Scaling Optimization-Oriented Hypernetworks for Implicit Neural Representations

Lulu Cai[1,*] **Hanxiang Ren**[2,3,*] **Siyan Dong**[1] **Li Sun**[1]
Jiefeng Wu[1] **Youyi Zheng**[3] **Yi Ma**[1] **Yanchao Yang**[1]
[1]The University of Hong Kong [2]TranscEngram [3]Zhejiang University [*] Equal contribution.
{lulu01cai,sunlids,jiefengwu}@connect.hku.hk
{siyan3d,mayi,yanchaoy}@hku.hk {hanxiang.ren,youyizheng}@zju.edu.cn

Abstract

We present a hypernetwork that generates the parameters of an implicit neural representation (INR) for a given signal in a single forward pass, without test-time optimization. Existing hypernetworks for this problem either ignore the chain-rule structure of the test-time optimization that produces an INR or, when they preserve it, instantiate the structure with MLPs whose parameter count grows quartically in target-network width. Our central contribution is to show that this parameter-count cost is structural to the MLP instantiation, not to the optimization-oriented principle of preserving chain-rule structure, and to resolve it by replacing the MLP body with attention. The resulting hypernetwork combines three design choices: an output-neuron-centric tokenization that aligns the token axis with each layer's output-neuron axis; an information-flow consistency rule that fixes the role of every cross-attention module from the representation its output should occupy; and a token-space iterative rollout that realizes the five quantities one optimization step on the target INR computes as five cross-attention modules with parameter count independent of target-network width. Across five 2D and 3D INR generation benchmarks, our method outperforms the state-of-the-art single-pass and optimization-oriented baselines; scales to target-network widths at which the MLP-based optimization-oriented paradigm exhausts GPU memory and produces the highest result on NeRF generation without ground-truth INR weights supervision.[1]

1 Introduction

Implicit neural representations (INRs) parameterize a signal as a neural network mapping coordinates to signal values, offering a continuous, resolution-independent, modality-agnostic functional view of data [1–3]. The price of this functional view is computational: producing the INR for a single new signal requires a per-instance optimization that takes thousands of gradient steps, shares no computation across signals, and is ill-suited to deployment scenarios in which signals arrive at inference time. Eliminating this per-instance cost – generating an INR for a given signal in a single forward pass, without test-time optimization – falls in the regime of hypernetworks.

Existing hypernetworks for INR generation face a tradeoff between scalability and explicit optimization structure. *Single-pass* hypernetworks regress the INR parameters directly [4–7]; they scale to large target networks and large training sets but do not explicitly encode the chain-rule dependencies that couple per-layer weights during optimization. *Optimization-oriented* hypernetworks instead unroll an optimization procedure into the architecture, with HyPoGen [8] predicting parameter updates layer by layer in the same forward-then-backward order that gradient descent on the target INR would follow, preserving the inter-layer dependencies of $\partial\mathcal{L}/\partial W_i$ in the hypernetwork architecture.

[1]Code repository: `https://github.com/lulucai435/hyper-inrs`.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

Yet, HyPoGen's MLP-based instantiation incurs a parameter cost quartic in target-network width per refinement step, leaving the larger INRs at which amortization is most useful out of reach.

Our position is that this tradeoff is structural to the MLP instantiation, not to the optimization-oriented principle. The factor that makes the MLP cost quartic is the joint dependence on both the row count and the row dimension of each weight matrix; attention removes the row-count dependence by sharing parameters across the token axis, and the row-dimension dependence is absorbed into a fixed embedding dimension by tokenization. The resulting hypernetwork tokenizes each weight matrix output-neuron by output-neuron; the five quantities a single optimization step on the target INR computes – one activation, two local Jacobians, the propagated loss-signal, and the parameter gradient – are realized one-for-one as learned token representations in five cross-attention modules, and the QKV assignment of every module is fixed by a single design rule we call *information-flow consistency*. The five modules together rollout for T refinement steps, mirroring the forward/backward dependencies of T optimization steps on the target INR (Sec. 3).

In summary, we make three contributions: (i) we identify the scalability barrier of optimization-oriented hypernetworks as a structural property of the MLP instantiation rather than of the principle, and remove it by replacing MLP sub-modules with attention; (ii) we derive a token-space realization of one optimization step in which five cross-attention modules follow from a single design rule; (iii) on five INR generation benchmarks, our method outperforms the strongest representatives of both prior paradigms and scales to target-network widths at which the MLP-based paradigm exhausts GPU memory. Initialization-only, compute-matched, and update-direction experiments further support the contribution of structured refinement (Sec. 4; Appendices 6.3.3 and 6.4.2).

2 Related Works

Implicit neural representations (INRs). INRs encode signals as continuous functions that map coordinates to signal values, parameterized by neural networks [1–3]. They underpin a range of applications including images and signed-distance fields [1, 9, 10], occupancy-based 3D shape reconstruction [11–14], and radiance fields [3, 15–17]. Subsequent work has improved their expressivity through periodic, wavelet, and learned activations [1, 18–24], multi-scale and band-limited encodings [2, 25, 26], hash- and grid-based encodings [27–30], and transferable initializations and features [31, 32]. These advances enrich what an INR can represent but leave the per-instance test-time optimization untouched; amortizing it across a training distribution is the focus of hypernetworks.

Hypernetworks for INR generation. A hypernetwork [33] produces target-network parameters from a conditioning input, used for audio [34], parameter-efficient transfer [35–37], continual [38] and few-shot learning [39–41], tabular foundation models [42], and weight prediction for unseen architectures [43, 44]. Within INR generation, the single-pass paradigm spans Transformer-based set-to-set or latent-to-weight mappings [4, 5], low-dimensional modulation [45], INR-VAE mixtures [6], architectural extensions [46–49], and a parallel line of weight-space generative models [7, 50–55]. Related approaches explore neural-process formulations [56], locality-aware modulation [57], and hypernetworks built on foundation models [58] for generalizable neural fields. The optimization-oriented paradigm is represented by HyPoGen [8], which unrolls a finite-step optimization procedure into the architecture and predicts per-step parameter updates whose layer-wise structure mirrors the chain-rule computation that gradient descent on the target INR would perform. We adopt this principle but observe that its MLP-based instantiation incurs a parameter cost quartic in target-network width per refinement step. Our method, to our knowledge, is the first to preserve the optimization-oriented principle while realizing it with attention. The chain-rule dependencies determine how the five cross-attention modules are connected, and the representation each module must produce determines which tokens supply its queries. At fixed embedding dimension, sharing the projections across output-neuron tokens makes the parameter count of these modules independent of target-network width.

Learned optimization and weight-space representations. Learned optimization studies how neural networks can produce parameter updates. Andrychowicz et al. [59] train a recurrent optimizer to map gradients to updates. For linear regression, von Oswald et al. [60] show how a linear self-attention layer can implement a step of gradient descent. Peebles et al. [61] learn from training checkpoints to generate parameter updates conditioned on a target metric. Modeling weights across networks also raises the question of how to represent them. For 3D neural fields, nf2vec [62] pools

features from individual weight rows into a single embedding for tasks such as shape retrieval. DeepWeightFlow [63] aligns hidden units across networks before learning a generative flow over their weights. Guided by the chain rule, our hypernetwork refines each output neuron's weight token by combining learned representations of the local parameter Jacobian and the loss signal propagated back to that layer.

Fast fitting and meta-learning for INRs. MetaSDF [64] and learned initializations [31] learn priors for gradient-based adaptation. Sparse INR meta-learning [65] constrains the adapted parameterization; COIN++ [66] optimizes low-dimensional modulations of a meta-learned network; context-pruned meta-learning [67] selects context points and corrects the induced meta-gradient bias. Transferable INR features [32] jointly train shared encoder layers and signal-specific decoders, then transfer the encoder for new-signal fitting, without bi-level meta-learning. These methods use gradient updates to fit an INR to each new signal. Our hypernetwork amortizes this fitting process across signals by learning to map a new signal to the full set of weights of its multilayer INR in one forward pass.

3 Method

3.1 From Test-Time Optimization to Feed-Forward Generation

Test-time optimization. An implicit neural representation (INR) encodes a signal ζ – such as an image, a 3D shape, or a radiance field – as a neural network $f_\theta : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ that maps a coordinate $\mathbf{x}$ to the signal value at that coordinate. The conventional way to obtain the parameters θ_ζ of an INR for a given signal ζ is to solve a per-instance optimization problem,

$$\theta_\zeta \;=\; \arg\min_\theta \mathcal{L}(f_\theta, \zeta), \tag{1}$$

where $\mathcal{L}$ is a reconstruction loss between f_θ and ζ. We refer to this procedure as *test-time optimization*, since the optimization is performed at the time the representation is needed, on the specific signal at hand, which is expensive on three counts: each signal incurs its own optimization run that takes minutes to hours, no computation is shared across signals, and the procedure is unsuited to deployment scenarios in which signals arrive at inference time.

Feed-forward generation and HyPoGen. A feed-forward alternative removes test-time optimization entirely: a hypernetwork $\mathcal{H}_\Omega$, trained once, produces the INR parameters in a single forward pass, $\theta_\zeta = \mathcal{H}_\Omega(\mathbf{z}_\zeta)$, given a per-signal conditioning input $\mathbf{z}_\zeta \in \mathbb{R}^{d_z}$. The obstacle is that the per-layer weights $W_1, \ldots, W_L$ of f_θ are not independent: they are coupled through the chain-rule computation that test-time optimization would normally exhibit, and regressing θ_ζ as a flat output does not explicitly preserve this coupling. [8] (HyPoGen) addressed this obstacle by *unrolling* the test-time optimization procedure into the hypernetwork's architecture: rather than regressing θ_ζ in one shot, HyPoGen produces it through T feed-forward steps,

$$\theta^{t+1} \;=\; \theta^t + \lambda^t \, \Delta\theta^t, \qquad t = 0, 1, \ldots, T-1, \tag{2}$$

where the step size $\lambda^t \in \mathbb{R}$ and the update direction $\Delta\theta^t$ are predicted by sub-modules of the hypernetwork, with the per-layer increment at layer i depending on signals from all subsequent layers $j > i$ to mirror the chain-rule computation of $\partial\mathcal{L}/\partial W_i$. We refer to this design as the *optimization-oriented* principle and adopt it as our starting point.

Scalability barrier of HyPoGen. The optimization-oriented principle is sound, but its instantiation in HyPoGen does not scale to the target networks of interest in INR generation. INRs are most commonly realized as MLPs with weight matrices $W_i \in \mathbb{R}^{n_i \times n_{i-1}}$, and HyPoGen's hypernetwork is itself MLP-based: each refinement step in Eq. 2 produces $\Delta\theta^{(t)}$ through MLP sub-modules whose parameter count per layer per refinement step is $\mathcal{O}(n_i^2 \cdot n_{i-1}^2)$, quartic in target-network width. Compressing each row of W_i to a d-dimensional latent before the MLP body reduces the per-row factor from n_{i-1}^2 to d^2 but leaves the n_i^2 factor intact; the full comparison is in Appendix 6.4.1. For large INRs and datasets that make hypernetwork-based amortization worthwhile, this cost is prohibitive. Preserving the principle while overcoming the scalability barrier requires an architectural ingredient whose parameter count is decoupled from the dimensions of the target-network layer, and attention is such an ingredient: a cross-attention module operates on a token sequence through projections whose size depends only on the per-token dimension. The remainder of the section develops a hypernetwork that retains the optimization-oriented principle and replaces the MLP body with attention, with the overall structure of one refinement step illustrated in Fig. 1.

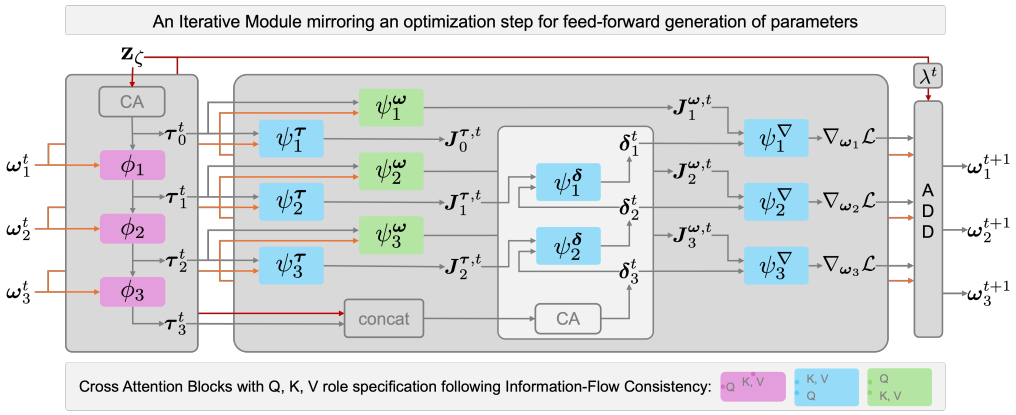


Figure 1: One iterative module of the proposed hypernetwork $\mathcal{H}_\Omega$, illustrated for a three-layer MLP target INR ($L = 3$); the construction extends to arbitrary L. At iteration t, the forward chain (left, modules ϕ_i) produces the pseudo-activation tokens $\boldsymbol{\tau}_i^t$ from the parameter tokens $\boldsymbol{\omega}_i^{(t)}$, seeded by $\boldsymbol{\tau}_0^t$ from $\mathbf{z}_\zeta$ through the Cross-Attention (CA) block. The backward block (right) preserves the chain-rule dependency structure at each layer: $\psi_i^{\boldsymbol{\tau}}$ and $\psi_i^{\boldsymbol{\omega}}$ produce the local Jacobian tokens $\boldsymbol{J}_{i-1}^{\boldsymbol{\tau},t}$ and $\boldsymbol{J}_i^{\boldsymbol{\omega},t}$, which $\psi_{i-1}^{\boldsymbol{\delta}}$ and ψ_i^{∇} combine with $\boldsymbol{\delta}_i^t$ via cross-attention to predict the propagated loss signal $\boldsymbol{\delta}_{i-1}^t$ and the parameter-gradient tokens, respectively; the backward recursion is seeded by $\boldsymbol{\delta}_L^t$, formed from $\boldsymbol{\tau}_L^t$ and $\mathbf{z}_\zeta$ via the concat-plus-CA module. The ADD block then applies the refinement with predicted step size and forwards to the next module.

3.2 What One Optimization Step Must Compute

By the optimization-oriented principle, the proposed hypernetwork is structured as a rollout of an iterative parameter-update procedure, in which each iterative module mirrors the operations of a single optimization step on the target INR. Before specifying the neural modules, we identify the quantities a single optimization step must compute in the target-network's own parameter and activation space.

Per-instance loss and forward pass. Test-time optimization on a signal ζ minimizes the reconstruction loss $\mathcal{L}(f_\theta, \zeta)$ over the INR parameters $\theta = (W_1, \ldots, W_L)$ by gradient-based updates. The forward pass of f_θ proceeds layer by layer,

$$\mathbf{h}_i \;=\; \sigma\big(W_i\,\mathbf{h}_{i-1}\big), \qquad i = 1, \ldots, L, \tag{3}$$

with $\mathbf{h}_0 = \mathbf{x}$ the input coordinate and $\mathbf{h}_L = f_\theta(\mathbf{x})$ the predicted signal value. Bias terms are omitted for clarity; their inclusion does not affect the structure of the derivation.

Loss-signal and chain rule. Define the *loss-signal at layer* i as $\delta_i := \partial\mathcal{L}/\partial\mathbf{h}_i$, with base case $\delta_L = \partial\mathcal{L}/\partial\mathbf{h}_L$ at the output layer. Applying the chain rule to Eq. 3 yields the two combinations a single optimization step requires at layer i,

$$\delta_{i-1} \;=\; \delta_i \cdot \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}}, \qquad\qquad \nabla_{W_i}\mathcal{L} \;=\; \delta_i \cdot \frac{\partial \mathbf{h}_i}{\partial W_i}, \tag{4}$$

the first propagating the loss-signal one layer backward and the second producing the parameter gradient that a gradient-based optimizer would use to update W_i.

Five per-layer quantities. Reading off Eqs. 3–4, one optimization step on the target INR computes, at each layer i, five quantities: (i) the activation $\mathbf{h}_i$; (ii) the activation Jacobian $\partial\mathbf{h}_i/\partial\mathbf{h}_{i-1}$; (iii) the parameter Jacobian $\partial\mathbf{h}_i/\partial W_i$; (iv) the propagated loss-signal δ_{i-1}; and (v) the parameter gradient $\nabla_{W_i}\mathcal{L}$. Quantities (i)–(iii) are local, each depending only on the current layer's parameters and the previous layer's activation; quantities (iv) and (v) are non-local, since they consume δ_i, which depends on every layer downstream of i through Eq. 4. The five cross-attention modules of the iterative module (Fig. 1) realize these five computational roles in token space, preserving their local and downstream dependencies. Next, we introduce the parameter tokens $\boldsymbol{\omega}_i$ that play the role of W_i in this token-space machinery.

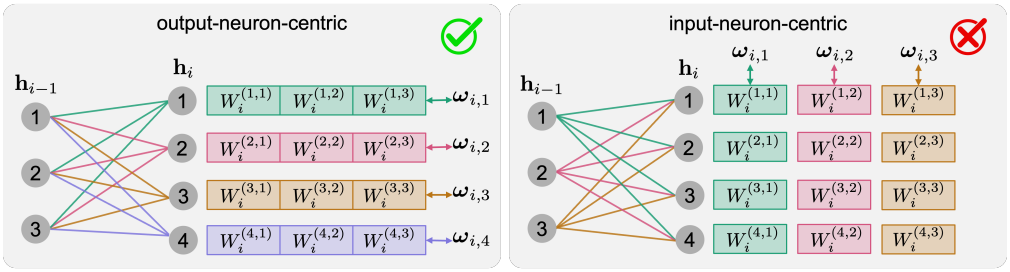


Figure 2: Two tokenization schemes for the layer-i weight matrix W_i. **Left (ours):** the *output-neuron-centric* scheme groups each row of W_i into one token $\boldsymbol{\omega}_{i,j}$, placing each output neuron's parameters into a single, co-adapted unit. **Right:** The *input-neuron-centric* scheme groups the outgoing weights of one input neuron into a single token. These weights feed into different output neurons, which may encode different features.

3.3 Output-Neuron-Centric Tokenization

To replace HyPoGen's MLP body with attention, we must choose how to tokenize the weight matrix $W_i \in \mathbb{R}^{n_i \times n_{i-1}}$ of each target-network layer. Two natural schemes are illustrated in Fig. 2: the *output-neuron-centric* scheme that slices W_i row by row, producing a token sequence of length n_i, and the *input-neuron-centric* scheme that slices W_i column by column, producing a token sequence of length n_{i-1}. Both fit the input format that attention requires, but the two schemes group different weight entries into one token, and the choice of which entries to group is what makes one scheme suit the hypernetwork's task while the other does not.

