathbb{R}$. Proof.* See Appendix. A.

Lemma 2. *The function $f(x)$ is a linear map from the vector space $\mathcal{X}$ to the vector space $\mathcal{Y}$.*

Proof. To prove linearity, we show that f satisfies the superposition principle.

$$\begin{aligned} f(a_1 \odot_x x_1 \oplus_x a_2 \odot_x x_2) &= g_y^{-1}(A g_x(g_x^{-1}(a_1 g_x(x_1) + a_2 g_x(x_2)))) \\ &= g_y^{-1}(a_1 g_y(g_y^{-1}(A g_x(x_1))) + a_2 g_y(g_y^{-1}(A g_x(x_2)))) \\ &= a_1 \odot_y f(x_1) \oplus_y a_2 \odot_y f(x_2) \end{aligned}$$

□

[1] Specifically, in our implementation each of g_x and g_y uses **6** invertible blocks (Appendix G.1). The specific INN architecture – affine coupling layers with a tiny U-Net conditioner – is detailed in Appendix G.2 and Appendix G.4. The linear core A is task-specific and its implementations are summarized in Appendix G.3.

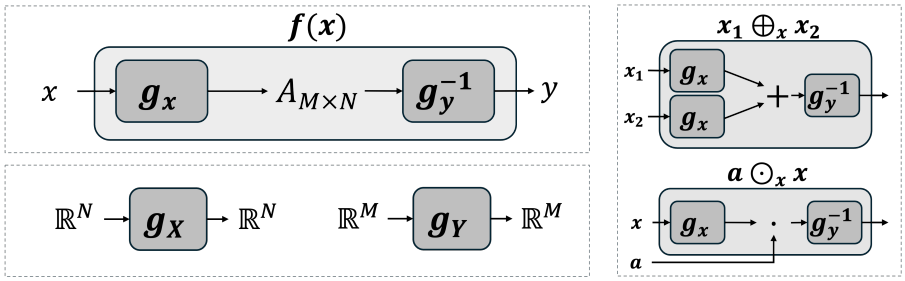

Figure 1: **Left.** The Linearizer structure (top) is a linear operation sandwiched between two invertible functions (bottom) **Right.** Vector addition and scalar multiplication define induced vector spaces for which f is linear.

2.3 Intuition

The Linearizer can be understood through several intuitive lenses.

Linear algebra analogy. The Linearizer is analogous to an eigendecomposition or SVD. Just as a matrix is diagonal (i.e. simplest) in its eigenbasis, our function f is a simple matrix multiplication in the "linear basis" defined by g_x and g_y. This provides a natural coordinate system in which to analyze and manipulate the function. For example, repeated application of a Linearizer with a shared basis ($g_x = g_y = g$) is equivalent to taking a power of its core matrix:

$$\mathbb{L}^{\circ N}_{\{g,g,A\}}(x) = \underbrace{f(f(\cdots f}_{N \text{ times}}(x))) = g^{-1}(A^N \cdot g(x)) = \mathbb{L}_{\{g,g,A^N\}}(x) \tag{4}$$

Geometric interpretation. g_x can be seen as a diffeomorphism. Equip the input with the pullback metric $g_x^*\langle\cdot,\cdot\rangle_{\mathbb{R}^N}$ induced by the Euclidean metric in latent space. With respect to this metric, straight lines in latent correspond to geodesics in input, and linear interpolation in the induced coordinates maps to smooth and semantically coherent curves in input space.

2.4 Properties

We review how basic properties of linear transforms are expressed by Linearizers. The following lemmas demonstrate how standard linear algebraic properties Axler (2015); Strang (2022b) are preserved under the learned change of coordinates. While these results follow directly from the vector space isomorphism established by g, we provide individual independent proofs and explicitly derive them here. Practically, we use neural networks, that are continuous and smooth and so are their inverses. All of the properties, with the exception of Lemma. 4 hold without this assumption.

2.4.1 *Composition*

Lemma 3. *The composition of two Linearizer functions with compatible spaces $f_1 : \mathcal{X} \to \mathcal{Y}$ and $f_2 : \mathcal{Y} \to \mathcal{Z}$, is also a Linearizer.*

Proof. $$(f_2 \circ f_1)(x) = g_z^{-1}(A_2 g_y(g_y^{-1}(A_1 g_x(x)))) = g_z^{-1}((A_2 A_1) \cdot g_x(x)) \quad \square \tag{5}$$

2.4.2 *Inner Product and Hilbert Space*

Definition 3 (Induced Inner Product)**.** *Given an invertible map $g : V \to \mathbb{R}^N$, the induced inner product on V is*

$$\langle v_1, v_2\rangle_g := \langle g(v_1), g(v_2)\rangle_{\mathbb{R}^N}, \tag{6}$$

where the right-hand side is the standard Euclidean dot product.

Equipped with this inner product, the induced vector spaces make *Hilbert spaces*.

Lemma 4 (Induced spaces are Hilbert)**.** *Let $g : V \to \mathbb{R}^n$ be a smooth bijection and endow V with the induced vector space operations $(\oplus_g, \odot_g)$ and inner product*

$$\langle u, v\rangle_g \;:=\; \langle g(u),\, g(v)\rangle_{\mathbb{R}^n}. \tag{7}$$

Then $(V, \langle\cdot,\cdot\rangle_g)$ is a Hilbert space. Proof. See Appendix C

2.4.3 *Transpose*

Lemma 5 (Transpose)**.** *Let $f(x) = g_y^{-1}(Ag_x(x))$ be a Linearizer. Its transpose $f^\top : \mathcal{Y} \to \mathcal{X}$ with respect to the induced inner products is*

$$f^\top(y) = g_x^{-1}\big(A^\top g_y(y)\big). \tag{8}$$

Proof. For all $x \in \mathcal{X}, y \in \mathcal{Y}$,

$$\langle f(x), y\rangle_{g_y} = \langle Ag_x(x), g_y(y)\rangle_{\mathbb{R}^N} = \langle g_x(x), A^\top g_y(y)\rangle_{\mathbb{R}^N} \tag{9}$$

$$= \langle x, g_x^{-1}(A^\top g_y(y))\rangle_{g_x} = \langle x, f^\top(y)\rangle_{g_x}. \quad \square \tag{10}$$

2.4.4 *Singular Value Decomposition*

Lemma 6 (SVD of a Linearizer)**.** *Let $A = U\Sigma V^\top$ be the singular value decomposition of A. Then the SVD of $f(x) = g_y^{-1}(Ag_x(x))$ is given by singular values Σ, input singular vectors $\tilde{v}_i = g_x^{-1}(v_i)$, and output singular vectors $\tilde{u}_i = g_y^{-1}(u_i)$.*

Proof. See Appendix B. $\square$

3 Applications

Having established the theoretical foundations, we now present proof-of-concept applications that highlight advantages of our framework, leaving scaling and engineering refinements for future work.

3.1 One-Step Flow Matching

Flow matching (Lipman et al., 2023; Song et al., 2021b) trains a network to predict the velocity field that transports noise samples to data points. In our setting, diffusion models Ho et al. (2020) and flow matching Lipman et al. (2023) are, from an algorithmic standpoint, equivalent under standard assumptions Gao et al. (2024); hence we use the two frameworks interchangeably. Traditionally, this velocity must be integrated over time with many small steps, making generation slow. We train flow matching using our Linearizer as the back bone, with a single vector space ($g_x = g_y = g$). The key property that helps us achieve one-step generation is the closure of linear operators. Applying the model sequentially with some simple linear operations in between steps is a chain of linear operations. This chain can be collapsed to a single linear operator so that the entire trajectory is realized in a single step.

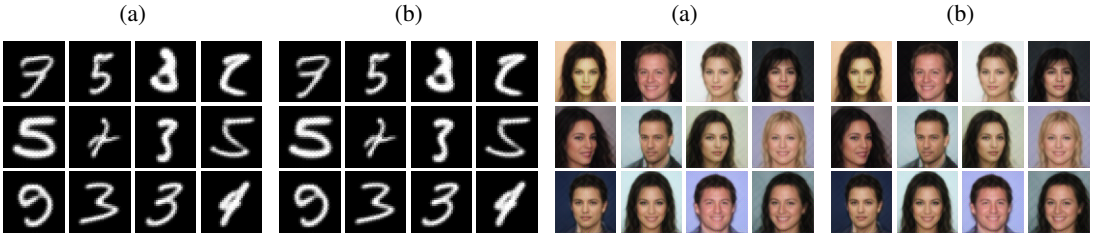

Figure 2: Comparison between multi-step and one-step flow matching. Panel labels: **(a)** multi-step Linear FM. **(b)** one-step Linear flow matching (FM);

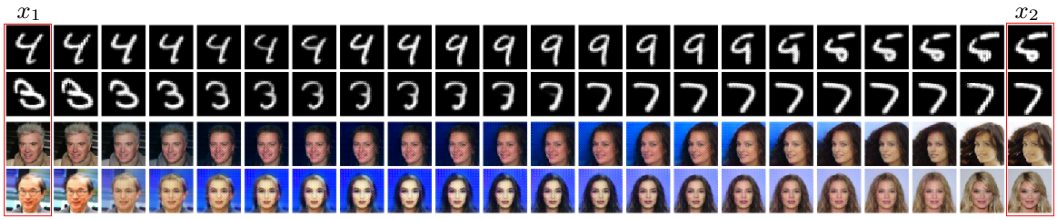

Figure 3: **One-step inversion examples:** Left and right (in red): original (not generated) data x_1 and x_2. Intermediate images obtained by latent interpolation. See Appendix. E.2 for more and higher-resolution results.

