

LiFT: LOOP FLOW TRANSFORMERS

LOOP IN DEPTH, FLOW IN TIME

Mohammad Mahdi Derakhshani^{1,4,*} Pedro M. P. Curvo^{1,4,*} Gertjan J. Burghouts^{3,†}

Jan-Willem van de Meent^{2,4,†} Cees G. M. Snoek^{1,4,†}

¹VISLab ²AMLab ³TNO, Intelligent Imaging ⁴University of Amsterdam

{m.m.derakhshani, p.m.pombeirocurvo}@uva.nl

*Equal co-authorship; random order. †Equal supervision.

ABSTRACT

We introduce Loop Flow Transformers (LiFT), a family of looped generative models that scales computation by repeatedly applying a shared Diffusion Transformer (DiT) core, with only light changes to the standard architecture. Rather than asking every recurrent step for the final prediction, LiFT trains each step with a single regression target: a point on a straight path from the model’s initial estimate to the flow-matching target. Because we index these targets by a continuous depth coordinate, a trained model can loop far beyond its training depth with no retraining, early exits, or other modifications. In our experiments, these longer rollouts improve generation, so inference computation can grow without adding parameters. On ImageNet at 256×256 , LiFT-L/2 achieves an FID 3.34 points lower than our dense DiT-XL/2 baseline while using approximately 60% fewer parameters, 32% fewer training FLOPs, and 52% fewer inference FLOPs.

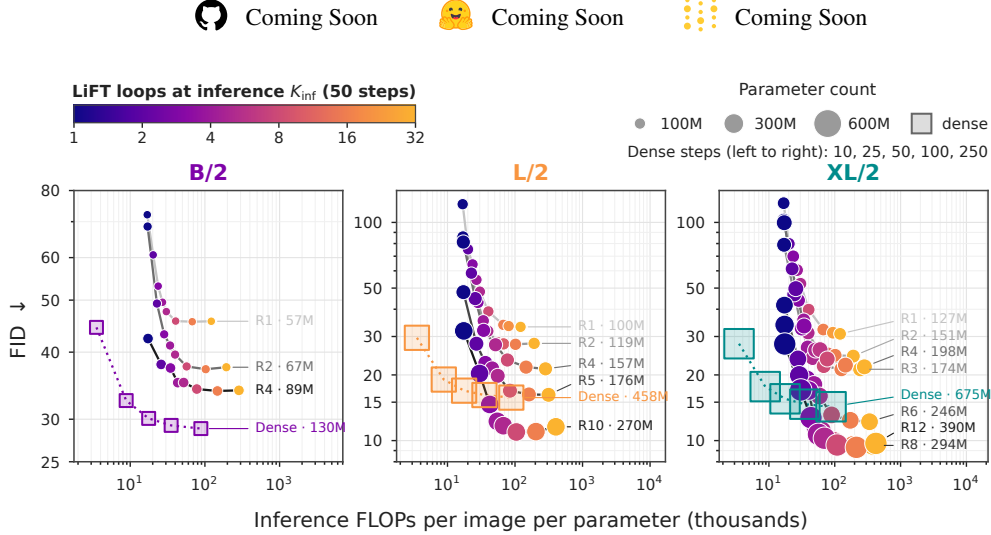


Figure 1: **Looping a smaller model beats larger dense DiTs at L/2 and XL/2 scale.** FID against inference FLOPs per parameter for LiFT checkpoints at 50 integration steps (solid; colour encodes inference depth, marker area parameter count) and for dense DiTs as the number of steps varies (dotted). Only at B/2, the dense model remains stronger.

1 INTRODUCTION

Diffusion Transformers (DiTs) have become a dominant architecture for generative modeling, with increasingly large models driving advances across image, video, and scientific domains [1–4]. Much of this progress, however, has come from increasing parameter count, so that generation quality remains closely tied to model size.

Looped transformers offer an alternative: by repeatedly applying a shared set of layers, they increase the network’s effective depth while keeping the number of trainable parameters fixed [5, 6]. In flow-based generation [7–9], this opens a second way to spend inference compute. A flow model generates an image by following a learned velocity field from noise to data, and the usual way to add compute is to take more integration steps, each requiring one network evaluation. Each step, however, can be no better than the network’s velocity estimate, so quality saturates once integration error is no longer the bottleneck [10, 11]. Looping instead spends additional compute within each step to refine the velocity estimate, so steps and loops act as complementary axes of inference compute [12]. Steps follow the flow more closely; loops estimate it more accurately. In principle, a trained looped model can therefore buy better predictions at test time by running more loops.

In practice, however, extra loops pay off only when two conditions are met. A model must be able to handle more loops than it was trained with, and the added compute must yield better quality than a dense model given the same compute. Results on both points have so far been mixed. Without dedicated training, a looped diffusion transformer produces coherent images only at its training depth [12]. Intermediate supervision can make looped models usable across depths, but loops beyond the training depth soon hurt rather than help: on ImageNet, FID worsens as soon as inference exceeds the training depth, and on video, quality peaks at 1.5 times the training depth [12]. In these methods every intermediate prediction is trained toward the complete output [12, 13], so no stage is given a distinct role. On the second condition, a looped DiT-B/4 trained with MeanFlow gave no advantage over a dense model at equal compute, and extra sampling steps paid off more than extra loops [14].

We address both conditions with the **Loop Flow Transformer (LiFT)**, which, rather than asking every loop for the complete output, gives each loop its own target. Within each network evaluation, LiFT draws a straight path from the model’s initial velocity estimate to its training target and tells each loop how far along this path it is. Each loop is supervised at the matching point, so successive loops carry out successive stages of a single correction, and the last loop is trained on the target itself. Because progress along the path is continuous, the loop count only sets how finely the path is traversed, much as the number of integration steps does for generative time; both can be chosen after training. LiFT thus separates computational depth from generative time:

“Loop in depth, flow in time.”

Our results on ImageNet 256×256 (Figure 1) show that at the DiT-L/2 and DiT-XL/2 scales [1], LiFT outperforms larger dense DiTs with fewer parameters. The figure separates the number of integration steps T from the number of loops used in training, K_{train} , and at inference, K_{inf} . Dense DiTs level off as T grows to 250 steps. A LiFT checkpoint trained at a fixed K_{train} trails a dense DiT that executes the same number of blocks, but keeps improving as K_{inf} grows until it reaches FIDs the dense models do not attain, at lower inference cost. At DiT-B/2, the dense model remains stronger.

Contributions

1. **A trajectory objective for recurrent generation.** We introduce LiFT, which gives each loop its own target on a path from the model’s initial estimate to the training target, and trains the last loop on the standard flow-matching objective.
2. **Useful recurrence far beyond training depth.** Without retraining, an XL/2 checkpoint trained with two loops reduces FID from **17.07 to 9.31** at sixteen loops, eight times its training loop count.
3. **Better generation with fewer parameters and lower inference cost.** On ImageNet 256×256 , LiFT-L/2 lowers FID by 3.34 relative to our dense DiT-XL/2 baseline, with about 60% fewer parameters, 52% fewer inference FLOPs, and the same training-token budget.

2 BACKGROUND

2.1 FLOW MATCHING

Let ρ_0 denote a tractable base distribution, typically $\mathcal{N}(0, I)$, and let $\rho_1 = p_{\text{data}}$ denote the target data distribution. Flow matching learns a time-dependent velocity field that transports samples from ρ_0 to ρ_1 [7]. A simple construction, also underlying rectified flow and a special case of stochastic interpolants [8, 9], starts from independent draws $x_0 \sim \rho_0$ and $x_1 \sim \rho_1$ and defines the linear conditional path and its velocity as

$$x_t = (1 - t)x_0 + tx_1, \quad u^* = \frac{dx_t}{dt} = x_1 - x_0, \quad t \in [0, 1]. \quad (1)$$

The marginal distribution ρ_t of x_t interpolates between the base and data distributions. Because the corresponding sample-specific velocity target is available directly, training can regress it without simulating an ODE. With $t \sim \mathcal{U}(0, 1)$ and d output coordinates, the conditional flow-matching objective is

$$\mathcal{L}_{\text{FM}}(\theta) = \mathbb{E}_{x_0, x_1, t} \left[\frac{1}{d} \|v_\theta(x_t, t) - u^*\|_2^2 \right]. \quad (2)$$

Its unrestricted regression optimum is the conditional mean $v^*(x, t) = \mathbb{E}[u^* \mid x_t = x, t]$, the marginal velocity associated with the path. Additional conditioning, such as a class label, can be included in both the predictor and this conditional expectation. At inference, the learned field defines a neural ODE [15]: generation starts from $X_0 \sim \rho_0$ and numerically integrates $dX_t/dt = v_\theta(X_t, t)$ to $t = 1$, yielding an approximation to ρ_1 . We write T for the number of integration steps; with ordinary Euler integration, each step calls the learned field once.

2.2 DIFFUSION TRANSFORMERS

Diffusion Transformers (DiTs) parameterize diffusion or flow-based generative models with a transformer backbone operating over image or latent patches [1]. The input is first embedded into a sequence of tokens and then processed by a stack of attention and feed-forward blocks. Generative time t and, when available, auxiliary information such as a class label condition these blocks through mechanisms such as adaptive normalization and gated residual branches. A final prediction head maps the resulting token representations back to the input geometry to produce the quantity required by the generative objective. In our setting, this quantity is the flow-matching velocity $v_\theta(x_t, t)$.

2.3 LOOPED TRANSFORMERS AND COMPUTATIONAL DEPTH

A conventional transformer assigns distinct parameters to each block. A looped transformer instead applies one shared block, or group of blocks, repeatedly, so executed depth grows with the number of passes while the parameter count stays fixed [5]. This distinction can be expressed through a prelude–core–coda decomposition, following Geiping et al. [6] and the architectural formulation of Chen et al. [16]. For an input x and conditioning c ,

$$h_0 = P_\theta(x, c), \quad h_k = R_\theta(\phi(h_{k-1}, h_0), c), \quad u_k = \mathcal{D}_\theta(h_k, c). \quad (3)$$

The *prelude* P_θ embeds the input once, producing the initial state h_0 . The *core* R_θ is applied K times with the same weights; the *boundary operator* ϕ links successive passes and may normalize the state and reinject h_0 so that every pass retains access to the input. The *coda* $\mathcal{D}_\theta$ maps any state to a prediction through its own transformer blocks, if any, and a prediction head, so u_0 is the prediction from the prelude alone and u_K the prediction after K passes. The core may additionally receive an embedding of its pass index; Section 3 builds on this. In our setting $x = x_t$, c encodes t and any class label, and $v_\theta(x_t, t) = u_K$.

With L_P , L_R , and L_D transformer blocks in the prelude, core, and coda, respectively, weight sharing separates the number of unique blocks from the number executed to produce the final prediction:

$$L_{\text{unique}} = L_P + L_R + L_D, \quad L_{\text{executed}} = L_P + KL_R + L_D.$$

These counts exclude the embedding and prediction heads and assume that the coda is evaluated only at the final state. This separation also underlies iso-depth scaling studies, which compare repeated

computation with distinct layers at matched executed depth [17]. The loop count K therefore controls computation within a single prediction, independently of any generative time coordinate. Training unrolls a chosen number of passes; at inference, their number can change without changing the trained parameters. Whether additional passes improve the prediction depends on the learned recurrence.

Parameter and compute budgets. Following Hoffmann et al. [18], let N denote the number of trainable parameters, D the number of training tokens processed, and C_{train} the total training compute in FLOPs. For image models, tokens are patches of an image or its latent representation. With n tokens per example, U optimizer updates, and global batch size B_{batch} , the training-data budget is

$$D = UB_{\text{batch}} n.$$

Repeated presentations of an image contribute to this count, whereas recurrent passes over the same tokens increase computation without increasing D or adding parameters. Equal parameter or data budgets therefore need not imply equal training compute.

We separately measure inference compute per generated image as

$$C_{\text{infer}} = T C_{\text{forward}}(K),$$

where T is the number of integration steps and $C_{\text{forward}}(K)$ is the cost of one velocity prediction with K loops. These costs exclude tokenizer decoding.

3 LiFT

We introduce the Loop Flow Transformer (LiFT), a flow-based generative model that increases computational depth by repeatedly applying a shared transformer core. To train this recurrence, we construct a continuous reference path from the model’s initial velocity estimate to the flow-matching target. During training, we randomly sample intermediate depth coordinates s , condition each core application on its coordinate, and supervise the resulting predictions at the corresponding points along the path. Each loop thus receives a distinct correction task, while the final prediction retains the ordinary flow-matching objective.

3.1 METHOD

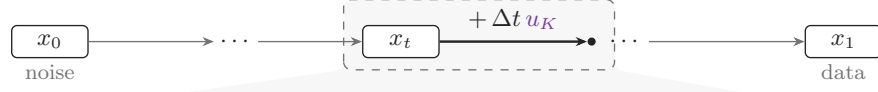
For a training example (x_0, x_1, t) , let x_t and u^* denote the interpolated state and conditional velocity target defined in Equation (1). We introduce a continuous computational-depth coordinate $s \in [0, 1]$ to describe progress within a single velocity evaluation, keeping the noisy input, generative time, and any class condition fixed. Figure 2 expands one such evaluation, the dashed sampler step in panel (a), into the recurrence of panel (b): starting from a prelude state h_0 , K applications of the shared core produce states $h_1, \dots, h_K$, and a shared readout maps each state h_k to a prediction u_k at depth s_k . The loop count is chosen separately for the two phases: training unrolls K_{train} loops, whereas inference may use a different depth K_{inf} ; plain K appears only in statements that hold for both. We suppress the fixed outer conditioning here and specify the recurrence in Section 3.2.

A reference path in prediction space. The initial prediction defines an anchor $b = \text{sg}(u_0)$, where sg denotes stop-gradient. Because it is computed before the recurrent rollout, this estimate depends on neither the sampled grid nor the loop count. We connect it to the sampled flow-matching target through the linear path

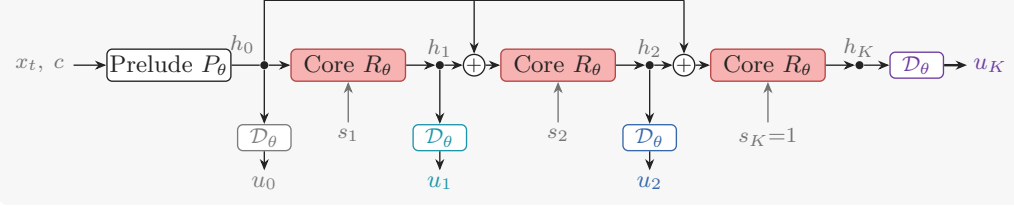
$$\bar{u}_s = (1 - s)b + su^*, \quad s \in [0, 1]. \quad (4)$$

For a fixed initial predictor $b = b(z)$, where z collects the noisy input, generative time, and any class condition, the reference path has conditional mean $\mathbb{E}[\bar{u}_s | z] = (1 - s)b(z) + sv^*(z)$, where $v^*(z) = \mathbb{E}[u^* | z]$ is the marginal velocity of Section 2.1. Its squared-loss regression optimum therefore progresses linearly from the initial estimate toward the marginal flow field, reaching the ordinary flow-matching optimum at $s = 1$. Each depth coordinate thus defines a prediction task along the same correction path; Section C.1 gives the assumptions and derivation.

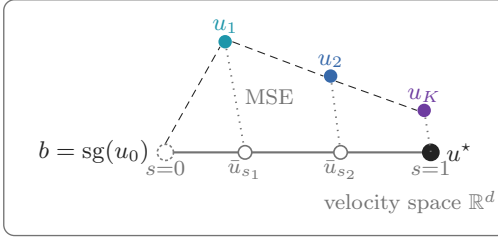
(a) Flow in time



(b) Loop in depth



(c) Training targets



(d) Depth coordinates

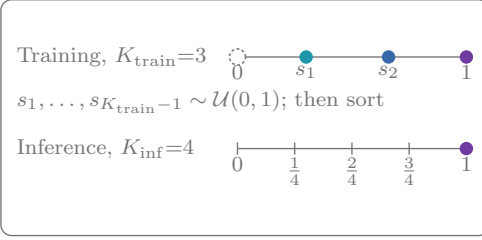


Figure 2: **Loop in depth, flow in time.** (a) The sampler advances through generative time, calling the network once per step. (b) Within each call, a shared core is applied K times, each time at its own depth coordinate s_k , and a shared readout maps every state to a prediction u_k . (c) In training, each u_k is regressed onto its point $\bar{u}_{s_k}$ on the path from $b = \text{sg}(u_0)$ to u^* . (d) Training samples the interior coordinates at random; inference uses a uniform grid. Colours mark the same loop across panels; normalization and the first reinjection of h_0 are omitted for clarity.

Random depth coordinates. To train the recurrence across this interval, we independently sample $K_{\text{train}} - 1$ coordinates from $\mathcal{U}(0, 1)$ for each example at every update (Figure 2d, top). Sorting these draws and appending the endpoints gives

$$s_0 = 0 < s_1 < \dots < s_{K_{\text{train}}-1} < s_{K_{\text{train}}} = 1. \quad (5)$$

We then unroll the same K_{train} core applications at these coordinates, supervising each readout against its corresponding reference value, as in Figure 2c. Varying the sampled grid changes both the intermediate prediction tasks and the recurrent histories through which they are reached, while the rollout length stays fixed. Every grid retains the endpoint $s_{K_{\text{train}}} = 1$, so the final readout always addresses the complete flow-matching task.

