TRAINING NEURAL ODES BY PATH MATCHING

Anonymous authors
Paper under double-blind review

ABSTRACT

The prohibitive training cost of Neural ODEs (NODEs) stems from backpropagation through the ODE solver, whose cost scales with the number of function evaluations (NFE). Flow Matching (FM) removes this backpropagation by reframing training as field regression, but supervising the field on pre-defined trajectories (e.g. straight lines) creates a *path-crossing* conflict: on paired data the field simply averages conflicting velocities. We present *Path Matching* (PM), which resolves this conflict by matching the field *on-policy*: rather than regressing onto a prescribed path, we roll out the model and supervise the field at the states x_t it actually visits, with a bridge target that realigns a drifted rollout back onto a target path (a straight line between endpoints by default). On image classification and a real, irregularly-sampled clinical time-series task, PM reaches accuracy on par with end-to-end NODE training at a fraction of the per-step cost and with no backprop through the solver, an efficiency we characterize with a runtime scaling law. A latent-dimension ablation then explains why the simulation-free alternative is fragile: it fails when a straight coupling cannot avoid path crossings—a limitation PM's curvable, on-policy objective escapes.

1 INTRODUCTION

Neural Ordinary Differential Equations (NODEs) (Chen et al., 2018) parameterize the derivative of a hidden state by a neural network, $\dot{z}(t) = v_\theta(z(t), t)$, and obtain the output by numerically integrating this field with an off-the-shelf ODE solver. This continuous-depth formulation is appealing for several reasons: it defines a model with *constant memory* (gradients are computed via the adjoint state rather than by storing activations), an *adaptive* amount of computation (the solver spends more evaluations where the dynamics are stiff), and a smooth, invertible map between input and output. These properties have made NODEs a natural modeling tool across domains. In *physically informed modeling* they impose known conservation laws and continuous-time structure on learned dynamics (Raissi et al., 2019); in *time-series modeling* they handle irregularly sampled observations by integrating the latent state between arbitrary timestamps (Rubanova et al., 2019); and in *generative modeling* they underlie continuous normalizing flows, where the same ODE transports a simple prior to a complex data distribution with a tractable change of variables (Grathwohl et al., 2019).

The cost bottleneck. Despite this promise, NODEs remain *under-explored at scale*, and the reason is training cost. Learning v_θ by gradient descent requires differentiating through the numerical integration itself: each optimization step backpropagates through the entire solver trajectory, whose length is the number of function evaluations (NFE). For adaptive solvers on non-trivial dynamics the NFE routinely reaches hundreds or thousands, so a single training step unrolls through as many composed applications of v_θ, making training slow and memory-intensive and destabilizing gradients as the field stiffens (Grathwohl et al., 2019; Finlay et al., 2020). Backpropagation through the solver is thus the central obstacle standing between NODEs and the scale at which other architectures operate.

Flow matching and its limits on paired data. *Flow Matching* (FM) (Lipman et al., 2023) removes this obstacle by reframing training as a *field-regression* task: instead of integrating and backpropagating, one prescribes a target path[1] $w(t)$ between endpoints and regresses v_θ onto its

[1]Throughout, "path" denotes the predefined/target path $w(t)$ and "trajectory" the learned/rollout trajectory; we avoid the word "flow" to distinguish our setting from generative modeling.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Table 1: Conceptual comparison of Flow Matching (FM), standard Neural ODE training, and Path Matching (PM). PM combines on-policy state visitation with regression-based training, avoiding backpropagation through the ODE solver.