Training. Training is just as standard FM, only using the Linearizer model and the induced spaces. We define the forward diffusion process:

$$x_t = (1-t) \odot x_0 \oplus t \odot x_1 = g^{-1}\big((1-t)\, g(x_0) + t\, g(x_1)\big). \tag{11}$$

This is a straight line in the g-space, mapped back to a curve in the data space. The target velocity is

$$v = x_1 \ominus x_0 = g^{-1}\big(g(x_1) - g(x_0)\big). \tag{12}$$

We parameterize a time-dependent Linearizer

$$f(x_t, t) = g^{-1}(A_t g(x_t)), \tag{13}$$

and train it to predict v. The loss is

$$\mathcal{L} = \mathbb{E}_{x_0, x_1, t} \left\| v \ominus f(x_t, t) \right\|^2 = \mathbb{E}_{x_0, x_1, t} \left\| g^{-1}\Big(g(x_1) - g(x_0) - A_t g(x_t)\Big) \right\|^2. \tag{14}$$

This objective ensures that the learned operators A_t approximate the true velocity of the flow within the latent space of the Linearizer.

Sampling as a collapsed operator. In inference time, standard practice is to discretize the ODE with many steps. In the induced space, an Euler update reads

$$x_{t+\Delta t} = x_t \oplus \big(\Delta t \odot f(x_t, t)\big), \tag{15}$$

which expands to

$$g(x_{t+\Delta t}) = (I + \Delta t\, A_t)\, g(x_t). \tag{16}$$

Iterating this N times produces

$$g(\hat{x}_1) \approx \underbrace{\left[\prod_{i=0}^{N-1} (I + \Delta t\, A_{t_i})\right]}_{:=B} g(x_0). \tag{17}$$

The entire product can be collected into a single operator B, resulting in

$$\hat{x}_1 = g^{-1}(B g(x_0)). \tag{18}$$

Thus, what was originally a long sequence of updates collapses into a single multiplication. In practice, higher-order integration such as Runge-Kutta (Runge, 1895; Kutta, 1901) may be used to calculate B more accurately (see Appendix F). B is calculated only once, after training. Then generation requires only a single feedforward activation of our model on noise, regardless of the number of steps discretizing the ODE.

Geometric intuition. g acts as a diffeomorphism: in its latent space, the path between x_0 and x_1 is a straight line, and the velocity is constant. When mapped back to the data space via g^{-1}, this straight line becomes a curved trajectory. The Linearizer exploits this change of coordinates so that, in the right space, the velocity field is trivial, and the expensive integration disappears.

Implementation. The operator A is produced by an MLP whose input is t. To avoid huge matrices multiplications we build A as a low-rank matrix, by multiplying two rectangular matrices. g does not take t as input. Full implementation details are provided in App. G.

One-step generation. Figure 2 shows qualitative results on MNIST LeCun et al. (1998) (32×32) and CelebA Liu et al. (2015) (64×64). As discussed, our framework supports both multi-step and one-step sampling and yields effectively identical outputs. Visually, the samples are indistinguishable across datasets; quantitatively, the mean squared error between one-step and multi-step generations is $\mathbf{3.0 \times 10^{-4}}$, confirming their near-equivalence in both datasets. Additionally, we evaluate FID across different step counts and our one-step simulation (see Figure 4 (a)). The FID from *100 full iterative steps* closely matches our *one-step* formulation, empirically validating the method's theoretical guarantee of one-step sampling. Moreover, simulating more steps (1000→1 vs. 100→1) improves FID by ∼8 points, highlighting the strength of the linear-operator formulation. Furthermore, we validate one-step vs. 100 step fidelity, showing high similarity as presented in Figure.4(c). Finally, while our absolute FID is not yet competitive with state-of-the-art systems, our goal here is to demonstrate the theory in practice rather than to exhaustively engineer for peak performance; scaling and optimization are left to future work.

Inversion and interpolation. A fundamental limitation of flow models is that, in contrast to VAEs (Kingma & Welling, 2014), they lack a natural encoder: they cannot map data back into the prior (noise) space, and thus act only as decoders. As a result, inversion methods for diffusion (Dhariwal & Nichol, 2021; Song et al., 2021a; Huberman-Spiegelglas et al., 2024) have become an active research area. However, these techniques are approximate and often suffer from reconstruction errors, nonuniqueness, or computational overhead. Our framework offers a new bridge between diffusion-based models and encoding ability.

Because the Linearizer is linear, its Moore–Penrose pseudoinverse is also a Linearizer:

Lemma 7 (Moore–Penrose pseudoinverse of a Linearizer)**.** *The Moore–Penrose pseudoinverse of f with respect to the induced inner products is*

$$f^{\dagger}(y) = g_x^{-1}(A^{\dagger} g_y(y)) \tag{19}$$

Proof. We verify the four Penrose equations (Penrose, 1955) in Appendix D. □

This property enables the exact encoding of data in the latent space. For example, given two data points x_1, x_2, we can encode them via $z^a = (1-a)\, f^{\dagger}(x_1) \;+\; a\, f^{\dagger}(x_2)$, and decode back by $\hat{x}^a = f(z^a)$. Fig 3 shows latent interpolation between two real (non-generated) images. Additionally, we evaluate reconstruction quality using two standard metrics: LPIPS Zhang et al. (2018) and PSNR. Figure 4(b) shows inversion-reconstruction consistency. our results confirm that the proposed approach preserves information with high quality.

3.2 Modular Style Transfer

Style transfer has been widely studied since Gatys et al. (2016) proposed optimizing image pixels to match style and content statistics. Johnson et al. (2016) introduced perceptual-loss training of feed-forward networks, making style transfer practical.

Linearizer formulation. We fix g_x and g_y and associate each style with a matrix A_{style}. Analogously to A_t, the matrix A_{style} is produced by a hypernetwork that takes a *style index* as input (rather than time, as in the one-step setting). Then

$$f_{\text{style}}(x) = g_y^{-1}(A_{\text{style}} g_x(x)). \tag{20}$$

This separates content representation from style, making styles modular and algebraically manipulable. In practice, we distill A_{style} from a pretrained Johnson-style network.

Full Steps			**One-steps**	
1	10	100	(100→1)	(1000→1)
405	152	127	131	123

(a) FID comparison.

	PSNR	**LPIPS**
MNIST	31.6	.008
CelebA	33.4	.006

(b) Inverse reconstruct consistency.

	PSNR	**LPIPS**
MNIST	32.4	.006
CelebA	32.9	.007

(c) 100 vs. 1 step fidelity.

Figure 4: Quantitative comparisons.

Figure 5: **Style transfer examples.** Left: original image. Middle: style transfer using the left-side and right-side style images. Right: interpolation between the two styles.

Style interpolation. Given two trained style operators, A^x_{style} and A^y_{style}, we form an interpolated operator $A^{(\alpha x+(1-\alpha)y)}_{\text{style}}$ that represents a linear interpolation between them. We evaluated $\alpha \in \{0,\ 0.35,\ 0.40,\ 0.45,\ 0.50,\ 0.55,\ 0.60,\ 0.65,\ 1\}$ and computed

$$f^{(\alpha x+(1-\alpha)y)}_{\text{style}}(x) = g_y^{-1}\left(A^{(\alpha x+(1-\alpha)y)}_{\text{style}}\, g_x(x)\right),$$

with results shown in Figure 5. In the first row, we interpolate between the *mosaic* and *candy* styles; in the second row between *rain-princess* and *udine*; and in the third row between *candy* and *rain-princess*. See App.E for additional transfers.

3.3 Linear idempotent generative network

Idempotency, $f(f(x)) = f(x)$, is a central concept in algebra and functional analysis. In machine learning, it has been used in Idempotent Generative Networks (IGNs) (Shocher et al., 2024), to create a projective generative model. It was also used for robust test time training (Durasov et al., 2025). Enforcing a network to be idempotent is tricky and is currently done by using sophisticated optimization methods such as (Jensen & Vicary, 2025). The result is only an approximately idempotent model over the training data. We demonstrate enforcing accurate idempotency through architecture using our Linearizer. The key observation is that idempotency is preserved between the inner matrix A and the full function f.