Tokenization. We adopt the output-neuron-centric scheme and represent W_i as

$$\boldsymbol{\omega}_i \;=\; \big(\boldsymbol{\omega}_{i,1}; \ldots ; \boldsymbol{\omega}_{i,n_i}\big) \;\in\; \mathbb{R}^{n_i \times d}, \tag{5}$$

where the j-th token $\boldsymbol{\omega}_{i,j} \in \mathbb{R}^{1\times d}$ is the encoding of the j-th row $W_i^{(j,:)} \in \mathbb{R}^{n_{i-1}}$ through a learned per-layer encoder, and a per-layer decoder $\mathrm{Decode}_i : \mathbb{R}^d \to \mathbb{R}^{n_{i-1}}$ recovers the row when the actual weight is needed. The token sequence $\boldsymbol{\omega}_i$ is the hypernetwork-side analogue of W_i named above.

Why row-wise rather than column-wise. The forward pass of the target INR (Eq. 3) is row-wise: the j-th component of $\mathbf{h}_i$ is determined entirely by the j-th row of W_i acting on the entire previous activation $\mathbf{h}_{i-1}$. The n_{i-1} entries of one row therefore form a co-adapted group, in the sense that they jointly determine one output neuron and their values must be consistent with one another for that neuron to compute the function the network needs. A column of W_i, by contrast, collects parameters that fan out from one input neuron to n_i different output neurons that may encode unrelated features. Row-wise tokenization lets the encoder and decoder represent and reconstruct each output neuron's incoming weights jointly. Column-wise tokenization splits those same weights across multiple tokens.

Token-axis as output-neuron-axis. Since the j-th row of W_i is exactly the parameters that produce the j-th output neuron, each token $\boldsymbol{\omega}_{i,j}$ corresponds to one output neuron of layer i. All five per-layer quantities of Sec. 3.2 live on this axis: $\mathbf{h}_i$ has one entry per output neuron, the activation Jacobian $\partial \mathbf{h}_i / \partial \mathbf{h}_{i-1}$ has one row per output neuron, the parameter Jacobian $\partial \mathbf{h}_i / \partial W_i$ is block-diagonal across output neurons (since $h_i^{(j)}$ depends only on the j-th row of W_i), and both δ_i and $\nabla_{W_i}\mathcal{L}$ have one entry or row per output neuron. Under output-neuron-centric tokenization, the token-space analogues of these quantities share the token axis with $\boldsymbol{\omega}_i$, so that attention's native operation of mixing across tokens corresponds to mixing across output neurons of the target layer. The empirical comparison against the input-neuron-centric scheme is reported in Sec. 4.4.

3.4 The Iterative Module

The iterative module (Fig. 1) produces the remaining five token-space quantities through five cross-attention modules: a forward chain of modules ϕ_i produces the pseudo-activation tokens $\boldsymbol{\tau}_i$, and a backward block of four modules $\psi_i^{\boldsymbol{\tau}}, \psi_i^{\boldsymbol{\omega}}, \psi_{i-1}^{\boldsymbol{\delta}}, \psi_i^{\nabla}$ produces the activation-Jacobian tokens $\boldsymbol{J}_{i-1}^{\boldsymbol{\tau}}$, the parameter-Jacobian tokens $\boldsymbol{J}_i^{\boldsymbol{\omega}}$, the loss-signal tokens $\boldsymbol{\delta}_{i-1}$, and the parameter-gradient tokens

$\nabla_{\boldsymbol{\omega}_i}\mathcal{L}$. The QKV projections and FFNs in each cross-attention module operate on d-dimensional tokens, so their parameter count is $\mathcal{O}(d^2)$, independent of the target-layer widths n_{i-1} and n_i.

Information-flow consistency. We design every cross-attention module in the iterative module by a single rule. Within each attention head, each query produces a convex combination of the projected value vectors. The values thus supply the content of each output token, while the queries determine the number of output tokens. The role of each module's output is fixed in advance – the forward block must produce pseudo-activation tokens, the activation-Jacobian module must produce activation-Jacobian tokens, and so on – so the slot V, and consequently K, must be occupied by the input that already lives in the role the output should occupy, with Q the remaining input. We call this rule *information-flow consistency* and apply it uniformly to derive the QKV assignment of every module below; the resulting assignments are also encoded by the color coding in Fig. 1.

Forward chain. The forward block at layer i takes $\boldsymbol{\omega}_i$ and $\boldsymbol{\tau}_{i-1}$ as inputs and produces an output in the role of pseudo-activation tokens for the current layer. By information-flow consistency, $\boldsymbol{\tau}_{i-1}$ is the input that already lives in the pseudo-activation role, so $V = K = \boldsymbol{\tau}_{i-1}$ and $Q = \boldsymbol{\omega}_i$, giving

$$\boldsymbol{\tau}_i \;=\; \phi_i(\boldsymbol{\omega}_i, \boldsymbol{\tau}_{i-1}) \;=\; \mathrm{softmax}\left(\frac{\boldsymbol{\omega}_i\, \boldsymbol{\tau}_{i-1}^{\top}}{\sqrt{d}}\right) \boldsymbol{\tau}_{i-1}. \tag{6}$$

This module reproduces the structural skeleton of a standard INR layer (Eq. 3) component by component: the score matrix plays the role of the linear mixing $W_i\mathbf{h}_{i-1}$, the row-wise softmax plays the role of the non-linearity σ, and the row-wise tokenization of Sec. 3.3 ensures output-unit alignment between the j-th row of W_i and the j-th output token. Recursive application of ϕ_i for $i = 1, \ldots, L$ mirrors the forward pass of Eq. 3 in token space; the recursion is seeded by $\boldsymbol{\tau}_0$, produced from $\mathbf{z}_\zeta$ through the top-left cross-attention module in Fig. 1. The full component-by-component correspondence is in Appendix 6.1.5.

Backward block. The backward block at layer i realizes the four computational roles (ii)–(v) of Sec. 3.2 with learned cross-attention modules, with each module's QKV assignment fixed by information-flow consistency once the role of its output is read off the specification:

$$\boldsymbol{J}^{\boldsymbol{\tau}}_{i-1} = \psi^{\boldsymbol{\tau}}_i(\boldsymbol{\omega}_i, \boldsymbol{\tau}_{i-1}), \qquad \boldsymbol{J}^{\boldsymbol{\omega}}_i = \psi^{\boldsymbol{\omega}}_i(\boldsymbol{\tau}_{i-1}, \boldsymbol{\omega}_i), \tag{7}$$

$$\boldsymbol{\delta}_{i-1} = \psi^{\boldsymbol{\delta}}_{i-1}(\boldsymbol{\delta}_i, \boldsymbol{J}^{\boldsymbol{\tau}}_{i-1}), \qquad \nabla_{\boldsymbol{\omega}_i}\mathcal{L} = \psi^{\nabla}_i(\boldsymbol{\delta}_i, \boldsymbol{J}^{\boldsymbol{\omega}}_i), \tag{8}$$

where each module argument list is written in the order (Q, shared $K = V$). The recursion runs with i ranging from L down to 1, each step producing $\boldsymbol{\delta}_{i-1}$ from $\boldsymbol{\delta}_i$ and the layer-i Jacobian token sequences. The recursion is seeded by $\boldsymbol{\delta}_L$, produced from $\boldsymbol{\tau}_L$ and $\mathbf{z}_\zeta$ through the concat-plus-cross-attention module on the right of Fig. 1; the two inputs carry the prediction-side and signal-side information that the target-network loss-signal $\delta_L = \partial\mathcal{L}/\partial\mathbf{h}_L$ depends on. Per-module derivations of the QKV assignments following the *information-flow consistency* principle are in Appendix 6.1.6.

3.5 Iterative Refinement and the Full Hypernetwork

The forward chain and the four backward modules furnish the per-layer machinery for one refinement step on the target INR. Now we compose the per-layer machinery into one refinement step of the target network and stack refinement steps across T iterations to realize the full hypernetwork.

One refinement step. One refinement step proceeds in three stages as shown in Fig. 1. The forward chain of $\phi_1, \ldots, \phi_L$ is run across all L target-network layers, producing the pseudo-activation tokens $\boldsymbol{\tau}_0, \ldots, \boldsymbol{\tau}_L$. The backward block is then run with i ranging from L down to 1, producing the loss-signal tokens $\boldsymbol{\delta}_{L-1}, \ldots, \boldsymbol{\delta}_0$ together with the parameter-gradient tokens $\nabla_{\boldsymbol{\omega}_i}\mathcal{L}$ at each layer as side outputs. Each layer's parameter tokens are then refined by

$$\boldsymbol{\omega}_i^{t+1} \;=\; \boldsymbol{\omega}_i^{t} \;-\; \lambda_i^t\, \nabla_{\boldsymbol{\omega}_i}\mathcal{L}^t, \qquad i = 1, \ldots, L, \tag{9}$$

where λ_i^t is a per-layer, per-signal step size predicted from the conditioning and parameter tokens (Appendix 6.1.4). Eq. 9 is one iteration of the rollout in Eq. 2, applied to the parameter token sequences directly; the per-layer decoder Decode_i then maps each refined state to target-INR weights.

Rollout and the full hypernetwork. Repeating the iterative module for T iterations produces parameter token sequences $\boldsymbol{\omega}_i^0, \ldots, \boldsymbol{\omega}_i^T$ at each layer i. The initial sequence $\boldsymbol{\omega}_i^0$ is generated from $\mathbf{z}_\zeta$ through an initialization sub-module, and the final sequence $\boldsymbol{\omega}_i^T$ is decoded to the target-network weight matrix W_i. The full hypernetwork is the composition of these T modules,

$$\theta_\zeta \;=\; \mathcal{H}_\Omega(\mathbf{z}_\zeta) \;=\; \big(W_1, \ldots, W_L\big), \qquad W_i \;=\; \mathrm{Decode}_i\big(\boldsymbol{\omega}_i^T\big), \tag{10}$$

producing the target-INR parameters in a single forward evaluation of $\mathcal{H}_\Omega$ as Sec. 3.1 required. The trainable parameters of $\mathcal{H}_\Omega$ comprise the per-layer encoders and decoders (Appendix 6.1.1), the FFNs inside the five cross-attention modules of the iterative module, the step-size sub-module, and the initialization sub-modules for $\boldsymbol{\tau}_0$, $\boldsymbol{\delta}_L$, and $\boldsymbol{\omega}_i^0$ at each layer; initialization details, including adaptive scaling for SIREN targets, are provided in Appendices 6.1.2–6.1.3. The parameter count of the core iterative body is independent of the target-network width at each layer, with the full comparison against HyPoGen given in Appendix 6.4.1. See Algorithm 1 for training and inference.

4 Experiments

We evaluate our method on a diverse set of INR generation tasks covering both 2D and 3D signals, across five datasets. For 2D signals, we adopt CelebA-HQ [68, 69] and ImageNet [70] for natural image tasks, and ERA5 [71] for climate data. For 3D signals, we use ShapeNet Chairs [72] for geometric reconstruction and Pre-trained NeRF [62] for novel view synthesis. Major baselines are HyPoGen [8], an MLP-based optimization-oriented hypernetwork, and LDMI [5], an attention-based hypernetwork without optimization-oriented design; together they represent the two existing paradigms that our method unifies. These baselines isolate the two axes of our contribution: HyPoGen instantiates the optimization-oriented principle without attention, LDMI uses attention without the optimization-oriented principle, and our method combines the two. We additionally include VaMoH [6] on the 2D datasets and ShapeNet Chairs, and HyperDiffusion [7] on Pre-trained NeRF; Trans-INR [4] and GINR-IPC [45] are deferred to Appendix 6.3.2. For each task, a shared encoder across all methods, maps the signal to $\mathbf{z}_\zeta$, from which the hypernetwork generates the target INR parameters θ_ζ (Appendix 6.2.3). Architectures, training configurations, and loss functions are in Appendix 6.2; additional qualitative results and the ImageNet LPIPS evaluation are in Appendices 6.3.1 and 6.3.2. Results across random seeds are reported in Appendix 6.3.6.

4.1 2D INR Generation

For 2D signals, we report reconstruction quality with PSNR, SSIM, and LPIPS in Table 1.

CelebA-HQ [68, 69] contains 30,000 face images (28K train, 2K test). Following [73] and [5], we use a resolution of 64×64. Our method outperforms all competitors across the three metrics, with a PSNR gain of 2.9 dB over LDMI, and produces reconstructions that closely match the ground truth. The gap against HyPoGen is noticeable: the MLP body's quartic parameter growth (Sec. 3.1) *does not* endow it the capacity to fit the 28K diverse facial samples, yielding collapsed reconstructions.

ERA5 [71] provides hourly atmospheric measurements from ECMWF's climate reanalysis at 31 km spatial resolution, with 8.5K training, 1.1K validation, and 2.4K test samples at 46×90 resolution; each pixel encodes a regional temperature. Our method again achieves the best score on all three metrics, confirming that the gains on natural images extend to scientific signals.

Table 1: **Quantitative comparison on 2D INR reconstruction** on CelebA-HQ, ERA5, and ImageNet.

Dataset	Metric	VAMoH	HyPoGen	LDMI	**Ours**
CelebA-HQ	PSNR ↑	21.3	16.2	24.8	**27.7**
	SSIM ↑	0.6303	0.4550	0.6947	**0.8635**
	LPIPS ↓	0.1332	0.1463	0.0556	**0.0296**
ERA5	PSNR ↑	39.0	40.0	44.6	**49.3**
	SSIM ↑	0.8090	0.9947	0.9952	**0.9991**
	LPIPS ↓	0.1663	0.0008	0.0003	**0.0001**
ImageNet	PSNR ↑	19.6	13.2	20.7	**22.9**
	SSIM ↑	0.5955	0.2473	0.5610	**0.6064**

ImageNet [70] operates at a substantially larger scale. We use the ILSVRC 2012 subset, with 1.28M training, 50K validation, and 100K test images across 1,000 classes at 256×256 resolution following [5]. Our method outperforms all baselines. HyPoGen fails to converge here, consistent with the scalability barrier of Sec. 3.1. LDMI converges but produces visible artifacts and inconsistencies (Fig. 3), as expected when per-layer weights are regressed without the chain-rule structure that the optimization-oriented principle provides.

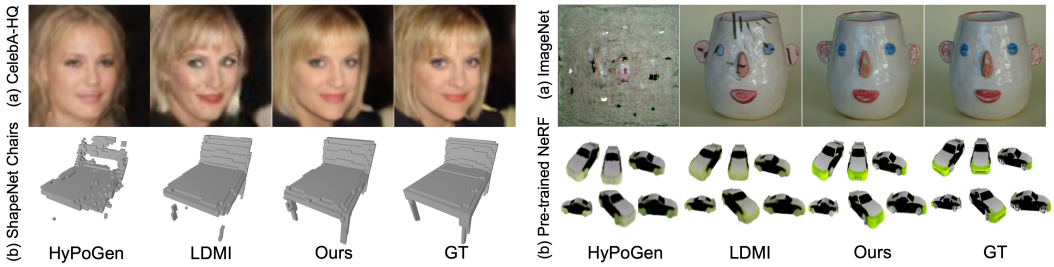


Figure 3: Visual comparison of generated INRs across four datasets. **Left:** 2D INR generation on (a) CelebA-HQ and 3D shape reconstruction on (b) ShapeNet Chairs. **Right:** 2D INR generation on (a) ImageNet and novel view synthesis on (b) Pre-trained NeRF. Columns show HyPoGen, LDMI, Ours, and the ground truth (GT). Qualitative results on ERA5, as well as additional samples for all four datasets shown here, are provided in Appendix 6.3.1.

4.2 3D INR Generation

We consider two 3D INR generation tasks: geometric reconstruction on occupancy grids using ShapeNet Chairs [72], and novel view synthesis on neural fields using the Pre-trained NeRF dataset [62], evaluated by occupancy grid accuracy and by PSNR with standard image-level metrics.

ShapeNet Chairs [72] contains 6,778 chair instances (5.4K train, 677 val, 679 test). Following [5], we adopt $32\times32\times32$ occupancy grids. Numerical results in Table 2 report occupancy accuracy and PSNR on rendered views, where our method achieves state-of-the-art performance on both metrics. Qualitatively (Fig. 3), HyPoGen fails to capture fine details such as chair legs, LDMI yields incomplete reconstructions with coarse surfaces, and our method produces complete shapes with smooth and plausible geometry.

Pre-trained NeRF [62] comprises 103K objects (98.6K train, 4K test). Each object is equipped with 36 rendered views, camera parameters, and an individual NeRF pre-trained via ray-based optimization on its own images. For each task, we take 6 conditional views as input to synthesize 6 novel views. We evaluate three supervision configurations (Table 2): *Weights only* uses only the per-object NeRF weights from [62], *Images only* uses only multi-view images, and *Weights & Images* uses both. HyperDiffusion is inapplicable to the images-only setting since it diffuses directly in weight space (see Appendix 6.2.1 for details). Our method outperforms all baselines across different configurations. Our images-only result of 28.8 dB is the highest overall in the table, showing that the framework does not rely on weight supervision – a practical advantage given the cost of high-quality ground-truth INR weights. The gain over LDMI is largest under weights-only (+2.7 dB), and the relative ranking of the two main baselines reverses there: HyPoGen (19.9 dB) outperforms LDMI (19.3 dB), contrary to their ordering elsewhere. This matches the design regimes of Sec. 3.1: when supervision is imposed directly in weight space, optimization-oriented designs fit the target distribution better than attention-based hypernetworks without such structural inductive bias. Additional NeRF evaluations and a discussion of hidden-neuron permutation symmetry are provided in Appendices 6.3.7–6.3.9.

Table 2: **Quantitative comparison on 3D INR generation.** 3D shape reconstruction on ShapeNet Chairs (top) and novel view synthesis on Pre-trained NeRF (bottom) under three supervision settings. "HyperDiff." denotes HyperDiffusion.

Dataset	Metric	VAMoH	HyPoGen	LDMI	**Ours**
ShapeNet Chairs	Accuracy ↑	96.8%	95.2%	97.3%	**97.9%**
	PSNR ↑	15.6	19.0	22.4	**23.8**
Pre-trained NeRF	Metric	HyperDiff.	HyPoGen	LDMI	**Ours**
Weights only	PSNR ↑	20.0	19.9	19.3	**22.0**
Images only	PSNR ↑	N/A	24.9	26.2	**28.8**
Weights & Images	PSNR ↑	17.5	25.5	26.5	**27.8**
	SSIM ↑	0.8774	0.921	0.929	**0.948**
	LPIPS ↓	0.1972	0.1182	0.1079	**0.0742**

4.3 Computational Efficiency

Table 3 reports parameter count, GPU memory, and inference latency at batch size 16 against HyPoGen and LDMI. Our hypernetwork uses fewer or comparable parameters across all five datasets,

with a similar trend in GPU memory; the gap is largest on Pre-trained NeRF, where the baselines require 5–6× the memory. Inference latency is also lowest on the two largest benchmarks, ImageNet and Pre-trained NeRF. Comparisons with per-instance SGD are in Appendix 6.3.4; additional FLOPs-matched comparisons and width-dependent cost analyses are in Appendix 6.4.2.