Method	ODE Rollout	On-policy States	Solver BP	Drift Correction	Training Objective
Flow Matching	✗	✗	✗	✗	Path regression
Neural ODE	✓	✓	✓	✓	End-to-end task loss
Path Matching	✓	✓	✗	✓	On-policy regression

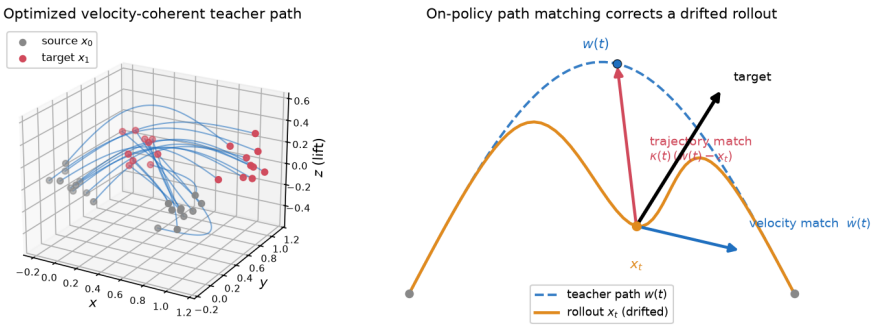


Figure 1: **Left:** the data-geometry target path. **Right:** on-policy correction—the field is supervised at the rollout state, and the bridge target pulls a drifted trajectory back onto the path.

time-derivative $\dot{w}(t)$ at sampled times—no solver in the loop, so no backpropagation through one. FM was designed for *generative* modeling, where the endpoints are *free*: the coupling between a prior sample and a data sample can be chosen, and the target paths (e.g. straight lines) can be arranged so they do not conflict. On *paired* data—where a specific x_0 must map to a specific x_1, as in regression, classification, or physical transport—this freedom disappears. Prescribed paths for different pairs generically *cross*: where two paths pass close in space but carry different velocities, the regression target becomes multivalued, the field averages the conflicting velocities, and samples are driven into the wrong basin. SFNO (Kim et al., 2024), the leading simulation-free method for paired data, avoids the conflict by running flow matching *in a learned latent space* that absorbs the crossings. This sidesteps rather than resolves the problem, at two costs: transport becomes uninterpretable in the original space (undermining exactly the physical and time-series applications above), and, as we show, the recipe is brittle—it collapses when the latent is too small for a non-crossing straight coupling to exist (Section 3.3).

Contributions. We present *Path Matching* (PM), which trains NODEs efficiently by *resolving* the path-crossing conflict in the original space rather than sidestepping it in a latent one—keeping FM's solver-free efficiency while restoring interpretable, crossing-free transport on paired data. Concretely:

1. **On-policy supervision.** PM supervises the field at the states the model's own rollout visits, not at points read off a prescribed path, so the supervision adapts to the learned dynamics—with no backprop through the solver.

2. **A contractive bridge target.** We pair path-velocity matching with a state-dependent correction whose gain makes the drift off the target path provably contract, stabilizing the rollout while adding no solver cost.

3. **Efficiency, characterized.** Across controlled transport tasks, image and time-series benchmarks, architecture ablations, and a runtime scaling law, PM matches end-to-end NODE accuracy at a fraction of the per-step cost, and we pin down why.

Table 2: Training complexity comparison. M denotes the number of function evaluations (NFEs) required by the ODE solver, and C_f and C_b denote the cost of one forward and backward evaluation of the velocity field, respectively.

Method	ODE Rollout	Solver BP	Forward Cost	Backward Cost
Flow Matching	✗	✗	$O(C_f)$	$O(C_b)$
Neural ODE	✓	✓	$O(MC_f)$	$O(MC_b)$
Path Matching	✓	✗	$O(MC_f)$	$O(C_b)$

2 METHOD

2.1 BACKGROUND: THE FLOW-MATCHING LOSS

Flow matching trains a velocity field v_θ to transport x_0 to x_1 along a *target path* $w(t)$ (with $w(0) = x_0$, $w(1) = x_1$) by regressing it onto that path's velocity $\dot{w}(t)$. Sampling a time t and the path point $x_t = w(t)$,

$$\mathcal{L} = \mathbb{E}_t \left\| v_\theta(x_t, t) - \mathrm{sg}[\dot{w}(t)] \right\|^2 . \tag{1}$$