Trajectory supervision over K_{train} loops. Writing $\mathcal{S} = (s_0, \dots, s_{K_{\text{train}}})$ for the sampled grid, we train the recurrent predictions with

$$\mathcal{L}_{\text{LiFT}}(\theta) = \mathbb{E}_{x_0, x_1, t, \mathcal{S}} \left[\frac{1}{K_{\text{train}} d} \sum_{k=1}^{K_{\text{train}}} \|u_k - ((1 - s_k)b + s_k u^*)\|_2^2 \right], \quad (6)$$

Here d denotes the prediction dimension, and the loss averages uniformly over the K_{train} recurrent readouts, with any class condition included in the expectation. At the endpoint $s_{K_{\text{train}}} = 1$, the reference recovers the conditional flow-matching target, while each intermediate coordinate specifies

how much of the correction remains:

$$u^* - \bar{u}_s = (1 - s)(u^* - b).$$

The objective therefore distributes the prediction task across depth, supervising a progression toward the endpoint rather than asking every readout to recover it immediately.

Although the anchor b is recomputed on every forward pass, it is detached when forming the targets, whereas gradients through the predictions propagate through the full recurrent rollout, including the prelude.

Supervising the initial estimate. The trajectory objective trains the recurrent predictions but does not directly require u_0 to approximate the target velocity. As the shared parameters change, its magnitude can drift, allowing the anchor contribution $(1 - s)b$ to dominate the intermediate targets and leaving the core to learn an unnecessarily large correction. To encourage a useful starting estimate, we add an auxiliary regression loss:

$$\begin{aligned}\mathcal{L}_{\text{train}}(\theta) &= \mathcal{L}_{\text{LiFT}}(\theta) + \lambda \mathcal{L}_{\text{prelude}}(\theta), \\ \mathcal{L}_{\text{prelude}}(\theta) &= \mathbb{E}_{x_0, x_1, t} \left[\frac{1}{d} \|u_0 - u^*\|_2^2 \right].\end{aligned}\tag{7}$$

The coefficient λ controls supervision of the initial prediction and is applied once outside the $1/K_{\text{train}}$ average, making its weight independent of the training loop count. Whereas this auxiliary term differentiates through u_0 , the same prediction remains detached when constructing the trajectory targets. Consequently, at $K_{\text{train}} = 1$, training combines the ordinary flow-matching objective with the separate prelude loss. [Algorithm 1](#) in [Section D.3](#) summarizes the procedure, with experimental settings reported in [Section D](#).

Inference with a chosen depth budget. At each integration step, LiFT initializes a fresh prelude state and applies the shared core at $s_k = k/K_{\text{inf}}$ ([Figure 2d](#), bottom). Changing the loop budget therefore changes the computational grid while preserving its endpoint $s = 1$. The final readout at this endpoint supplies the velocity to the outer sampler of [Figure 2a](#), while intermediate readouts are needed only during training. With Euler integration, the update is

$$x_{t+\Delta t} = x_t + \Delta t u_{\theta}^{(K_{\text{inf}})}(x_t, t),\tag{8}$$

where $u_{\theta}^{(K_{\text{inf}})}(x_t, t)$ denotes the final readout $u_{K_{\text{inf}}}$, and $\Delta t = 1/T$ for T uniform steps.

3.2 ARCHITECTURE

LiFT partitions a DiT backbone into a prelude P_{θ} , a shared recurrent core R_{θ} , and a prediction-only coda $\mathcal{D}_{\theta}$ that includes the velocity readout. The prelude embeds the input patches and processes them once per integration step, while the core reuses the same transformer blocks at every inner step. The fixed outer conditioning $c = e_t(t) + e_y(y)$ sums embeddings of the generative time t and the class label y , omitting e_y when no label is available, and the computation is

$$\begin{aligned}h_0 &= P_{\theta}(x_t, c), \\ h_k &= R_{\theta}(\text{RMSNorm}(h_{k-1}) + h_0, c + e_s(s_k)), \quad k = 1, \dots, K, \\ u_k &= \mathcal{D}_{\theta}(h_k, c), \quad k = 0, \dots, K.\end{aligned}\tag{9}$$

The boundary operation normalizes the previous state and reinjects h_0 (the $\oplus$ nodes in [Figure 2b](#)), preserving access to the input representation throughout the rollout. In addition to this input reinjection, we adapt the shared transformation to its position along the reference path by conditioning the core on $e_s(s_k)$ through adaptive normalization and residual gates, following timestep-modulated looped transformers [19]. This depth-dependent conditioning is confined to the core, whereas the prelude and coda use only the fixed outer condition c . Within the core, however, the depth embedding supplies only the current coordinate s_k ; the preceding coordinate and interval length remain implicit in the recurrent history rather than appearing as explicit inputs.

The coda $\mathcal{D}_{\theta}$ maps each hidden state to a velocity prediction using parameters shared across depths. Because it serves only as a prediction branch, it does not modify the state passed to the next core application.

With L_P , L_R , and L_D transformer blocks in the prelude, core, and coda, LiFT contains $L_P + L_R + L_D$ unique blocks but executes $L_P + K_{\text{inf}}L_R + L_D$ blocks at inference. Because trajectory training also evaluates the coda at the initial state and every intermediate state, its forward pass instead executes $L_P + K_{\text{train}}L_R + (K_{\text{train}} + 1)L_D$ transformer blocks. These counts exclude embeddings and prediction heads; [Section D.5.1](#) provides analytic FLOP accounting that includes these operations and backward computation.

4 EXPERIMENTS

LiFT makes recurrent depth adjustable at inference, allowing a fixed set of parameters to support different computational budgets. We evaluate how this flexibility affects generation quality on class-conditional ImageNet, addressing four questions:

- Q1.** Does LiFT improve beyond its training depth, and overtake dense DiTs?
- Q2.** How does LiFT compare with larger dense models at comparable compute?
- Q3.** How should compute be split between recurrent depth and integration steps?
- Q4.** Which evaluated settings best fit a compute or parameter budget?

4.1 EXPERIMENTAL SETUP

We evaluate class-conditional generation on ImageNet-1k at 256×256 resolution, using SD-VAE representations of shape $4 \times 32 \times 32$.

Models. Our dense DiT baselines and LiFT use the same transformer blocks, adapted from SpeedrunDiT [20]. These blocks use RMSNorm, per-head RMS normalization of attention queries and keys, and two-dimensional rotary positional embeddings, while retaining GELU MLPs and adaLN-zero-style conditioning with gated residual branches. Sharing these modifications across model families controls for the underlying block design. We use the B/2, L/2, and XL/2 backbone dimensions, each with 2×2 latent patches.

In the main comparisons, LiFT uses four prelude blocks ($L_P = 4$), a shared recurrent core, and no transformer blocks in the coda ($L_D = 0$). The coda therefore consists only of a shared prediction head, which normalizes and conditionally modulates the tokens before projecting and unpatchifying the velocity prediction. During training, this head reads both the prelude and recurrent states; during sampling, it reads only the final state. [Table 1](#) lists representative configurations; [Table 15](#) in [Section D.1](#) gives the complete list.

Training. We fix the training-data budget at $D = 3.2768 \times 10^{10}$ latent-patch tokens per model across all main experiments. With 256 tokens per image, this corresponds to 500,000 optimizer updates at a global batch size of 256, independently of recurrent depth. All models share the data pipeline and optimization settings, using AdamW with a learning rate of 10^{-4} . Dense baselines are trained with conditional flow matching, while LiFT uses trajectory supervision with an auxiliary prelude-loss weight of $\lambda = 0.01$.

Within each backbone scale, we vary the shared-core size L_R and training depth K_{train} while keeping the executed transformer depth $4 + K_{\text{train}}L_R$ equal to the dense reference: 12 blocks for B/2, 24 for L/2, and 28 for XL/2. With the same training-token budget, this approximately matches transformer training computation while changing the balance between unique parameters and repeated computation. Our FLOP estimates additionally account for conditioning and intermediate readouts. With no transformer coda, their additional cost is less than 0.2% of the dense reference across these configurations; [Section D.5.1](#) derives the comparison and gives the numerical budgets.

Evaluation. We sample with exponential-moving-average weights and Euler integration without classifier-free guidance, pairing the initial noise and class labels across configurations. For each setting, we generate 50,000 images and compute FID, spatial FID (sFID), and Inception Score (IS) from those same images using the official ADM evaluator. FID serves as the primary metric in the main comparisons. [Section A](#) lists every evaluated setting with its inference cost, FID, sFID, and IS, and [Table 15](#) collects parameter counts and training costs, so every number quoted in this section

Table 1: **Same executed depth, far fewer parameters.** Representative LiFT configurations and their dense references; L_R is the shared-core block count and $L_{\text{exec}}^{\text{train}}$ the executed training depth, including the $L_P = 4$ prelude blocks, and N the parameter count. Table 15 lists all configurations.

Model	L_R	K_{train}	$L_{\text{exec}}^{\text{train}}$	N (M)
DiT-B/2	–	–	12	130.30
DiT-L/2	–	–	24	457.83
DiT-XL/2	–	–	28	674.82
LiFT-B/2-R1	1	8	12	56.69
LiFT-B/2-R4	4	2	12	88.58
LiFT-L/2-R5	5	4	24	175.79
LiFT-L/2-R10	10	2	24	270.24
LiFT-XL/2-R8	8	3	28	293.96
LiFT-XL/2-R12	12	2	28	389.58

can be traced to a table. Full optimization settings, evaluation details, and compute accounting are provided in Section D.

4.2 Q1: GENERALIZATION BEYOND TRAINING DEPTH

Question: Does LiFT improve beyond its training depth, and overtake dense DiTs?

To assess depth generalization, we vary K_{inf} for each checkpoint with the sampler fixed at 50 integration steps. Figure 3 shows FID against inference depth, with training depths marked and dense B/2, L/2, and XL/2 drawn as horizontal references.

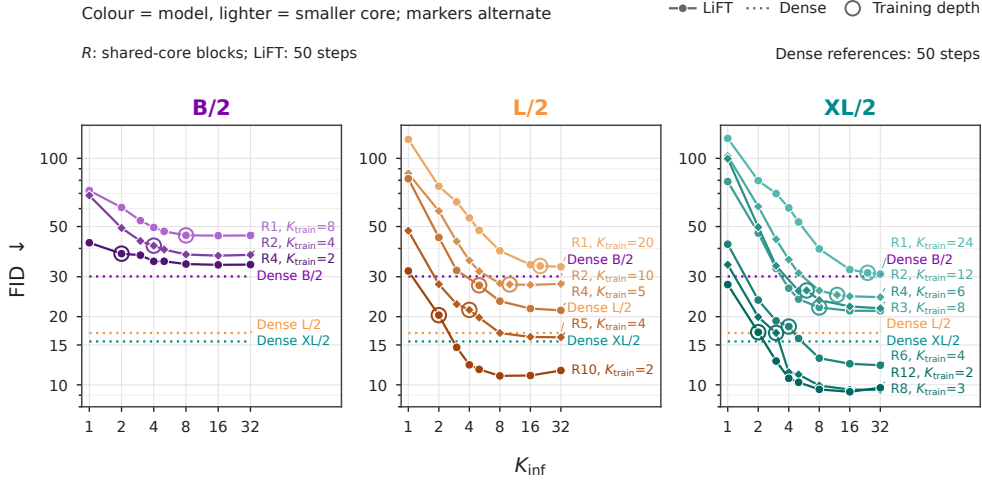


Figure 3: **LiFT improves beyond its training depth, and its larger cores overtake dense DiTs.** FID versus inference depth at 50 integration steps; rings mark training depth and dotted lines the dense DiTs. At L/2 and XL/2, the largest cores keep improving well past training depth and overtake the dense model of their scale, whereas single-block cores plateau. Training FLOPs match within each scale.

Increasing inference depth substantially reduces FID without further optimization. L/2 R10 improves from 20.30 at its two-loop training depth to 10.95 at eight inference loops; XL/2 R12 from 17.07 to 9.31 at sixteen inference loops; XL/2 R8 from 16.95 at three training loops to 9.50 at thirty-two inference loops. Checkpoints trained at larger depth follow the same trend with smaller gains (L/2 R5: 21.42 at four training loops to 16.21 at thirty-two inference loops), while the single-block cores plateau near their training depth, consistent with the limited capacity of a single shared block [12]. The curves of the largest cores flatten after eight to sixteen inference loops, with L/2 R10 and

Table 2: **The best inference depth is often far beyond the training depth.** FID at training depth K_{train} and best FID over inference depth K_{inf} , at ratio $r = K_{\text{inf}}/K_{\text{train}}$; Table 4 lists all checkpoints.

Model	K_{train}	FID at K_{train}	Best K_{inf}	Best FID	r
B/2 R4	2	37.95	16	33.87	$8\times$
L/2 R10	2	20.30	8	10.95	$4\times$
XL/2 R12	2	17.07	16	9.31	$8\times$
XL/2 R8	3	16.95	32	9.50	$10.67\times$
L/2 R5	4	21.42	32	16.21	$8\times$
B/2 R1	8	45.70	16	45.58	$2\times$

XL/2 R12 rising slightly at thirty-two loops. Table 2 summarizes the best depth for representative checkpoints.

Inference depth alone also carries several checkpoints past the dense references. L/2 R10 overtakes both dense L/2 and dense XL/2 at three inference loops, and XL/2 R12 overtakes dense XL/2 at three as well. XL/2 R8 and R6 follow at four and eight inference loops, whereas L/2 R5 draws level with dense L/2 at eight inference loops and overtakes it at sixteen. Because T is fixed, most of these crossings cost more inference FLOPs than the dense model they beat. The exception is L/2 R10 at three inference loops, which outperforms dense XL/2 (FID 14.64 versus 15.54) with slightly less inference compute (11.43 versus 11.86 TFLOPs) and 60% fewer parameters.

Finding: Training depth is not a ceiling: a checkpoint trained with few loops can be run with many more, turning a small model into a deeper one at inference.

4.3 Q2: GENERATION QUALITY AT COMPARABLE COMPUTE

Question: How does LiFT compare with larger dense models at comparable compute?

In Q1 we fixed T , so most crossings of a dense reference cost more inference FLOPs than the dense model they beat. Here we compare on cost, $C_{\text{infer}} = T C_{\text{forward}}(K_{\text{inf}})$: LiFT keeps 50 integration steps and varies K_{inf} , while the dense references vary T (Figure 4).

At its training depth, each LiFT checkpoint executes as many transformer blocks as its dense reference and therefore matches its inference cost, yet every evaluated checkpoint attains a higher FID than the same-scale dense model at 50 steps. For example, L/2 R10 reaches FID 20.30 at its two training loops, compared with 16.94 for dense L/2 at nearly identical cost. This gap is expected: at matched executed depth, a looped model replaces unique blocks with shared ones, and iso-depth scaling laws for looped language models find that a shared recurrence is worth only about half of a unique block [17]. The advantage of LiFT therefore emerges not at the training operating point but once test-time compute is reallocated, whether by extending recurrent depth beyond training or, as Q3 will show, by trading integration steps for loops.

Figure 4 shows that, once depth is extended, the outcome depends on scale. At B/2, the dense model remains stronger at every evaluated cost: the best LiFT B/2 result, FID 33.87, does not reach the 32.50 that dense B/2 attains with 25 integration steps at 1.150 TFLOPs. We attribute this gap to capacity: with at most 89M parameters for the same 32.8B training tokens, the B/2 cores appear too small for the data budget, so additional inference loops cannot compensate. At L/2 and XL/2, by contrast, recurrence yields operating points that improve on the dense reference while costing less. LiFT L/2 R10 with four inference loops attains FID 12.25 at 14.79 TFLOPs per image, whereas dense L/2 needs 16.14 TFLOPs and 100 steps to reach 16.17. Similarly, LiFT XL/2 R8 with four inference loops reaches FID 11.36 at 15.25 TFLOPs, compared with 14.69 at 23.72 TFLOPs for dense XL/2. Because each pair shares a backbone scale, these gains hold under comparable training FLOPs and identical training-token budgets, while LiFT uses 41% and 56% fewer parameters, respectively.

The same sweeps also permit a comparison across scales, although training budgets then differ. With eight inference loops, LiFT L/2 R10 attains FID 10.95 at 28.24 TFLOPs, whereas dense XL/2 reaches

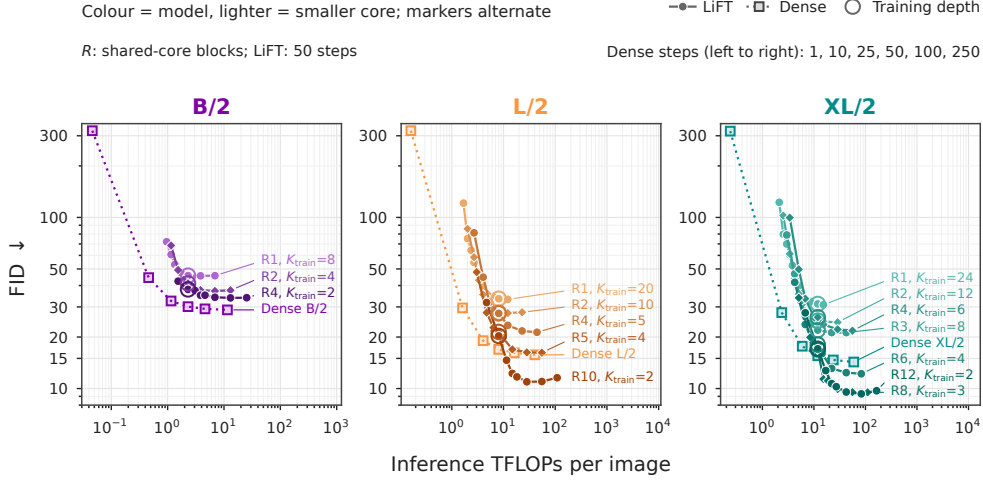


Figure 4: **Looping more reaches lower FID than sampling more for L/2 and XL/2.** FID versus inference cost: LiFT varies loops at 50 integration steps (solid), and dense DiTs vary the number of steps (dotted); rings mark training depth. Training FLOPs match within each model scale only.