Lemma 8. *The function f is idempotent $\iff$ The matrix A is idempotent.*

Proof. For all x,

$$\begin{aligned} A^2 = A \quad &\iff \quad g^{-1}\big(A^2 g(x)\big) = g^{-1}\big(Ag(x)\big) \\ &\iff \quad \big\{\, g^{-1} \circ A \circ \underbrace{(g \circ g^{-1})}_{\text{Id}} \circ A \circ g \,\big\}(x) = \big\{\, g^{-1} \circ A \circ g \,\big\}(x) \\ &\iff \quad f(f(x)) = f(x) \quad \text{(by the definition of } f\text{)} \quad \square \end{aligned}$$

Method. Figure 6 left shows the method. Recall that idempotent (projection) matrices have eigenvalues that are either 0 or 1. We assume a diagonalizable projection matrix $A = Q\Lambda Q^{-1}$ where Λ is a diagonal matrix with entries $\{0, 1\}$. The matrices Q, Q^{-1} can be absorbed into g, g^{-1} without loss of expressivity. So we can train a Linearizer $g^{-1}(\Lambda g(\cdot))$. In order to have Λ that is binary yet differentiable, we use straight-through estimation (Bengio et al., 2013) having underlying probabilities P

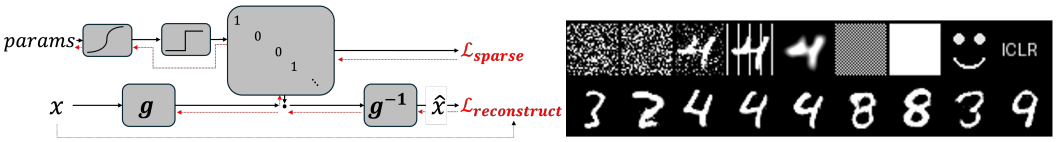

Figure 6: **Left: IGN training diagram.** Black solid arrows denote the forward pass; the red dashed arrow shows backpropagation. The parameter logits are passed through a sigmoid and then thresholded to form the binary projector A; during backprop, the straight–through estimator (STE) bypasses the threshold so gradients flow through the underlying probabilities. **Right: Projection results.** Our Linear IGN makes a global projector that projects any input to the target distribution. Top are inputs and bottom are matching outputs.

as parameters in $[0, 1]$ and then use $A = round(P) + P - P.detach()$ where $detach$ means stopping gradients. We forward propagate rounded values (0 or 1) and back propagate continuous values.

Idempotency by architecture allows reducing the idempotent loss used in Shocher et al. (2024). We need the data to be fixed points with the tightest possible latent manifold. The losses are:

$$\mathcal{L}_{rec} = ||f(x) - x||^2 \tag{21}$$

$$\mathcal{L}_{sparse} = \frac{1}{N} Rank(A) = \frac{1}{N} \sum_{i}^{N} \lambda_i \tag{22}$$

We further enforce regularization that encourages g to preserve the norms.

$$\mathcal{L}_{isometry} = \Big\| \|g(x) - g(0)\|^2 - \|x\|^2 \Big\|_1 \tag{23}$$

This nudges g towards a near–isometry around the data, thus resisting mapping far-apart inputs to close outputs. It mitigates collapse and improves stability. We apply it with a small weight (0.001).

From local to global projectors. As Shocher et al. (2024) put it, *"We view this work as a first step towards a global projector."* In practice, IGNs achieve idempotency only around the training data: near the source distribution or near the target distribution they are trained to reproduce. If you venture further away, then the outputs degenerate into arbitrary artifacts not related to the data distribution. In contrast, our Linearizer is idempotent by architecture. It does not need to be trained to approximate projection, it is one. Figure 6 right shows various inputs projected by our model onto the distribution. Interestingly, there is not even a notion of a separate *source distribution*: we never inject latent noise during training, and in a precise sense the entire ambient space serves as the source. This makes our construction a particularly unusual generative model and a natural next step toward the global projector envisioned in Shocher et al. (2024).

3.4 Additional Experiments

In this section, we broadened our experimental scope with **three** additional, diverse small tasks: face attribute classification, disentanglement distillation, and weather prediction. These experiments further strengthening the empirical evidence for the Linearizer's applicability and robustness.

CelebFaces Attributes (CelebA). We evaluate on the CelebA face-attribute classification task, which includes large pose variation, background clutter, diverse identities, and rich annotations (40 binary attributes such as *Male*, *Eyeglasses*, *Blond Hair*). After training, our Linearizer attains a test accuracy of **87.6%**; for reference, a ResNet-50 baseline reaches **86.1%**. These results suggest that the Linearizer handles moderately complex, real-world visual variability.

Distillation of nonlinear disentanglement models. In this experiment, we distill the encoder of a pretrained disentanglement model into a Linearizer (using pretrained checkpoints). We consider VAE, FactorVAE, and β-VAE on the **dSprites** dataset, which provides ground-truth factors of variation. We assess quality with standard metrics: **FactorVAE**, **SAP**, and **DCI** (disentanglement, completeness, informativeness). Across models, the Linearizer closely matches the original encoders (rows prefixed with *Linearizer-*) on these metrics (often slightly improving FactorVAE/SAP, e.g., BetaH: +0.10/+0.01), indicating that it can capture highly nonlinear transformations with no loss in disentanglement quality of the encoded representations. For the table with full results please see Appendix E.3.

Weather Prediction. To further explore the applicability of our framework, we conduct a small experiment in a new domain: regression of future steps. We use a standard weather-prediction benchmark where the model must predict the future given 96 past time steps. We compare our method with two well-known baselines tailored for regression, **Autoformer** Wu et al. (2021) and **PatchTST** Nie (2022). Briefly, although our method is generic, it shows competitive results, surpassing the highly nonlinear Autoformer and being comparative (e.g., gaps of 0.008–0.034 MSE depending on horizon) with the strong PatchTST baseline. For the table with full results please see Appendix E.3.

4 Analysis and Discussion

4.1 Expressiveness and Limitations

A natural question is whether any arbitrary function can be represented by a Linearizer. The answer is no; the architecture imposes certain topological constraints. For example, the null space of f is completely determined by the null space of the matrix A. As linear operators can only have null spaces containing a single point (the zero vector) or an infinite subspace, f is restricted to these behaviors. This prevents a Linearizer from, for example, mapping three distinct points to zero.

However, for many practical machine learning tasks, we empirically find that this theoretical limitation is less restrictive than it appears. High-dimensional data, such as images or text embeddings, often reside on a low-dimensional manifold granting the network a large degree of freedom for mapping off-manifold inputs. We hypothesize that in such cases most of the expressiveness constraints may not impact the task for the relevant data. Specifically, regarding the null space example, a finite number of on-manifold inputs can be mapped to zero, while infinitely many more off-manifold.

4.2 On the Role of the Core Operator A

One might also ask whether the matrix A is strictly necessary or if its action could be fully absorbed by the powerful invertible networks g_x and g_y. Let the SVD of A be $U\Sigma V^T$. In the general case where g_x and g_y are distinct, we could define new invertible networks $g'_y = g_y \circ U$ and $g'_x = V^T \circ g_x$. The function would then be $f(x) = (g'_y)^{-1}(\Sigma g'_x(x))$. If we further allow the networks to absorb the scaling defined by the singular values in Σ, then the core operator could be reduced to a diagonal matrix of zeros and ones (note that this does not imply idempotency unless $g_x = g_y$). In this sense, the fundamental role of A is simply to define the rank of the function f.

The situation is different in the constrained case where $g_x = g_y = g$. Here, the two networks are not independent. To preserve the structure $f(x) = g^{-1}(Ag(x))$, any transformation absorbed by g must be inverted by g^{-1}. This rules out g absorbing SVD elements U, V as they are not necessarily mutual inverses. For diagonalizable matrices however, Eigen-decomposition can be applied. Then, $A = Q\Lambda Q^{-1}$, where one could define $g' = Q^{-1} \circ g$ and $f(x) = (g')^{-1}(\Lambda g'(x))$. In this scenario, A cannot be reduced to a projection matrix, its role is to determine the eigenvalues of the function, a much stronger responsibility than just its rank. However, this is limited to diagonalizable matrices.

In practice, parameterizing an explicit and expressive A is needed in some cases, particularly when we wish to represent a family of functions (e.g., $A_1, A_2 \dots$) that share the same linear basis (g_x, g_y). The expressiveness of A allows more degrees of freedom for the diversity of different functions. For example, in our diffusion application, we learn a time-dependent operator A_t. A single pair of networks, g_x and g_y, defines a fixed basis, while explicit A_t allows for a rich variety of linear operations within that basis, controlled by the continuous parameter t.

5 Limitations

While the Linearizer introduces a principled framework for exact linearization of neural maps, several limitations remain. First, invertible networks are inherently more challenging to train than standard architectures, often requiring careful design. Second, this work represents a broad and general first step: our applications demonstrate feasibility, but none are yet scaled to state-of-the-art benchmarks. Finally, the precise expressivity of the Linearizer remains an open theoretical question.

6 Related Work

Linearization in Dynamical Systems: The Koopman operator theory linearizes nonlinear dynamics by advancing observables on a single state space, yielding linear evolution $z_{t+1} = Kz_t$ in discrete time or $\dot{z} = Az$ in continuous time (Mezić, 2005). Data-driven realizations such as DMD and EDMD construct finite-dimensional approximations of this infinite-dimensional operator on a chosen dictionary of observables, which typically produces a square matrix because the dictionary is mapped into itself, although variants with different feature sets exist (Schmid, 2010; Lusch et al., 2018; Mardt et al., 2018). Linear algebraic analyses such as SVD, eigendecomposition, and taking powers are standard in these Koopman approximations. Our work addresses a different object: an arbitrary learned mapping $f : \mathcal{X} \to \mathcal{Y}$. We learn invertible coordinate maps g_x and g_y and a finite matrix A such that $g_y \circ f \circ g_x^{-1} = A$, which gives the exact finite-dimensional linearity of f between

induced spaces. This formulation allows for distinct input and output coordinates, allows A to be rectangular when $\dim \mathcal{X} \neq \dim \mathcal{Y}$, and transfers linear-algebraic structure in A directly to f in a controlled way. In particular, one can compute $\mathrm{SVD}(f)$ via $\mathrm{SVD}(A)$, perform analytic test-time inversion with A^{+}, impose spectral constraints such as idempotency by restricting $\sigma(A)$, and more.