Table 3: Computational efficiency evaluated at batchsize 16 on each dataset.

	Params (M)			**GPU Memory (GB)**			**Infer Time @ b=16 (ms)**		
Dataset	HyPoGen	LDMI	Ours	HyPoGen	LDMI	Ours	HyPoGen	LDMI	Ours
CelebA-HQ	128.8	40.5	41.2	0.72	0.35	0.42	6.6	17.2	24.5
ImageNet	578.7	544.9	124.5	8.20	8.12	5.51	66.8	84.5	58.5
ERA5	93.1	48.9	21.3	0.59	0.51	0.33	21.2	25.0	24.6
Shapes3D	8.3	30.9	4.1	0.82	0.79	0.80	4.7	8.3	10.2
Pretrained NeRF	1049.9	515.5	482.5	14.51	12.47	2.51	2382	2541	2018

4.4 Ablation and Analysis

We ablate the two architectural principles of Sec. 3 and validate the scalability claim of Sec. 3.1 empirically. Ablations on the forward chain, the backward decomposition, and the number of refinement steps T are in Appendix 6.3.3.

Output-neuron-centric tokenization. *Column-wise Tokenization* replaces the output-neuron-centric scheme of Sec. 3.3 with the input-neuron-centric scheme of Fig. 2: each token of $\boldsymbol{\omega}_i$ encodes a column of W_i rather than a row, breaking the correspondence between the j-th token and the j-th output neuron of layer i, while every other component is kept unchanged. PSNR drops by 2.4 / 3.1 / 2.7 dB on CelebA-HQ / ImageNet / NeRF (Table 4), confirming that grouping the parameters that produce one output neuron into a co-adapted token, rather than the parameters that fan out from one input neuron to unrelated output neurons, is what aligns the token axis with the output-neuron axis on which all five per-layer quantities of Sec. 3.2 live.

Table 4: Ablations on the two architectural principles of Sec. 3. PSNR (↑) on CelebA-HQ, ImageNet, and Pre-trained NeRF (*Weights & Images*).

Variant	CelebA-HQ	ImageNet	NeRF
Ours (Full)	**27.7**	**22.9**	**27.8**
Column-wise Tokenization	25.3	19.8	25.1
Reversed Q-KV	23.5	18.3	22.9

Information-flow consistency in QKV assignment. *Reversed Q-KV* swaps Q with K/V in every cross-attention module of the iterative module, violating the information-flow-consistency rule of Sec. 3.4. The forward block, for instance, becomes $\boldsymbol{\tau}_i = \mathrm{CrossAttn}(\boldsymbol{\tau}_{i-1}, \boldsymbol{\omega}_i, \boldsymbol{\omega}_i)$, which places V in the parameter space and so writes its output into a role inconsistent with the pseudo-activation position of $\boldsymbol{\tau}_i$; the four backward modules (Eqs. 7–8) are reversed analogously. PSNR drops by 4.2 / 4.6 / 4.9 dB across the three datasets (Table 4). Since only the QKV assignment changes and all parameter counts are preserved, the gap directly measures the value of the principle.

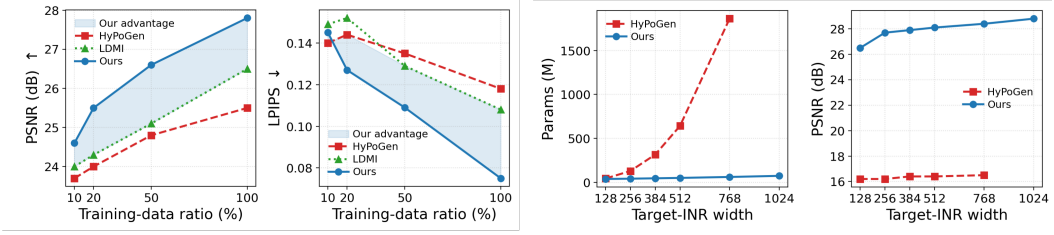
Descent behavior of learned updates. On 20 fixed ImageNet validation images, we compare learned updates with true negative-gradient and random directions from the same rollout states. The control directions are rescaled to match the learned update's norm at each layer. True gradients with respect to the parameter tokens are computed by backpropagating reconstruction MSE through the decoder and target INR. Table 5 reports reconstruction MSE at different step-size multipliers α, with $\alpha = 1$ denoting the learned update's full length.

Table 5 shows that learned directions reduce mean reconstruction MSE at every reported nonzero step size across all three transitions, while norm-matched random directions provide no consistent improvement. The negative-gradient direction gives a larger decrease for $T_0 \to T_1$ at $\alpha = 0.05$. At $\alpha = 1$, however, the network's actual updates reduce mean MSE in every transition, whereas norm-matched gradient steps increase it. These results show that the learned rollout produces successive loss-reducing updates at the step lengths used for INR generation. Decoded rollout reconstructions are in Appendix 6.3.5.

Scalability Analysis The parameter-count derivation in Appendix 6.4.1 formalizes the scalability barrier mentioned in Sec. 3.1: per target-network layer per refinement step, HyPoGen's core body

Table 5: Matched-norm direction diagnostic at selected step sizes. **Learned**: our model's updates; **GD**: true negative-gradient directions; **Random**: random directions. GD and Random are matched to Learned's norm at each layer. Entries are reconstruction MSE (↓) averaged over 20 ImageNet validation images; Random reports the mean and standard deviation of this average across 10 seeds.

	$T_0 \to T_1$			$T_1 \to T_2$			$T_2 \to T_3$		
α	Learned	GD	Random	Learned	GD	Random	Learned	GD	Random
0	.718	.718	$.718 \pm .000$	.192	.192	$.192 \pm .000$	.096	.096	$.096 \pm .000$
.05	.681	.295	$.717 \pm .006$	.189	.596	$.192 \pm .003$	.091	.254	$.097 \pm .002$
.1	.642	1.404	$.717 \pm .011$	.186	.862	$.193 \pm .006$	.086	.261	$.098 \pm .003$
.5	.348	.697	$.725 \pm .056$	.144	1.108	$.233 \pm .034$	.051	.442	$.129 \pm .019$
1	.192	.734	$.780 \pm .115$	.096	1.002	$.368 \pm .085$	.042	.637	$.236 \pm .054$



(a) Effect of training-data scale.

(b) Effect of target-INR width.

Figure 4: **Empirical scalability analysis.** (a) Performance versus training-data ratio on NeRF generation, testing how accuracy improves with data scale. (b) Parameter count and PSNR versus target-INR width on CelebA-HQ at batch size 64, testing the parameter-count prediction of Eq. 40: HyPoGen's parameter count grows rapidly while ours grows mildly with much more gains in PSNR.

costs $\mathcal{O}(n_i^2 \cdot d^2)$ parameters while ours costs $\mathcal{O}(d^2)$, the factor of n_i^2 being the structural difference (Eq. 40). We test the two consequences empirically: *data scaling* (Fig. 4a) and *width scaling* (Fig. 4b).

Data scaling. Fig. 4a shows how each method's performance evolves as the fraction of Pre-trained NeRF training data grows from 10% to 100%. All three methods improve, but at very different rates: our method gains $+3.2$ dB PSNR over the range, LDMI $+2.5$ dB, and HyPoGen only $+1.8$ dB; LPIPS follows the same ranking. The gap between our method and LDMI also grows with scale, from $+0.6$ dB at 10% to $+1.3$ dB at full data, indicating that the methods diverge precisely in the regime that matters for amortized INR generation.

Width scaling. Fig. 4b tests the parameter-count prediction of Eq. 40 empirically. The prediction is that HyPoGen's core body carries a factor of n_i^2 and ours does not; the test sweeps the target-INR width from 128 to 1024 on CelebA-HQ at batch size 64, comparing HyPoGen and ours – the two methods that both encode the optimization-oriented principle, isolating the MLP body versus the attention body as the architectural variable. The prediction is borne out: HyPoGen's parameter count grows rapidly with width and exhausts GPU memory at 1024, while ours grows mildly across the entire range. PSNR follows the same pattern: ours improves from 26.5 to 28.8 dB as width grows, while HyPoGen remains around 16 dB across all tractable widths and fails outright at 1024. Appendix 6.4.3 evaluates token compression and layer fusion.

5 Conclusion

We have shown that the scalability barrier of optimization-oriented hypernetworks for INR generation is a property of the MLP body rather than of the principle of preserving chain-rule structure, and that replacing the body with attention dissolves the barrier without compromising the principle. Reconstruction benchmarks and the additional controlled experiments support the central observation that single-pass scalability and chain-rule structure are not opposing design goals; they are simultaneously achievable through a combination of weight-matrix tokenization, role-driven module specification, and a token-space rollout that mirrors one optimization step. This construction motivates further work on amortizing structured optimization in meta-learning, learned optimizers, and weight generation beyond INRs.

Acknowledgments

We thank the anonymous reviewers for their valuable comments and suggestions. This work was supported by the Early Career Scheme of the Research Grants Council (grant no. 27207224), the HKU-100 Award, and a donation from the Musketeers Foundation. This work was also supported in part by the JC STEM Lab of Autonomous Intelligent Systems funded by The Hong Kong Jockey Club Charities Trust. We acknowledge the computing resources provided by the HKU-Shanghai Intelligent Computing Research Center.

References

[1] Vincent Sitzmann, Julien N.P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*, 2020.

[2] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *NeurIPS*, 2020.

[3] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[4] Yinbo Chen and Xiaolong Wang. Transformers as meta-learners for implicit neural representations. In *European Conference on Computer Vision*, 2022.

[5] Ignacio Peis, Batuhan Koyuncu, Isabel Valera, and Jes Frellsen. Hyper-transforming latent diffusion models. In *Forty-second International Conference on Machine Learning*, 2025.

[6] Batuhan Koyuncu, Pablo Sanchez-Martin, Ignacio Peis, Pablo M. Olmos, and Isabel Valera. Variational mixture of hypergenerators for learning distributions over functions. In *International Conference on Machine Learning*, 2023.

[7] Ziya Erkoç, Fangchang Ma, Qi Shan, Matthias Nießner, and Angela Dai. Hyperdiffusion: Generating implicit neural fields with weight-space diffusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14300–14310, October 2023.

[8] Hanxiang Ren, Li Sun, Xulong Wang, Pei Zhou, Zewen Wu, Siyan Dong, Difan Zou, Youyi Zheng, and Yanchao Yang. Hypogen: Optimization-biased hypernetworks for generalizable policy generation. In *The Thirteenth International Conference on Learning Representations*, 2025.

[9] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[10] Matan Atzmon and Yaron Lipman. Sal: Sign agnostic learning of shapes from raw data. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[11] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[12] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[13] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3D shape. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4857–4866, 2020.

[14] Julian Chibane, Thiemo Alldieck, and Gerard Pons-Moll. Implicit functions in feature space for 3d shape reconstruction and completion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2020.

[15] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *CVPR*, 2021.

[16] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul Srinivasan, Howard Zhou, Jonathan T. Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibrnet: Learning multi-view image-based rendering. In *CVPR*, 2021.

[17] Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo. *arXiv preprint arXiv:2103.15595*, 2021.

[18] Vishwanath Saragadam, Daniel LeJeune, Jasper Tan, Guha Balakrishnan, Ashok Veeraraghavan, and Richard G Baraniuk. Wire: Wavelet implicit neural representations. In *Conf. Computer Vision and Pattern Recognition*, 2023.

[19] Zhen Liu, Hao Zhu, Qi Zhang, Jingde Fu, Weibing Deng, Zhan Ma, Yanwen Guo, and Xun Cao. Finer: Flexible spectral-bias tuning in implicit neural representation by variable-periodic activation functions. In *Proceedings of the IEEE/CVF Computer Vision and Pattern Recognition Conference (CVPR)*, 2024.

[20] Amirhossein Kazerouni, Reza Azad, Alireza Hosseini, Dorit Merhof, and Ulas Bagci. Incode: Implicit neural conditioning with prior knowledge embeddings. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1298–1307, 2024.

[21] Rizal Fathony, Anit Kumar Sahu, Devin Willmott, and J Zico Kolter. Multiplicative filter networks. In *International Conference on Learning Representations*, 2021.

[22] Kexuan Shi, Xingyu Zhou, and Shuhang Gu. Improved implicit neural representation with fourier reparameterized training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 25985–25994, 2024.

[23] Rui Gao and R. Jaiman. H-siren: Improving implicit neural representations with hyperbolic periodic functions. *ArXiv*, abs/2410.04716, 2024.

[24] Jiaming Zhao, Wenbo Qiao, Peng Zhang, and Hui Gao. Quantum implicit neural representations. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 60940–60956. PMLR, 21–27 Jul 2024.

[25] David B. Lindell, Dave Van Veen, Jeong Joon Park, and Gordon Wetzstein. Bacon: Band-limited coordinate networks for multiscale scene representation. In *CVPR*, 2022.

[26] Ishit Mehta, Michaël Gharbi, Connelly Barnes, Eli Shechtman, Ravi Ramamoorthi, and Manmohan Chandraker. Modulated periodic activations for generalizable local functional representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 14214–14223, 2021.

[27] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022.

[28] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision (ECCV)*, 2022.

[29] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[30] Shaowen Xie, Hao Zhu, Zhen Liu, Qi Zhang, You Zhou, Xun Cao, and Zhan Ma. Diner: Disorder-invariant implicit neural representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[31] Matthew Tancik, Ben Mildenhall, Terrance Wang, Divi Schmidt, Pratul P. Srinivasan, Jonathan T. Barron, and Ren Ng. Learned initializations for optimizing coordinate-based neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2846–2855, 2021.

[32] Kushal Vyas, Ahmed Imtiaz Humayun, Aniket Dashpute, Richard Baraniuk, Ashok Veeraraghavan, and Guha Balakrishnan. Learning transferable features for implicit neural representations. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[33] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017.

[34] Filip Szatkowski, Karol J. Piczak, Przemysław Spurek, Jacek Tabor, and Tomasz Trzcinski. Hypersound: Generating implicit neural representations of audio signals with hypernetworks. In *Sixth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2022.

[35] Rabeeh Karimi Mahabadi, Sebastian Ruder, Mostafa Dehghani, and James Henderson. Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks. In *Annual Meeting of the Association for Computational Linguistics*, 2021.

[36] Jesus-German Ortiz-Barajas, Helena Gomez-Adorno, and Thamar Solorio. Hyperloader: Integrating hypernetwork-based lora and adapter layers into multi-task transformers for sequence labelling, 2024.

[37] Hamish Ivison and Matthew Peters. Hyperdecoders: Instance-specific decoders for multi-task NLP. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 1715–1730, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[38] Johannes von Oswald, Christian Henning, Benjamin F. Grewe, and João Sacramento. Continual learning with hypernetworks. In *International Conference on Learning Representations*, 2020.

[39] Marcin Sendera, Marcin Przewięźlikowski, Konrad Karanowski, Maciej Zięba, Jacek Tabor, and Przemysław Spurek. Hypershot: Few-shot learning by kernel hypernetworks. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 2469–2478, 2023.

[40] Marcin Przewięźlikowski, Przemysław Przybysz, Jacek Tabor, Maciej Zięba, and Przemysław Spurek. HyperMAML: Few-shot adaptation of deep models with hypernetworks. *Neurocomputing*, 598:128179, 2024.

[41] Andrey Zhmoginov, Mark Sandler, and Maksym Vladymyrov. HyperTransformer: Model generation for supervised and semi-supervised few-shot learning. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pages 27075–27098. PMLR, 2022.

[42] Andreas C Mueller, Carlo A Curino, and Raghu Ramakrishnan. Mothernet: Fast training and inference via hyper-network transformers. In *The Thirteenth International Conference on Learning Representations*, 2025.

[43] Boris Knyazev, Michal Drozdzal, Graham W. Taylor, and Adriana Romero. Parameter prediction for unseen deep architectures. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[44] Boris Knyazev, Doha Hwang, and Simon Lacoste-Julien. Can we scale transformers to predict parameters of diverse imagenet models? In *International Conference on Machine Learning*, 2023.

[45] Chiheon Kim, Doyup Lee, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Generalizable implicit neural representations via instance pattern composers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11808–11817, 2023.

[46] Sudarshan Babu, Richard Liu, Avery Zhou, Michael Maire, Greg Shakhnarovich, and Rana Hanocka. Hyperfields: Towards zero-shot generation of nerfs from text. In *Forty-first International Conference on Machine Learning*, 2024.

[47] Eric Hedlin, Munawar Hayat, Fatih Porikli, Kwang Moo Yi, and Shweta Mahajan. HyperNet fields: Efficiently training hypernetworks without ground truth by learning weight trajectories. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 22129–22138, 2025.

[48] Xingyi Yang and Xinchao Wang. Neural metamorphosis. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 1–19, 2024.

[49] Qi Wu, David Bauer, Yuyang Chen, and Kwan-Liu Ma. Hyperinr: A fast and predictive hypernetwork for implicit neural representations via knowledge distillation. *arXiv preprint arXiv:2304.04188*, 2023.

[50] Kai Wang, Dongwen Tang, Boya Zeng, Yida Yin, Zhaopan Xu, Yukun Zhou, Zelin Zang, Trevor Darrell, Zhuang Liu, and Yang You. Neural network diffusion, 2024.

[51] Bedionita Soro, Bruno Andreis, Hayeon Lee, Wonyong Jeong, Song Chong, Frank Hutter, and Sung Ju Hwang. Diffusion-based neural network weights generation. In *The Thirteenth International Conference on Learning Representations*, 2025.

[52] Konstantin Schürholt, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Hyper-representations as generative models: Sampling unseen neural network weights. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[53] Konstantin Schürholt, Michael W. Mahoney, and Damian Borth. Towards scalable and versatile weight space learning. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*. PMLR, 2024.

[54] Kai Wang, Dongwen Tang, Wangbo Zhao, Konstantin Schürholt, Zhangyang Wang, and Yang You. Scaling up parameter generation: A recurrent diffusion approach, 2025.

[55] Soro Bedionita, Bruno Andreis, Song Chong, and Sung Ju Hwang. Instruction-guided autoregressive neural network parameter generation, 2025.

[56] Jeffrey Gu, Kuan-Chieh Wang, and Serena Yeung. Generalizable neural fields as partially observed neural processes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5330–5339, 2023.