14.30 only with 250 integration steps at 59.31 TFLOPs. LiFT therefore lowers FID by 3.34 points while using approximately 60% fewer parameters and 52% fewer inference FLOPs. Moreover, LiFT reaches this quality with about 32% less training computation (61.98 versus 91.10 EFLOPs).

Finding: A smaller recurrent model can stand in for a larger dense one. At equal depth it trails, as its fewer parameters predict, but, once the shared core has enough parameters, looping more at inference closes and then reverses the gap.

4.4 Q3: ALLOCATING COMPUTATION ACROSS DEPTH AND TIME

Question: How should compute be split between recurrent depth and integration steps?

The comparisons in Q2 fix LiFT at 50 steps, leaving open how a given budget should be divided between recurrent depth and integration steps. We therefore evaluate LiFT L/2 R10 over the joint grid $T \in \{10, 25, 50, 100\}$ and $K_{\text{inf}} \in \{1, 2, 4, 8, 16, 32\}$. For each inference budget, Figure 5(a) reports the lowest FID among the settings that fit within it, alongside the dense envelopes, whereas Figure 5(b) maps FID across the full grid, so that allocations of equal cost can be compared directly.

Allowing both budgets to vary sharpens the comparison with dense L/2. With 25 integration steps and four inference loops, LiFT L/2 R10 achieves FID 13.16 at 7.397 TFLOPs per image, compared with 16.94 at 8.069 TFLOPs for dense L/2 with 50 steps, and thus improves generation with approximately 41% fewer parameters and 8% less inference computation. In Figure 5(b), this setting lies on the cheaper side of the white curve that marks the cost of dense L/2. The same setting also improves on the checkpoint’s own training configuration: relative to two training loops and 50 steps, it reduces FID from 20.30 to 13.16 while lowering cost from 8.070 to 7.397 TFLOPs. In this regime, recurrent refinement is a more efficient use of inference compute than additional integration steps.

Recurrence is not, however, always the better destination for additional compute. At sixteen inference loops and ten integration steps, the same model obtains FID 15.09 at 11.027 TFLOPs, which is both worse and more expensive than the four-loop, 25-step setting. Here, shifting computation toward integration steps improves the trade-off, so the preferable direction of reallocation depends on the starting configuration.

At larger budgets, the FID surface also flattens. With eight inference loops, doubling the number of steps from 50 to 100 leaves FID nearly unchanged at 10.95 and 10.96, whereas extending depth to

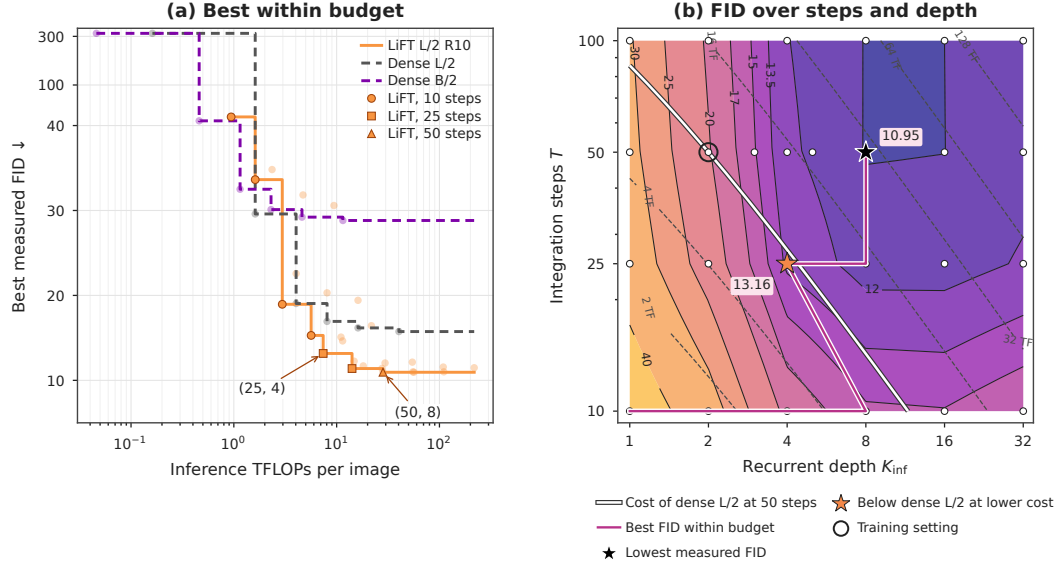


Figure 5: **Balancing inference loops and integration steps gives the best quality per budget.** (a) Lowest measured FID within each inference budget for LiFT L/2 R10, which varies both, and for dense DiTs, which vary integration steps; labels give (T, K_{inf}) . (b) FID across inference loops and integration steps, interpolated between the measured dots; dashed lines connect allocations of equal cost, the white line marks the cost of dense L/2 at 50 steps (FID 16.94), and the pink path traces the best allocation as the budget grows. Dense L/2 matches LiFT’s training FLOPs.

thirty-two inference loops raises FID at both budgets. Even at this plateau, the best measured setting, FID 10.95 at 28.24 TFLOPs, remains well below the 15.74 that dense L/2 reaches with 250 steps at 40.35 TFLOPs. Section A.3 reports the selected operating points, and Section A the complete FID, sFID, and IS sweeps.

Finding: Inference quality depends on balancing K_{inf} against T : neither more loops nor more integration steps alone yields the best result at a given budget.

4.5 Q4: MODEL SELECTION UNDER RESOURCE BUDGETS

Question: Which evaluated settings best fit a compute or parameter budget?

Q3 varies the inference settings of a fixed checkpoint, but deployment also requires choosing the checkpoint itself. To support this choice, Figure 6 reports, for each target FID, the least inference compute and the fewest parameters with which any evaluated checkpoint reaches it. Conversely, a fixed budget on either resource identifies the best measured FID it can support. The two resources need not favor the same model, however. For example, at a target FID of 20, minimizing compute selects LiFT L/2 R10 with ten integration steps and four inference loops (2.96 TFLOPs per image), whereas minimizing parameters selects the smaller LiFT L/2 R5 (176M parameters).

Finding: When memory is the constraint, LiFT reaches any quality target with fewer parameters than a dense DiT, and strict targets with less compute as well.

4.6 QUALITATIVE COMPARISONS

Figure 7 complements these comparisons with paired generations from LiFT L/2 R10 and dense L/2.

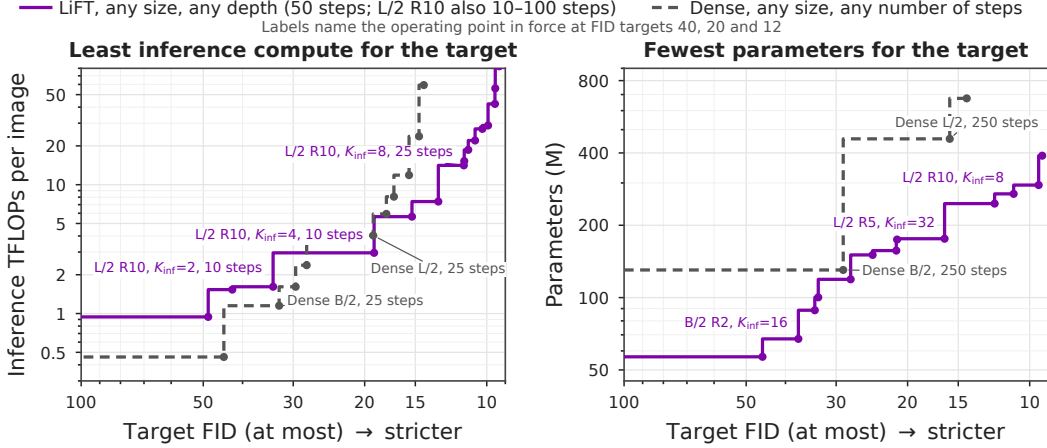


Figure 6: **LiFT needs fewer parameters for every quality target, and less compute for strict ones.** Least inference compute (left) and fewest parameters (right) with which any evaluated LiFT or dense setting reaches each target FID; settings span scales with different training FLOPs.

5 RELATED WORK

Recurrent computation and depth generalization. Reusing a shared transformation separates the number of trainable parameters from the depth of the executed computation, as in Universal Transformers [5]. The ability to benefit from additional iterations at inference also predates recent looped transformers: on algorithmic reasoning tasks, Schwarzschild et al. [21] demonstrate generalization beyond a fixed training depth, while Bansal et al. [22] investigate training strategies that sustain useful computation over longer rollouts. Recurrent language models extend this direction to larger scales, including training over distributions of recurrence counts [6]. Together, these works establish recurrent depth as a resource for inference-time computation; LiFT investigates how intermediate generative predictions should be supervised to make use of that resource.

Depth conditioning and computational trajectories. To adapt a shared transformation across recurrent applications, Xu and Sato [19] condition it on computational depth. LoopFormer [23] extends this perspective to normalized trajectories, using position and step-size conditioning to train shorter computations toward a full-depth solution. Whereas LoopFormer aligns trajectory endpoints across computational budgets, LiFT assigns distinct targets to intermediate velocity predictions within each rollout. Its normalized coordinate specifies the corresponding fraction of the correction toward the flow-matching target, while training keeps the number of recurrent applications fixed and randomizes their intermediate coordinates.

Recurrence in generative transformers. Within generative networks, DiT-Air [24] investigates layer sharing in diffusion transformers, while Ai et al. [25] examine recurrent layer arrangements inside flow-matching text-to-speech models. LoopDiT [14] provides a close architectural comparison: both methods combine a prelude, a shared recurrent core with depth-dependent conditioning, and an output stage. Within this shared layout, LoopDiT trains its final prediction with MeanFlow, whereas LiFT supervises the intermediate readouts of a flow-matching velocity predictor. The two methods also allocate shorter computations differently: LoopDiT retains a prefix of its training conditioning schedule, while LiFT changes the grid over the full interval $[0, 1]$, so every budget reaches $s = 1$.

Intermediate supervision for generative prediction. Beyond the sharing layout, the training objective determines what intermediate predictions are asked to recover. DeepFlow [13] studies intermediate flow-matching supervision and velocity alignment, while R-MDM [26] examines how the generative loss should be distributed across recurrent outputs. ELT [12] supplements supervision at sampled intermediate exits with self-distillation from a detached full-depth prediction and also reports depth extrapolation in UCF-101 video generation. LiFT organizes these prediction tasks along the reference $\bar{u}_s = (1 - s) \text{sg}(u_0) + s u^*$, assigning each readout a fraction of the correction

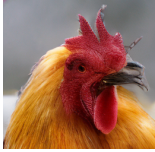
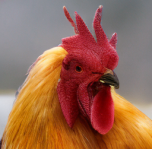
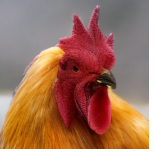
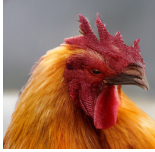
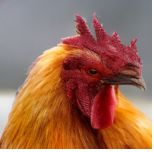
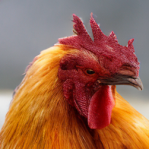
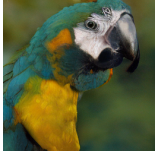
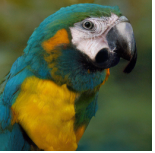
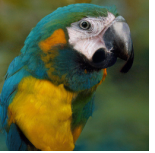
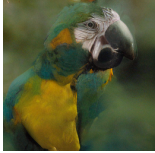
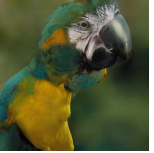
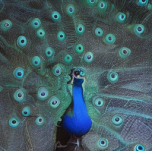
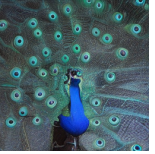
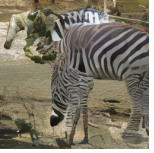
LiFT L/2 R10				Dense L/2			
270.24M parameters · $K_{\text{train}} = 2 \cdot 50$ steps				457.83M parameters · varying steps			
	$K_{\text{inf}} = 2$ (train)	$K_{\text{inf}} = 8$	$K_{\text{inf}} = 16$	$T = 50$	$T = 100$	$T = 250$	
TFLOPs	8.07	28.24	55.14	8.07	16.14	40.35	
Rooster							
Macaw							
Golden retriever							
Peacock							
Tabby cat							
Zebra							
Red panda							

Figure 7: **More loops refine samples from the same noise.** Selected paired samples from LiFT L/2 R10 at 50 integration steps and dense L/2 at varying numbers of steps; each row shares class and initial noise, and headers give inference TFLOPs per image. Compare the second column with the last: at eight inference loops, LiFT uses 30% less compute than dense L/2 at 250 steps and reaches a lower FID (10.95 versus 15.74).

toward the flow-matching target. Unlike a full-depth teacher used in self-distillation, the detached initial prediction defines the starting point from which this correction is measured. The resulting depth-indexed targets define different prediction tasks along the trajectory, while the final readout retains the complete endpoint task.

Computational depth and generative time. The distinction between computation within an evaluation and progression through the generative process also appears in continuous-depth models. $\text{ODE}_t(\text{ODE}_l)$ [27] explicitly separates sampling time from network length and uses length consis-

tendency to relate predictions produced with different computational budgets. LiFT instead uses its continuous coordinate to define a reference trajectory in prediction space. Its core remains a discrete, depth-conditioned recurrence: additional loops change the rollout rather than refine a prescribed numerical solver for an inner ODE. A separate distinction concerns whether recurrent state is reused within a velocity evaluation or carried across the generative trajectory. RIN [28] and Thinking with Looped Flows [29] carry representations as the generative input changes, whereas LiFT initializes its recurrent state separately at each integration step. Flow Reasoning Models [30] provide a closer connection to refinement at fixed generative input: they recurrently condition on previous clean predictions and use Fixed-Point Forcing to construct detached, rollout-derived conditioning states. LiFT instead carries hidden features, differentiates through the recurrent rollout, and assigns depth-dependent velocity targets to its readouts.

Other forms of recursion and equilibrium. Recursive Flow Matching [31] relates rescaled generative trajectories through velocity and consistency supervision; its recursion connects transport problems rather than intermediate predictions within one velocity evaluation. Fixed Point Diffusion Models [32], Generative Equilibrium Transformers [33], and CoFRe [34] instead define repeated computation through a hidden-state equilibrium. These perspectives are complementary to LiFT’s finite, depth-conditioned rollout, whose reference path specifies how its predictions should progress without imposing a fixed-point equation on the hidden states.

6 CONCLUSION

LiFT gives each training loop of a recurrent DiT its own target, a point on a reference path from the model’s initial velocity estimate to the flow-matching target. With this supervision, recurrent depth behaves as a second inference-time axis: a fixed checkpoint keeps improving well beyond its training depth, and the best quality at a given budget comes from balancing inference loops against integration steps rather than maximizing either. Although LiFT trails a dense DiT of the same executed depth at its training depth, reallocating inference compute toward recurrence lets its larger-core L/2 and XL/2 checkpoints outperform larger dense DiTs with fewer parameters and less inference compute.

Implications. By making recurrent depth selectable at inference, LiFT allows a fixed checkpoint to adapt its computation to the available budget. Because the shared core adds executed depth without adding parameters, this flexibility could be particularly useful when parameter storage is more restrictive than computation, including memory-constrained edge devices. It also motivates further study of budget-aware generation in video and generative dynamics models.

Limitations. Our evaluation concerns class-conditional image generation at 256×256 resolution without classifier-free guidance. Whether the same depth-scaling behavior extends to other modalities or guided sampling remains open. Within this setting, the advantage of LiFT depends on scale and operating point: at B/2, the dense model remains stronger at every evaluated cost, and at its training depth every LiFT checkpoint trails its same-scale dense reference. Each result also comes from a single training run and sample set. The deployment implications require hardware evaluation, since we report parameter counts and analytic FLOP estimates rather than device-level latency, energy consumption, or peak memory. Finally, inference depths are prescribed, and the joint allocation study covers one checkpoint over a finite grid; adaptive depth selection and broader optimization of inner and outer compute budgets remain for future study.

ACKNOWLEDGEMENTS

This project was supported by the ELLIS Unit Amsterdam and by SURF through access to the Snellius supercomputer. Cees G. M. Snoek is (partially) funded by the Horizon Europe project ELLIOT (GA No. 101214398).