Neural Tangent Kernel: The NTK framework shows that infinitely wide networks trained with small steps evolve linearly in parameter space (Jacot et al., 2018). However, the resulting model remains nonlinear in input, output mapping. In comparison, our contribution achieves input-output linearity in a learned basis for any network width, independent of training dynamics.

One-Step Diffusion Distillation: Reducing sampling cost in diffusion models often uses student distillation: Progressive Distillation (Salimans & Ho, 2022), Consistency Models (Song et al., 2023), Distribution Matching Distillation (DMD) and f-distillation fall into this category. A recent work, the Koopman Distillation Model (KDM) (Berman et al., 2025), uses Koopman-based encoding to distill a diffusion model into a one-step generator, achieving strong FID improvements through offline distillation. The Koopman Distillation Model (KDM) distills a pre-trained diffusion teacher into an encoder linear decoder for diffusion only. We train from scratch, enforce exact finite-dimensional linearity by construction, and target multiple use cases including composition, inversion, idempotency, and continuous evolution. Unlike some invertible Koopman setups that force a shared map, KDM employs distinct encoder components and a decoder and does not enforce $g_x = g_y$.

Invertible Neural Networks: Invertible models like NICE (Dinh et al., 2015), RealNVP (Dinh et al., 2017) and Glow (Kingma & Dhariwal, 2018) use invertible transforms for density modeling and generative sampling. In this work, we make use of such invertible neural networks, to impose a bijective change of coordinates that allows linearity (lemma 2 only holds given this invertability of g_x, g_y) rather than modeling distribution.

Group-theoretic and Equivariant Representations: Let G act on $\mathcal{X}$ via $(g, x) \mapsto g \cdot x$. Many works learn an encoder $e : \mathcal{X} \to V$ and a linear representation $\rho : G \to \mathrm{GL}(V)$ such that the equivariance constraint $e(g \cdot x) = \rho(g)\, e(x)$ holds approximately, often with a decoder d that reconstructs x or enforces $d(\rho(g)z) \approx g \cdot d(z)$ (Quessard et al., 2020). The objective is to make the *action* of a symmetry group linear in latent coordinates. In contrast, we target a different object: an *arbitrary learned map* $f : \mathcal{X} \to \mathcal{Y}$. We construct invertible g_x, g_y and a finite matrix A so that $g_y \circ f \circ g_x^{-1} = A$, which yields *exact* finite-dimensional linearity of f between induced spaces and enables direct linear algebra on f via A.

Manifold Flattening and Diffeomorphic Autoencoders: Manifold-flattening methods assume data lie on a manifold $M \subset \mathbb{R}^N$ and learn or construct a map $\Phi : \mathbb{R}^N \to \mathbb{R}^k$ so that $\Phi(M)$ is close to a linear subspace L, often with approximate isometry on M (Psenka et al., 2024). Diffeomorphic autoencoders parameterize deformations $\varphi \in \mathrm{Diff}(\Omega)$ with a latent z in a Lie algebra and use a decoder to warp a template, with variants using a log map to linearize the deformation composition (Bône et al., 2019). These approaches *flatten data geometry or deformation fields*. This contrasts with our Linearizer in which the flattening is over a mapping function rather than a dataset manifold or a deformation group.

Deep Linear Networks: Networks composed solely of linear layers are linear in the Euclidean basis and are primarily used to study optimization paths (Saxe et al., 2014; Arora et al., 2018; Cohen et al., 2016). They lack expressive power in standard coordinates. In contrast, our Linearizer is expressive thanks to g_x and g_y that are nonlinear over the standard vector spaces, yet maintains exact linearity in the induced coordinate system.

Normalizing flows: Normalizing flows Rezende & Mohamed (2015) use invertible neural networks to transport probability measures: they map a simple Gaussian distribution into a complex data distribution while preserving density through the change-of-variables formula. Linearizers use the same building block, an invertible network, but instead of transporting measures, they transport algebraic structure. In this sense, Linearizers may be viewed as the "linear-algebraic analogue" of normalizing flows: both frameworks exploit invertible transformations to pull back complex objects into domains where they become simple and tractable.

7 CONCLUSION

We introduced the Linearizer, a framework that learns invertible coordinate transformations such that neural networks become exact linear operators in the induced space. This yields new applications in one-step flow matching, modular style transfer, and architectural enforcement of idempotency.

Looking ahead, several directions stand out. First, scaling one-step flow matching and IGN to larger datasets and higher resolutions could provide competitive generative models with unprecedented efficiency. Second, the matrix structure of the Linearizer opens the door to modeling motion dynamics: by exploiting matrix exponentials, one could simulate continuous-time evolution directly, extending the approach beyond generation to physical and temporal modeling. More broadly, characterizing the theoretical limits of Linearizer expressivity, and integrating it with other operator-learning frameworks, remains an exciting avenue for future research.

REPRODUCIBILITY STATEMENT

Full source code and trained models will be released publicly upon acceptance. All theoretical claims are stated with complete proofs or derivations in the Appendix (see Appendices A, B, C, D, F), and implementation details of the architectures and training procedures are described in Appendix G. Experimental setups, including loss weights, backbones, and evaluation metrics (FID, PSNR, LPIPS), are specified in Section 3. Datasets (MNIST and CelebA) are standard and used without modification, as noted in Section 3. Together, these details are intended to enable exact reproduction of both theoretical and empirical results.

REFERENCES

Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pp. 244–253. PMLR, 2018.

Sheldon Axler. *Linear Algebra Done Right*. Springer, 3rd edition, 2015.

Yoshua Bengio, Nicolas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR, Workshop Track)*, 2013.

Nimrod Berman, Ilan Naiman, Moshe Eliasof, Hedi Zisling, and Omri Azencot. One-step offline distillation of diffusion-based models via koopman modeling. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*. PMLR, 2025.

Alexandre Bône, Maxime Louis, Olivier Colliot, and Stanley Durrleman. Learning low-dimensional representations of shape data sets with diffeomorphic autoencoders. In *Information Processing in Medical Imaging (IPMI)*, volume 11492 of *Lecture Notes in Computer Science*, pp. 195–207. Springer, 2019.

Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. In *Proceedings of the 29th Annual Conference on Learning Theory (COLT)*, 2016.

Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, 2021.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR, Workshop Track)*, 2015.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

Nikita Durasov, Assaf Shocher, Doruk Oner, Gal Chechik, Alexei A. Efros, and Pascal Fua. IT^3: Idempotent test-time training. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*. PMLR, 2025.

Ruiqi Gao, Emiel Hoogeboom, Jonathan Heek, Valentin De Bortoli, Kevin P Murphy, and Tim Salimans. Diffusion meets flow matching: Two sides of the same coin. 2024. *URL https://diffusionflow. github. io*, 2024.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2414–2423, 2016.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

Yarden Huberman-Spiegelglas, Shai Bagon, and Michal Irani. Diffusion inversion: A universal technique for diffusion editing and interpolation. *Transactions on Machine Learning Research (TMLR)*, 2024.

Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018.

Nikolaj Banke Jensen and Jamie Vicary. Enforcing idempotency in neural networks. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision (ECCV)*, pp. 694–711, 2016.

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.

Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 31, 2018.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR)*, 2014.

Wilhelm Kutta. *Beitrag zur näherungsweisen Integration totaler Differentialgleichungen*. Teubner, 1901.

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.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9:4950, 2018.

Saunders Mac Lane and Garrett Birkhoff. *Algebra*. AMS Chelsea Publishing, 3rd edition, 1999.

Andreas Mardt, Luca Pasquali, Hao Wu, and Frank Noé. Vampnets for deep learning of molecular kinetics. *Nature Communications*, 9:5, 2018.

Igor Mezić. Spectral properties of dynamical systems, model reduction and decompositions. *Nonlinear Dynamics*, 41:309–325, 2005.

Y Nie. A time series is worth 64words: Long-term forecasting with transformers. *arXiv preprint arXiv:2211.14730*, 2022.

Roger Penrose. A generalized inverse for matrices. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51(3):406–413, 1955.

Michael Psenka, Druv Pai, Vishal Raman, Shankar Sastry, and Yi Ma. Representation learning via manifold flattening and reconstruction. *Journal of Machine Learning Research*, 2024.

Robin Quessard, Thomas D. Barrett, and William R. Clements. Learning group structure and disentangled representations of dynamical environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pp. 1530–1538. PMLR, 2015.

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*, pp. 234–241. Springer, 2015.

Carl Runge. Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46(2):167–178, 1895.

Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

Peter J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010.

Assaf Shocher, Amil Dravid, Yossi Gandelsman, Inbar Mosseri, Michael Rubinstein, and Alexei A. Efros. Idempotent generative network. In *International Conference on Learning Representations (ICLR)*, 2024.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations (ICLR)*, 2021a.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations (ICLR)*, 2021b.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023.

Gilbert Strang. *Introduction to Linear Algebra*. SIAM, 2022a.

Gilbert Strang. *Introduction to Linear Algebra*. SIAM, 2022b.

Steven H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. Chapman and Hall/CRC, 2024.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in neural information processing systems*, 34:22419–22430, 2021.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 586–595, 2018.