[57] Doyup Lee, Chiheon Kim, Minsu Cho, and Wook-Shin Han. Locality-aware generalizable implicit neural representation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[58] Jeffrey Gu and Serena Yeung-Levy. Foundation models secretly understand neural network weights: Enhancing hypernetwork architectures with foundation models. In *The Thirteenth International Conference on Learning Representations*, 2025.

[59] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[60] Johannes von Oswald, Eyvind Niklasson, Ettore Randazzo, Joao Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 35151–35174, 2023.

[61] William Peebles, Ilija Radosavovic, Tim Brooks, Alexei A. Efros, and Jitendra Malik. Learning to learn with generative models of neural network checkpoints. *arXiv preprint arXiv:2209.12892*, 2022.

[62] Pierluigi Zama Ramirez, Luca de Luigi, Daniele Sirocchi, Adriano Cardace, Riccardo Spezialetti, Francesco Ballerini, Samuele Salti, and Luigi Di Stefano. Deep learning on object-centric 3d neural fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):9940–9956, 2024.

[63] Saumya Gupta, Scott Biggs, Moritz Laber, Zohair Shafi, Robin Walters, and Ayan Paul. DeepWeightFlow: Re-basined flow matching for generating neural network weights. In *The Fourteenth International Conference on Learning Representations*, 2026.

[64] Vincent Sitzmann, Eric Chan, Richard Tucker, Noah Snavely, and Gordon Wetzstein. MetaSDF: Meta-learning signed distance functions. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

[65] Jaeho Lee, Jihoon Tack, Namhoon Lee, and Jinwoo Shin. Meta-learning sparse implicit neural representations. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

[66] Emilien Dupont, Hrushikesh Loya, Milad Alizadeh, Adam Golinski, Yee Whye Teh, and Arnaud Doucet. COIN++: Neural compression across modalities. *Transactions on Machine Learning Research*, 2022.

[67] Jihoon Tack, Subin Kim, Sihyun Yu, Jaeho Lee, Jinwoo Shin, and Jonathan Richard Schwarz. Learning large-scale neural fields via context pruned meta-learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[68] 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 (ICCV)*, pages 3730–3738, December 2015.

[69] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *International Conference on Learning Representations (ICLR)*, 2018.

[70] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[71] Hans Hersbach, Bill Bell, Paul Berrisford, G Biavati, András Horányi, Joaquín Muñoz Sabater, Julien Nicolas, C Peubey, R Radu, I Rozum, et al. ERA5 monthly averaged data on single levels from 1979 to present. Copernicus Climate Change Service (C3S) Climate Data Store (CDS), 2019.

[72] Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, L. Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *ArXiv*, abs/1512.03012, 2015.

[73] Emilien Dupont, Hyunjik Kim, S. M. Ali Eslami, Danilo Jimenez Rezende, and Dan Rosenbaum. From data to functa: Your data point is a function and you can treat it like one. In *International Conference on Machine Learning*, 2022.

[74] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9650–9660, 2021.

[75] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[76] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[77] Patrick Esser, Robin Rombach, and Björn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

6 Appendix

6.1 Method and Algorithm Details

6.1.1 Per-Layer Encoder and Decoder

The tokenization scheme of Sec. 3.3 introduces a per-layer encoder Encode_i and a per-layer decoder Decode_i for layer i of the target network. They sit at the two ends of the iterative module: Encode_i converts each row of W_i into one parameter token to form $\boldsymbol{\omega}_i$, and Decode_i recovers each row at the end of the rollout (Eq. 10). This appendix specifies both.

The encoder maps the j-th row of W_i to the j-th parameter token through an MLP, adds a learnable positional embedding $\boldsymbol{p}_{i,j} \in \mathbb{R}^d$ to distinguish the row identity, and applies layer normalization:

$$\boldsymbol{\omega}_{i,j} \;=\; \mathrm{Encode}_i\big(W_i^{(j,:)}\big) \;=\; \mathrm{LayerNorm}\Big(\mathrm{MLP}_i^{\mathrm{enc}}\big(W_i^{(j,:)}\big) \;+\; \boldsymbol{p}_{i,j}\Big), \qquad j = 1, \ldots, n_i. \quad (11)$$

The decoder maps the refined tokens back to weight rows through layer normalization and an MLP:

$$W_i^{(j,:)} \;=\; \mathrm{Decode}_i(\boldsymbol{\omega}_{i,j}) \;=\; \mathrm{MLP}_i^{\mathrm{dec}}\big(\mathrm{LayerNorm}(\boldsymbol{\omega}_{i,j})\big), \qquad j = 1, \ldots, n_i. \quad (12)$$

Both $\mathrm{MLP}_i^{\mathrm{enc}} : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^d$ and $\mathrm{MLP}_i^{\mathrm{dec}} : \mathbb{R}^d \to \mathbb{R}^{n_{i-1}}$ are shared across the n_i rows of layer i, so each module's parameter count is independent of n_i; the rows are distinguished only by the positional embeddings $\boldsymbol{p}_{i,1}, \ldots, \boldsymbol{p}_{i,n_i}$, which contribute a term that scales linearly in n_i rather than quadratically.

6.1.2 Initialization Sub-modules

The full hypernetwork $\mathcal{H}_\Omega$ specified in Sec. 3.5 requires three initial token sequences for the iterative module of Sec. 3.4 to be unrolled: the pseudo-activation $\boldsymbol{\tau}_0$ that seeds the forward chain at the input side of every iteration, the loss-signal $\boldsymbol{\delta}_L^t$ that seeds the backward block at the output side of iteration t, and the parameter token sequence $\boldsymbol{\omega}_i^0$ at each layer i that seeds the parameter-token rollout of Eq. 9. The three sub-modules differ in when they are evaluated: $\boldsymbol{\tau}_0$ depends only on $\mathbf{z}_\zeta$ and is computed once at the start of the rollout, $\boldsymbol{\delta}_L^t$ depends on the layer-L pseudo-activation $\boldsymbol{\tau}_L^t$ and is therefore re-evaluated at every iteration t, and $\boldsymbol{\omega}_i^0$ is consumed only at $t = 0$ and overwritten by the rollout's update rule at every subsequent iteration. This appendix specifies the three sub-modules that produce them. All three take the conditioning input $\mathbf{z}_\zeta$ as input and emit token sequences in the format expected by the iterative module.

Pseudo-activation Initialization $\boldsymbol{\tau}_0$ The pseudo-activation tokens $\boldsymbol{\tau}_0 \in \mathbb{R}^{n_0 \times d}$ play the role at the input of the forward chain that the input coordinate $\mathbf{x}$ plays in the target INR's forward pass (Eq. 3). They are produced by a sequence of n_0 learnable query tokens $\boldsymbol{Q}^{\boldsymbol{\tau}} \in \mathbb{R}^{n_0 \times d}$ cross-attending to $\mathbf{z}_\zeta$:

$$\boldsymbol{\tau}_0 \;=\; \mathrm{CrossAttn}_{\boldsymbol{\tau}}\big(\boldsymbol{Q}^{\boldsymbol{\tau}}, \mathbf{z}_\zeta, \mathbf{z}_\zeta\big) \;\in\; \mathbb{R}^{n_0 \times d}, \quad (13)$$

shown as the leftmost CA block in Fig. 1. The query tokens supply the row count n_0 — the number of input neurons of the target INR — and absorb task-specific content from $\mathbf{z}_\zeta$ through the cross-attention's keys and values. The same $\boldsymbol{\tau}_0$ is reused as the input to ϕ_1 at every iteration t, since the construction depends only on $\mathbf{z}_\zeta$.

Loss-signal Initialization $\boldsymbol{\delta}_L^t$ The loss-signal tokens $\boldsymbol{\delta}_L^t \in \mathbb{R}^{n_L \times d}$ play the role at the output of the backward block that the output-layer loss-signal $\delta_L = \partial\mathcal{L}/\partial\mathbf{h}_L$ plays in the target INR's backward pass. At the target network, δ_L depends on the prediction $\mathbf{h}_L$ and on the supervision target through the reconstruction loss $\mathcal{L}$. The hypernetwork mirrors this dependence by feeding the $\boldsymbol{\delta}_L^t$ sub-module with two inputs that carry the same two information sources in token form: the pseudo-activation $\boldsymbol{\tau}_L^t$, which is the prediction-side analogue at iteration t, and the conditioning input $\mathbf{z}_\zeta$, which is the signal-side analogue and carries the supervision target through the upstream encoder that produces $\mathbf{z}_\zeta$ from ζ. The two inputs are concatenated along the token axis, and a sequence of n_L learnable query tokens $\boldsymbol{Q}^{\boldsymbol{\delta}} \in \mathbb{R}^{n_L \times d}$ cross-attends to the concatenated sequence:

$$\boldsymbol{\delta}_L^t \;=\; \mathrm{CrossAttn}_{\boldsymbol{\delta}}\big(\boldsymbol{Q}^{\boldsymbol{\delta}}, \boldsymbol{C}^t, \boldsymbol{C}^t\big) \;\in\; \mathbb{R}^{n_L \times d}, \qquad \boldsymbol{C}^t = \big[\,\boldsymbol{\tau}_L^t \;;\; \mathbf{z}_\zeta\,\big], \quad (14)$$

shown as the concat-plus-CA block on the right of Fig. 1. The query tokens supply the row count n_L to match $\boldsymbol{\tau}_L^t$ token-by-token, and absorb both prediction-side and signal-side content from the concatenated keys and values. Unlike $\boldsymbol{\tau}_0$, this sub-module is re-evaluated at every iteration t because $\boldsymbol{\tau}_L^t$ changes between iterations as the parameter tokens are refined.

Parameter Token Initialization $\boldsymbol{\omega}_i^0$ The parameter token sequence $\boldsymbol{\omega}_i^0 \in \mathbb{R}^{n_i \times d}$ at each layer i seeds the parameter-token rollout of Eq. 9. For each layer i, a sequence of n_i learnable query tokens $\boldsymbol{Q}_i \in \mathbb{R}^{n_i \times d}$ cross-attends to $\mathbf{z}_\zeta$:

$$\boldsymbol{\omega}_i^0 \;=\; \mathrm{CrossAttn}_i\big(\boldsymbol{Q}_i, \mathbf{z}_\zeta, \mathbf{z}_\zeta\big) \;\in\; \mathbb{R}^{n_i \times d}. \tag{15}$$

The query tokens supply the row count n_i — the number of output neurons of layer i — and absorb task-specific content from $\mathbf{z}_\zeta$ through the keys and values. Each layer has its own cross-attention module and its own query tokens, so layer-specific initialization patterns are stored in the queries and the cross-attention's projections, while task-specific information enters uniformly through $\mathbf{z}_\zeta$.

This design contrasts with HyPoGen [8], which initializes all signals from a shared, signal-independent parameter vector θ^0 and relies on the rollout alone to introduce signal dependence; tying $\boldsymbol{\omega}_i^0$ to $\mathbf{z}_\zeta$ at iteration zero shortens the rollout the hypernetwork must perform from a random starting point to a task-conditioned one.

6.1.3 Adaptive Scaling for Initialization-Sensitive Target Networks

The hypernetwork $\mathcal{H}_\Omega$ controls the target-INR weights end-to-end: at every iteration t, including $t = 0$, the layer-i weight matrix is the decoded parameter token sequence as specified in Eq. 10, and the target INR is evaluated on whatever weights the hypernetwork produces. For target architectures whose stability is robust to the absolute scale of their weights, this end-to-end control raises no further issue. For target architectures with strict initialization requirements, however, the scale of decoded weights strongly affects target-network activations and training stability. We introduce an adaptive scaling module to match the scale of decoded weights to the target architecture.

SIREN [1] is the canonical such case. SIREN replaces the standard non-linearity σ in Eq. 3 with the periodic activation $\sin(\Omega\cdot)$, where Ω is a frequency hyperparameter, and is sensitive to weight initialization in a way standard ReLU MLPs are not: its standard hidden-layer initialization uses a scale $\sqrt{6/n_{i-1}}/\Omega$ to control activation and gradient statistics. We adopt this scale to initialize our learnable scaling factors. The per-layer decoder of Sec. 3.3 maps each parameter token to its corresponding weight row, a representation-level transformation that is shared across target architectures and is independent of any one architecture's initialization scheme. A separate sub-module therefore adapts decoded-weight magnitudes to the initialization scale appropriate for the target architecture.

Concretely, we introduce a per-layer adaptive scaling sub-module for SIREN-style targets. A learnable per-element scaling matrix $S_W^{(i)} \in \mathbb{R}^{n_i \times n_{i-1}}$ is initialized from $\mathcal{U}(-\beta_i, +\beta_i)$ with the bound

$$\beta_i \;=\; s_\zeta(i) \cdot \frac{\sqrt{6/n_{i-1}}}{\Omega}, \qquad s_\zeta(i) \;=\; \sqrt{\frac{\mathrm{var}_{\mathrm{target}}(i)}{\mathrm{var}\big(W_i^0\big)}}, \tag{16}$$

where $s_\zeta(i)$ is a variance-matching factor that aligns the empirical variance of the decoded weight W_i^0 with the target variance $\mathrm{var}_{\mathrm{target}}(i)$ implied by SIREN's initialization scale, deriving the multiplier from the target architecture's own initialization scheme rather than fixing it by hand. The effective layer-i weight matrix that the target INR is evaluated with becomes

$$\widetilde{W}_i \;=\; S_W^{(i)} \odot W_i, \tag{17}$$

with $\odot$ the element-wise product, and $S_W^{(i)}$ is optimized jointly with the rest of $\mathcal{H}_\Omega$. The scaling sub-module is required only for target architectures with strict initialization constraints; for target networks without such constraints it can be omitted, and the hypernetwork's end-to-end control of the decoded weights is then sufficient on its own.

Necessity of adaptive scaling. We verify that the adaptive scaling sub-module is necessary on the two SIREN-based settings of our experiments by removing $S_W^{(i)}$ and retraining $\mathcal{H}_\Omega$ end-to-end with all other components held fixed. Reconstruction PSNR drops from 27.7 to 11.8 on CelebA-HQ and from 49.3 to 8.6 on ERA5, the latter a complete collapse: the large performance loss demonstrates the practical importance of matching decoded-weight scales to SIREN in these settings.

Adaptive scaling restores trainability of LDMI across widths. A complementary line of evidence comes from porting our adaptive scaling sub-module onto a different hypernetwork that targets SIREN INRs. Using the width-scaling setting of Fig. 4b (CelebA-HQ at batch size 64, target-INR width swept from 128 to 1024), we compare the original LDMI [5] with an LDMI variant in which $S_W^{(i)}$ from Eq. 17 is inserted in front of each decoded SIREN weight matrix and trained jointly with the rest of the network, all other components held fixed. Original LDMI trains successfully at widths 128 and 768, but collapses at widths 384, 512, and 1024, consistent with the instability observed when removing $S_W^{(i)}$ from our hypernetwork. With our adaptive scaling sub-module attached, LDMI trains stably at every width in the sweep, and PSNR rises monotonically from 20.4 at width 128 to 26.1 at width 1024 (Table 6). The result shows that adaptive scaling transfers beyond our hypernetwork: matching the initialization scale of a SIREN target improves trainability across widths, enabling LDMI to benefit from increased target capacity.

Table 6: Width-scaling stability of LDMI with and without our adaptive scaling sub-module on CelebA-HQ at batch size 64, with SIREN as the target INR. PSNR (dB) is reported; "–" marks runs that diverge at iteration zero. The original LDMI fails on three of the five widths in the sweep, while LDMI augmented with $S_W^{(i)}$ from Eq. 17 trains across the full range.

Target-INR width	128	384	512	768	1024
LDMI (original)	22.1	–	–	25.3	–
LDMI + Adaptive Scaling	20.4	22.8	25.2	25.3	26.1

6.1.4 Step-Size Sub-Module

The refinement update of Eq. 9 multiplies the parameter-gradient tokens by a scalar step size λ_i^t, which the main text introduces as a sub-module of the hypernetwork that takes $\mathbf{z}_\zeta$ as input. This appendix specifies that sub-module. We drop the iteration superscript t; the construction repeats identically at every iteration.

Eq. 9 unrolls test-time optimization into T feed-forward steps, but unlike standard gradient descent the hypernetwork is amortized over signals: different ζ traverse different optimization trajectories, and the step size that works for one signal does not work for another (high-frequency signals require smaller steps, smoother signals can take larger steps). A fixed scalar would underfit some signals and oscillate on others. We therefore predict λ per-signal and per-iteration, from inputs that are available at the corresponding point in the rollout.

By analogy with the classical picture, a good step size depends on the geometry of the task (loss-surface curvature, anisotropy) and the current iterate's location in parameter space. We supply the sub-module with one input for each: $\mathbf{z}_\zeta$ encodes the per-signal conditioning that determines the task geometry, and $\boldsymbol{\omega}_i$ encodes the current parameter-token state of the layer being updated.

The sub-module concatenates $\mathbf{z}_\zeta$ and $\boldsymbol{\omega}_i$ along the token axis, applies self-attention to the joint sequence, and reduces the output to a scalar through two MLPs:

$$\lambda_i \;=\; \mathrm{MLP}_{\lambda,2}\Big(\mathrm{vec}\big(\mathrm{MLP}_{\lambda,1}(\boldsymbol{a}_i)\big)\Big), \qquad \boldsymbol{a}_i \;=\; \mathrm{SelfAttn}\big([\mathbf{z}_\zeta;\, \boldsymbol{\omega}_i]\big). \tag{18}$$

Self-attention lets every $\mathbf{z}_\zeta$ token attend to every $\boldsymbol{\omega}_i$ token and vice versa, so the predicted step size is informed jointly by the conditioning signal and the current iterate.

6.1.5 Forward-Block Correspondence with a Standard INR Layer

This section elaborates on the claim in Sec. 3.4 that the cross-attention module ϕ_i derived from information-flow consistency reproduces the structural skeleton of a standard INR layer (Eq. 3) component by component. Table 7 summarizes these correspondences.

Linear mixing. A standard INR layer mixes the previous activation $\mathbf{h}_{i-1} \in \mathbb{R}^{n_{i-1}}$ through the rows of $W_i \in \mathbb{R}^{n_i \times n_{i-1}}$ to produce the pre-activation $W_i\mathbf{h}_{i-1} \in \mathbb{R}^{n_i}$, whose j-th component is $W_i^{(j,:)}\mathbf{h}_{i-1}$, the inner product of the j-th row of W_i with the entire previous activation. The cross-attention module ϕ_i in Eq. 6 produces a score matrix $\boldsymbol{\omega}_i\boldsymbol{\tau}_{i-1}^\top/\sqrt{d}$, whose (j,k)-entry is

the inner product $\boldsymbol{\omega}_{i,j}^{\top}\boldsymbol{\tau}_{i-1,k}/\sqrt{d}$ between the j-th parameter token and the k-th previous-layer pseudo-activation token. Both operations therefore combine each output neuron's parameters with previous-layer activations through inner products. The factor of $1/\sqrt{d}$ is a normalization specific to attention that has no counterpart in a standard layer; it does not affect the structural correspondence.