REFERENCES

- [1] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023. URL <https://arxiv.org/abs/2212.09748>.
- [2] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 12606–12633, 2024. URL <https://proceedings.mlr.press/v235/esser24a.html>.
- [3] Adam Polyak et al. Movie Gen: A cast of media foundation models. *arXiv preprint arXiv:2410.13720*, 2024. URL <https://arxiv.org/abs/2410.13720>.
- [4] NVIDIA. Cosmos world foundation model platform for physical AI. *arXiv preprint arXiv:2501.03575*, 2025. URL <https://arxiv.org/abs/2501.03575>.
- [5] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL <https://arxiv.org/abs/1807.03819>.
- [6] Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/3b01972cf31e6fa0fe29e4b8b5c2a0a1-Abstract-Conference.html.
- [7] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023. URL <https://arxiv.org/abs/2210.02747>.
- [8] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations*, 2023. URL <https://arxiv.org/abs/2209.03003>.
- [9] Michael S. Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. In *International Conference on Learning Representations*, 2023. URL <https://arxiv.org/abs/2209.15571>.
- [10] Yilun Xu, Mingyang Deng, Xiang Cheng, Yonglong Tian, Ziming Liu, and Tommi Jaakkola. Restart sampling for improving generative processes. In *Advances in Neural Information Processing Systems*, 2023. URL <https://arxiv.org/abs/2306.14878>.
- [11] Nanye Ma, Shangyuan Tong, Haolin Jia, Hexiang Hu, Yu-Chuan Su, Mingda Zhang, Xuan Yang, Yandong Li, Tommi Jaakkola, Xuhui Jia, and Saining Xie. Inference-time scaling for diffusion models beyond scaling denoising steps. *arXiv preprint arXiv:2501.09732*, 2025. URL <https://arxiv.org/abs/2501.09732>.
- [12] Sahil Goyal, Swayam Agrawal, Gautham Govind Anil, Prateek Jain, Sujoy Paul, and Aditya Kusupati. ELT: Elastic looped transformers for visual generation. In *European Conference on Computer Vision*, 2026. URL <https://arxiv.org/abs/2604.09168>.
- [13] Inkyu Shin, Chenglin Yang, and Liang-Chieh Chen. Deeply supervised flow-based generative models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025. URL <https://arxiv.org/abs/2503.14494>.
- [14] Wenhao Chai. LoopDiT: Loop transformers for diffusion models. Technical blog post, 2026. URL <https://wenhaochai.com/blogs/loopdit.html>.

- [15] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, 2018. URL <https://arxiv.org/abs/1806.07366>.
- [16] Zixi Chen, Akshay Vegesna, Samip Dahal, and Andrew Gordon Wilson. How model growth, recursion, and boundary operators influence scaling exponents. *arXiv preprint arXiv:2609.19107*, 2026. URL <https://arxiv.org/abs/2609.19107>.
- [17] Kristian Schwethelm, Daniel Rückert, and Georgios Kaissis. How much is one recurrence worth? iso-depth scaling laws for looped language models. *arXiv preprint arXiv:2604.21106*, 2026. URL <https://arxiv.org/abs/2604.21106>.
- [18] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/c1e2faff6f588870935f114ebe04a3e5-Paper-Conference.pdf.
- [19] Kevin Xu and Issei Sato. On expressive power of looped transformers: Theoretical analysis and enhancement via timestep encoding. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 69613–69646, 2025. URL <https://proceedings.mlr.press/v267/xu25x.html>.
- [20] Swayam Bhanded. Speedrunning ImageNet diffusion. *Transactions on Machine Learning Research*, August 2026. URL <https://openreview.net/forum?id=0mYu3uPM3j>.
- [21] Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL <https://proceedings.neurips.cc/paper/2021/hash/3501672ebc68a5524629080e3ef60aef-Abstract.html>.
- [22] Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/7f70331dbe58ad59d83941dfa7d975aa-Abstract-Conference.html.
- [23] Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. LoopFormer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *International Conference on Learning Representations*, 2026. URL <https://arxiv.org/abs/2602.11451>.
- [24] Chen Chen, Rui Qian, Wenze Hu, Tsu-Jui Fu, Jialing Tong, Xinze Wang, Lezhi Li, Bowen Zhang, Alex Schwing, Wei Liu, and Yinfei Yang. DiT-Air: Revisiting the efficiency of diffusion model architecture design in text to image generation. *arXiv preprint arXiv:2503.10618*, 2025. URL <https://arxiv.org/abs/2503.10618>.
- [25] Jiabao Ai, Peng Han, Yuchen Song, and Zhengjun Yue. Depth through recurrence: Looped transformers for flow-matching TTS. *arXiv preprint arXiv:2609.29768*, 2026. URL <https://arxiv.org/abs/2609.29768>.
- [26] Alba Carballo-Castro, Julianna Piskorz, Paulius Rauba, Mihaela van der Schaar, and Pascal Frossard. Recursive scaling in masked diffusion models. *arXiv preprint arXiv:2606.18022*, 2026. URL <https://arxiv.org/abs/2606.18022>.
- [27] Denis Gudovskiy, Wenzhao Zheng, Tomoyuki Okuno, Yohei Nakata, and Kurt Keutzer. $\text{ODE}_t(\text{ODE}_l)$: Shortcutting the time and the length in diffusion and flow models for faster sampling. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2026. URL <https://arxiv.org/abs/2506.21714>.

- [28] Allan Jabri, David J. Fleet, and Ting Chen. Scalable adaptive computation for iterative generation. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 14569–14589, 2023. URL <https://proceedings.mlr.press/v202/jabri23a.html>.
- [29] Ayhan Suleymanzade, Chanhuk Lee, Floor Eijkelboom, Nicholas M. Boffi, İsmail İlkan Ceylan, and Jinwoo Kim. Thinking with looped flows. *arXiv preprint arXiv:2609.11801*, 2026. URL <https://arxiv.org/abs/2609.11801>.
- [30] Alec Helbling, Andrey Bryutkin, Mauro Martino, Duen Horng Chau, Nima Dehmamy, and Hendrik Strobelt. Flow reasoning models: Turning flows into efficient recurrent reasoners. *arXiv preprint arXiv:2606.29150*, 2026. URL <https://arxiv.org/abs/2606.29150>.
- [31] Jiahe Huang, Sihan Xu, Sharvaree Vadgama, and Rose Yu. Recursive flow matching. *arXiv preprint arXiv:2605.26535*, 2026. URL <https://arxiv.org/abs/2605.26535>.
- [32] Xingjian Bai and Luke Melas-Kyriazi. Fixed point diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024. URL <https://arxiv.org/abs/2401.08741>.
- [33] Zhengyang Geng, Ashwini Pople, and J. Zico Kolter. One-step diffusion distillation via deep equilibrium models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL <https://arxiv.org/abs/2401.08639>.
- [34] Andrea Miele, Yiming Qin, Alba Carballo-Castro, Justin Deschenaux, and Pascal Frossard. Fixed-point masked generative modeling. *arXiv preprint arXiv:2605.31215*, 2026. URL <https://arxiv.org/abs/2605.31215>.
- [35] Frank Nielsen. Fisher–Rao and pullback hilbert cone distances on the multivariate Gaussian manifold with applications to simplification and quantization of mixtures. In *Proceedings of 2nd Annual Workshop on Topology, Algebra, and Geometry in Machine Learning*, volume 221 of *Proceedings of Machine Learning Research*, pages 488–504, 2023. URL <https://proceedings.mlr.press/v221/nielsen23b.html>.

APPENDIX INDEX

	Page
A Complete and extended experimental results	
A.1 Complete numerical results corresponding to the main figures	19
A.2 Depth-generalization matrices	21
A.3 Joint allocation of inference computation	23
B Ablations and diagnostics	
B.1 Random versus fixed depth coordinates	26
B.2 Prelude-loss / λ ablation	27
B.3 Prediction trajectories	27
B.4 Computational-grid consistency	30
C Analysis of the LiFT objective	
C.1 Conditional regression result	32
C.2 Gradient flow with a detached anchor	32
C.3 Euclidean minimum action	33
C.4 Gaussian, Fisher, and KL interpretations	33
D Reproducibility and implementation	
D.1 Data and architecture	35
D.2 Training protocol	35
D.3 Training and sampling pseudocode	35
D.4 Evaluation protocol	37
D.5 Compute accounting	38

A COMPLETE AND EXTENDED EXPERIMENTAL RESULTS

This appendix reports every measured setting behind the main figures, including small regressions, and extends the analysis of depth extrapolation and of the allocation of inference compute. Each operating point is evaluated once on 50,000 generated images, so small FID differences, such as those near the plateaus, have not been tested for statistical significance. [Section D](#) gives the architecture, training, and evaluation details.

A.1 COMPLETE NUMERICAL RESULTS CORRESPONDING TO THE MAIN FIGURES

[Table 3](#) lists the dense step sweeps and LiFT depth sweeps plotted in the main comparisons, with sampling costs computed as in [Section D.5](#). Training depths, parameter counts, and training costs are collected in [Table 15](#).

Table 3: Complete numerical results corresponding to the main figures. Each row reports FID, sFID, and IS from the same 50,000 generated images. R denotes the number of blocks in the shared core; a dash denotes a dense model.

Model	K_{inf}	T	TFLOPs/image	FID	sFID	IS
Dense B/2	–	1	0.046	321.17	408.50	2.04
	–	10	0.460	44.49	13.18	45.92
	–	25	1.150	32.50	8.92	53.42
	–	50	2.300	30.10	7.84	54.64
	–	100	4.600	29.22	7.39	55.11
	–	250	11.501	28.82	7.19	54.86
Dense L/2	–	1	0.161	321.15	410.26	2.02
	–	10	1.614	29.59	10.99	68.33
	–	25	4.035	19.05	6.90	82.98
	–	50	8.069	16.94	5.98	86.15
	–	100	16.139	16.17	5.67	86.99
	–	250	40.347	15.74	5.52	87.61
Dense XL/2	–	1	0.237	319.32	409.10	2.03
	–	10	2.372	27.80	9.86	72.85
	–	25	5.931	17.68	6.57	89.16
	–	50	11.862	15.54	5.81	92.56
	–	100	23.723	14.69	5.52	93.38
	–	250	59.308	14.30	5.43	93.83
LiFT B/2 R1	1	50	0.959	72.11	10.90	23.70
	2		1.151	60.71	11.14	28.14
	3		1.342	53.09	9.84	31.52
	4		1.534	49.54	9.24	32.91
	5		1.726	47.60	8.91	33.52
	8		2.301	45.70	8.32	33.79
	16		3.834	45.58	7.99	33.61
	32		6.901	45.67	7.92	33.46
LiFT B/2 R2	1	50	1.151	68.55	11.82	24.45
	2		1.534	49.27	9.63	33.50
	3		1.917	43.18	8.49	36.79
	4		2.301	41.16	7.83	37.60
	5		2.684	39.58	7.06	38.22
	8		3.834	37.62	6.33	39.29
	16		6.900	37.18	6.15	39.34
	32		13.033	37.50	6.21	39.16

Table 3 (continued)

Model	K_{inf}	T	TFLOPs/image	FID	sFID	IS
LiFT B/2 R4	1	50	1.534	42.40	7.50	39.06
	2		2.300	37.95	6.67	40.19
	3		3.067	37.37	6.41	40.61
	4		3.833	35.10	5.66	41.95
	5		4.600	35.11	5.68	42.00
	8		6.900	34.15	5.41	42.71
	16		13.032	33.87	5.34	43.06
	32		25.296	33.96	5.34	43.21
LiFT L/2 R1	1	50	1.682	121.21	104.66	7.27
	2		2.018	75.29	10.05	22.40
	3		2.355	64.22	10.11	26.97
	4		2.691	54.58	9.60	31.10
	5		3.027	48.16	9.24	34.91
	8		4.036	39.08	8.42	40.80
	16		6.727	33.92	7.42	44.82
	20		8.072	33.50	7.28	45.18
LiFT L/2 R2	32		12.108	33.21	7.17	45.60
	1	50	2.018	85.76	10.23	17.47
	2		2.691	58.55	8.86	28.08
	3		3.363	42.97	7.98	38.21
	4		4.036	35.41	7.55	44.81
	5		4.708	31.74	7.34	48.56
	8		6.726	28.04	6.96	53.47
	10		8.071	27.75	6.89	53.92
LiFT L/2 R4	16		12.106	27.63	6.74	53.71
	32		22.865	27.90	6.71	53.21
	1	50	2.691	81.29	10.49	19.10
	2		4.036	44.75	9.04	36.04
	3		5.380	32.01	6.69	48.01
	5		8.070	27.50	6.36	53.61
	8		12.104	23.41	5.03	59.29
	16		22.863	21.71	4.50	61.80
	32		44.380	21.32	4.36	62.56
LiFT L/2 R5	1	50	3.027	47.91	8.58	35.52
	2		4.708	27.80	7.22	56.95
	3		6.389	22.71	6.58	65.41
	4		8.070	21.42	6.22	67.99
	5		9.751	19.81	5.47	70.97
	8		14.794	16.94	4.73	77.30
	16		28.242	16.27	4.55	79.02
	32		55.138	16.21	4.52	79.18
LiFT L/2 R10	1	50	4.708	31.83	6.22	51.22
	2		8.070	20.30	6.34	71.80
	3		11.431	14.64	4.22	84.12
	4		14.793	12.25	4.12	92.01
	5		18.155	11.70	4.33	93.53
	8		28.241	10.95	4.74	97.17
	16		55.136	10.99	4.62	96.87
	32		108.926	11.57	4.07	93.75
LiFT XL/2 R1	1	50	2.119	122.52	18.87	11.80
	2		2.543	79.77	12.75	18.44
	3		2.967	69.91	10.71	23.90
	4		3.391	60.46	11.55	27.93
	5		3.814	52.36	10.87	31.92
	8		5.086	39.83	8.97	40.41
	16		8.476	32.28	7.29	47.04
	24		11.866	31.23	7.04	47.95
LiFT XL/2 R1	32		15.256	30.85	6.93	48.22

Table 3 (continued)

Model	K_{inf}	T	TFLOPs/image	FID	sFID	IS
LiFT XL/2 R2	1	50	2.543	102.40	44.09	12.42
	2		3.390	61.26	8.36	26.55
	3		4.238	43.92	7.90	37.17
	4		5.085	35.72	7.88	44.80
	5		5.932	31.14	7.52	50.13
	8		8.474	26.08	6.79	57.16
	12		11.864	24.95	6.42	58.63
	16		15.253	24.60	6.25	58.94
	32		28.810	24.39	6.14	59.21
LiFT XL/2 R3	1	50	2.967	78.97	11.00	18.98
	2		4.238	46.80	8.62	35.51
	3		5.508	32.63	7.84	49.36
	4		6.779	26.70	7.28	57.76
	5		8.050	23.90	6.93	62.75
	8		11.863	21.96	6.40	66.05
	16		22.030	21.23	5.89	67.20
	32		42.365	21.23	5.83	67.27
LiFT XL/2 R4	1	50	3.390	99.69	15.79	14.56
	2		5.085	49.72	10.07	32.10
	3		6.779	33.52	7.56	46.02
	5		10.168	26.01	5.62	55.15
	6		11.863	26.11	6.32	54.92
	8		15.252	23.66	5.41	58.32
	16		28.808	22.18	4.93	60.49
	32		55.919	21.78	4.76	61.32
LiFT XL/2 R6	1	50	4.237	41.77	9.40	41.59
	2		6.779	23.67	7.33	64.82
	3		9.321	19.17	6.65	74.46
	4		11.862	18.08	6.22	76.97
	5		14.404	16.02	5.30	82.10
	8		22.029	13.11	4.55	91.00
	16		42.362	12.39	4.38	93.69
	32		83.029	12.20	4.33	94.54
LiFT XL/2 R8	1	50	5.085	33.91	8.97	50.74
	2		8.473	19.93	6.95	74.82
	3		11.862	16.95	6.02	81.87
	4		15.251	11.36	4.33	100.08
	5		18.640	11.11	4.25	101.13
	8		28.806	9.93	4.02	107.47
	16		55.917	9.53	3.94	109.25
	32		110.138	9.50	3.93	109.29
LiFT XL/2 R12	1	50	6.779	27.71	7.15	60.14
	2		11.862	17.07	6.41	82.66
	3		16.945	12.74	4.42	93.63
	4		22.028	10.69	4.13	102.06
	5		27.111	10.25	4.02	103.72
	8		42.361	9.55	3.96	106.79
	16		83.026	9.31	3.93	107.88
	32		164.356	9.71	3.89	106.20

A.2 DEPTH-GENERALIZATION MATRICES

Figure 8 and Table 5 report every measured point in the main depth sweeps, with rows identifying checkpoints and columns specifying inference depth. We express depth extrapolation through the ratio

$$r = \frac{K_{\text{inf}}}{K_{\text{train}}},$$

so that evaluating a two-loop checkpoint at sixteen loops corresponds to $r = 8$. Because rows differ in backbone and core size as well as in training depth, comparisons across rows also reflect the architecture.

Table 4: **Depth extrapolation for all evaluated checkpoints.** All settings use 50 integration steps and 50,000 generated images. The best tested depth minimizes FID within each checkpoint’s measured sweep; $r = K_{\text{inf}}/K_{\text{train}}$ is evaluated at that depth.