A PROOF OF LEMMA 1 (VALID VECTOR SPACES)

Proof. We verify that $(V, \oplus_g, \odot_g)$ is a vector space over $\mathbb{R}$ by transporting each axiom via g and g^{-1}, using only the definitions

$$u \oplus_g v := g^{-1}(g(u) + g(v)), \qquad a \odot_g u := g^{-1}(a\, g(u)).$$

For $u, v, w \in V$ and $a, b \in \mathbb{R}$:

1. **Closure**

$$u \oplus_g v \;=\; g^{-1}\big(g(u) + g(v)\big) \;\in\; V. \tag{24}$$

2. **Associativity**

$$\begin{aligned}(u \oplus_g v) \oplus_g w &= g^{-1}\big(g(u \oplus_g v) + g(w)\big)\\ &= g^{-1}\big((g(u) + g(v)) + g(w)\big)\\ &= g^{-1}\big(g(u) + (g(v) + g(w))\big)\\ &= g^{-1}\big(g(u) + g(v \oplus_g w)\big)\\ &= u \oplus_g (v \oplus_g w).\end{aligned} \tag{25}$$

3. **Commutativity**

$$\begin{aligned}u \oplus_g v &= g^{-1}\big(g(u) + g(v)\big)\\ &= g^{-1}\big(g(v) + g(u)\big)\\ &= v \oplus_g u.\end{aligned} \tag{26}$$

4. **Additive identity** (with $0_V := g^{-1}(0)$)

$$u \oplus_g 0_V = g^{-1}\big(g(u) + g(0_V)\big) = g^{-1}\big(g(u) + 0\big) = u. \tag{27}$$

5. **Additive inverse** (with $(-u) := g^{-1}(-g(u))$)

$$u \oplus_g (-u) = g^{-1}\big(g(u) + g(-u)\big) = g^{-1}\big(g(u) - g(u)\big) = g^{-1}(0) = 0_V. \tag{28}$$

6. **Compatibility of scalar multiplication**

$$\begin{aligned}a \odot_g (b \odot_g u) &= g^{-1}\big(a\, g(b \odot_g u)\big) = g^{-1}\big(a\,(b\, g(u))\big)\\ &= g^{-1}\big((ab)\, g(u)\big) = (ab) \odot_g u.\end{aligned} \tag{29}$$

7. **Scalar identity**

$$1 \odot_g u = g^{-1}\big(1 \cdot g(u)\big) = g^{-1}\big(g(u)\big) = u. \tag{30}$$

8. **Distributivity over vector addition**

$$\begin{aligned}a \odot_g (u \oplus_g v) &= g^{-1}\big(a\, g(u \oplus_g v)\big) = g^{-1}\big(a\,(g(u) + g(v))\big)\\ &= g^{-1}\big(a\, g(u) + a\, g(v)\big) = g^{-1}\big(a\, g(u)\big) \;\oplus_g\; g^{-1}\big(a\, g(v)\big)\\ &= (a \odot_g u) \;\oplus_g\; (a \odot_g v).\end{aligned} \tag{31}$$

9. **Distributivity over scalar addition**

$$\begin{aligned}(a + b) \odot_g u &= g^{-1}\big((a + b)\, g(u)\big) = g^{-1}\big(a\, g(u) + b\, g(u)\big)\\ &= g^{-1}\big(a\, g(u)\big) \;\oplus_g\; g^{-1}\big(b\, g(u)\big) = (a \odot_g u) \;\oplus_g\; (b \odot_g u).\end{aligned} \tag{32}$$

□

B PROOF OF LEMMA 6 (SVD OF A LINEARIZER)

Let $f : \mathcal{X} \to \mathcal{Y}$ be a Linearizer

$$f(x) = g_y^{-1}\big(A\, g_x(x)\big), \tag{33}$$

where $g_x : \mathcal{X} \to \mathbb{R}^N$ and $g_y : \mathcal{Y} \to \mathbb{R}^M$ are invertible and $A \in \mathbb{R}^{M\times N}$. Equip $\mathcal{X}$ and $\mathcal{Y}$ with the induced inner products

$$\langle u, v\rangle_{g_x} := \langle g_x(u), g_x(v)\rangle_{\mathbb{R}^N}, \qquad \langle p, q\rangle_{g_y} := \langle g_y(p), g_y(q)\rangle_{\mathbb{R}^M}. \tag{34}$$

Let the (Euclidean) SVD of A be $A = U\Sigma V^\top$, where $U = [u_1, \dots, u_M] \in \mathbb{R}^{M\times M}$, $V = [v_1, \dots, v_N] \in \mathbb{R}^{N\times N}$ are orthogonal and $\Sigma = \mathrm{diag}(\sigma_1, \dots, \sigma_r, 0, \dots)$ with $\sigma_1 \geq \cdots \geq \sigma_r > 0$. Define

$$\tilde{u}_i := g_y^{-1}(u_i) \in \mathcal{Y}, \qquad \tilde{v}_i := g_x^{-1}(v_i) \in \mathcal{X}. \tag{35}$$

Then $\{\tilde{v}_i\}$ and $\{\tilde{u}_i\}$ are orthonormal sets in $(\mathcal{X}, \langle\cdot,\cdot\rangle_{g_x})$ and $(\mathcal{Y}, \langle\cdot,\cdot\rangle_{g_y})$, respectively, and for each $i \leq r$,

$$f(\tilde{v}_i) = \sigma_i \odot_{g_y} \tilde{u}_i, \qquad \text{and} \qquad f^*(\tilde{u}_i) = \sigma_i \odot_{g_x} \tilde{v}_i, \tag{36}$$

where f^* is the adjoint with respect to the induced inner products and $a\odot_g$ denotes the induced scalar multiplication. Hence $(\{\tilde{u}_i\}_{i=1}^r, \{\sigma_i\}_{i=1}^r, \{\tilde{v}_i\}_{i=1}^r)$ is an SVD of f between the induced Hilbert spaces.

Proof. **Orthonormality.** For i, j,

$$\langle \tilde{v}_i, \tilde{v}_j\rangle_{g_x} = \langle g_x(\tilde{v}_i), g_x(\tilde{v}_j)\rangle_{\mathbb{R}^N} = \langle v_i, v_j\rangle_{\mathbb{R}^N} = \delta_{ij}, \tag{37}$$

and similarly $\langle \tilde{u}_i, \tilde{u}_j\rangle_{g_y} = \delta_{ij}$. Thus g_x and g_y are isometric isomorphisms from the induced spaces to Euclidean space.

Action on right singular vectors. Using $Av_i = \sigma_i u_i$,

$$f(\tilde{v}_i) = g_y^{-1}\big(A\, g_x(\tilde{v}_i)\big) = g_y^{-1}\big(Av_i\big) = g_y^{-1}\big(\sigma_i u_i\big) = \sigma_i \odot_{g_y} \tilde{u}_i. \tag{38}$$

Adjoint and action on left singular vectors. The adjoint $f^* : \mathcal{Y} \to \mathcal{X}$ with respect to the induced inner products is characterized by

$$\langle f(x), y\rangle_{g_y} = \langle x, f^*(y)\rangle_{g_x} \quad \text{for all } x \in \mathcal{X},\ y \in \mathcal{Y}. \tag{39}$$

Transporting through g_x, g_y and using Euclidean adjoints shows that

$$f^*(y) = g_x^{-1}\big(A^\top g_y(y)\big). \tag{40}$$

Therefore, since $A^\top u_i = \sigma_i v_i$,

$$f^*(\tilde{u}_i) = g_x^{-1}\big(A^\top g_y(\tilde{u}_i)\big) = g_x^{-1}\big(A^\top u_i\big) = g_x^{-1}\big(\sigma_i v_i\big) = \sigma_i \odot_{g_x} \tilde{v}_i. \tag{41}$$

Conclusion. The triples $\big(\tilde{u}_i, \sigma_i, \tilde{v}_i\big)$ satisfy the defining relations of singular triplets for f between the induced inner product spaces, with $\{\tilde{v}_i\}$ and $\{\tilde{u}_i\}$ orthonormal bases of the right and left singular subspaces. For zero singular values, the same construction holds with images mapped to the null spaces as usual. □

C PROOF OF LEMMA 4 (HILBERT SPACE)

Let $g : V \to \mathbb{R}^n$ be a bijection and endow V with the induced vector space operations $(\oplus_g, \odot_g)$ and inner product

$$\langle u, v \rangle_g \;:=\; \langle g(u),\, g(v) \rangle_{\mathbb{R}^n}. \tag{42}$$

Then $(V, \langle \cdot, \cdot \rangle_g)$ is a Hilbert space.

Proof. By Lemma 1, $(V, \oplus_g, \odot_g)$ is a vector space and g is a vector-space isomorphism from $(V, \oplus_g, \odot_g)$ onto $(\mathbb{R}^n, +, \cdot)$. The form $\langle \cdot, \cdot \rangle_g$ is an inner product because it is the pullback of the Euclidean inner product: bilinearity, symmetry, and positive definiteness follow immediately from injectivity of g and the corresponding properties in $\mathbb{R}^n$.