Non-linearity. A standard INR layer applies the activation function σ element-wise to $W_i\mathbf{h}_{i-1}$, producing $\mathbf{h}_i = \sigma(W_i\mathbf{h}_{i-1})$. The cross-attention module ϕ_i applies a row-wise softmax to the score matrix. Both operations introduce a non-linearity between the linear mixing and the next stage of the computation; the softmax differs from σ in its normalization (it produces a convex combination, σ does not) and in its row-wise rather than element-wise structure, but both serve the structural role of a non-linearity in the layer.

Output-unit alignment. A standard INR layer satisfies an output-unit alignment property: the j-th component of $\mathbf{h}_i$ is produced by the j-th row of W_i alone, and consequently the j-th output neuron inherits its identity from the j-th row. The cross-attention module ϕ_i inherits the same property from the row-wise tokenization of Sec. 3.3: the j-th output token of ϕ_i is produced by the j-th row of the score matrix and, after softmax, by the row-wise convex combination of $\boldsymbol{\tau}_{i-1}$ that the j-th query token $\boldsymbol{\omega}_{i,j}$ specifies. The j-th query token is the encoding of the j-th row of W_i, so the j-th output token of ϕ_i is in one-to-one correspondence with the j-th output neuron of the target layer.

Table 7: Component-by-component correspondence between a standard INR layer (Eq. 3) and the cross-attention module ϕ_i (Eq. 6) that realizes the forward block.

	Standard layer	Cross-attention module ϕ_i
Linear mixing	$W_i\,\mathbf{h}_{i-1}$	$\boldsymbol{\omega}_i\,\boldsymbol{\tau}_{i-1}^{\top}/\sqrt{d}$
Non-linearity	activation σ	row-wise softmax
Output-unit alignment	j-th row $\to$ j-th neuron	j-th query $\to$ j-th output token

6.1.6 Per-Module QKV Derivations for the Backward Block

This section justifies the four output-representation claims that Sec. 3.4 uses to fix the QKV assignments of the backward modules $\psi_i^{\boldsymbol{\tau}}, \psi_i^{\boldsymbol{\omega}}, \psi_{i-1}^{\boldsymbol{\delta}}, \psi_i^{\nabla}$ by information-flow consistency. For each module, we identify the target-network quantity it surrogates, identify the representation that quantity's output lives in, and read off the QKV assignment.

Activation-Jacobian module $\psi_i^{\boldsymbol{\tau}}$. The module surrogates the activation Jacobian $\partial\mathbf{h}_i/\partial\mathbf{h}_{i-1}$, which appears in Eq. 4 as the linear map that transforms the layer-i loss-signal δ_i into the layer-$(i-1)$ loss-signal $\delta_{i-1} = \delta_i \cdot \partial\mathbf{h}_i/\partial\mathbf{h}_{i-1}$. The output of this contraction is in the layer-$(i-1)$ activation representation, since it is the gradient of $\mathcal{L}$ with respect to $\mathbf{h}_{i-1}$. The activation-Jacobian token sequence $\boldsymbol{J}_{i-1}^{\boldsymbol{\tau}}$ that $\psi_i^{\boldsymbol{\tau}}$ produces therefore lives in this representation. Among the module's two inputs $\boldsymbol{\omega}_i$ and $\boldsymbol{\tau}_{i-1}$, the sequence $\boldsymbol{\tau}_{i-1}$ is the layer-$(i-1)$ pseudo-activation token sequence and already lives in the required representation, while $\boldsymbol{\omega}_i$ lives in the layer-i parameter representation. By information-flow consistency, $V = K = \boldsymbol{\tau}_{i-1}$ and $Q = \boldsymbol{\omega}_i$.

Parameter-Jacobian module $\psi_i^{\boldsymbol{\omega}}$. The module surrogates the parameter Jacobian $\partial\mathbf{h}_i/\partial W_i$, which appears in Eq. 4 as the linear map that transforms δ_i into the parameter gradient $\nabla_{W_i}\mathcal{L} = \delta_i \cdot \partial\mathbf{h}_i/\partial W_i$. The output of this contraction is in the layer-i parameter representation, since it is the gradient of $\mathcal{L}$ with respect to W_i. The parameter-Jacobian token sequence $\boldsymbol{J}_i^{\boldsymbol{\omega}}$ that $\psi_i^{\boldsymbol{\omega}}$ produces therefore lives in this representation. The module's two inputs are again $\boldsymbol{\omega}_i$ and $\boldsymbol{\tau}_{i-1}$, but now $\boldsymbol{\omega}_i$ is the input that lives in the layer-i parameter representation. By information-flow consistency, $V = K = \boldsymbol{\omega}_i$ and $Q = \boldsymbol{\tau}_{i-1}$, the QKV roles are reversed relative to $\psi_i^{\boldsymbol{\tau}}$ even though the two inputs are the same.

Loss-signal-propagation module $\psi_{i-1}^{\boldsymbol{\delta}}$. The module preserves the dependency structure of the chain-rule combination $\delta_{i-1} = \delta_i \cdot \partial\mathbf{h}_i/\partial\mathbf{h}_{i-1}$ of Eq. 4, which produces the layer-$(i-1)$ loss-signal. The output token sequence $\boldsymbol{\delta}_{i-1}$ therefore lives in the layer-$(i-1)$ activation representation. The module's two inputs are $\boldsymbol{\delta}_i$ and $\boldsymbol{J}_{i-1}^{\boldsymbol{\tau}}$. The sequence $\boldsymbol{\delta}_i$ lives in the layer-i activation representation; the sequence $\boldsymbol{J}_{i-1}^{\boldsymbol{\tau}}$ was just established to live in the layer-$(i-1)$ activation representation. By information-flow consistency, $V = K = \boldsymbol{J}_{i-1}^{\boldsymbol{\tau}}$ and $Q = \boldsymbol{\delta}_i$.

Parameter-gradient module ψ_i^{∇}. The module preserves the dependency structure of the chain-rule combination $\nabla_{W_i}\mathcal{L} = \delta_i \cdot \partial \mathbf{h}_i / \partial W_i$ of Eq. 4, which produces the parameter gradient. The output token sequence $\nabla_{\boldsymbol{\omega}_i}\mathcal{L}$ therefore lives in the layer-i parameter representation. The module's two inputs are $\boldsymbol{\delta}_i$ and $\boldsymbol{J}_i^{\boldsymbol{\omega}}$. The sequence $\boldsymbol{\delta}_i$ lives in the layer-i activation representation; the sequence $\boldsymbol{J}_i^{\boldsymbol{\omega}}$ was just established to live in the layer-i parameter representation. By information-flow consistency, $V = K = \boldsymbol{J}_i^{\boldsymbol{\omega}}$ and $Q = \boldsymbol{\delta}_i$.

Summary. The four QKV assignments derived above are collected in Table 8 and match the color coding in Fig. 1. Each assignment follows from information-flow consistency once the output representation is identified from the target-network quantity the module surrogates. The same rule therefore specifies all four modules, and the QKV ablation in Table 4 supports these assignments.

Table 8: QKV assignments of the four backward modules, derived from information-flow consistency given the output representation each module's target-network counterpart lives in.

Module	Surrogate of	Output representation	$V = K$	Q
$\psi_i^{\boldsymbol{\tau}}$	$\partial \mathbf{h}_i / \partial \mathbf{h}_{i-1}$	layer-$(i-1)$ activation	$\boldsymbol{\tau}_{i-1}$	$\boldsymbol{\omega}_i$
$\psi_i^{\boldsymbol{\omega}}$	$\partial \mathbf{h}_i / \partial W_i$	layer-i parameter	$\boldsymbol{\omega}_i$	$\boldsymbol{\tau}_{i-1}$
$\psi_{i-1}^{\boldsymbol{\delta}}$	$\delta_i \cdot \partial \mathbf{h}_i / \partial \mathbf{h}_{i-1}$	layer-$(i-1)$ activation	$\boldsymbol{J}_{i-1}^{\boldsymbol{\tau}}$	$\boldsymbol{\delta}_i$
ψ_i^{∇}	$\delta_i \cdot \partial \mathbf{h}_i / \partial W_i$	layer-i parameter	$\boldsymbol{J}_i^{\boldsymbol{\omega}}$	$\boldsymbol{\delta}_i$

6.1.7 Training and Inference Procedure

The pseudocode below summarizes the training and inference procedure.

Algorithm 1 Signal-conditioned INR generation

Input: signal ζ, rollout length T.
Output: target-INR parameters θ_ζ.
  $\mathbf{z}_\zeta \leftarrow E(\zeta)$.
  Initialize $\boldsymbol{\tau}_0$ and $\{\boldsymbol{\omega}_i^0\}_{i=1}^L$ from $\mathbf{z}_\zeta$.
  for $t = 0, \ldots, T-1$ **do**
    Forward chain ($i = 1, \ldots, L$): compute pseudo-activation and local Jacobian tokens.
    Initialize $\boldsymbol{\delta}_L^t$ from $[\boldsymbol{\tau}_L^t; \mathbf{z}_\zeta]$.
    Backward chain ($i = L, \ldots, 1$): compute $\boldsymbol{\delta}_{i-1}^t$ and $\nabla_{\boldsymbol{\omega}_i}\mathcal{L}^t$.
    Predict λ_i^t from $[\mathbf{z}_\zeta; \boldsymbol{\omega}_i^t]$ for each layer i.
    $\boldsymbol{\omega}_i^{t+1} \leftarrow \boldsymbol{\omega}_i^t - \lambda_i^t \nabla_{\boldsymbol{\omega}_i}\mathcal{L}^t$ for all i.
  end for
  $W_i \leftarrow \mathrm{Decode}_i(\boldsymbol{\omega}_i^T)$; for SIREN, use $\widetilde{W}_i = S_W^{(i)} \odot W_i$.
  Training: compute $\mathcal{L}_{\mathrm{total}}$ and backpropagate through generation to update trainable components.
  Inference: return θ_ζ.

6.2 Experimental Setup

6.2.1 Baseline Methods

We compare our approach against six baselines, which we organize into two groups based on whether their hypernetwork architecture is optimization-oriented (Sec. 3.1). HyPoGen is the only prior work whose architecture mirrors the structure of test-time optimization, and our method belongs to the same group. The remaining five baselines—LDMI, Trans-INR, GINR-IPC, VaMoH, and HyperDiffusion—generate INR parameters without such mirroring, through single-pass mappings, modulation, variational inference, or weight-space diffusion.

Optimization-oriented baseline. **HyPoGen [8]** is the immediate predecessor of our method and the only prior work whose hypernetwork architecture is explicitly biased toward an optimization process. Originally proposed for generalizable policy generation in robot manipulation, HyPoGen synthesizes target-network parameters from a task specification by approximating an optimization process executed over a finite number of steps in a compressed latent space, with each step instantiated by MLP sub-modules that follow the chain-rule structure of the forward and backward computation. We adopt its optimization-oriented principle as our starting point and replace the MLP body with attention to overcome the quartic-in-width scalability barrier identified in Sec. 3.1.

Non-optimization-oriented baselines. **LDMI [5]** extends latent diffusion models to INR generation by replacing the standard convolutional decoder with a Transformer-based hyper-decoder that produces INR parameters from latent variables in a single-pass mapping. As a strong attention-based hypernetwork without an optimization-oriented design, LDMI is the direct counterpart for isolating the contribution of the optimization-oriented principle in our method.

Trans-INR [4] uses a Transformer as a meta-learner to produce the entire set of INR weights from input observations through a set-to-set mapping. Although its design is motivated by gradient-based meta-learning, the resulting architecture predicts the parameters in a single forward pass without explicitly mirroring the steps of test-time optimization. We include it as an early Transformer-based hypernetwork baseline on the Pre-trained NeRF setting.

GINR-IPC [45] takes a different approach to amortization: instead of generating the full INR weights, it modulates only the weights of an early MLP layer of a coordinate-based network as an instance-specific pattern, while the remaining layers learn shared pattern-composition rules across instances. By restricting weight generation to a low-dimensional modulation, GINR-IPC achieves strong generalization across images, audio, and 3D objects, and we include it as a representative modulation-based hypernetwork baseline on Pre-trained NeRF.

VaMoH [6] couples INR-based variational autoencoders with a mixture of hypernetworks, using a normalizing flow as the prior over the latent space, in order to learn distributions over continuous functions. We include it as a representative variational baseline for function-space generative modeling on the 2D datasets and ShapeNet Chairs.

HyperDiffusion [7] casts INR generation as a generative-modeling problem in weight space. It first overfits a separate MLP to each data instance to obtain a dataset of pre-trained INR weights, and then trains a Transformer diffusion model directly on these weights to model their distribution. The original method is unconditional and was demonstrated on 3D shapes and 4D mesh animations. We include HyperDiffusion as a baseline only on the Pre-trained NeRF [62] dataset, because it is the only dataset in our experimental setup that provides a collection of pre-trained INR weights—one individually optimized NeRF per object—which is a prerequisite for HyperDiffusion's weight-space diffusion process. The other four datasets (CelebA-HQ, ImageNet, ERA5, and ShapeNet Chairs) only provide raw signals (images, climate fields, or occupancy grids), and obtaining a per-instance INR weight dataset on them would require running the very test-time optimization procedure that hypernetwork-based INR generation aims to amortize away in the first place, defeating the purpose of the comparison. To enable a fair comparison under our setup, we adapt HyperDiffusion to the conditional setting by employing a DINO encoder [74] to provide image-based conditioning signals, and instantiate it under two of our three supervision settings as follows. Under *weights only* supervision, we recover HyperDiffusion's original training scheme: the diffusion model is trained to denoise toward the ground-truth NeRF weights. Under *weights & images* supervision, we additionally render novel views from the generated weights using the target NeRF and add an image

reconstruction loss against the ground-truth views, on top of the original weight-supervised diffusion objective. The *images only* supervision setting is fundamentally inapplicable to HyperDiffusion: its training objective requires ground-truth weights as the diffusion target, and removing them leaves the diffusion process without any signal to define what the model should denoise toward. We therefore report "N/A" for HyperDiffusion under the images-only setting in Table 2.

6.2.2 Implementation Details

Training Configuration All experiments were conducted on NVIDIA H800 GPUs with mixed precision training. We employ the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and a fixed learning rate of 5×10^{-5} across all datasets without learning rate scheduling.

Architecture Specifications Our hypernetwork architecture scales with dataset complexity, as shown in Table 9. The task tokenizer dimension and hidden dimensions are adjusted based on the signal complexity: simpler datasets like ShapeNet Chairs use 64-dimensional representations, while complex datasets like Pre-trained NeRF require 768 dimensions.

Table 9: Model hyperparameter across different datasets.

	CelebA-HQ 64×64	ImageNet	ERA5	Chairs	Pre-trained NeRF
Learning rate	5e-5	5e-5	5e-5	5e-5	5e-5
Optimizer	Adam	Adam	Adam	Adam	Adam
Mini-batch size	64	128	128	32	96
Task Tokenizer dim	128	256	128	64	768
Each Block Hidden dim	128	768	128	64	768
Transformer layers	1	1	1	1	1
Token len	384,257,257,257,4	384,257,257,257,4	342,257,257,257,2	171,129,2	64,64,64,16
INR type	SIREN	SIREN	SIREN	SIREN	NeRF
INR layers	5	5	5	3	3
INR hidden dim	256	256	256	128	64
INR Ω / encode frequency	30	30	30	30	24

6.2.3 Encoder Architectures

For each dataset, the encoder maps an input signal ζ to its conditioning input $\mathbf{z}_\zeta$, from which the hypernetwork generates the target INR parameters θ_ζ. The encoder is shared across all compared methods on the same dataset. We follow the encoder choices of LDMI [5]: a 2D convolutional VAE for CelebA-HQ and ERA5, a pre-trained and frozen VQ-VAE [75] for ImageNet, and a 3D convolutional VAE for ShapeNet Chairs. For Pre-trained NeRF, we use a DINO ViT-B/16 encoder [74] with per-view camera-pose conditioning injected through DiT-style adaLN-Zero modulation [76]; the encoder is initialized from the public self-supervised DINO checkpoint and fine-tuned end-to-end on $N{=}6$ conditional views per object, whose token sequences are concatenated to form the conditioning input. Per-dataset specifications are summarized in Table 10.

Table 10: Encoder used for each dataset.

Dataset	Encoder
CelebA-HQ	2D Conv VAE
ERA5	2D Conv VAE
ImageNet	VQ-VAE (pre-trained, frozen)
ShapeNet Chairs	3D Conv VAE
Pre-trained NeRF	DINO ViT-B/16 + adaLN-Zero pose cond.

6.2.4 Loss Function

For CelebA-HQ, ImageNet, ERA5, and ShapeNet Chairs, we follow the training losses of LDMI. For Pre-trained NeRF, we use L_1 losses on images and weights.

Notation. N denotes the number of samples per mini-batch (for losses computed over a patch-discriminator's output, N is the total number of patch positions across the batch). The hyperparameters λ_{kl}, λ_{disc}, $\lambda_{\text{codebook}}$, $\lambda_{\text{perceptual}}$, λ_{weight}, and λ_{img} scale the corresponding loss terms. The reconstruction loss $\mathcal{L}_{\text{rec}}$ defined here is the per-instance loss $\mathcal{L}$ used in Section 3.2 for the chain-rule derivation; the dataset-level training objective $\mathcal{L}_{\text{total}}$ is its sum (with KL/adversarial/codebook terms) over a mini-batch. To avoid clashes with the activation σ used in Eq. 3, we write the likelihood variance as s^2 and the sigmoid function as $\text{sig}(\cdot)$.

For the GAN-trained datasets (CelebA-HQ and ImageNet), we additionally use an *adaptive discriminator weight* β and a *warmup gate* γ defined as

$$\beta = \lambda_{\text{disc}} \cdot \text{clip}\left(\frac{\|\nabla_{\theta_L}\mathcal{L}_{\text{rec}}\|}{\|\nabla_{\theta_L}\mathcal{L}_{\text{g}}\| + 10^{-4}},\, 0,\, 10^4\right), \qquad \gamma = \mathbf{1}[\text{global_step} \geq \text{threshold}], \tag{19}$$

where θ_L denotes the parameters of the last decoder layer [77]. We write $\text{Weight}_{\text{adj}} := \gamma \cdot \beta$ for the combined weight applied to the generator's adversarial loss; the discriminator is gated by γ alone.