Model	K_{train}	FID at K_{train}	Best K_{inf}	Best FID	r
B/2 R4	2	37.95	16	33.87	$8\times$
L/2 R10	2	20.30	8	10.95	$4\times$
XL/2 R12	2	17.07	16	9.31	$8\times$
XL/2 R8	3	16.95	32	9.50	$10.67\times$
B/2 R2	4	41.16	16	37.18	$4\times$
L/2 R5	4	21.42	32	16.21	$8\times$
XL/2 R6	4	18.08	32	12.20	$8\times$
L/2 R4	5	27.50	32	21.32	$6.4\times$
XL/2 R4	6	26.11	32	21.78	$5.33\times$
B/2 R1	8	45.70	16	45.58	$2\times$
XL/2 R3	8	21.96	32	21.23	$4\times$
L/2 R2	10	27.75	16	27.63	$1.6\times$
XL/2 R2	12	24.95	32	24.39	$2.67\times$
L/2 R1	20	33.50	32	33.21	$1.6\times$
XL/2 R1	24	31.23	32	30.85	$1.33\times$

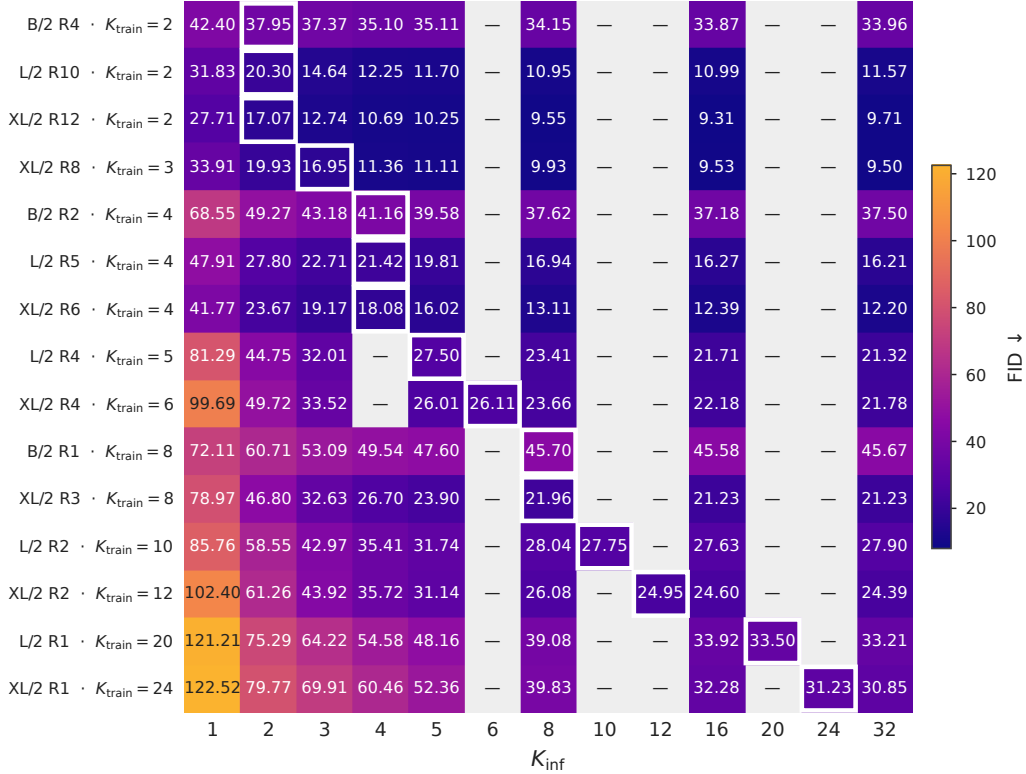


Figure 8: FID of every checkpoint in the main depth sweeps across inference depths, on a shared colour scale. The outlined cell in each row marks training depth, and grey cells indicate unevaluated settings.

Observed extrapolation. Several checkpoints improve substantially beyond training depth before their gains diminish. L/2 R10, for example, reduces FID from 20.30 to 10.95 at $r = 4$, and XL/2 R12

Table 5: FID of every checkpoint across inference depths, with the depth ratio r beneath each row. Bold values mark training depth, dashes denote unevaluated settings, and $\dagger$ marks an increase in FID relative to the preceding evaluated depth.

Model	K_{train}	K_{inf}												
		1	2	3	4	5	6	8	10	12	16	20	24	32
B/2 R4	2	42.40	37.95	37.37	35.10	35.11 [†]	–	34.15	–	–	33.87	–	–	33.96 [†]
r	–	0.50	1.00	1.50	2.00	2.50	3.00	4.00	5.00	6.00	8.00	10.00	12.00	16.00
L/2 R10	2	31.83	20.30	14.64	12.25	11.70	–	10.95	–	–	10.99 [†]	–	–	11.57 [†]
r	–	0.50	1.00	1.50	2.00	2.50	3.00	4.00	5.00	6.00	8.00	10.00	12.00	16.00
XL/2 R12	2	27.71	17.07	12.74	10.69	10.25	–	9.55	–	–	9.31	–	–	9.71 [†]
r	–	0.50	1.00	1.50	2.00	2.50	3.00	4.00	5.00	6.00	8.00	10.00	12.00	16.00
XL/2 R8	3	33.91	19.93	16.95	11.36	11.11	–	9.93	–	–	9.53	–	–	9.50
r	–	0.33	0.67	1.00	1.33	1.67	2.00	2.67	3.33	4.00	5.33	6.67	8.00	10.67
B/2 R2	4	68.55	49.27	43.18	41.16	39.58	–	37.62	–	–	37.18	–	–	37.50 [†]
r	–	0.25	0.50	0.75	1.00	1.25	1.50	2.00	2.50	3.00	4.00	5.00	6.00	8.00
L/2 R5	4	47.91	27.80	22.71	21.42	19.81	–	16.94	–	–	16.27	–	–	16.21
r	–	0.25	0.50	0.75	1.00	1.25	1.50	2.00	2.50	3.00	4.00	5.00	6.00	8.00
XL/2 R6	4	41.77	23.67	19.17	18.08	16.02	–	13.11	–	–	12.39	–	–	12.20
r	–	0.25	0.50	0.75	1.00	1.25	1.50	2.00	2.50	3.00	4.00	5.00	6.00	8.00
L/2 R4	5	81.29	44.75	32.01	–	27.50	–	23.41	–	–	21.71	–	–	21.32
r	–	0.20	0.40	0.60	0.80	1.00	1.20	1.60	2.00	2.40	3.20	4.00	4.80	6.40
XL/2 R4	6	99.69	49.72	33.52	–	26.01	26.11[†]	23.66	–	–	22.18	–	–	21.78
r	–	0.17	0.33	0.50	0.67	0.83	1.00	1.33	1.67	2.00	2.67	3.33	4.00	5.33
B/2 R1	8	72.11	60.71	53.09	49.54	47.60	–	45.70	–	–	45.58	–	–	45.67 [†]
r	–	0.12	0.25	0.38	0.50	0.62	0.75	1.00	1.25	1.50	2.00	2.50	3.00	4.00
XL/2 R3	8	78.97	46.80	32.63	26.70	23.90	–	21.96	–	–	21.23	–	–	21.23
r	–	0.12	0.25	0.38	0.50	0.62	0.75	1.00	1.25	1.50	2.00	2.50	3.00	4.00
L/2 R2	10	85.76	58.55	42.97	35.41	31.74	–	28.04	27.75	–	27.63	–	–	27.90 [†]
r	–	0.10	0.20	0.30	0.40	0.50	0.60	0.80	1.00	1.20	1.60	2.00	2.40	3.20
XL/2 R2	12	102.40	61.26	43.92	35.72	31.14	–	26.08	–	24.95	24.60	–	–	24.39
r	–	0.08	0.17	0.25	0.33	0.42	0.50	0.67	0.83	1.00	1.33	1.67	2.00	2.67
L/2 R1	20	121.21	75.29	64.22	54.58	48.16	–	39.08	–	–	33.92	33.50	–	33.21
r	–	0.05	0.10	0.15	0.20	0.25	0.30	0.40	0.50	0.60	0.80	1.00	1.20	1.60
XL/2 R1	24	122.52	79.77	69.91	60.46	52.36	–	39.83	–	–	32.28	–	31.23	30.85
r	–	0.04	0.08	0.12	0.17	0.21	0.25	0.33	0.42	0.50	0.67	0.83	1.00	1.33

from 17.07 to 9.31 at $r = 8$; both then regress slightly at larger depths, whereas L/2 R5 and XL/2 R6 reach their best tested FID at 32 loops. By contrast, the single-block cores, as well as the two-block cores in L/2 and XL/2, gain little beyond training depth.

A.3 JOINT ALLOCATION OF INFERENCE COMPUTATION

Increasing recurrent depth and increasing the number of integration steps both add sampling computation, but their effects need not be interchangeable. We therefore evaluate LiFT L/2 R10 over the Cartesian grid

$$T \in \{10, 25, 50, 100\}, \quad K_{\text{inf}} \in \{1, 2, 4, 8, 16, 32\}.$$

Because all settings use the same frozen 500,000-update checkpoint, trained at $K = 2$ with four prelude blocks, ten core blocks, and auxiliary prelude-loss weight $\lambda = 0.01$, the sweep holds trainable parameters, training tokens, and training compute fixed. Every setting follows the evaluation protocol of Section D.4, so initial noise and class labels are paired across settings.

Section 4.4 draws its main allocation results from this grid: four loops at 25 integration steps improve on the two-loop training setting at lower cost, whereas sixteen loops at ten steps are both worse and more expensive than that setting.

Diminishing returns. The gains from both recurrence and integration steps diminish at larger budgets. At eight loops, increasing the number of steps from 25 to 50 reduces FID from 11.39 to 10.95, whereas doubling the steps again to 100 leaves FID nearly unchanged at 10.96. Eight loops is

the best tested depth at all three of these step counts, while extending the rollout from eight to 32 loops worsens FID at all four budgets.

Metric dependence. Even at a fixed number of steps, the preferred depth depends on the quality metric. At 25 integration steps, for example, increasing depth from eight to 32 improves sFID from 4.36 to 3.96, even as FID worsens from 11.39 to 12.14 and IS decreases from 97.79 to 94.34.

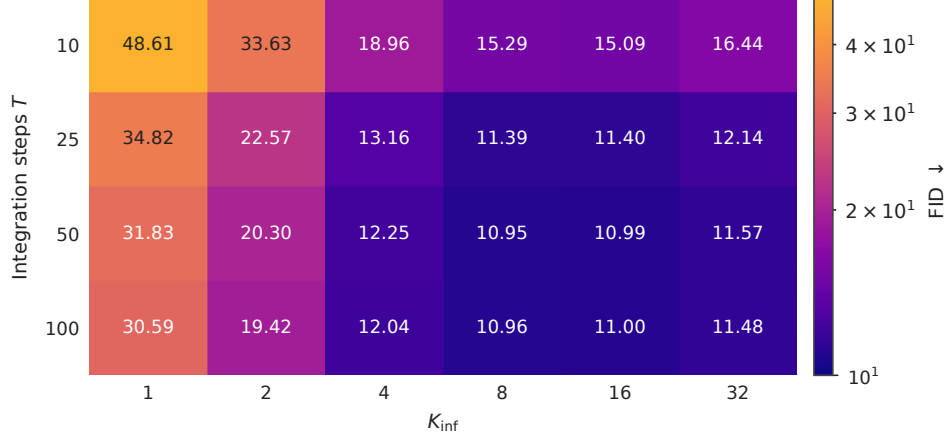


Figure 9: FID across integration steps and recurrent depth. Colour encodes FID on a logarithmic scale.

Table 6: Complete joint-sweep operating points for LiFT L/2 R10, including additional depths 3 and 5 at 50 integration steps. All metrics in a row use the same images.

T	K_{inf}	TFLOPs/image	FID	sFID	IS
10	1	0.942	48.613	10.751	41.770
	2	1.614	33.633	11.497	58.830
	4	2.959	18.955	6.107	81.803
	8	5.648	15.289	5.428	90.433
	16	11.027	15.094	5.481	90.670
	32	21.785	16.435	5.606	86.694
25	1	2.354	34.819	6.898	49.804
	2	4.035	22.570	7.270	69.810
	4	7.397	13.165	4.041	91.327
	8	14.120	11.390	4.360	97.795
	16	27.568	11.402	4.308	97.103
	32	54.463	12.142	3.963	94.345
50	1	4.708	31.828	6.223	51.216
	2	8.070	20.304	6.345	71.805
	3	11.431	14.642	4.217	84.120
	4	14.793	12.255	4.123	92.013
	5	18.155	11.702	4.326	93.526
	8	28.241	10.952	4.744	97.169
	16	55.136	10.994	4.615	96.865
	32	108.926	11.567	4.074	93.746
100	1	9.415	30.586	5.933	51.696
	2	16.139	19.422	5.926	72.730
	4	29.587	12.043	4.372	91.961
	8	56.482	10.961	5.180	95.980
	16	110.271	10.999	5.002	95.332
	32	217.851	11.484	4.330	93.193

Measured operating-point envelopes. To compare these allocations within an inference budget B , we select the measured LiFT operating point with the lowest FID:

$$(n^*, k^*) \in \underset{(n,k) \in \mathcal{G}: nC_{\text{forward}}(k) \leq B}{\operatorname{argmin}} \operatorname{FID}(n, k),$$

where $\mathcal{G}$ contains all measured operating points from the joint sweep, together with the additional depths $K_{\text{inf}} = 3, 5$ evaluated with 50 integration steps. Selecting within this set gives an observed FID, step count, and recurrent depth for each budget, without interpolating between settings. We apply the same budget constraint to dense B/2 and L/2 over $T \in \{1, 10, 25, 50, 100, 250\}$, using the analytic FLOP accounting of the main figures and excluding VAE decoding.

LiFT L/2 R10 and dense L/2 also have comparable training computation, with an estimated LiFT overhead of 0.019% (Section D.5.1), whereas dense B/2 has a smaller training budget.

Figure 5 shows these envelopes in the main text, while Table 7 records the setting selected at each measured budget.

Table 7: FID-minimizing measured settings within each inference budget. The dense column selects among both evaluated dense backbones. Budgets are rounded for display; nearby distinct costs may share a printed value.

Budget (TFLOPs)	LiFT T	LiFT K	LiFT FID	Dense setting	Dense FID
0.046	–	–	–	Dense B/2, $T=1$	321.169
0.161			–	Dense L/2, $T=1$	321.150
0.460			–	Dense B/2, $T=10$	44.486
0.942		1	48.613	Dense B/2, $T=10$	44.486
1.150			48.613	Dense B/2, $T=25$	32.501
1.614			48.613	Dense L/2, $T=10$	29.591
1.614	10	2	33.633	Dense L/2, $T=10$	29.591
2.300			33.633	Dense L/2, $T=10$	29.591
2.354			33.633	Dense L/2, $T=10$	29.591
2.959		4	18.955	Dense L/2, $T=10$	29.591
4.035			18.955	Dense L/2, $T=25$	19.047
4.035			18.955	Dense L/2, $T=25$	19.047
4.600		4	18.955	Dense L/2, $T=25$	19.047
4.708			18.955	Dense L/2, $T=25$	19.047
5.648			18.955	Dense L/2, $T=25$	19.047
5.648		8	15.289	Dense L/2, $T=25$	19.047
7.397			15.289	Dense L/2, $T=25$	19.047
8.069			15.289	Dense L/2, $T=25$	19.047
8.070		4	13.165	Dense L/2, $T=25$	19.047
9.415			13.165	Dense L/2, $T=50$	16.940
11.027			13.165	Dense L/2, $T=50$	16.940
11.431	25	4	13.165	Dense L/2, $T=50$	16.940
11.501			13.165	Dense L/2, $T=50$	16.940
11.501			13.165	Dense L/2, $T=50$	16.940
14.120		8	11.390	Dense L/2, $T=50$	16.940
14.793			11.390	Dense L/2, $T=50$	16.940
16.139			11.390	Dense L/2, $T=100$	16.173
16.139		8	11.390	Dense L/2, $T=100$	16.173
18.155			11.390	Dense L/2, $T=100$	16.173
21.785			11.390	Dense L/2, $T=100$	16.173
27.568			11.390	Dense L/2, $T=100$	16.173

Table 7 (continued)

Budget (TFLOPs)	LiFT T	LiFT K	LiFT FID	Dense setting	Dense FID
28.241	50	8	10.952	Dense L/2, $T=100$	16.173
29.587			10.952	Dense L/2, $T=100$	16.173
40.347			10.952	Dense L/2, $T=250$	15.743
54.463			10.952	Dense L/2, $T=250$	15.743
55.136			10.952	Dense L/2, $T=250$	15.743
56.482			10.952	Dense L/2, $T=250$	15.743
108.926			10.952	Dense L/2, $T=250$	15.743
110.271			10.952	Dense L/2, $T=250$	15.743
217.851			10.952	Dense L/2, $T=250$	15.743

B ABLATIONS AND DIAGNOSTICS

This appendix first ablates two training choices, the depth coordinates and the auxiliary prelude loss, and then examines how a trained model responds to changes in its inference grid. The training ablations share one setup: LiFT B/2 with four prelude blocks, a two-block shared core, four training loops, and no transformer coda. Each setting is trained once for 100,000 updates at a global batch size of 256, otherwise following the pipeline and optimization settings of [Section D](#), and generation follows the evaluation protocol of [Section D.4](#). [Sections B.3](#) and [B.4](#) then turn to a frozen L/2 checkpoint and ask how its intermediate predictions and final velocities vary with the inference grid.