Let $\| \cdot \|_g$ be the norm induced by $\langle \cdot, \cdot \rangle_g$. For any $u, v \in V$,

$$\|u - v\|_g \;=\; \|g(u) - g(v)\|_2, \tag{43}$$

so g is an isometry from $(V, \| \cdot \|_g)$ onto $(\mathbb{R}^n, \| \cdot \|_2)$. Hence a sequence $\{u_k\} \subset V$ is Cauchy in $\| \cdot \|_g$ iff $\{g(u_k)\}$ is Cauchy in $\mathbb{R}^n$. Since $\mathbb{R}^n$ is complete, $\{g(u_k)\}$ converges to some $y \in \mathbb{R}^n$. By surjectivity of g, there exists $u^\star \in V$ with $g(u^\star) = y$, and then $\|u_k - u^\star\|_g = \|g(u_k) - y\|_2 \to 0$. Thus $(V, \| \cdot \|_g)$ is complete. Therefore $(V, \langle \cdot, \cdot \rangle_g)$ is a Hilbert space. $\square$

D PROOF OF LEMMA 7 (PSEUDO-INVERSE OF A LINEARIZER)

Pseudoinverse of a Linearizer. Let $f : \mathcal{X} \to \mathcal{Y}$ be a Linearizer $f(x) = g_y^{-1}\big(A\, g_x(x)\big)$ under the induced inner products $\langle u, v\rangle_{g_x} := \langle g_x(u), g_x(v)\rangle$ on $\mathcal{X}$ and $\langle u, v\rangle_{g_y} := \langle g_y(u), g_y(v)\rangle$ on $\mathcal{Y}$. Write $f^*(y) = g_x^{-1}\big(A^\mathsf{T} g_y(y)\big)$ for the adjoint and let $A^\dagger$ be the (Euclidean) Moore–Penrose pseudoinverse of A (Penrose, 1955).

The Moore–Penrose pseudoinverse of f with respect to the induced inner products is

$$f^\dagger \;=\; g_x^{-1} \circ A^\dagger \circ g_y. \tag{44}$$

Proof. Following Penrose (1955), the Moore–Penrose pseudoinverse of a linear operator is uniquely characterized as the map satisfying four algebraic conditions (the Penrose equations). To establish that $f^\dagger = g_x^{-1} \circ A^\dagger \circ g_y$ is indeed the pseudoinverse of f, it therefore suffices to verify these four identities explicitly:

1. $f f^\dagger f = f$,
2. $f^\dagger f f^\dagger = f^\dagger$,
3. $(f f^\dagger)^* = f f^\dagger$,
4. $(f^\dagger f)^* = f^\dagger f$.

We verify the Penrose equations in the induced spaces.

(1) $f\, f^\dagger\, f = f$:

$$\begin{aligned} f\, f^\dagger\, f &= \big(g_y^{-1} \circ A \circ g_x\big)\,\big(g_x^{-1} \circ A^\dagger \circ g_y\big)\,\big(g_y^{-1} \circ A \circ g_x\big) \\ &= g_y^{-1}\big(A A^\dagger A\, g_x\big) \\ &= g_y^{-1}\big(A\, g_x\big) \;=\; f, \end{aligned} \tag{45}$$

using $A A^\dagger A = A$.

(2) $f^\dagger f f^\dagger = f^\dagger$:

$$\begin{aligned} f^\dagger f f^\dagger &= \big(g_x^{-1} \circ A^\dagger \circ g_y\big)\,\big(g_y^{-1} \circ A \circ g_x\big)\,\big(g_x^{-1} \circ A^\dagger \circ g_y\big) \\ &= g_x^{-1}\big(A^\dagger A A^\dagger\, g_y\big) \\ &= g_x^{-1}\big(A^\dagger g_y\big) \;=\; f^\dagger, \end{aligned} \tag{46}$$

using $A^\dagger A A^\dagger = A^\dagger$.

(3) $(f f^\dagger)^* = f f^\dagger$ on $\mathcal{Y}$:

$$\begin{aligned} f f^\dagger &= g_y^{-1}\big(A A^\dagger\, g_y\big), \\ (f f^\dagger)^* &= g_y^{-1}\big((A A^\dagger)^\mathsf{T} g_y\big) = g_y^{-1}\big(A A^\dagger\, g_y\big) = f f^\dagger, \end{aligned} \tag{47}$$

since $A A^\dagger$ is symmetric.

(4) $(f^\dagger f)^* = f^\dagger f$ on $\mathcal{X}$:

$$\begin{aligned} f^\dagger f &= g_x^{-1}\big(A^\dagger A\, g_x\big), \\ (f^\dagger f)^* &= g_x^{-1}\big((A^\dagger A)^\mathsf{T} g_x\big) = g_x^{-1}\big(A^\dagger A\, g_x\big) = f^\dagger f, \end{aligned} \tag{48}$$

since $A^\dagger A$ is symmetric. All four conditions hold, hence $f^\dagger$ is the Moore–Penrose pseudoinverse of f in the induced inner-product spaces. □

E ADDITIONAL RESULTS

E.1 STYLE TRANSFER

We expand the results of style transfer made in the main paper and show multiple more transformations in Figure7 following the same setup.

Figure 7: **Style transfer examples.** Left: original image. Middle: style transfer using the left-side and right-side style images. Right: interpolation between the two styles.

E.2 ONE-STEP INVERSE INTERPOLATION

We extend the results presented in the main text by providing additional interpolation examples on the MNIST (Figure 9) and CelebA (Figure 8) datasets.

Figure 8: Additional interpolations of CelebA via inversion.

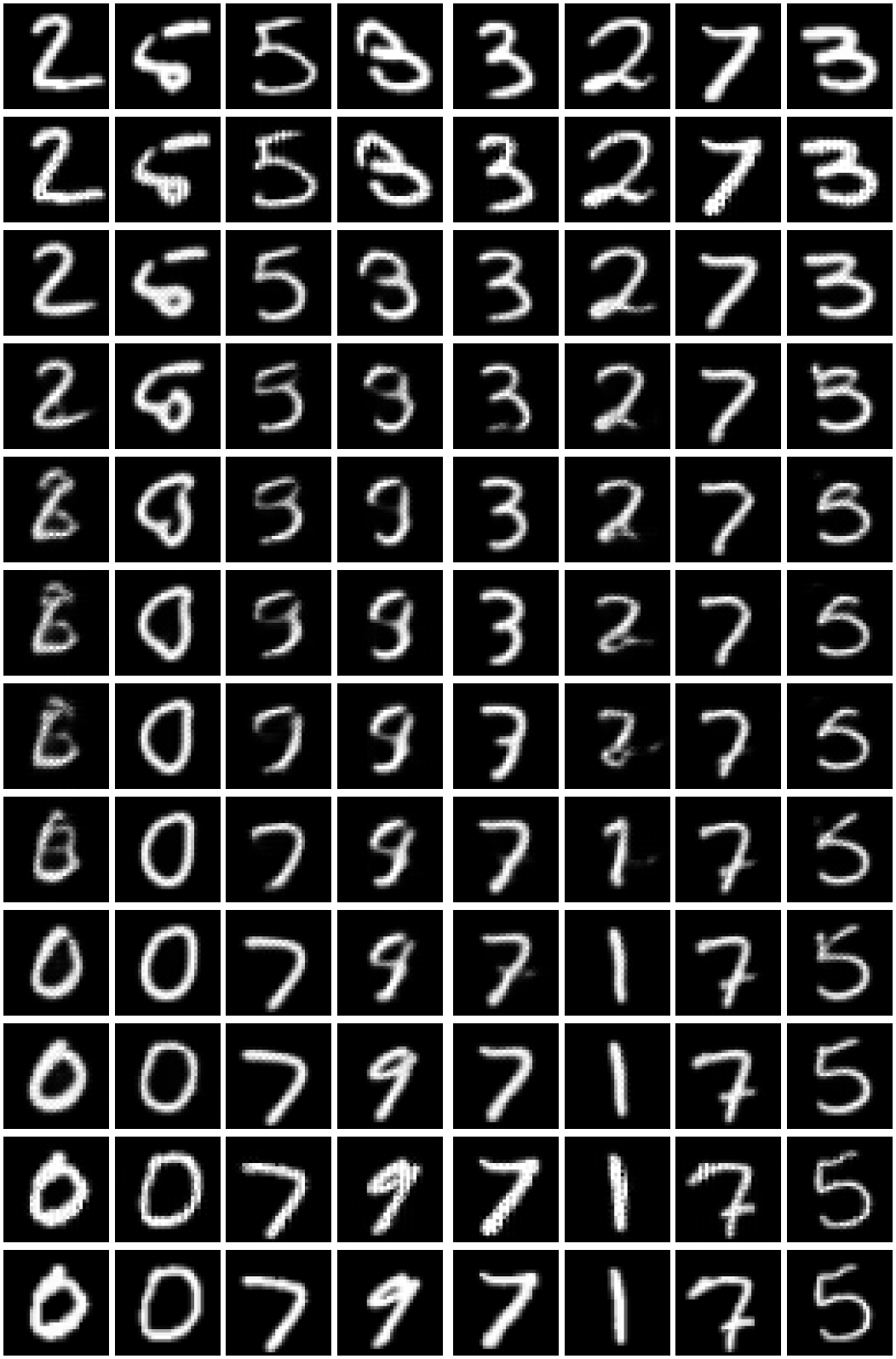

Figure 9: Additional interpolations of MNIST via inversion.

E.3 Additional Applications - Full Results

CelebFaces Attributes (CelebA). The dataset includes large pose variation, background clutter, diverse identities, and 40 binary attributes (e.g., *Male*, *Eyeglasses*, *Blond Hair*). After training, our Linearizer attains **87.6%** test accuracy; a ResNet-50 baseline reaches **86.1%**.