CelebA-HQ. For the generator,

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{rec}} + \lambda_{\text{kl}} \cdot \mathcal{L}_{\text{kl}} + \text{Weight}_{\text{adj}} \cdot \mathcal{L}_{\text{g}}^{\text{hinge}}. \tag{20}$$

The reconstruction loss is a per-pixel weighted Gaussian negative log-likelihood (with learned variance) whose numerator combines pixel and perceptual terms,

$$\mathcal{L}_{\text{rec}} = \frac{1}{N}\sum_{i=1}^{N} \alpha_i \cdot \left(\frac{|x_i - \hat{x}_i| + \lambda_{\text{perceptual}} \cdot \text{LPIPS}(x_i, \hat{x}_i)}{\exp(\log s^2)} + \log s^2\right), \tag{21}$$

between the real input image x_i and the reconstruction $\hat{x}_i$, where s^2 is the (learned) likelihood variance and α_i is the per-sample weight. The KL divergence loss is

$$\mathcal{L}_{\text{kl}} = \frac{1}{N}\sum_{i=1}^{N} \text{KL}(q(z_i) \,\|\, p(z_i)), \tag{22}$$

between the posterior $q(z_i)$ and prior $p(z_i)$ of the latent z_i. The generator hinge loss is

$$\mathcal{L}_{\text{g}}^{\text{hinge}} = -\frac{1}{N}\sum_{i=1}^{N} \text{logits_fake}_i, \tag{23}$$

where logits_fake_i is the discriminator's pre-sigmoid logit on the reconstruction $\hat{x}_i$. For the discriminator,

$$\mathcal{L}_{\text{discriminator}} = \gamma \cdot \mathcal{L}_{\text{d}}^{\text{hinge}}, \qquad \mathcal{L}_{\text{d}}^{\text{hinge}} = \tfrac{1}{2}\big(\mathcal{L}_{\text{real}} + \mathcal{L}_{\text{fake}}\big), \tag{24}$$

with $\mathcal{L}_{\text{real}} = \frac{1}{N}\sum_{i=1}^{N} \max(0, 1 - \text{logits_real}_i)$ and $\mathcal{L}_{\text{fake}} = \frac{1}{N}\sum_{i=1}^{N} \max(0, 1 + \text{logits_fake}_i)$, and logits_real_i, logits_fake_i the discriminator's pre-sigmoid logits on x_i and $\hat{x}_i$. The discriminator update is gated only by the warmup indicator γ; neither λ_{disc} nor β is applied here.

ImageNet. The total loss comprises an autoencoder loss and a discriminator loss. The autoencoder loss combines reconstruction, codebook, and adversarial terms,

$$\mathcal{L}_{\text{autoencoder}} = \mathcal{L}_{\text{rec}} + \lambda_{\text{codebook}} \cdot \mathcal{L}_{\text{codebook}} + \text{Weight}_{\text{adj}} \cdot \mathcal{L}_{\text{g}}^{\text{hinge}}. \tag{25}$$

The reconstruction loss combines pixel-wise and perceptual terms,

$$\mathcal{L}_{\text{rec}} = \mathcal{L}_{\text{pixel}} + \lambda_{\text{perceptual}} \cdot \mathcal{L}_{\text{perceptual}}, \tag{26}$$

with $\mathcal{L}_{\text{pixel}} = \|x - y\|_1$ the L_1 pixel loss between the original image x and the reconstruction y, and $\mathcal{L}_{\text{perceptual}} = \text{LPIPS}(x, y)$ the LPIPS distance. The codebook loss regularizes the discrete latent space,

$$\mathcal{L}_{\text{codebook}} = \|\mathbf{z}_e - \mathbf{e}\|_2^2, \tag{27}$$

where $\mathbf{z}_e$ is the encoder's continuous latent and $\mathbf{e}$ is its nearest codebook entry. The generator hinge loss $\mathcal{L}_{\text{g}}^{\text{hinge}}$ is defined as in the CelebA-HQ paragraph above. The discriminator loss is

$$\mathcal{L}_{\text{discriminator}} = \gamma \cdot \mathcal{L}_{\text{d}}^{\text{hinge}}, \tag{28}$$

with the hinge form $\mathcal{L}_{\text{d}}^{\text{hinge}} = \frac{1}{2}(\mathcal{L}_{\text{real}} + \mathcal{L}_{\text{fake}})$ identical to the CelebA-HQ definition. As in the CelebA-HQ case, the discriminator update is gated only by γ.

ERA5.

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{rec}} + \lambda_{\text{kl}} \cdot \mathcal{L}_{\text{kl}}, \qquad \mathcal{L}_{\text{rec}} = \frac{1}{N} \sum_{i=1}^{N} \left(\frac{|x_i - \hat{x}_i|}{\exp(\log s^2)} + \log s^2 \right), \tag{29}$$

where x_i and $\hat{x}_i$ are the original and reconstructed temperature fields, s^2 is the (learned) likelihood variance, and $\mathcal{L}_{\text{kl}}$ is the KL divergence defined above. ERA5 uses neither per-sample weighting nor a perceptual term.

ShapeNet Chairs.

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{rec}} + \lambda_{\text{kl}} \cdot \mathcal{L}_{\text{kl}}, \tag{30}$$

with the per-voxel binary cross-entropy reconstruction loss

$$\mathcal{L}_{\text{rec}} = -\frac{1}{N} \sum_{i=1}^{N} \alpha_i \cdot \big(x_i \log \text{sig}(\ell_i) + (1 - x_i) \log\big(1 - \text{sig}(\ell_i)\big) \big), \tag{31}$$

where $x_i \in \{0, 1\}$ is the binary occupancy of voxel i, ℓ_i is the predicted logit, and $\text{sig}(\cdot)$ is the sigmoid function. The factor α_i is an optional per-voxel weight that can be supplied to compensate for the imbalance between occupied and empty voxels; in our default ShapeNet configuration no such weight is passed and $\alpha_i \equiv 1$.

Pre-trained NeRF.

$$\mathcal{L}_{\text{total}} = \lambda_{\text{weight}} \cdot \mathcal{L}_{\text{weight}} + \lambda_{\text{img}} \cdot \mathcal{L}_{\text{img}}. \tag{32}$$

The weight loss is the L_1 distance between the predicted and ground-truth NeRF parameters,

$$\mathcal{L}_{\text{weight}} = \frac{1}{N} \sum_{i=1}^{N} \big\| \theta_i^{\text{pred}} - \theta_i^{\text{gt}} \big\|_1, \tag{33}$$

and the image loss is the L_1 distance between rendered and ground-truth novel-view RGB,

$$\mathcal{L}_{\text{img}} = \frac{1}{N} \sum_{i=1}^{N} \big\| \hat{\mathbf{c}}_i - \mathbf{c}_i^{\text{gt}} \big\|_1, \tag{34}$$

where θ_i^{pred} and θ_i^{gt} are the hypernetwork-predicted and ground-truth NeRF weight vectors of the i-th object, and $\hat{\mathbf{c}}_i$ and $\mathbf{c}_i^{\text{gt}}$ are the rendered and ground-truth RGB at the corresponding novel views. For weight-only supervision, $\lambda_{\text{weight}} = 1$ and $\lambda_{\text{img}} = 0$; for image-only supervision, $\lambda_{\text{weight}} = 0$ and $\lambda_{\text{img}} = 1$; for combined supervision, $\lambda_{\text{weight}} = 0.1$ and $\lambda_{\text{img}} = 1$.

6.3 Additional Results and Ablations

6.3.1 Additional Qualitative results

We provide additional visual comparisons on all five datasets: CelebA-HQ (Figure 5), ERA5 (Figure 6), ImageNet (Figure 7), ShapeNet Chairs (Figure 8), and Pre-trained NeRF (Figure 9). In each figure, rows from top to bottom correspond to HyPoGen, LDMI, Ours, and the ground truth (GT), while columns show different samples. Across all datasets, our method produces reconstructions that most closely match the ground truth.

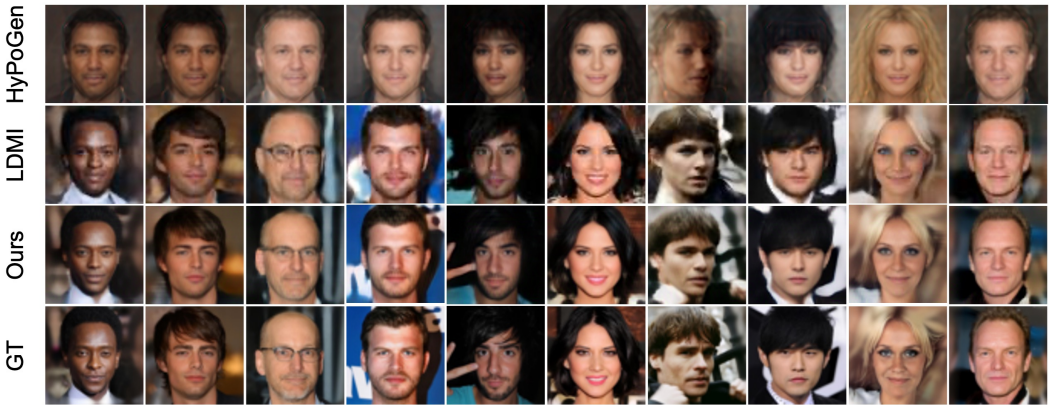


Figure 5: Additional visual comparison on CelebA-HQ.

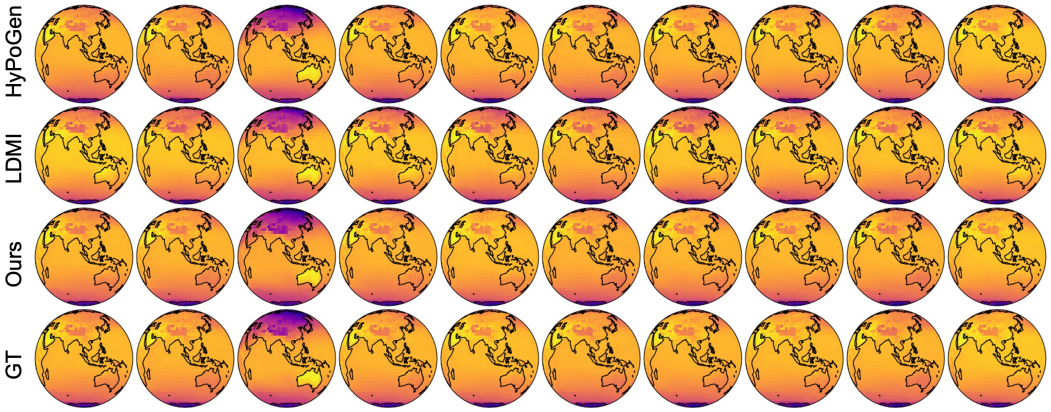


Figure 6: Additional visual comparison on ERA5.

6.3.2 Additional Quantitative Results

Additional NeRF comparisons. We further compare against two generalizable INR methods, Trans-INR [4] and GINR-IPC [45], on the Pre-trained NeRF dataset (Table 11). Our method outperforms both across all three supervision settings.

Table 11: Additional INR-generation baselines on the Pre-trained NeRF dataset under three supervision settings. Our method outperforms both Trans-INR and GINR-IPC across all settings.

Supervision	Metric	Trans-INR	GINR-IPC	**Ours**
Weights only	PSNR ↑	16.5	17.1	**22.0**
Images only	PSNR ↑	21.9	26.1	**28.8**
Weights & Images	PSNR ↑	16.6	18.2	**27.8**
	SSIM ↑	0.8659	0.8899	**0.948**
	LPIPS ↓	0.2618	0.1855	**0.0742**

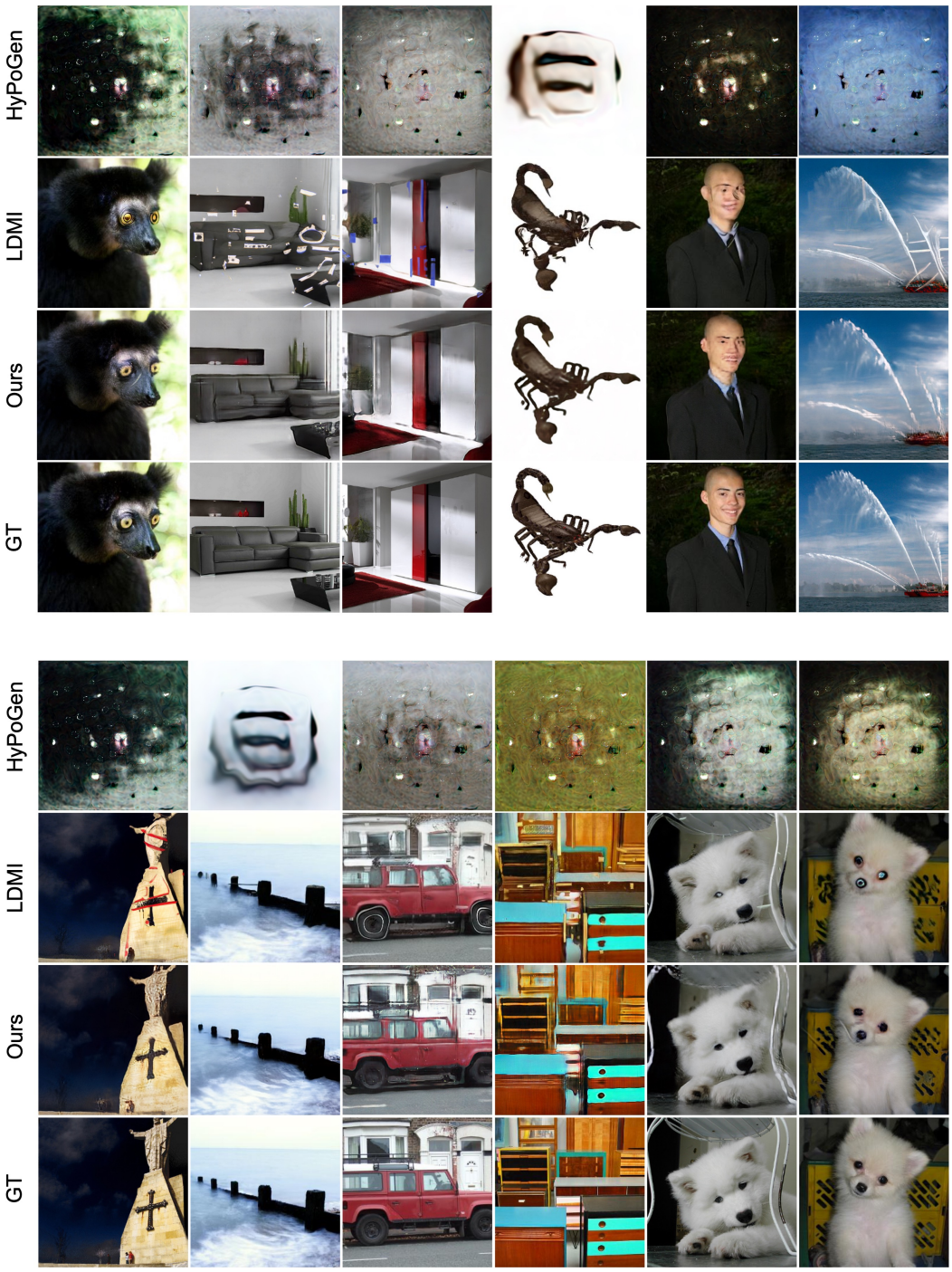


Figure 7: Additional visual comparison on ImageNet.

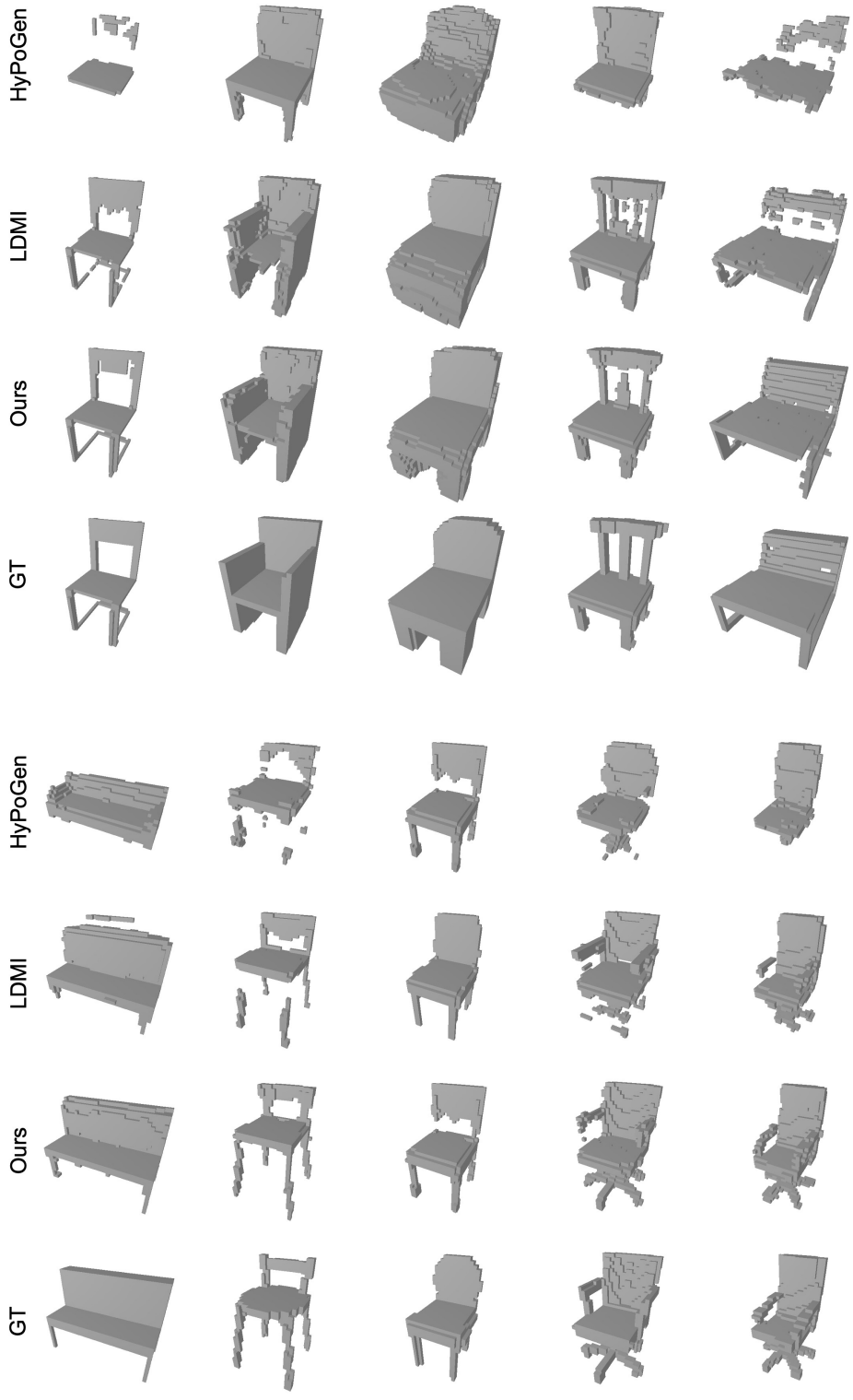


Figure 8: Additional visual comparison on ShapeNet Chairs.