B.1 RANDOM VERSUS FIXED DEPTH COORDINATES

This comparison isolates the effect of sampling the depth coordinates at random. The fixed-grid variant replaces the random interior coordinates with the uniform grid $s_k = k/K_{\text{train}}$ and uses it both to condition the core and to construct the reference targets, while the random-grid variant keeps its sampled coordinates for both. Everything else is shared: the two runs use the same architecture, depth embeddings, and trajectory supervision, and both are evaluated on the uniform grid $s_k = k/K_{\text{inf}}$. To keep the comparison free of auxiliary prelude regression, both variants set $\lambda = 0$, whereas the main experiments use $\lambda = 0.01$.

Random coordinates improve robustness away from the training depth. At four loops, the two checkpoints attain similar FID: 68.51 with random coordinates and 68.31 with fixed coordinates. The fixed-grid model reaches its best tested FID of 66.35 at five loops, slightly below the random-grid model’s best of 66.97 at eight loops. The difference between the two models becomes clearer at larger depths, where fixed-grid FID rises to 75.26 at 16 loops and 86.95 at 32, while the random-grid model stays close to its best, at 67.71 and 68.38, respectively ([Figure 10](#)). Random coordinates also improve FID at depths below the training budget, particularly at one and two loops.

sFID and IS show the same pattern at large depth: at 32 loops, random coordinates give sFID 9.26 and IS 18.92, compared with 14.87 and 15.78 for fixed coordinates. [Table 8](#) lists every evaluated setting.

Table 8: **Random versus fixed training coordinates.** Complete metrics for B/2 R2 with $K_{\text{train}} = 4$, $\lambda = 0$, and 100,000 updates. All settings use 50,000 images and 50 integration steps.

K_{inf}	Random s			Fixed s		
	FID ↓	sFID ↓	IS ↑	FID ↓	sFID ↓	IS ↑
1	98.48	15.80	14.58	151.23	57.03	6.64
2	78.57	10.75	18.19	99.47	13.12	14.88
3	70.87	9.00	19.32	75.81	8.07	18.48
4	68.51	9.43	19.49	68.31	9.45	19.55
5	67.46	9.51	19.47	66.35	10.65	19.74
8	66.97	9.43	19.31	66.60	10.44	19.25
16	67.71	9.26	19.05	75.26	11.86	17.51
32	68.38	9.26	18.92	86.95	14.87	15.78

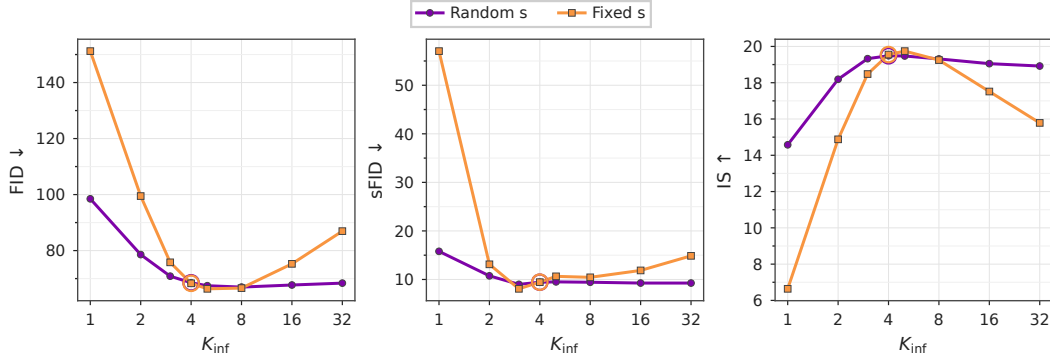


Figure 10: **Training-grid randomization improves depth robustness.** Both models use trajectory supervision with $\lambda = 0$ and are evaluated on uniform grids ending at $s = 1$. Each point uses the same 50,000 initial noise samples and class labels, EMA weights, 50 Euler steps, and no classifier-free guidance. Hollow rings mark the training depth, $K = 4$.

B.2 PRELUDE-LOSS / λ ABLATION

The prelude-loss sweep keeps trajectory supervision and random depth coordinates and varies only $\lambda \in \{0, 0.01, 0.1, 1\}$; its $\lambda = 0$ run is the random-grid run of Section B.1. As in Equation (7), the auxiliary term is added once, outside the average over recurrent predictions. Its role is to keep the initial prediction, and hence the anchor, at the scale of the target, so we track the anchor’s magnitude and error throughout training.

A small auxiliary weight controls anchor drift. Without prelude supervision, the anchor RMS rises throughout training, reaching 3.39 over the final 10,000 updates against a target RMS of 1.30, and its median MSE to the flow-matching target grows to 7.96. All three nonzero weights prevent this drift, keeping the anchor RMS near 0.93 and its MSE below 0.84 (Figure 11 and Table 9), so even $\lambda = 0.01$ suffices.

Table 9: **Prelude-loss training diagnostics.** Anchor RMS, prelude MSE, and gradient norm are medians of logged values over updates 90,000–100,000. Validation MSE uses EMA weights at update 100,000.

λ	Anchor RMS	Prelude MSE	Gradient norm	Validation MSE
0	3.3875	7.9626	0.0978	0.777666
0.01	0.9238	0.8316	0.0568	0.775813
0.1	0.9311	0.8100	0.0661	0.776069
1	0.9380	0.7992	0.1525	0.777948

Larger weights bring no further benefit. Stronger prelude regression reduces the prelude’s own error further but does not help downstream. The final validation MSE is lowest at $\lambda = 0.01$, if only by about 0.24% relative to the zero-weight run (Table 9), and FID is lowest there as well: 64.81 at $K_{\text{inf}} = 8$, compared with 66.97 without prelude supervision, 65.32 with $\lambda = 0.1$, and 70.03 with $\lambda = 1$ (Figure 12). We therefore use $\lambda = 0.01$ throughout, and Table 10 lists every evaluated depth.

B.3 PREDICTION TRAJECTORIES

Whereas the generation experiments measure how additional recurrence affects sample quality, this analysis looks inside a single evaluation of the velocity field. We first compare the intermediate readouts with the supervised reference path toward the sampled target, and then with the chord joining the model’s own initial and final estimates.

Paired evaluation protocol. We use the EMA weights of the frozen LiFT L/2 R10 checkpoint from Section A.3, trained with $K_{\text{train}} = 2$. From the ImageNet validation set, we select 512

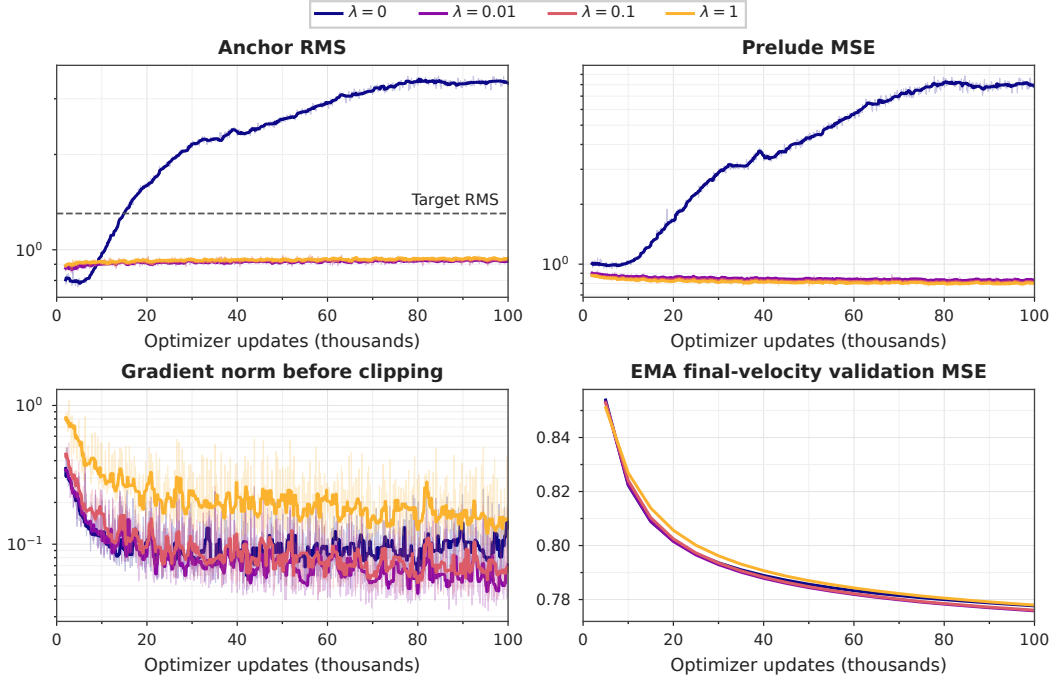


Figure 11: **Prelude supervision controls the scale of the initial prediction.** LiFT B/2 R2, trained with $K = 4$ for 100,000 updates, varying only the auxiliary weight λ . Training panels show logged values faintly and an 11-point moving median prominently, omitting the first 2,000 updates; validation curves show unsmoothed full-validation EMA velocity MSE. The dashed line marks the approximate target RMS.

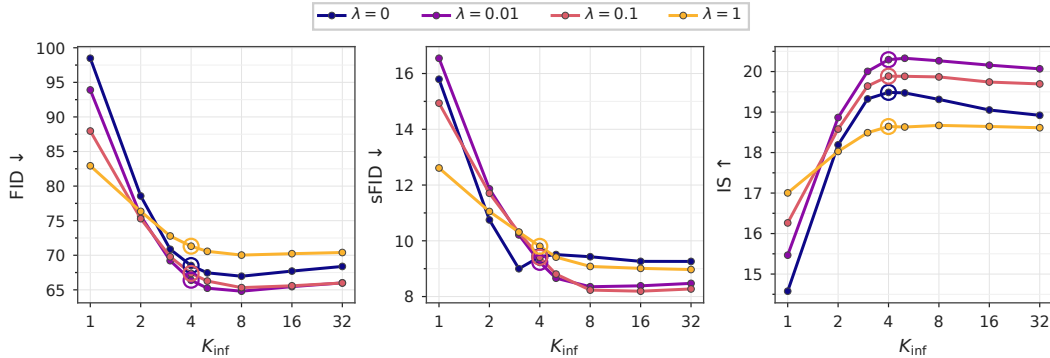


Figure 12: **Generation quality across prelude-loss weights.** FID, sFID, and IS for the same four 100,000-update checkpoints used in Figure 11. Each point uses 50,000 generated images with 50 integration steps, with common initial noise and class labels. Hollow rings mark the training depth, $K = 4$.

Table 10: **Complete prelude-loss sweep.** All runs use the same B/2 R2 architecture, $K_{\text{train}} = 4$, and 100,000 updates. Each setting uses 50,000 images, EMA weights, 50 Euler steps, and no classifier-free guidance.

λ	Metric	$K = 1$	2	3	4	5	8	16	32
0	FID	98.48	78.57	70.87	68.51	67.46	66.97	67.71	68.38
	sFID	15.80	10.75	9.00	9.43	9.51	9.43	9.26	9.26
	IS	14.58	18.19	19.32	19.49	19.47	19.31	19.05	18.92
0.01	FID	93.90	75.69	69.22	66.39	65.24	64.81	65.46	66.01
	sFID	16.55	11.86	10.21	9.23	8.66	8.35	8.39	8.47
	IS	15.47	18.87	20.00	20.29	20.33	20.27	20.16	20.07
0.1	FID	87.96	75.29	69.79	67.38	66.27	65.32	65.59	65.99
	sFID	14.94	11.71	10.30	9.41	8.81	8.23	8.19	8.27
	IS	16.26	18.58	19.64	19.89	19.88	19.87	19.74	19.69
1	FID	82.94	76.34	72.79	71.31	70.57	70.03	70.23	70.40
	sFID	12.61	11.05	10.31	9.80	9.41	9.08	9.01	8.97
	IS	17.00	18.03	18.49	18.64	18.63	18.67	18.64	18.61

unique examples, excluding exact source duplicates of training images, and evaluate each at $t \in \{0.05, 0.25, 0.5, 0.75, 0.95\}$. Every depth and grid receives the same image, noise, class label, and noisy state $x_t = (1 - t)x_0 + tx_1$. Uniform inference grids use $K_{\text{inf}} \in \{1, 2, 3, 4, 5, 8, 16, 32\}$; the trajectory plots show five representative depths from two to 32. We average measurements over generative times within each image before summarizing across images, with approximate 95% intervals given by 1.96 standard errors.

Progress toward the supervised target. For each input, let $b = u_0$ and $\Delta = u^* - b$. We decompose the displacement of readout k into progress along the target direction and an orthogonal residual:

$$\alpha_k = \frac{\langle u_k - b, \Delta \rangle}{\|\Delta\|_2^2}, \quad e_k = (u_k - b) - \alpha_k \Delta. \quad (10)$$

The reference path has $\alpha_k = s_k$ and $e_k = 0$. At uniform depth four, the measured mean projections are 0.246, 0.493, 0.737, and 0.982 at $s = 0.25, 0.5, 0.75, 1$, respectively. At the same coordinates, depth 32 gives 0.249, 0.495, 0.738, and 0.982, showing that this depth-indexed progression persists with substantially more recurrent applications than were used in training (Figure 13a).

Geometry between the model’s own predictions. To examine the realized path without reference to the sampled target, we replace Δ by the chord $q = u_K - b$ of each rollout and define

$$\beta_k = \frac{\langle u_k - b, q \rangle}{\|q\|_2^2}, \quad r_k = (u_k - b) - \beta_k q. \quad (11)$$

The projection β_k tracks s_k closely across the tested depths (Figure 13b): the mean absolute discrepancy over interior readouts is 0.0006 at $K = 2$ and 0.0051 at $K = 32$. The orthogonal residual is similar across depths as well: it is already 6.07% of the final prediction norm at the training depth $K = 2$ and reaches 8.68% at $K = 32$ (Figure 13c), so running sixteen times as many loops changes the shape of the trajectory only slightly. Table 11 reports absolute and normalized residuals over the interior readouts.

Table 11: Intermediate deviations from each rollout’s own endpoint chord. We average over interior readouts, then generative times and images; the endpoints are excluded because they lie on the chord by construction. The last two columns use different normalization scales.

K_{inf}	$ \beta_k - s_k $	$\ r_k\ _2 / \sqrt{d}$	$100\ r_k\ _2 / \ u_K - b\ _2$	$100\ r_k\ _2 / \ u_K\ _2$
2	0.0006	0.0556	0.56	6.07
4	0.0030	0.0710	0.64	7.72
8	0.0040	0.0764	0.68	8.29
16	0.0049	0.0782	0.70	8.48
32	0.0051	0.0800	0.72	8.68

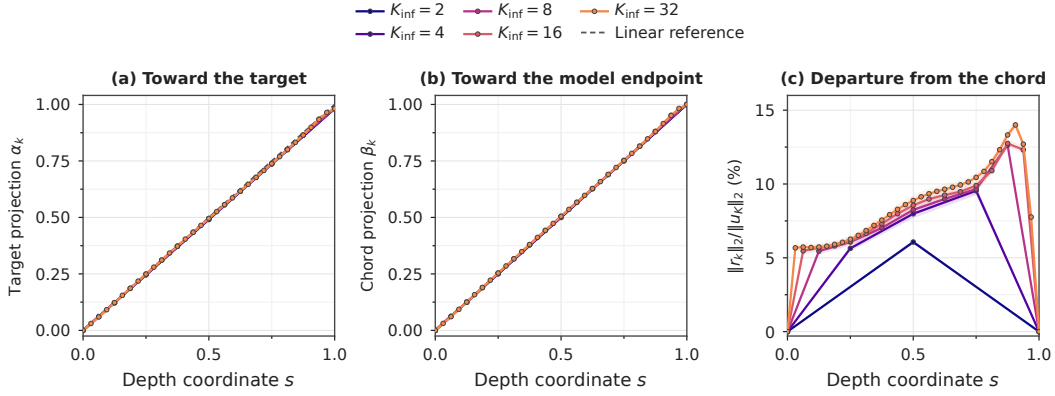


Figure 13: Prediction trajectories of LiFT L/2 R10, trained with two loops. (a) Projected progress toward the sampled target follows the assigned depth coordinate. (b) Projection onto each rollout’s own endpoint chord is close to linear, and (c) the orthogonal residual remains similar across depths. Curves average five generative times within each of 512 images; shaded bands show approximate 95% intervals across images. Chord endpoints agree by construction.

A learned progression beyond training depth. For this two-loop checkpoint, projections of intermediate predictions onto their own endpoint chord remain almost linear in the depth coordinate through 32 inference loops.

B.4 COMPUTATIONAL-GRID CONSISTENCY

Although the preceding analysis shows a regular progression within each rollout, different intermediate coordinates could still lead to different endpoints. We therefore perturb the computational grid while holding the generative state fixed, testing whether the final velocity remains consistent as the route through computational depth changes.

Protocol. We retain the checkpoint, 512 paired examples, and five generative times from [Section B.3](#). At inference depth four, we compare the uniform grid (0.25, 0.5, 0.75, 1) with front-loaded (0.1, 0.2, 0.4, 1), back-loaded (0.6, 0.8, 0.9, 1), and three independently seeded random-grid variants. We construct each random grid by sorting three uniform draws and appending the endpoint one; the same per-image grid is used at all five generative times. Uniform grids at depths 1, 2, 3, 5, 8, 16, 32 additionally test sensitivity to the number of intermediate predictions. The initial depth coordinate is zero in every case.