Distillation of nonlinear disentanglement models. We distill pretrained encoders (VAE, FactorVAE, β-VAE; checkpoints from the public repository[2]) on **dSprites** and evaluate with **FactorVAE**, **SAP**, and **DCI** (disentanglement, completeness, informativeness). The Linearizer closely matches the original encoders (rows prefixed with *Linearizer-*).

Table 1: Disentanglement metrics on dSprites. Higher is better (↑).

Model	FactorVAE ↑	SAP ↑	DCI-D ↑	DCI-C ↑	DCI-I ↑
VAE	.28±.03	.02±.01	.49±.02	.27±.01	.94±.00
Linearizer-VAE	.29±.03	.02±.00	.49±.01	.27±.01	.93±.01
FactorVAE	.33±.01	.06±.01	.64±.01	.51±.01	.94±.00
Linearizer-FactorVAE	.35±.03	.05±.01	.64±.01	.50±.00	.94±.00
BetaH	.59±.05	.23±.01	.83±.02	.71±.00	.95±.00
Linearizer-BetaH	.69±.05	.24±.01	.80±.00	.71±.00	.95±.00

Weather Prediction. We evaluate multi-step regression where the model predicts future values given 96 past steps. We compare against **Autoformer** [3] and **PatchTST** [2]. Columns denote horizons (steps ahead); scores are MSE (lower is better).

Table 2: Weather forecasting (MSE; lower is better) at different horizons.

Model	96	192	336	720
Linearizer	0.167	0.224	0.257	0.368
PatchTST	0.159	0.211	0.241	0.334
Autoformer	0.247	0.321	0.341	0.411

[2] https://github.com/YannDubs/disentangling-vae

F DERIVATION FOR RUNGE–KUTTA COLLAPSE TO ONE STEP

We follow the setup in Sec. 3.1. Let $z_t := g(x_t)$ and $f(x,t) = g^{-1}(A_t\, g(x))$. In the induced coordinates, addition and scaling are Euclidean, so the ODE update acts linearly on z_t.

RK4 step in the induced space. Define $t_n = n\Delta t$ and $z_n := z_{t_n} = g(x_{t_n})$. The classical RK4 step from t_n to $t_{n+1} = t_n + \Delta t$ is

$$k_1 = A_{t_n}\, z_n, \tag{49}$$

$$k_2 = A_{t_n+\frac{\Delta t}{2}}\left(z_n + \tfrac{\Delta t}{2}\, k_1\right), \tag{50}$$

$$k_3 = A_{t_n+\frac{\Delta t}{2}}\left(z_n + \tfrac{\Delta t}{2}\, k_2\right), \tag{51}$$

$$k_4 = A_{t_n+\Delta t}(z_n + \Delta t\, k_3), \tag{52}$$

$$z_{n+1} = z_n + \tfrac{\Delta t}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right). \tag{53}$$

Since each k_i is linear in z_n, there exists a matrix M_n such that $z_{n+1} = M_n\, z_n$. Expanding the linearity gives the explicit one-step operator

$$M_n = I + \tfrac{\Delta t}{6}\Big[A_{t_n} + 2\, A_{t_n+\frac{\Delta t}{2}}\Big(I + \tfrac{\Delta t}{2}\, A_{t_n}\Big) \tag{54}$$

$$+ 2\, A_{t_n+\frac{\Delta t}{2}}\Big(I + \tfrac{\Delta t}{2}\, A_{t_n+\frac{\Delta t}{2}}\big(I + \tfrac{\Delta t}{2}\, A_{t_n}\big)\Big) \tag{55}$$

$$+ A_{t_n+\Delta t}\Big(I + \Delta t\, A_{t_n+\frac{\Delta t}{2}}\big(I + \tfrac{\Delta t}{2}\, A_{t_n+\frac{\Delta t}{2}}\big(I + \tfrac{\Delta t}{2}\, A_{t_n}\big)\big)\Big)\Big]. \tag{56}$$

Collapsed one-step operator. Iterating N steps and collecting the product in latent space yields

$$B \;:=\; \prod_{n=0}^{N-1} M_n, \qquad \text{so that} \qquad g(\hat{x}_1) \;=\; B\, g(x_0), \quad \hat{x}_1 \;=\; g^{-1}\big(B\, g(x_0)\big). \tag{57}$$

The last two equations (equation 54 and equation 57) are the RK4 analogues of the Euler collapse in the main text: the multi-step solver is replaced by a single multiplication with B computed once after training.

Time-invariant sanity check (optional). If $A_t \equiv A$ is constant, then equation 54 reduces to the degree-4 Taylor polynomial of $e^{\Delta t A}$:

$$M_n \;=\; I + \Delta t\, A + \tfrac{(\Delta t)^2}{2}\, A^2 + \tfrac{(\Delta t)^3}{6}\, A^3 + \tfrac{(\Delta t)^4}{24}\, A^4, \qquad B \;=\; M^{\,N}. \tag{58}$$

This matches the behavior of classical RK4 on linear time-invariant systems.

G Implementation Details

G.1 Backbones and Block Structure

We use **6 invertible blocks** in all models. The two architectures differ only in the internal block type:

Model family	Per-block flow (forward)
Diffusion / Style	(optional Squeeze2x2 for 1ch) $\rightarrow$ ActNorm $\rightarrow$ Affine Coupling $(x_1 \,\vert\, x_2) \rightarrow$ Affine Coupling $(x_2 \,\vert\, y_1) \rightarrow$ Invertible 1×1 Conv
IGN	SpatialSplit2x2Rand $\rightarrow$ Additive Couplings with $\{F, G\} \rightarrow$ concat $\rightarrow$ PixelShuffle $(2\times)$

Table 3: Block-level flow for the two architectures; repeated 6 times.

Notes.

- **ActNorm**: per-channel affine with data-dependent init on first batch.
- **Affine coupling (Diffusion/Style)**: shift/log-scale predicted by a tiny U-Net conditioner; clamped $\log s$.
- **Invertible 1×1 conv (Diffusion/Style)**: learned channel permutation/mixer (orthogonal init).
- **SpatialSplit2x2Rand (IGN)**: deterministic 2×2 space partition into two streams; inverse merges.
- **PixelShuffle / unshuffle (IGN)**: reversible $2\times$ spatial reindexing between blocks.
- **We do not use `InvTanhScaled`**.

The blocks:

1. **Diffusion/Style.** We use a U-Net (Ronneberger et al., 2015), adapting the implementation of Karras et al. (2022). We modify only two hyperparameters: `model_channels` $= 16$ and `channel_mult` $= [1, 1]$.
2. **IGN.** We use botlleneck architecture, gradually downscaling to spatial size 1×1 by stride 2 convolutions, and then using transposed convolutions back to original resolution. See table below for exact layers.

G.2 Time Path (Diffusion)

In diffusion/flow-matching models, the scalar t is embedded and fed to: (i) the affine-coupling conditioners inside g, and (ii) the time-dependent core A_t (low-rank MLP). Style and IGN models do not use t.

G.3 Cores A

- **Diffusion**: Two non-linear MLPs that produces two matrices for the low-rank parameterization $A = A_1 A_2$ with $\mathrm{rank}(A) = 16$.
- **Style.** A kernel 4×4 is produced by a non-linear hypernetwork $L(\text{style}) = A_{\text{ker}}^{\text{style}}$, which we then apply to the image via a 2D convolution (PyTorch).
- **IGN**: diagonal A (elementwise scale in latent); for idempotent IGN we threshold the diagonal with STE.

G.4 Conditioners

Diffusion / Style (Affine-coupling conditioners). Each coupling uses a tiny U-Net that maps $C_{\text{in}} = C/2 \rightarrow 2\,C_{\text{in}}$ (shift, $\log s$).

#	Layer (per conditioner)
1	GroupNorm $\to$ Conv3 $\times$ 3 $\to$ SiLU
2	Residual block $\times 2$ (GroupNorm, Conv3 $\times$ 3, SiLU)
3	Final Conv3 $\times$ 3 to $2\,C_{\text{in}}$; clamp $\log s$

Table 4: Minimal U-Net-style conditioner used in affine couplings.

IGN (Additive-coupling CNNs). Each coupling uses a plain CNN (*no U-Net skip concatenations*); forward is additive, inverse is exact.

#	**F** or **G** layer stack
1	Conv2d $2 \to 8$, kernel 4, stride 2
2	Conv2d $8 \to 32$, kernel 4, stride 2
3	Conv2d $32 \to 128$, kernel 4, stride 2
4	Conv2d $128 \to 512$, kernel 4, stride 2
5	ConvTranspose2d $512 \to 128$, kernel 4, stride 2
6	ConvTranspose2d $128 \to 32$, kernel 4, stride 2
7	ConvTranspose2d $32 \to 8$, kernel 4, stride 2
8	ConvTranspose2d $8 \to 2$, kernel 4, stride 2 $\to \tanh$

Table 5: CNN used for F and G inside IGN additive couplings.

G.5 Losses (weights fixed)

IGN.

$$\lambda_{\text{rec}} = 1.0, \quad \lambda_{\text{sparse}} = 0.75, \quad \lambda_{\text{iso}} = 0.001.$$

We use (i) reconstruction MSE, (ii) sparsity/rank on the diagonal of A, and (iii) a light isometry term on g.

Diffusion / Flow-Matching. We use the main training loss described in the paper. For additional stability, we empirically found that adding the perceptual reconstruction terms $d(x_0,\, g^{-1}(g(x_0)))$ and $d(x_1,\, g^{-1}(g(x_1)))$ (with d as LPIPS (Zhang et al., 2018)), together with the alignment term $\|A\, g(x_0) - g(x_1)\|^2$, yields more stable training and higher-quality models. The primary loss in the paper also uses LPIPS as the distance metric. No explicit weighting between losses was required.