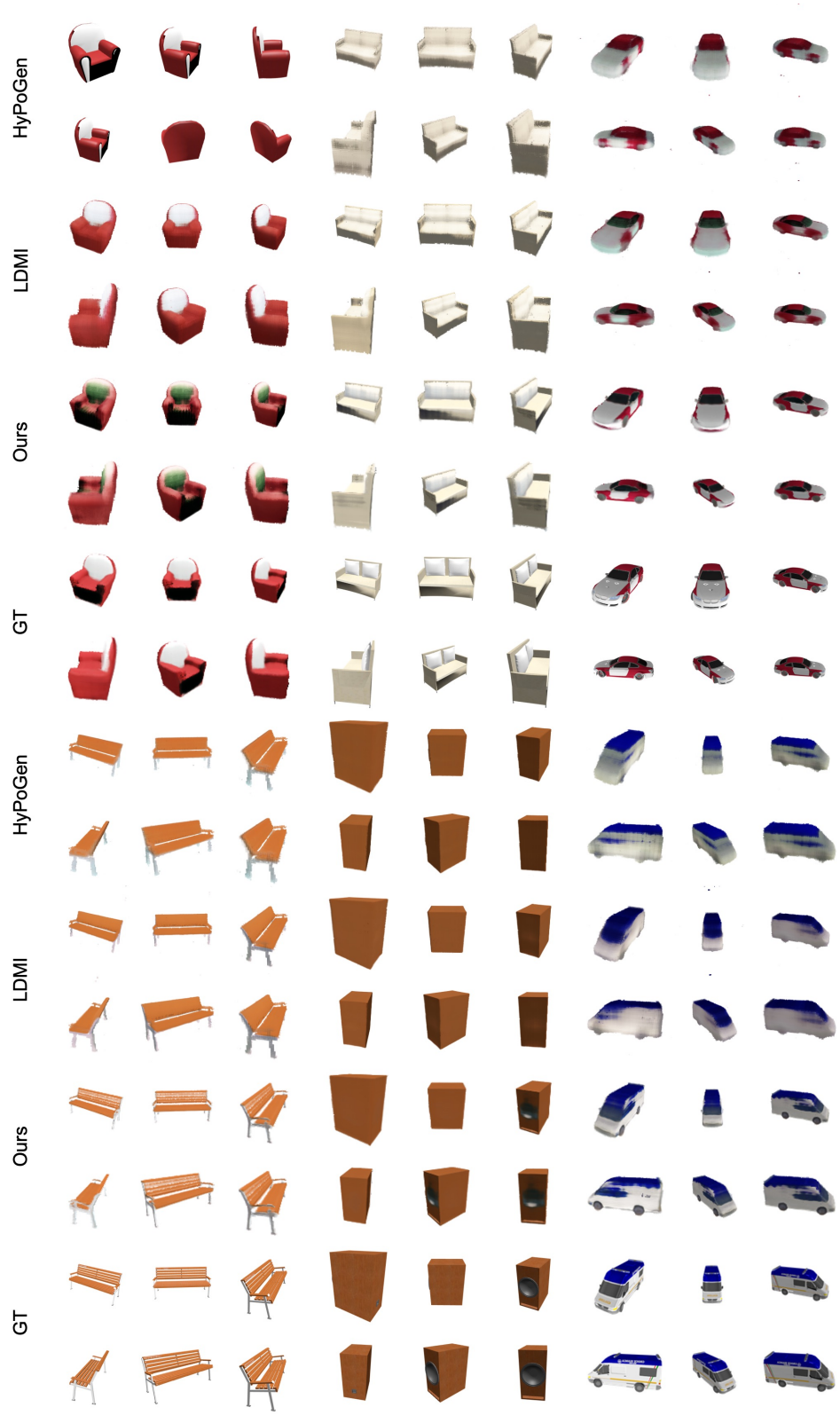


Figure 9: Additional visual comparison on Pre-trained NeRF.

LPIPS on ImageNet. For completeness, we report LPIPS on ImageNet, which was omitted from Table 1 due to space constraints: VAMoH 0.4128, HyPoGen 0.6141, LDMI **0.1468**, and Ours 0.1636. LDMI achieves slightly lower LPIPS than our method on this dataset, while our method consistently outperforms all baselines on PSNR and SSIM.

6.3.3 Additional Ablations

Forward chain. **w/o Forward Chain** removes the recursive application of ϕ_i along $i = 1, \ldots, L$ and replaces the pseudo-activation tokens $\boldsymbol{\tau}_i$ with learnable token sequences that do not depend on $\boldsymbol{\tau}_{i-1}$, while keeping the backward block, the rollout depth T, and the parameter count comparable (Table 12). PSNR drops by 2.8 / 4.4 / 3.2 dB. The drop is the cost of removing the token-space analogue of Eq. 3: when $\boldsymbol{\tau}_i$ no longer carries the layer-wise activation history that ϕ_i would have produced from $\boldsymbol{\tau}_{i-1}$ and $\boldsymbol{\omega}_i$, the backward block must predict its learned surrogates without the same forward-layer context.

Backward block decomposition. **w/o Backward Structure** replaces the four role-specific backward modules of Section 3.4 ($\psi_i^{\boldsymbol{\tau}}$, $\psi_i^{\boldsymbol{\omega}}$, $\psi_{i-1}^{\boldsymbol{\delta}}$, and ψ_i^{∇}) with four identical $\Delta\boldsymbol{\omega}$-generating cross-attentions stacked sequentially, matched in parameter count (Table 12). PSNR drops by 3.3 / 4.8 / 3.6 dB, showing that what drives the backward block's contribution is not raw attention capacity alone, but the chain-rule-structured decomposition into the four computational roles of Eqs. 7–8. Removing the decomposition removes the structure that the optimization-oriented principle requires.

Table 12: Ablations on the structural blocks of the iterative module of Sec. 3.4. Each variant disables one structural block—the forward chain, the role-specific backward decomposition, while keeping all other components and total compute matched to the full model. PSNR (↑) on CelebA-HQ, ImageNet, and Pre-trained NeRF (*Weights & Images*).

	CelebA-HQ		**ImageNet**		**NeRF**	
Variant	Params	PSNR	Params	PSNR	Params	PSNR
Full model	41.2	27.7	124.5	22.9	482.5	27.8
w/o Forward module	33.1	24.9	101.2	18.5	441.9	24.6
w/o Backward module	38.4	24.4	110.1	18.1	456.8	24.2

Number of refinement steps T. Table 13 compares $T = 0, 1, 3$ on ImageNet and Pre-trained NeRF. The initialization-only variant ($T = 0$) removes the iterative module while retaining signal-conditioned initialization and decoding; the single-step variant ($T = 1$) performs one forward–backward pass before decoding. Without refinement, PSNR is 9.31 dB on ImageNet and 20.93 dB on NeRF. The full three-step model improves these results by 13.59 and 6.87 dB, respectively, demonstrating the contribution of refinement beyond initialization and decoding. A single step already reaches 22.1 and 26.7 dB; increasing T from one to three further improves PSNR by 0.8 and 1.1 dB. Together, these results show that refinement accounts for a substantial part of the reconstruction quality, with successive updates providing further gains.

Table 13: Ablations on the number of refinement steps in the iterative module of Sec. 3.4. PSNR (↑) on ImageNet and Pre-trained NeRF (*Weights & Images*).

Variant	ImageNet	NeRF
Initialization Only (T=0)	9.31	20.93
Single-Step Rollout (T=1)	22.1	26.7
Ours (Full, T=3)	**22.9**	**27.8**

6.3.4 Amortized Generation versus Per-Instance Optimization

We compare mean PSNR and total generation time on 100 test instances per dataset, starting from the same signal-conditioned initialization (Table 14). Our hypernetwork generates all 100 INRs in one batched forward pass with three internal refinement steps, while the SGD baseline performs iterative optimization independently for each instance. On Pre-trained NeRF, our model generates the 100

NeRFs in 0.54 s at 22.11 dB, compared with 610.99 s at 15.45 dB for 100-step SGD. The same quality and efficiency advantage holds on ImageNet, illustrating the benefit of amortizing INR generation across signals.

Table 14: Mean PSNR and total generation time for 100 test instances per dataset. Both methods use the same signal-conditioned initialization; ours generates all 100 INRs in one batched forward pass, whereas SGD optimizes each instance independently.

		ImageNet		Pre-trained NeRF	
Method	Steps	Mean PSNR (dB)	Total time (s)	Mean PSNR (dB)	Total time (s)
True-gradient SGD	1	9.17	11.78	14.95	14.25
True-gradient SGD	3	9.23	18.98	14.97	20.38
True-gradient SGD	10	9.64	27.42	15.11	35.90
True-gradient SGD	100	14.14	80.15	15.45	610.99
Ours	3	19.35	0.0975	22.11	0.5375

6.3.5 Reconstruction Across the Rollout

Reconstruction across the rollout. We decode all four rollout states on the 20 ImageNet validation images used in the direction study (Table 5). Mean reconstruction MSE falls at each step, from 0.72 at initialization to 0.04 after the third update (Table 15).

Table 15: Reconstruction of 20 ImageNet validation images at each decoded rollout state.

Decoded state	MSE ↓	PSNR (dB) ↑
T_0	0.72	7.6
T_1	0.19	13.3
T_2	0.10	16.5
T_3	0.04	20.0

6.3.6 Stability Across Random Seeds

To assess robustness with respect to random initialization, we evaluate our method across multiple random seeds on three datasets that span the diversity of our experimental setup: Pre-trained NeRF (3D radiance fields), CelebA-HQ (2D natural images), and ERA5 (2D climate data). Table 16 reports PSNR for five different seeds, along with mean and standard deviation. The variability captured corresponds to random-initialization differences while keeping all other experimental conditions fixed.

Our method exhibits strong consistency across all three datasets. On Pre-trained NeRF, PSNR remains within 27.7–27.8 dB (mean: 27.76 ± 0.05). On CelebA-HQ, the standard deviation is 0.10 dB. On ERA5, PSNR ranges from 48.9 to 49.5 dB (mean: 49.26 ± 0.23). The observed standard deviations (0.05–0.23 dB) are an order of magnitude smaller than the gaps between our method and the baselines reported in Tables 1 and 2, indicating that the improvements are not attributable to seed variance.

Table 16: PSNR stability across random seeds on multiple datasets. Each row reports our method's PSNR under five independent training runs that differ only in the random seed used for initialization, together with the mean and standard deviation. NeRF results are reported under the *Weights & Images* supervision setting.

Dataset	Seed 444	Seed 555	Seed 777	Seed 888	Seed 999	Mean $\pm$ Std
Pre-trained NeRF	27.7	27.8	27.7	27.8	27.8	27.76 ± 0.05
CelebA-HQ	27.8	27.7	27.7	27.5	27.6	27.66 ± 0.10
ERA5	48.9	49.3	49.4	49.2	49.5	49.26 ± 0.23

6.3.7 Unseen Conditioning-Input Resolutions

We examine the effect of changing the conditioning-image resolution at test time, without retraining the hypernetwork.

CelebA-HQ encoder restriction. We inherit the LDMI conditioning encoder for the controlled CelebA-HQ comparison. Although its convolutional VAE is fully convolutional, the subsequent LatentTokenizer has learned positional embeddings tied to the 64-token training grid. Inputs at 128×128 and 256×256 would produce 256 and 1,024 tokens and cannot be processed by the unmodified checkpoint. We therefore do not report an unseen-input-resolution result for this checkpoint; doing so would require modifying the inherited tokenizer.

Variable-resolution NeRF conditioning. Our DINO-based NeRF encoder bicubically interpolates positional embeddings across patch grids. We evaluate a checkpoint trained at 112×112, without retraining, at 56×56 and native 224×224. The generated NeRF is rendered and evaluated at the matching resolution. Table 17 averages PSNR over 10 held-out objects. Time and peak memory are measured at batch size 1, include image encoding and INR parameter generation, and exclude NeRF rendering. Conditioning-token counts refer to the complete multiview conditioning sequence. The different evaluation resolutions also change the reconstruction task, so the PSNR entries are not a controlled comparison at one fixed output resolution.

Table 17: Unseen conditioning resolutions for the NeRF checkpoint trained at 112×112.

Input resolution	Status	Tokens	PSNR (dB)	Time (ms)	Memory (GB)
56×56	Unseen	60	21.83	75.6	1.86
112×112	Training	300	28.08	80.3	1.87
224×224	Unseen, native	1,182	25.09	83.8	1.93

6.3.8 Optional Test-Time NeRF Adaptation

Using the same 10 held-out objects and unseen 224×224 inputs, we compare standard random NeRF initialization with weights generated by the hypernetwork. Only the target NeRF parameters are optimized; adaptation settings are identical between these two initializations. Table 18 reports average novel-view PSNR at 224×224. The hypernetwork was neither trained at this resolution nor explicitly meta-trained as an initialization for adaptation.

Table 18: Optional adaptation from random or generated NeRF weights, novel-view PSNR in dB.

Optimization steps	Random initialization	Hypernetwork initialization
0	16.81	25.09
1	16.84	25.22
5	16.97	25.43
10	17.24	25.60
20	18.11	25.85
100	19.74	29.06

After 100 adaptation steps, generated initialization improves by 3.97 dB and remains 9.32 dB above random initialization. In the reported implementation, random initialization requires 500 optimization steps and 8,065.49 ms per NeRF to reach 25.19 dB; generated weights reach a comparable 25.09 dB in an 83.8 ms image-to-INR forward pass. Adaptation is an optional downstream use and is excluded from the primary feed-forward generation claim.

6.3.9 Hidden-Neuron Permutation Symmetry

A hidden-neuron permutation can preserve the represented function when the corresponding parameter groups are transformed consistently. For compatible adjacent dense layers, a permutation matrix P_i acts as $W_i' = P_i W_i$, $b_i' = P_i b_i$, and $W_{i+1}' = W_{i+1} P_i^\top$. More generally, for multiple hidden layers, $W_i' = P_i W_i P_{i-1}^\top$ with input/output permutations fixed to the identity. Fixed row queries and

positional embeddings do not guarantee that our hypernetwork is equivariant to this transformation, and an element-wise weight loss can penalize functionally equivalent but differently ordered targets.

Our weight-supervised setting uses the official pre-trained NeRF weights provided by nf2vec [62]. We retain the original hidden-neuron ordering in each checkpoint, without additional permutation, canonicalization, or alignment. Canonicalization methods such as those used in DeepWeightFlow [63] remain a complementary direction.

6.4 Complexity and Scalability

6.4.1 Parameter-Count Derivation for the Core Iterative Body

This section expands the parameter-count comparison summarized in Sec. 3.1 and Sec. 3.5. We derive the per-layer per-refinement-step parameter count of HyPoGen's MLP-based core and of our attention-based core, in two formulations: in the original target-network dimensions n_i and n_{i-1}, and under the row-level compression to embedding dimension d that the parameter tokens of Sec. 3.3 implement. Throughout, we count parameters of the *core iterative body* only, that is, the sub-modules that consume a representation of W_i and produce an update of the same shape. The per-layer encoders and decoders that convert between W_i and the token representation are shared between the two architectures and excluded from the count.

Original target-network dimensions Consider a single fully connected layer of an MLP target network with weight matrix $W_i \in \mathbb{R}^{n_i \times n_{i-1}}$. Without any compression, the layer is presented to the hypernetwork's core body as a flat vector of $n_i \cdot n_{i-1}$ scalars, and the body must produce an update of the same dimension.

HyPoGen's MLP body. HyPoGen's core body is a stack of linear transformations that operate on this flat representation. Each linear transformation maps a vector of dimension $n_i \cdot n_{i-1}$ to a vector of the same dimension and therefore carries $(n_i \cdot n_{i-1})^2 = n_i^2 \cdot n_{i-1}^2$ parameters. The parameter count per layer per refinement step is therefore

$$\mathcal{C}_{\text{orig}}^{\text{HyPoGen}} \;=\; \mathcal{O}\big(n_i^2 \cdot n_{i-1}^2\big)\,, \tag{35}$$

quartic in target-network width and growing with both layer dimensions n_i and n_{i-1}.

Our attention body. Our core body represents W_i as a token sequence of length n_i in which each token is the n_{i-1}-dimensional encoding of one row of W_i, following the output-neuron-centric tokenization of Sec. 3.3. Five cross-attention modules operate on token sequences of this shape: one forward module ϕ_i and four backward modules $\psi_i^{\boldsymbol{\tau}}, \psi_i^{\boldsymbol{\omega}}, \psi_i^{\boldsymbol{\delta}}, \psi_i^{\nabla}$. Each module has query, key, value, and output projections of size $n_{i-1} \times n_{i-1}$, contributing $\mathcal{O}(n_{i-1}^2)$ parameters in total, and the same projections are applied across all n_i tokens by the attention mechanism rather than parameterized separately for each token. The number n_i therefore enters as the length of the sequence the same module is applied to, not as a factor in the module's own parameter count. With a constant number of modules per refinement step, the parameter count per layer per refinement step is

$$\mathcal{C}_{\text{orig}}^{\text{Ours}} \;=\; \mathcal{O}\big(n_{i-1}^2\big)\,, \tag{36}$$

independent of n_i and quadratic rather than quartic in n_{i-1}.

Headline contrast. Combining Eqs. 35–36, the contrast in the original dimensions is

$$\underbrace{\mathcal{O}(n_i^2 \cdot n_{i-1}^2)}_{\text{HyPoGen}} \quad\longrightarrow\quad \underbrace{\mathcal{O}(n_{i-1}^2)}_{\text{Ours}}\,. \tag{37}$$

The structural difference is the factor of n_i^2, which our hypernetwork removes because n_i governs the length of the token sequence processed by a module whose parameter count is independent of that length, rather than entering as a multiplier on the dimension of a flat vector whose representation requires a parameter count that grows with n_i.

Compression to embedding dimension d The cost of either core can be reduced further by compressing each row of W_i from n_{i-1} scalars to a d-dimensional latent through the per-layer encoder, with the per-layer decoder recovering the row when the actual weights are needed. Under

this compression, the layer is presented to the core body as either a flat vector of dimension $n_i \cdot d$ (for HyPoGen) or a token sequence of length n_i with each token of dimension d (for ours).