Endpoint agreement at fixed depth. For grid g , let $u^{(g)}$ denote its final velocity and $u^{(4)}$ the prediction on the uniform four-step grid. We measure

$$\delta_g = \frac{\|u^{(g)} - u^{(4)}\|_2}{\|u^{(4)}\|_2}, \quad c_g = \frac{\langle u^{(g)}, u^{(4)} \rangle}{\|u^{(g)}\|_2 \|u^{(4)}\|_2}. \quad (12)$$

The front-loaded and back-loaded grids alter the endpoint by a mean 1.89% and 1.82%, respectively, with mean cosine similarity above 0.9997. The three random-grid variants give similar mean differences of 1.72–1.85%, but have heavier tails: the 95th percentiles of per-image time-averaged differences are 4.26–4.61%, compared with 2.58–2.66% for the two fixed nonuniform grids ([Table 12](#)).

Changing the number of recurrent applications. Relative to uniform $K = 4$, the two-step endpoint differs by 5.12% on average, and the 32-step endpoint by 4.24% (mean cosine 0.9985), even at 16 times the training depth. These averages conceal variation with generative time ([Figure 14](#)): at $K = 32$ and $t = 0.5$, the mean difference is 6.90% and its 95th percentile 15.48%. Taking each image’s largest difference across the five times at this depth gives a 95th percentile of 18.72%. Target MSE, by contrast, barely moves, staying within 0.7667–0.7689 for all grids with $K \geq 2$.

Table 12: Paired endpoint agreement with the uniform $K = 4$ prediction. Differences are percentages of the reference prediction norm; means have approximate 95% intervals across 512 images. The image-mean p95 averages the five times within each image before taking the percentile, whereas the worst-time p95 first takes their maximum. All nonuniform grids have $K = 4$. MSE is measured against u^* .

Grid	Mean (%)	Image-mean p95	Worst-time p95	Cosine	Target MSE
Uniform $K = 1$	11.01 ± 0.20	14.88	26.86	0.99198	0.7765
Uniform $K = 2$	5.12 ± 0.12	7.63	16.59	0.99805	0.7669
Uniform $K = 3$	2.61 ± 0.07	4.08	8.92	0.99948	0.7668
Uniform $K = 4$	0.00 ± 0.00	0.00	0.00	1.00000	0.7670
Uniform $K = 5$	1.70 ± 0.05	2.80	6.64	0.99977	0.7675
Uniform $K = 8$	2.65 ± 0.10	4.88	13.98	0.99927	0.7684
Uniform $K = 16$	3.66 ± 0.14	6.85	18.94	0.99872	0.7689
Uniform $K = 32$	4.24 ± 0.13	7.03	18.72	0.99850	0.7686
Front-loaded	1.89 ± 0.04	2.66	4.72	0.99977	0.7667
Back-loaded	1.82 ± 0.04	2.58	4.84	0.99977	0.7673
Random 1	1.85 ± 0.11	4.61	7.43	0.99968	0.7671
Random 2	1.82 ± 0.11	4.51	6.95	0.99970	0.7671
Random 3	1.72 ± 0.10	4.26	7.01	0.99972	0.7671

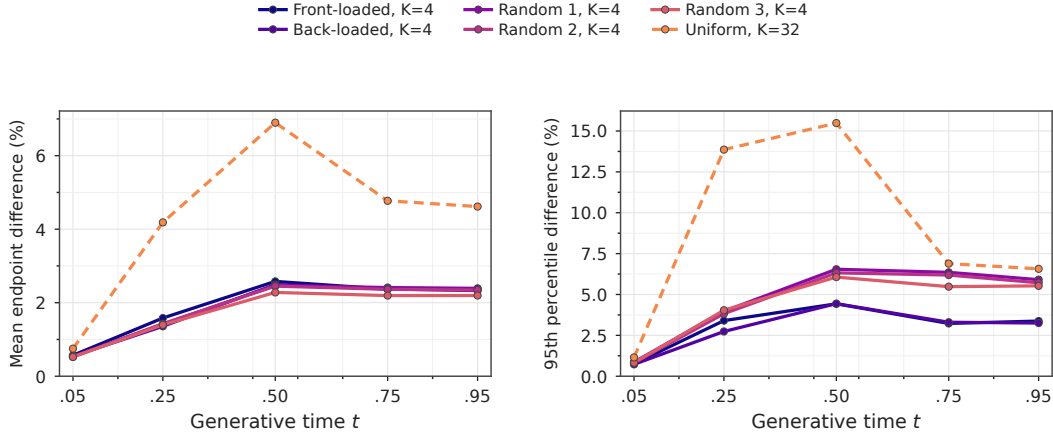


Figure 14: Sensitivity of the final velocity to the computational grid, relative to uniform $K = 4$ at the identical noisy input. The left panel shows the mean relative distance and the right shows its 95th percentile across 512 images at each generative time. Solid curves change coordinates while holding $K = 4$ fixed; the dashed curve also increases the number of recurrent applications to 32.

Numerical precision. We recompute the uniform four-step endpoint in FP32 for the same first 16 examples at each time and compare it with the BF16 prediction, yielding pooled RMSE ϵ between 0.0065 and 0.0150 (Table 13). On these identical examples, fixed-depth grid differences are comparable to this numerical scale near $t = 0.05$ but generally exceed it at intermediate times. Among the listed grids, the largest ratio of grid-difference RMSE to ϵ is 7.5, for uniform depth 32 at $t = 0.5$, so precision explains only the smallest of the observed differences.

Table 13: Numerical precision check on the same 16 examples at each time. The second column gives pooled BF16–FP32 endpoint RMSE for uniform $K = 4$. The remaining columns divide each grid’s pooled RMSE relative to that BF16 reference by the numerical baseline, using the identical examples.

t	BF16–FP32 RMSE	Front/ ϵ	Back/ ϵ	Random 1/ ϵ	$K = 32/\epsilon$
0.05	0.00663	1.2	0.9	0.8	1.2
0.25	0.00654	3.5	4.3	2.4	5.9
0.50	0.00717	3.5	3.0	2.7	7.5
0.75	0.00937	2.4	2.5	1.9	4.9
0.95	0.01500	1.5	1.4	1.1	2.8

Interpretation. Together, the trajectory and endpoint measurements support a regular depth-conditioned progression that is only mildly sensitive to the tested changes of intermediate coordinates.

C ANALYSIS OF THE LIFT OBJECTIVE

This appendix states the assumptions behind the conditional-regression result of Section 3.1, derives the training gradient with a detached anchor, and interprets the reference path through Euclidean minimum action, Gaussian transport, and Fisher geometry, with Gaussian KL divergence recovering its fitting objective.

C.1 CONDITIONAL REGRESSION RESULT

The target u^* depends on sampled endpoints that are not individually available at inference, so its regression optimum is conditional on the predictor’s inputs. Fix an initial predictor $b = b(x_t, t)$ and write $z = (x_t, t)$, including any conditioning variable when present. Let $v^*(z) = \mathbb{E}[u^* | z]$ denote the conditional outer velocity. For a fixed depth s , the reference target and its conditional mean are

$$Y_s = (1 - s)b(z) + su^*, \quad m_s(z) = \mathbb{E}[Y_s | z] = (1 - s)b(z) + sv^*(z). \quad (13)$$

For any deterministic prediction $f(z)$ and finite second moments, the squared-error decomposition is

$$\mathbb{E}[\|f(z) - Y_s\|_2^2 | z] = \|f(z) - m_s(z)\|_2^2 + s^2 \mathbb{E}[\|u^* - v^*(z)\|_2^2 | z]. \quad (14)$$

The random depth grid is independent of the endpoint pair, so conditioning additionally on the full grid $\mathcal{S}$ leaves $\mathbb{E}[u^* | z, \mathcal{S}] = v^*(z)$. The same decomposition therefore applies at each s_k , even though a recurrent prediction can depend on the preceding depth coordinates. Consequently, the unrestricted regression optimum progresses from the fixed initial predictor toward the population velocity field, reaching the ordinary flow-matching optimum at $s = 1$. These optima assume a fixed anchor function and unrestricted predictions; parameter sharing, finite capacity, and optimization can prevent their simultaneous realization.

C.2 GRADIENT FLOW WITH A DETACHED ANCHOR

During training, b is recomputed from the current model but detached within each update. If $J_\theta u_k$ is the total Jacobian of the k th readout through the unrolled computation, the trajectory gradient is

$$g_\theta = \mathbb{E}_{x_0, x_1, t, \mathcal{S}} \left[\frac{2}{Kd} \sum_{k=1}^K (J_\theta u_k)^\top (u_k - \bar{u}_{s_k}) \right]. \quad (15)$$

The Jacobian includes dependence on h_0 through both initialization and boundary reinjection, excluding only the target-side dependence through $b = \text{sg}(u_0)$. Thus, the fixed-anchor path arguments apply to the reference within each update, while the anchor itself evolves across updates.

The prelude loss in Equation (7) adds $\frac{2\lambda}{d} \mathbb{E}[(J_\theta u_0)^\top (u_0 - u^*)]$ to this gradient, directly supervising how the initial prediction is learned across updates. Since the anchor remains detached in the trajectory terms, this supervision leaves the form of the reference path and its fixed-endpoint derivations unchanged.

Scope of the construction. The regression and path arguments characterize the supervision: conditional regression describes optimal predictions for a fixed anchor, and the path arguments below characterize the reference connecting the endpoints. The recurrent core is not required to satisfy a fixed-point equation or implement a numerical solver for an inner ODE. Consequently, changing the inference-depth grid tests how the learned recurrence generalizes; these analyses provide no guarantee of reduced numerical integration error.

C.3 EUCLIDEAN MINIMUM ACTION

For the path arguments, fix a training example (x_0, x_1, t) and the endpoints $b = \text{sg}(u_0)$ and $u^* = x_1 - x_0$. Paths have unit duration and absolutely continuous means with square-integrable derivatives. The Gaussian constructions restrict every admissible path to a common, fixed covariance $\sigma^2 I$, where $\sigma > 0$. The model fits predictions at the random coordinates $\mathcal{S} = (s_0, \dots, s_K)$ defined in Equation (5), using weights $1/K$ and coordinate normalization $1/d$. These regression weights specify how the selected path is fitted, separately from the assumptions that select the path itself.

Fix $b, u^* \in \mathbb{R}^d$ and let $\Delta = u^* - b$. For any absolutely continuous path γ with square-integrable derivative and endpoints $\gamma(0) = b$, $\gamma(1) = u^*$, the endpoint constraints give $\int_0^1 \dot{\gamma}(s) ds = \Delta$. Expanding a squared residual therefore yields

$$\mathcal{A}[\gamma] = \frac{1}{2} \int_0^1 \|\dot{\gamma}(s)\|_2^2 ds = \frac{1}{2} \|\Delta\|_2^2 + \frac{1}{2} \int_0^1 \|\dot{\gamma}(s) - \Delta\|_2^2 ds. \quad (16)$$

The second term is nonnegative and vanishes if and only if $\dot{\gamma}(s) = \Delta$ almost everywhere. Integrating with the initial boundary condition gives the unique minimizer $\gamma(s) = b + s\Delta = (1-s)b + su^*$.¹

The same argument applies to a nonuniform depth grid. Let $\delta_k = s_k - s_{k-1} > 0$, $z_0 = b$, $z_K = u^*$, and interpolate the knots linearly between successive coordinates. The resulting action is

$$\begin{aligned} \mathcal{A}_{\mathcal{S}}(z_{0:K}) &= \frac{1}{2} \sum_{k=1}^K \frac{\|z_k - z_{k-1}\|_2^2}{\delta_k} \\ &= \frac{1}{2} \|\Delta\|_2^2 + \frac{1}{2} \sum_{k=1}^K \frac{\|z_k - z_{k-1} - \delta_k \Delta\|_2^2}{\delta_k}. \end{aligned} \quad (17)$$

Since $\sum_k \delta_k = 1$, the minimizing increments are $z_k - z_{k-1} = \delta_k \Delta$, giving $z_k = b + s_k \Delta = \bar{u}_{s_k}$ for every sampled grid. Uniform grids recover increments Δ/K as a special case. Thus randomizing the coordinates changes the reference knots without changing the underlying minimum-action path.

The factors $1/\delta_k$ belong to the action of the interpolated path and thereby determine its minimizing knots. Once these knots are selected as targets, Equation (6) averages their prediction errors with weights $1/K$. Applying the action directly to the model's successive increments would instead define a different loss and produce different gradients.

C.4 GAUSSIAN, FISHER, AND KL INTERPRETATIONS

C.4.1 GAUSSIAN PROBABILITY TRANSPORT

A probabilistic view lifts a path of means μ_s into the translated Gaussian family

$$q_s(w) = \mathcal{N}(w; \mu_s, \sigma^2 I), \quad \mu_0 = b, \quad \mu_1 = u^*. \quad (18)$$

¹The same straight path minimizes the action under any constant positive-definite metric; a prediction-dependent metric can instead yield a curved path.

Here $w \in \mathbb{R}^d$ is a velocity-space variable, and the auxiliary distributions describe the fixed training example. The translation field $a_s(w) = \dot{\mu}_s$ transports this family and satisfies

$$\partial_s q_s(w) + \nabla_w \cdot (q_s(w) a_s(w)) = 0, \quad \mathcal{A}_{\text{prob}}[q, a] = \frac{1}{2} \int_0^1 \int_{\mathbb{R}^d} \|a_s(w)\|_2^2 q_s(w) dw ds. \quad (19)$$

Any admissible field transporting this density path satisfies $\mathbb{E}_{q_s}[a_s(w)] = \dot{\mu}_s$, assuming finite energy and sufficient decay for integration by parts. Jensen's inequality then gives $\mathbb{E}_{q_s} \|a_s(w)\|_2^2 \geq \|\dot{\mu}_s\|_2^2$, with equality for the constant field $a_s = \dot{\mu}_s$. Minimizing the transport action within this fixed-covariance family thus reduces to minimizing $\frac{1}{2} \int_0^1 \|\dot{\mu}_s\|_2^2 ds$. By the preceding derivation, the reference distribution is

$$\bar{q}_s = \mathcal{N}((1-s)b + su^*, \sigma^2 I). \quad (20)$$

Associating each readout with $q_{\theta,k} = \mathcal{N}(u_k, \sigma^2 I)$ and fitting it to $\bar{q}_{s_k}$ by KL divergence gives the trajectory objective, as derived in [Section C.4.3](#). In this interpretation, the auxiliary Gaussians are transported by shifting their means while keeping covariance fixed across depth. The deterministic network passes the terminal mean to the outer sampler as its velocity prediction; it neither samples w nor estimates posterior uncertainty.

C.4.2 FISHER GEOMETRY WITH FIXED COVARIANCE

The fixed-covariance family also gives the reference an information-geometric interpretation. For $q(w; \mu) = \mathcal{N}(w; \mu, \sigma^2 I)$, the score with respect to the mean and its Fisher information matrix are

$$\nabla_\mu \log q(w; \mu) = \sigma^{-2}(w - \mu), \quad G_F(\mu) = \mathbb{E}_q [(\nabla_\mu \log q)(\nabla_\mu \log q)^\top] = \sigma^{-2} I. \quad (21)$$

The Fisher metric on this mean submanifold is constant, so its path energy is a scaled Euclidean action:

$$\mathcal{A}_F[\mu] = \frac{1}{2} \int_0^1 \dot{\mu}_s^\top G_F \dot{\mu}_s ds = \sigma^{-2} \mathcal{A}[\mu]. \quad (22)$$

The unique constant-speed energy minimizer between the specified endpoints is therefore $\mu_s = \bar{u}_s$. Fitting the predicted Gaussian means to this geodesic by KL again gives [Equation \(6\)](#).

This conclusion requires covariance to be fixed along every admissible path. Equal endpoint covariances alone are insufficient, because allowing covariance to vary changes the manifold and can give geodesics that leave this subfamily [\[35\]](#). A common, constant positive-definite covariance Σ would still give a straight mean geodesic under $G_F = \Sigma^{-1}$, but its KL fitting criterion would be a Mahalanobis error. The isotropic assumption makes that error proportional to the unweighted MSE used here.

C.4.3 GAUSSIAN KL PROJECTION

Once the reference path is specified, Gaussian KL provides a direct derivation of its fitting loss. At a depth s_k , let $\bar{q}_k = \mathcal{N}(\bar{u}_{s_k}, \sigma^2 I)$ and $q_{\theta,k} = \mathcal{N}(u_k, \sigma^2 I)$. Their log-density ratio gives

$$\begin{aligned} D_{\text{KL}}(\bar{q}_k \| q_{\theta,k}) &= \frac{1}{2\sigma^2} \mathbb{E}_{w \sim \bar{q}_k} [\|w - u_k\|_2^2 - \|w - \bar{u}_{s_k}\|_2^2] \\ &= \frac{1}{2\sigma^2} \|u_k - \bar{u}_{s_k}\|_2^2. \end{aligned} \quad (23)$$

The cross term vanishes because $\mathbb{E}_{\bar{q}_k} [w - \bar{u}_{s_k}] = 0$. Thus, after averaging over training examples and recurrent depths,

$$\mathcal{L}_{\text{KL}} = \mathbb{E}_{x_0, x_1, t, \mathcal{S}} \left[\frac{1}{K} \sum_{k=1}^K D_{\text{KL}}(\bar{q}_k \| q_{\theta,k}) \right] = \frac{d}{2\sigma^2} \mathcal{L}_{\text{LiFT}}. \quad (24)$$

The factor d accounts for coordinate averaging in the implemented MSE. Because σ is fixed, the objectives differ only by a positive constant and have proportional gradients under the same detached-boundary convention. Their equivalence therefore requires no Gaussian sampling or variance parameter in the implementation. Whereas the action and geometric arguments select a reference path, KL recovers the criterion used to fit it once that path is specified.