G.6 One-line Hyperparameter Summary

Item	Setting
Invertible blocks	6 (both families)
Permutations	Diff/Style: invertible 1×1 conv; IGN: SpatialSplit2x2Rand + PixelShuffle
ActNorm	Enabled (both)
Core A	Diff/Style: low-rank 16; IGN: diagonal (STE for projector)
Time t	Used in Diffusion/Style (conditioners + A_t); not used in IGN

Table 6: Everything needed to reproduce the scaffolding.

H Linearizer Universal Final Fit Theorem

Definition 4 (Linearizer-realizable map)**.** *A function $F^* : \mathbb{R}^N \to \mathbb{R}^M$ is* Linearizer-realizable *if there exist a linear map $A^* : \mathbb{R}^N \to \mathbb{R}^M$ and diffeomorphisms $g_x^* : \mathbb{R}^N \to \mathbb{R}^N$ and $g_y^* : \mathbb{R}^M \to \mathbb{R}^M$ such that*

$$F^*(x) \;=\; (g_y^*)^{-1}\big(A^*\, g_x^*(x)\big) \qquad \textit{for all } x \in \mathbb{R}^N.$$

Fact 1 (Diffeomorphism Extension on Finite Sets)**.** *Let $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^N$ be a finite set of n distinct points, and let $P = \{p_1, \dots, p_n\} \subset \mathbb{R}^N$ be any other finite set of n distinct points. There exists a global diffeomorphism $g^* : \mathbb{R}^N \to \mathbb{R}^N$ such that $g^*(x_i) = p_i$ for all $i = 1, \dots, n$.*

Fact 2 (INN Approximation, Teshima et al. 2020; informal)**.** *Let $K \subset \mathbb{R}^d$ be compact, and let $h : \mathbb{R}^d \to \mathbb{R}^d$ be a C^1 diffeomorphism. For any $\eta > 0$, there exists an invertible neural network (INN) $H_\eta : \mathbb{R}^d \to \mathbb{R}^d$ such that $\sup_{z \in K} \|H_\eta(z) - h(z)\| < \eta$ and $\sup_{z \in h(K)} \|H_\eta^{-1}(z) - h^{-1}(z)\| < \eta$.*

Lemma 9 (Existence of a Realizable Fit)**.** *Let $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^N$ be a finite set of distinct data points, and let $Y = \{y_1, \dots, y_n\} \subset \mathbb{R}^M$ be the corresponding target points. There exists an ideal, Linearizer-realizable function F^* that perfectly fits the data, i.e., $F^*(x_i) = y_i$ for all i.*

Proof. Let $P = \{p_1, \dots, p_n\} \subset \mathbb{R}^N$ be a set of n distinct latent points lying on the first coordinate axis, defined as $p_i = (i, 0, \dots, 0)^\top$.

We construct the linear map $A^* : \mathbb{R}^N \to \mathbb{R}^M$ to preserve the distinctness of these points by mapping the first axis of the domain to the first axis of the codomain. We distinguish two cases based on the dimensions:

- **Case 1 ($M \le N$): We define A^* as the projection onto the first M coordinates. For any vector $v = (v_1, \dots, v_N)^\top \in \mathbb{R}^N$:**
$$A^*(v) = (v_1, \dots, v_M)^\top \in \mathbb{R}^M.$$
- **Case 2 ($M > N$): We define A^* as the canonical embedding (zero-padding) into the first N coordinates. For any vector $v = (v_1, \dots, v_N)^\top \in \mathbb{R}^N$:**
$$A^*(v) = (v_1, \dots, v_N, \underbrace{0, \dots, 0}_{M-N})^\top \in \mathbb{R}^M.$$

In both cases, we define the target latent points as $Q = \{q_1, \dots, q_n\}$ where $q_i = A^* p_i$. Since the points p_i differ in their first coordinate (i), and A^* preserves the first coordinate in both cases (as $M, N \ge 1$), the resulting points q_i are distinct in $\mathbb{R}^M$.

By **Fact 1**, there exists a global diffeomorphism $g_x^* : \mathbb{R}^N \to \mathbb{R}^N$ such that $g_x^*(x_i) = p_i$ for all i. Similarly, there exists a global diffeomorphism $g_y^* : \mathbb{R}^M \to \mathbb{R}^M$ such that $g_y^*(y_i) = q_i$ for all i.

We define the ideal Linearizer F^* as:
$$F^*(x) := (g_y^*)^{-1}(A^* g_x^*(x)).$$
Checking this function on the data points x_i:
$$F^*(x_i) = (g_y^*)^{-1}(A^* p_i) = (g_y^*)^{-1}(q_i) = y_i.$$
Thus, a Linearizer-realizable function F^* exists that perfectly fits the finite dataset. □

Theorem 1 (A Linearizer can fit any finite dataset)**.** *Let $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^N$, $N \ge 2$, be a finite set and $F : X \to \mathbb{R}^M$ be the target function. For every $\varepsilon > 0$, there exist invertible neural networks G_x, G_y and a linear map A such that the Linearizer $\hat{F}(x) := G_y^{-1}(A\, G_x(x))$ achieves arbitrarily low error on the dataset:*
$$\sup_{x \in X} \|\hat{F}(x) - F(x)\| < \varepsilon.$$

Proof. By Lemma 9, there exists an ideal, Linearizer-realizable function $F^*(x) = (g_y^*)^{-1}(A^* g_x^*(x))$ such that $F(x) = F^*(x)$ for all $x \in X$. We need to prove that our INN-based architecture $\hat{F}$ can approximate F^* on the compact set X.

Let $\varepsilon > 0$ be given. Set $A := A^*$. The proof follows the standard ϵ-δ error decomposition.

$$\begin{aligned}\hat{F}(x) - F^*(x) &= G_y^{-1}\big(A^* G_x(x)\big) - (g_y^*)^{-1}\big(A^* g_x^*(x)\big) \\ &= \underbrace{G_y^{-1}\big(A^* G_x(x)\big) - (g_y^*)^{-1}\big(A^* G_x(x)\big)}_{(I)} + \underbrace{(g_y^*)^{-1}\big(A^* G_x(x)\big) - (g_y^*)^{-1}\big(A^* g_x^*(x)\big)}_{(II)}.\end{aligned}$$

The ideal diffeomorphisms g_x^* and $(g_y^*)^{-1}$ are continuous and are being approximated on compact sets. By Fact 2, we can choose INNs G_x and G_y that are arbitrarily close to g_x^* and g_y^* (and their inverses) on these sets.

As both $(g_y^*)^{-1}$ and A^* are continuous (and **uniformly continuous** on any compact set), we can make the norms of both terms (I) and (II) arbitrarily small by choosing sufficiently accurate INN approximations G_x and G_y (i.e., for a small enough δ). We can thus choose δ such that the total error is $< \varepsilon$. $\square$

I ADDITIONAL ILLUSTRATIONS

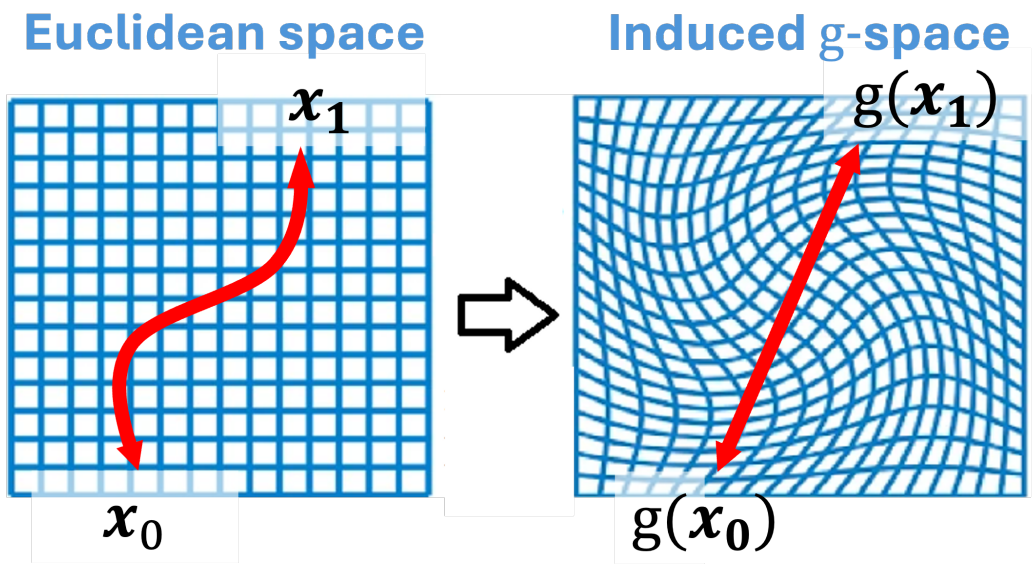

Figure 10: **Conceptual illustration of the geometric intuition.** We visualize the Linearizer in the context of Flow Matching (where $g_x = g_y = g$). (Left) The data manifold in standard Euclidean space. The optimal transport path between noise x_0 and data x_1 is a non-linear curve. (Right) The induced g-space. The network learns a coordinate transformation where this complex path is rectified into a simple straight line (a geodesic), enabling exact one-step generation.