HyPoGen's MLP body under compression. A linear transformation between two spaces of dimension $n_i \cdot d$ carries $(n_i \cdot d)^2 = n_i^2 \cdot d^2$ parameters, giving a parameter count per layer per refinement step of

$$\mathcal{C}_{\text{cmp}}^{\text{HyPoGen}} \;=\; \mathcal{O}\big(n_i^2 \cdot d^2\big)\,. \tag{38}$$

The factor of d^2 replaces the factor of n_{i-1}^2 from Eq. 35, and the factor of n_i^2 remains because the MLP body still processes n_i rows through a transformation whose parameter count grows with n_i.

Our attention body under compression. Each of the five cross-attention modules now has projections of size $d \times d$, contributing $\mathcal{O}(d^2)$ parameters per module, and the same projections are applied across all n_i tokens. The parameter count per layer per refinement step is

$$\mathcal{C}_{\text{cmp}}^{\text{Ours}} \;=\; \mathcal{O}\big(d^2\big)\,, \tag{39}$$

independent of both n_i and n_{i-1}. The dependence on target-network width is removed entirely; the embedding dimension d is a hypernetwork hyperparameter chosen independently of the target-network width and controls hypernetwork capacity in isolation.

Headline contrast under compression. The contrast under compression is

$$\underbrace{\mathcal{O}(n_i^2 \cdot d^2)}_{\text{HyPoGen}} \quad\longrightarrow\quad \underbrace{\mathcal{O}(d^2)}_{\text{Ours}}\,. \tag{40}$$

The factor of n_i^2 is again the structural difference between the two architectures, and the mechanism that removes it is the same as in the uncompressed case: n_i governs sequence length rather than module parameter count.

Summary Table 19 summarizes the four counts derived above. In both formulations, the parameter count of HyPoGen's core body retains a factor of n_i^2 that our attention-based core does not, and this factor is the structural origin of the scalability barrier identified in Sec. 3.1. The empirical confirmation of these scalings – parameter count, GPU memory, and PSNR as the target-network width N varies from 128 to 1024 – is reported in Sec. 4.4 (Fig. 4b).

Table 19: Parameter count of the core iterative body per target-network layer per refinement step, for HyPoGen's MLP-based core and our attention-based core, in the original target-network dimensions and under row-level compression to embedding dimension d. Encoders and decoders are shared between the two architectures and excluded from the count.

	Original dimensions	Under compression
HyPoGen (MLP body)	$\mathcal{O}(n_i^2 \cdot n_{i-1}^2)$	$\mathcal{O}(n_i^2 \cdot d^2)$
Ours (attention body)	$\mathcal{O}(n_{i-1}^2)$	$\mathcal{O}(d^2)$
Structural difference	factor of n_i^2	factor of n_i^2

6.4.2 Compute-Matched Comparisons and End-to-End Scaling

Matched inference FLOPs. On CelebA-HQ, we compare with an LDMI variant at matched inference FLOPs, keeping the encoder, target INR, supervision, and training protocol fixed (Table 20). The 2.1 dB gap supports the benefit of allocating computation to our structured rollout. Together with the parameter-matched backward-structure ablation, this supports the contribution of the optimization-oriented decomposition beyond increased capacity or attention computation.

Table 20: CelebA-HQ reconstruction: ours versus LDMI at matched inference FLOPs.

Method	FLOPs (G/image)	PSNR (dB)
Ours	25.4	27.7
LDMI	25.5	25.6

Width-dependent costs. Table 21 follows the width-scaling setting of Fig. 4b, with inference measurements at batch size 1. Our peak inference memory grows from 0.14 to 0.25 GB, compared with 0.19 to 10.39 GB for HyPoGen, yielding approximately $41.6\times$ lower peak memory at width 768. Latency grows moderately from 15.2 to 17.2 ms. These results demonstrate the practical memory benefit of the scalable attention body, alongside its higher FLOPs and latency than HyPoGen at these three batch-size-1 settings.

Table 21: End-to-end width sweep at batch size 1. Each entry is HyPoGen / Ours.

Width	FLOPs (G)	Peak inference memory (GB)	Latency (ms)
128	5.4 / 11.0	0.19 / 0.14	6.7 / 15.2
512	12.6 / 71.0	3.56 / 0.19	9.1 / 16.4
768	23.1 / 138.3	10.39 / 0.25	14.6 / 17.2

6.4.3 Engineering Strategies for Scalability

We explore two simple strategies to mitigate the depth-linear and attention-quadratic cost factors of Appendix 6.5. ***Token Compression*** (n=2) merges every n adjacent parameter tokens after the row-wise tokenization of Sec. 3.3 into a single token, halving the attention sequence length (from n_i to n_i/n); the target network's width is unchanged, but each cross-attention module operates on a shorter sequence, addressing wider target networks. ***Layer Fusion*** (m=2) groups every m consecutive target-network layers and lets each group share a single set of forward/backward modules (the five cross-attentions of Sec. 3.4); the target network's depth is unchanged, but the hypernetwork now instantiates only $\lceil L/m \rceil$ rather than L such module sets, addressing deeper target networks. We evaluate both strategies on ImageNet and Pre-trained NeRF in Table 22.

On ImageNet, all three configurations share the same target network (5×256 SIREN), so the comparison is direct: Token Compression and Layer Fusion reduce inference time at batch size 16 by 8% ($58.5 \rightarrow 54.1$ ms) and 30% ($58.5 \rightarrow 41.2$ ms) respectively, Layer Fusion also cuts parameter count by 20% (124.5M $\rightarrow$ 99.4M), and PSNR drops by less than 0.3 dB.

On Pre-trained NeRF (under the *Images only* supervision setting), the two variants intentionally use *larger* target networks than the baseline (Token Comp. 3×128, Layer Fus. 6×64), so PSNR is not directly comparable (†). The relevant question is how much these larger targets inflate the hypernetwork's cost: inference time rises by only 4–6% and parameter count by under 2%, while PSNR stays within 0.4 dB of the baseline. Together with the ImageNet result, this shows the two strategies either reduce cost at fixed target size or absorb larger targets at near-flat cost.

Table 22: Efficiency comparison (batch size: 16) under two practical engineering strategies for scaling to wider or deeper target networks. NeRF results are reported under the *Images only* supervision setting.

	ImageNet			**Pre-trained NeRF**		
Metric	Token Comp.	Layer Fus.	Ours	Token Comp.	Layer Fus.	Ours
PSNR (dB)	22.7	22.6	22.9	29.0†	29.2†	28.8
Infer Time @ b=16 (ms)	54.1	41.2	58.5	2098	2135	2018
Params (M)	132.82	99.43	124.5	484.39	491.43	482.5

† Uses a larger target network than the baseline, so PSNR is not directly comparable. For ImageNet, all methods use the same target network (5×256). For Pre-trained NeRF, the two variants use larger target networks: Token Comp. uses 3×128, Layer Fus. uses 6×64.

6.5 Limitations

We discuss computational, architectural, and evaluation limitations.

Per-instance inference latency. Our forward and backward blocks are realized as stacks of cross-attention modules, each of which carries non-trivial constant-factor cost relative to a single MLP layer. Although Table 3 shows that our method matches or improves on HyPoGen and LDMI in

parameter count and GPU memory at batch size 16, single-instance inference (batch size 1) carries higher latency than the MLP-based HyPoGen on the smaller benchmarks.

Depth and width scaling of the iterative module. Our parameter-count derivation in Appendix 6.4.1 establishes that the cost of one refinement step is independent of target-network width, but it is linear in the number of target-network layers L, and the attention sub-modules are quadratic in their sequence length. For target networks substantially deeper or wider than the 3–5 layer SIRENs and NeRFs evaluated in this work, the depth-linear and attention-quadratic factors would dominate the iterative module's cost. We discuss two simple engineering strategies that mitigate this cost in Appendix 6.4.3.

Scope of statistical evaluation. We report multi-seed stability on three of the five datasets (Appendix 6.3.6). Running the full multi-seed protocol on all five datasets and all six baselines was beyond our compute budget, and we therefore cannot report mean $\pm$ std for every cell of Tables 1–2. The seed-to-seed standard deviations we do measure (0.05–0.23 dB) are an order of magnitude smaller than the gaps between our method and the baselines, so we expect this restriction not to affect our conclusions, but a fully seeded benchmark across all configurations remains future work.

Architectural scope. Experiments cover real-valued dense SIRENs and NeRF MLPs with fixed positional encoding. Generating grid/hash features or complex-valued INRs requires new parameter representations and validation. Input-resolution generalization also depends on the conditioning encoder.

6.6 Asset Licenses and Attribution

We use the following datasets and baseline implementations. All assets are used in accordance with their respective licenses and terms of use.

Datasets.

- **CelebA-HQ** [69], derived from CelebA [68]. Released for non-commercial research use under the original CelebA terms.
- **ImageNet** [70]. Released for non-commercial research use under the ImageNet Terms of Access.
- **ERA5** [71]. Released by the Copernicus Climate Change Service (C3S) under the Copernicus Licence; downloaded from the Copernicus Climate Data Store.
- **ShapeNet Chairs** [72]. Released for non-commercial research use under the ShapeNet Terms of Use.
- **Pre-trained NeRF dataset** [62]. Used under the license stated by the original release.

Baseline implementations. We compare with HyPoGen [8], LDMI [5], VaMoH [6], Trans-INR [4], GINR-IPC [45], and HyperDiffusion [7]. We use the official implementations where available, under the licenses specified by their authors. Encoder backbones for the conditioning networks of Appendix 6.2.3 are used under their respective public licenses.

6.7 Broader Impacts

This research develops a hypernetwork architecture for generating Implicit Neural Representations and does not involve human subjects, animal experiments, or sensitive data collection. All experiments use publicly available benchmark datasets (CelebA-HQ, ImageNet, ERA5, ShapeNet Chairs, Pre-trained NeRF) under their original licenses; no privacy or ethical issues arise from data usage.

The proposed methodology advances fundamental research in neural representation learning. We see two classes of broader impact worth noting.

Positive impacts. Our method reduces the parameter count of the hypernetwork's core body relative to MLP-based predecessors by a factor that scales with target-network width (Appendix 6.4.1), and reduces inference-time GPU memory on the largest benchmark by 5–6$\times$ (Table 3). For downstream

applications such as 3D reconstruction, neural rendering, and scientific signal modeling that rely on hypernetwork-generated INRs, this lowers the compute and memory barrier and makes such tools more accessible to researchers and practitioners with limited hardware.

Potential negative impacts. As with most generative-modeling research, improvements in neural-field generation could in principle be combined with other systems to produce synthetic visual content. Our work focuses on the architecture for parameter generation rather than on photorealistic media synthesis, and we do not foresee a direct path from our contribution to harmful applications. The CelebA-HQ dataset contains images of human faces; we use it only as a reconstruction-quality benchmark and do not train any identification, attribution, or generation model targeting specific individuals.

6.8 Use of Large Language Models

Large language models were used in this work only as auxiliary writing aids, specifically for surface-level revisions to phrasing and grammar. They were not used to design the method, derive results, run experiments, or generate any scientific content. All ideas, derivations, experimental designs, and conclusions are the authors' own intellectual contributions, and the use of LLMs does not affect the methodology, scientific rigor, or originality of the research.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and Section 1 state our three contributions, framed against the tradeoff between single-pass and optimization-oriented hypernetworks discussed in Section 2 : (i) we identify the scalability barrier of optimization-oriented hypernetworks as a structural property of the MLP instantiation rather than the principle (Section 3.1); (ii) we derive a token-space rollout that preserves the chain-rule dependency structure with five cross-attention modules specified by one information-flow rule (Sections 3.3–3.4, with the chain-rule derivation in Section 3.2); and (iii) on five INR generation benchmarks our method outperforms the strongest representatives of both prior paradigms, scales to widths at which the MLP-based optimization-oriented paradigm exhausts GPU memory, and produces the highest result on NeRF generation without weight supervision (Section 4). The Q/K/V assignment claim is supported by Table 4, the scalability claim by Section 4.4, and the efficiency claim by Table 3 in Section 4.3.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations in Appendix 6.5, with the corresponding mitigation experiments in Appendix 6.4.3. Specifically: (i) single-instance inference latency is higher than the MLP-based HyPoGen baseline on the smaller datasets; (ii) the cost of one refinement step is linear in target-network depth and quadratic in attention sequence length, which we mitigate with two engineering strategies (Token Compression and Layer Fusion). On ImageNet at fixed target-network size, these reduce inference time by up to 30% and parameter count by up to 20% at less than 0.3 dB PSNR loss; on Pre-trained NeRF they let us absorb substantially wider or deeper target networks at only 4–6% extra inference time and under 2% extra parameters (Table 22); and (iii) due to compute constraints, exhaustive multi-seed evaluation is reported on three of the five datasets only.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.

- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not contain formal theorems. The gradient-inspired / chain-rule motivation in Section 3 is used as a design principle to justify architectural choices, namely the paired forward/backward blocks and information-flow-consistent Q/K/V assignment of Section 3.4, derived from the chain-rule decomposition in Section 3.2. We explicitly analyze core parameter counts and the structural correspondence between the token-space forward block and an INR layer: the parameter-count derivation in Appendix 6.4.1 (with all assumptions stated inline, e.g., counting only the core iterative body) and the forward-block correspondence with a standard INR layer in Appendix 6.1.5. The full per-module Q/K/V derivations are in Appendix 6.1.6. None of these constitutes a formal theorem.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The full architecture is described across Section 3.3 (row-wise tokenization), Section 3.4 (forward and backward blocks), and Section 3.5 (iterative refinement and the full hypernetwork), with per-module Q/K/V derivations in Appendix 6.1.6, initialization submodules in Appendix 6.1.2, and adaptive scaling in Appendix 6.1.3. Implementation details—including token sequence lengths, embedding dimensions, number of Transformer layers, per-dataset target INR specifications (Table 9), encoder architectures (Appendix 6.2.3), and per-dataset loss formulations (Appendix 6.2.4)—are in Appendix 6.2. The training configuration is fixed across all datasets (Adam with β_1=0.9, β_2=0.999, learning rate 5×10^{-5}, mixed-precision training on NVIDIA H800 GPUs). Code is provided in the supplementary material, with an initial public release at `https://github.com/lulucai435/hyper-inrs`.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: All datasets used (CelebA-HQ, ImageNet, ERA5, ShapeNet Chairs, Pretrained NeRF) are publicly available; we cite their original sources and follow their licenses (Appendix 6.6). We provide a code package as supplementary material containing the implementation of our hypernetwork (corresponding to Sections 3.3–3.5), per-dataset configuration files, and training scripts for the main results in Tables 1 and 2. An initial public release is available at `https://github.com/lulucai435/hyper-inrs`.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.

- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4 specifies the data splits and evaluation metrics for each setting: PSNR, SSIM, and LPIPS for 2D INR generation (Section 4.1); accuracy and PSNR for ShapeNet Chairs and PSNR for NeRF novel view synthesis (Section 4.2). The six baselines (HyPoGen, LDMI, VaMoH, Trans-INR, GINR-IPC, HyperDiffusion) are described in Appendix 6.2.1, and the shared per-dataset encoders used across all compared methods are in Appendix 6.2.3. Appendix 6.2 reports the optimizer (Adam), learning rate (5×10^{-5}, no scheduling), per-dataset batch sizes, and all model hyperparameter choices in Table 9; per-dataset loss formulations are in Appendix 6.2.4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We report multi-seed stability analysis on three representative datasets (Pretrained NeRF, CelebA-HQ, ERA5), each trained with five independent random seeds, in Appendix 6.3.6. We report mean $\pm$ standard deviation of PSNR across seeds; the variability captured corresponds to random-initialization differences with all other experimental conditions fixed. The observed standard deviations (0.05–0.23 dB) are an order of magnitude smaller than the gaps between our method and the baselines reported in Tables 1 and 2, indicating that our improvements are not attributable to seed variance. Due to compute constraints, multi-seed experiments are not run for every dataset and baseline configuration; this is acknowledged in Appendix 6.5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All experiments were conducted on NVIDIA H800 GPUs with mixed-precision training. Table 3 (Section 4.3) reports per-experiment resource use—parameter count, peak GPU memory, and inference latency at batch size 16—for our method and the two main baselines (HyPoGen, LDMI) on all five datasets. The total compute across all experiments reported in this paper, including the main results in Tables 1 and 2, the architectural ablations in Table 4 (Section 4.4) and Appendix 6.3.3, the multi-seed stability runs of Appendix 6.3.6, the data-scale and width-scale studies of Section 4.4, the engineering-strategy experiments of Appendix 6.4.3, and preliminary runs that did not make it into the paper, is approximately **300** H800 GPU-days.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and confirm that this work conforms in all respects. The research uses only publicly available benchmark datasets (CelebA-HQ, ImageNet, ERA5, ShapeNet Chairs, Pre-trained NeRF) listed with their licenses in Appendix 6.6, involves no human subjects or private data, and concerns fundamental architectural research on hypernetworks for implicit neural representations. See Appendix 6.7 for a discussion of broader impacts.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

Justification: We discuss broader impacts in Appendix 6.7. Positive impacts: more parameter-efficient INR generation reduces compute and memory cost for downstream applications such as 3D reconstruction, neural rendering, and scientific signal modeling, lowering the resource barrier for these tools (Section 4.3, Table 3; further analysis in Appendix 6.4.1). Potential negative impacts: as with most generative-modeling research, improvements in neural-field generation could in principle be combined with other systems to produce synthetic visual content; however, our work focuses on architecture for parameter generation rather than on photorealistic media synthesis, and we do not foresee a direct path to harmful applications. The CelebA-HQ dataset contains images of human faces; we use it only as a reconstruction-quality benchmark, in line with its original license.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release pretrained generative models for media synthesis, scraped datasets, or other high-risk assets. Our contribution is an architecture for hypernetwork-based INR parameter generation, trained on publicly available benchmark datasets; it does not pose dual-use or misuse risks beyond those of the underlying public datasets.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cite the original papers for all datasets and baselines in Section 2 and Section 4, and list their licenses and terms of use in Appendix 6.6. The datasets used are CelebA-HQ, ImageNet, ERA5, ShapeNet Chairs, and Pre-trained NeRF; the baselines are HyPoGen, LDMI, VaMoH, Trans-INR, GINR-IPC, and HyperDiffusion (described in Appendix 6.2.1). Encoder backbones (DINO ViT-B/16, VQ-VAE) used in Appendix 6.2.3 are also under their respective public licenses. Our use of each asset complies with its license.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The supplementary code package includes configuration files for all five datasets, training scripts, and a description of the model architecture (corresponding to Sections 3.3–3.5 of the main paper). An initial public release is available at `https://github.com/lulucai435/hyper-inrs`. We do not release new datasets or pretrained model checkpoints with the initial submission.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or human-subjects research.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or human-subjects research, so no IRB approval is required.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs were not used as a component of the proposed method or in any core scientific contributions of this work. Any use of LLMs was limited to surface-level writing assistance (grammar and phrasing), which does not impact the methodology, scientific rigor, or originality of the research. See Appendix 6.8.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.