D REPRODUCIBILITY AND IMPLEMENTATION

The main comparisons share the same data pipeline, transformer block design, and optimization settings, allowing dense and recurrent architectures to be evaluated under a common protocol. [Section A](#) provides the numerical results for these comparisons.

D.1 DATA AND ARCHITECTURE

We use class-conditional ImageNet-1k at 256×256 resolution. Images follow the ADM/DiT preprocessing convention: BOX downsampling, bicubic resizing, and a 256×256 center crop. Each preprocessed image is encoded into one cached SD-VAE posterior sample, with posterior noise drawn on CPU from a seed determined by the cache seed and global image index. The resulting $4 \times 32 \times 32$ latents are scaled by 0.18215 and stored in FP16. Training reuses these cached representations while sampling fresh flow noise and generative times.

Dense DiT and LiFT use the same SpeedrunDiT-adapted transformer blocks, with RMSNorm, per-head query/key normalization, two-dimensional rotary embeddings, GELU MLPs, and adaLN-zero-style conditioning with gated residual branches. [Table 14](#) gives the backbone dimensions; all models use 2×2 latent patches, yielding 256 tokens per image.

Table 14: Backbone dimensions shared by the dense and recurrent models. Dense depth counts transformer blocks.

Backbone	Width	Attention heads	Dense depth
B/2	768	12	12
L/2	1024	16	24
XL/2	1152	16	28

Within this shared block design, LiFT uses four prelude blocks and no transformer blocks in the coda for the main comparisons. A shared prediction head reads out the prelude and recurrent states during training, whereas sampling requires only the final readout. Because the core is shared, each LiFT model has fewer parameters than its dense backbone. The depth embedding adds a small overhead relative to a dense model with the same number of unique blocks, and the parameter count remains fixed when the loop count changes. [Table 15](#) collects the configurations and their analytic training costs.

D.2 TRAINING PROTOCOL

The main protocol uses 500,000 optimizer updates with a global batch size of 256 and the optimization and numerical settings in [Table 16](#). Under this common protocol, dense models use conditional flow matching, while LiFT uses the trajectory objective with an auxiliary prelude loss of weight $\lambda = 0.01$, added once outside the average over recurrent predictions.

Both model families use standard Gaussian flow noise, uniform generative times, and a linear noise-to-data interpolant, retaining class labels throughout training without label dropout. LiFT also samples interior depth coordinates independently for each example and update, then sorts them and appends the endpoint $s = 1$. The number of recurrent applications remains fixed within each run.

The reported B/2, L/2, and XL/2 checkpoints in [Section A.1](#) were trained on four NVIDIA H100 GPUs, with 64 examples per GPU and no gradient accumulation; [Table 15](#) reports their analytic training costs, computed as in [Section D.5](#).

D.3 TRAINING AND SAMPLING PSEUDOCODE

In each training update, [Algorithm 1](#) independently samples the noise, generative time, and depth coordinates for every example in the minibatch. Each MSE averages over examples and latent coordinates, while the loop average is written explicitly. The prelude weight λ controls direct supervision of the initial prediction, using the experimental value given in [Section D.2](#). Only the copy of that prediction used in the trajectory target is detached; the auxiliary loss differentiates through the prediction itself.

Table 15: Complete model configurations and training resources for the checkpoints in [Section A.1](#), with R denoting the shared-core block count. All LiFT models use four prelude blocks and no transformer coda, giving $L_{\text{unique}} = 4 + L_R$ unique blocks and $L_{\text{exec}}^{\text{train}} = 4 + K_{\text{train}}L_R$ executed blocks per training forward pass. Executed depth matches the dense reference within each backbone scale, with its shared value centered across each LiFT group. Parameter counts N are obtained from the implementation and include conditioning and readout parameters, whereas block counts omit this work. The analytic forward-plus-backward estimate C_{train} accounts for it using the protocol in [Section D.2](#) and the accounting in [Section D.5](#), and is reported in EFLOPs (10^{18} FLOPs).

Model	L_R	K_{train}	L_{unique}	$L_{\text{exec}}^{\text{train}}$	N (M)	C_{train} (EFLOPs)
<i>Dense references</i>						
DiT-B/2	–	–	12	12	130.30	17.666
DiT-L/2	–	–	24	24	457.83	61.973
DiT-XL/2	–	–	28	28	674.82	91.098
<i>LiFT configurations</i>						
LiFT-B/2-R1	1	8	5	12	56.69	17.697
LiFT-B/2-R2	2	4	6		67.32	17.681
LiFT-B/2-R4	4	2	8		88.58	17.673
LiFT-L/2-R1	1	20	5	24	100.23	62.089
LiFT-L/2-R2	2	10	6		119.12	62.031
LiFT-L/2-R4	4	5	8		156.90	62.002
LiFT-L/2-R5	5	4	9		175.79	61.996
LiFT-L/2-R10	10	2	14		270.24	61.984
LiFT-XL/2-R1	1	24	5	28	126.62	91.263
LiFT-XL/2-R2	2	12	6		150.53	91.181
LiFT-XL/2-R3	3	8	7		174.43	91.153
LiFT-XL/2-R4	4	6	8		198.34	91.139
LiFT-XL/2-R6	6	4	10		246.15	91.125
LiFT-XL/2-R8	8	3	12		293.96	91.118
LiFT-XL/2-R12	12	2	16		389.58	91.111

Table 16: Shared settings for the main training protocol.

Setting	Value
Optimizer	AdamW
Training updates	500,000
Global batch size	256
Learning rate	10^{-4}
Schedule	500 warmup updates, then constant
Adam coefficients	(0.9, 0.999)
Adam ϵ	10^{-8}
Weight decay	0
Gradient clipping	Global norm 1
EMA decay	0.999
Training seed	42
Computation precision	BF16
Parameter and loss-accumulation precision	FP32

Algorithm 1 LiFT trajectory training with random depth coordinates**Require:** Parameters θ , training depth $K \geq 1$, minibatch (x_1, y) , prelude weight λ

- 1: Sample $x_0 \sim \mathcal{N}(0, I)$ and $t \sim \mathcal{U}(0, 1)$ per example
- 2: $x_t \leftarrow (1 - t)x_0 + tx_1$; $u^* \leftarrow x_1 - x_0$
- 3: Draw $K - 1$ independent uniform depths and sort them into $s_1, \dots, s_{K-1}$
- 4: Set $s_0 \leftarrow 0$, $s_K \leftarrow 1$ {No interior samples when $K = 1$ }
- 5: $c \leftarrow e_t(t) + e_y(y)$; $h_0 \leftarrow P_\theta(x_t, c)$; $h \leftarrow h_0$
- 6: $u_0 \leftarrow \mathcal{D}_\theta(h_0, c)$; $b \leftarrow \text{sg}(u_0)$
- 7: $\ell \leftarrow \lambda \text{MSE}(u_0, u^*)$ {Outside the loop average}
- 8: **for** $k = 1, \dots, K$ **do**
- 9: $h \leftarrow R_\theta(\text{RMSNorm}(h) + h_0, c + e_s(s_k))$
- 10: $u_k \leftarrow \mathcal{D}_\theta(h, c)$
- 11: $\bar{u}_k \leftarrow (1 - s_k)b + s_k u^*$
- 12: $\ell \leftarrow \ell + \text{MSE}(u_k, \bar{u}_k)/K$
- 13: **end for**
- 14: Backpropagate ℓ through all predictions and recurrent hidden states
- 15: Update θ with the optimizer; update the EMA parameters

To preserve numerical ordering and endpoint spacing, the implementation draws and sorts interior coordinates in FP64, then applies $s_i = (1 - K\epsilon)r_{(i)} + i\epsilon$, with $\epsilon = 10^{-6}$, before casting the grid to FP32. Here $r_{(i)}$ is the i th sorted uniform draw, and the endpoints remain exactly zero and one. The resulting separation leaves the loss weights unchanged and introduces no step-size factor into the core.

Algorithm 2 LiFT sampling with T Euler steps and K_{inf} inner loops**Require:** EMA parameters, condition y , integration steps T , inner loops K_{inf}

- 1: Sample $x \sim \mathcal{N}(0, I)$
- 2: **for** $j = 0, \dots, T - 1$ **do**
- 3: $t \leftarrow j/T$; $c \leftarrow e_t(t) + e_y(y)$
- 4: $h_0 \leftarrow P_\theta(x, c)$; $h \leftarrow h_0$
- 5: **for** $k = 1, \dots, K_{\text{inf}}$ **do**
- 6: $s \leftarrow k/K_{\text{inf}}$
- 7: $h \leftarrow R_\theta(\text{RMSNorm}(h) + h_0, c + e_s(s))$
- 8: **end for**
- 9: $x \leftarrow x + \mathcal{D}_\theta(h, c)/T$
- 10: **end for**
- 11: **return** Decode x with the latent tokenizer’s decoder

At inference, only the final prediction advances the outer sample, so the sampling algorithm omits intermediate readouts and the training anchor. It uses no classifier-free guidance (scale 1), as specified in the evaluation protocol.

D.4 EVALUATION PROTOCOL

Each setting generates 50,000 images, with 50 samples per class. Sample $i \in \{0, \dots, 49,999\}$ receives class label $i \bmod 1000$ and initial Gaussian noise generated in FP32 on CPU with seed $200042 + i$. This assignment pairs the initial noise and class labels across models and sampling budgets. We preserve these sample identities across distributed generation and resumed jobs to maintain the pairing throughout evaluation.

Sampling uses EMA weights and Euler integration from noise to data, without classifier-free guidance (scale 1). Each integration step requires one velocity evaluation. Depth sweeps hold the number of steps at 50, while step sweeps vary it. At every integration step, LiFT reinitializes the prelude state and uses the uniform inference-depth coordinates $s_k = k/K_{\text{inf}}$.

Transformer inference uses BF16, while Euler updates and VAE decoding use FP32. We decode samples with `stabilityai/sd-vae-ft-mse`, using a latent scaling factor of 0.18215.

The decoded samples are evaluated with the official ADM Inception evaluator. FID and sFID compare features from all 50,000 generated images with the published ImageNet reference statistics, while IS is computed from the same generated images.

D.5 COMPUTE ACCOUNTING

The 32×32 latent grid contains $n = (32/2)^2 = 256$ patch tokens. With $U = 500,000$ updates and global batch size $B_{\text{batch}} = 256$, the training-data budget is

$$D = UB_{\text{batch}} n = 3.2768 \times 10^{10} \quad (25)$$

latent-patch tokens, corresponding to 128 million image presentations. This data budget counts repeated image presentations, with recurrent computation over the same tokens accounted for separately below. Parameter counts include all trainable embedding, conditioning, transformer, and readout parameters.

To account for computation, we estimate FLOPs from the architecture, counting two operations per multiply-add. Let $C_{\text{forward}}^{\text{train}}(K_{\text{train}})$ denote the forward cost per example during training. For LiFT, this includes the prelude, all recurrent applications, depth conditioning, and the anchor and intermediate readouts. For the dense baseline, it includes the ordinary forward pass and final prediction. Total training computation is estimated as

$$C_{\text{train}} \approx 3UB_{\text{batch}} C_{\text{forward}}^{\text{train}}(K_{\text{train}}), \quad (26)$$

with the factor of three approximating the combined forward and backward cost. The depth argument is omitted for dense models. [Section D.5.1](#) expands both costs in terms of the individual model components.

Sampling requires only the final readout. Its cost per generated image is therefore

$$C_{\text{infer}} = T C_{\text{forward}}(K_{\text{inf}}), \quad (27)$$

where C_{forward} includes patch embedding, the prelude, every core application, conditioning, and the final prediction head, but excludes SD-VAE decoding.

Within each backbone scale, matching executed transformer depth approximately matches training computation; the following derivation quantifies the remaining overhead from conditioning and readouts when the coda contains no transformer blocks.

D.5.1 COST OF TRAJECTORY SUPERVISION

Because trajectory supervision reads out every recurrent state, each operation in the prediction branch contributes repeatedly to the training cost. A zero-block coda, paired with the four-block prelude used in our comparisons, limits this repeated computation to the prediction head. We quantify its cost using the analytic accounting of [Section D.5](#), with two operations per multiply-add.

General training and sampling costs. Let L_P , L_R , and L_D denote the numbers of transformer blocks in the prelude, shared core, and prediction-only coda, respectively. Write f_{blk} for the forward cost of one block, f_{head} for the prediction head after the coda, f_s for one depth embedding, and f_{io} for the patch embedding and outer-time conditioning computed once per example. With $K = K_{\text{train}}$, training computes $u_0, \dots, u_K$, giving

$$C_{\text{forward}}^{\text{train}}(K) = f_{\text{io}} + K f_s + [L_P + K L_R + (K + 1) L_D] f_{\text{blk}} + (K + 1) f_{\text{head}}. \quad (28)$$

The $K + 1$ prediction branches include the initial estimate used as the anchor and every recurrent readout. Sampling requires only the final prediction, reducing the cost at depth $K = K_{\text{inf}}$ to

$$C_{\text{forward}}(K) = f_{\text{io}} + K f_s + (L_P + K L_R + L_D) f_{\text{blk}} + f_{\text{head}}. \quad (29)$$

Although all recurrent applications share the same parameters, each application contributes to the computation in these expressions.

Because the stop-gradient on $b = \text{sg}(u_0)$ removes only the target-side derivative, backpropagation still traverses the prelude state and every recurrent transition, and the auxiliary prelude loss also

differentiates through u_0 . The usual approximation of the backward pass as twice the forward cost therefore applies, and substituting these forward costs into [Equations \(26\) and \(27\)](#) gives the total training and inference budgets. The forward cost includes the anchor computation whether or not the auxiliary prelude loss is enabled.

Why we use a zero-block coda. Consider a dense model with the same block design and width, and $L = L_P + KL_R + L_D$ blocks. Its forward cost is $C_{\text{forward}}^{\text{dense}} = f_{\text{io}} + Lf_{\text{blk}} + f_{\text{head}}$. This dense model matches LiFT’s executed transformer depth at sampling depth K , but LiFT’s trajectory training incurs an additional

$$C_{\text{forward}}^{\text{train}}(K) - C_{\text{forward}}^{\text{dense}} = K(L_D f_{\text{blk}} + f_{\text{head}} + f_s). \quad (30)$$

The repeated coda evaluations account for the transformer-block overhead that remains when only sampling depth is matched. We therefore choose $L_D = 0$ and $L_P + K_{\text{train}}L_R = L$ in the matched-training comparisons. This choice equalizes transformer-block computation, leaving the additional prediction heads and depth embeddings as the remaining overhead. Under the same training-cost approximation, their relative cost is

$$\frac{C_{\text{train}}^{\text{LiFT}}}{C_{\text{train}}^{\text{dense}}} - 1 = \frac{K_{\text{train}}(f_{\text{head}} + f_s)}{C_{\text{forward}}^{\text{dense}}}, \quad (31)$$

for equal update and global-batch budgets. The total FLOPs are therefore approximately, rather than exactly, matched.

Numerical accounting for our models. For n patch tokens, hidden width w , MLP width m , patch dimension q , and sinusoidal embedding dimension f , our implementation counts

$$\begin{aligned} f_{\text{blk}} &= n(8w^2 + 4wm + 4nw) + 12w^2, & f_s &= 2(fw + w^2), \\ f_{\text{io}} &= 2nqw + 2(fw + w^2), & f_{\text{head}} &= 4w^2 + 2nwq. \end{aligned} \quad (32)$$

The block expression includes attention projections, both attention matrix products, the MLP, and the conditioning projection. For our SD-VAE models, $n = 256$, $q = 2^2 \times 4 = 16$, $m = 4w$, and $f = 256$. Including all supervised readouts gives the representative comparisons in [Table 17](#).

Table 17: Analytic training compute for representative matched-depth configurations. Each model uses 500,000 updates and a global batch size of 256; one EFLOP is 10^{18} operations. LiFT uses $L_P = 4$ and $L_D = 0$. The totals include the forward and approximate backward cost of conditioning and all trajectory readouts. Overheads are computed before rounding the totals.

Backbone	L_R	K_{train}	Dense (EFLOPs)	LiFT (EFLOPs)	Extra cost
B/2	2	4	17.6655	17.6812	0.0889%
L/2	10	2	61.9726	61.9842	0.0188%
XL/2	4	6	91.0976	91.1391	0.0455%

Across all matched-depth configurations in [Table 15](#), the additional cost ranges from 0.044% to 0.178% for B/2, from 0.019% to 0.188% for L/2, and from 0.015% to 0.182% for XL/2. Thus, every configuration differs from its same-scale dense reference by less than 0.2% under this accounting.