2.2 THE TARGET PATH

A target path is only learnable if nearby particles move *coherently*: where two paths pass close in space their velocities must agree, or the field regressed on them is multivalued and stiff. This is the sole requirement PM places on the path $w(t)$ between the fixed endpoints. By default we take $w(t)$ to be the straight line between endpoints,

$$w(t) = (1 - t)\, x_0 + t\, x_1, \qquad \dot{w}(t) = x_1 - x_0, \tag{2}$$

the simplest coherent choice and the one used throughout our main experiments (Section 3); crucially, each pair (x_0, x_1) gets its own straight segment, so—unlike a shared class anchor—coherence only needs to hold locally between nearby pairs, not globally across a class. Section 3 additionally tests target paths built from the data geometry (the explicit construction is in Appendix B) as a supplementary refinement, and introduces a *path-stiffness* measure (σ) used to evaluate candidate paths.

2.3 PATH MATCHING (ON-POLICY) AND THE BRIDGE LOSS

The on-policy state x_t is obtained by *running the solver*: starting from x_0, we integrate the current field forward, $x_t = x_0 + \int_0^t v_\theta(x_s, s)\, ds$, rather than reading a point off the prescribed path $w(t)$—so x_t is wherever the model's own rollout actually lands at time t, not the corresponding path point $w(t)$. We supervise the field at this rollout state x_t with a *bridge* target that adds a trajectory-matching term, pulling a drifted rollout back onto $w(t)$,

$$\mathrm{target}(x_t, t) = \underbrace{\dot{w}(t)}_{\text{reference velocity}} + \underbrace{\kappa(t)\,(w(t) - x_t)}_{\text{velocity correction}}, \qquad \kappa(t) = 1/(1 - t). \tag{3}$$

The gain $\kappa(t) = 1/(1 - t)$ grows as $t \to 1$, landing the rollout on the exact endpoint x_1. At convergence the rollout tracks the path, $x_t = w(t)$, the trajectory-matching term vanishes, and the loss reduces to pure velocity matching. The bridge is contractive in the drift $e(t) = x_t - w(t)$ ($\dot{e} = -\kappa(t)\, e$), so a perturbed rollout is pulled back onto the path; Appendix A derives this, contrasts it with the endpoint-aligned target it replaces, and shows empirically (Table 9) that PM is insensitive to the schedule's exact shape or cap—even the velocity-only target ($\kappa \equiv 0$) stays within 0.5% on CIFAR-10.

3 EXPERIMENTS

After fixing the shared setup (Section 3.1), our main result benchmarks PM against the NODE-training baselines on image classification (Section 3.2), and asks both where its accuracy comes

from and why it is cheaper per step. We then vary the architecture to expose when the simulation-free baseline breaks (Section 3.3), and finally isolate the crossing conflict PM is designed for on controlled paired-transport tasks (Section 3.4).

3.1 EXPERIMENTAL SETUP

Baselines and datasets. We follow the SFNO (Kim et al., 2024) latent-flow setup—an encoder lifts an image to a latent z_0, a NODE transports $z_0 \to z_1$, and a head decodes z_1 into logits—and evaluate on *MNIST*, *SVHN*, and *CIFAR-10* (standard train/test splits, 10% of train held out for validation). Every arm trains the *same* architecture and parameter count and differs *only* in how the field is supervised (Table 3). Four baselines—*NODE* (Chen et al., 2018), *RNODE* (Finlay et al., 2020), *FW* (Anonymous, 2025), and *STEER* (Ghosh et al., 2020)—train the full stack end-to-end with cross-entropy and thus *backpropagate through the solver*, differing only in how they regularize or discretize the trajectory. *SFNO* (Kim et al., 2024) instead uses its own *simulation-free* recipe, regressing the field onto the straight-line velocity $z_1 - z_0$ with a noised label auto-encoder and no solver. *PM (ours)* is two-phase distillation: we pretrain $\mathrm{enc} + \mathrm{mid} + \mathrm{head}$ end-to-end, freeze it, and train the field to regress the frozen block's per-sample output $z_1 = \mathrm{mid}(z_0)$ via the bridge loss on *detached* endpoints. Like SFNO it uses no solver backprop, but it supervises a *per-sample* in-space target rather than a shared latent anchor.

Table 3: The six arms compared in Table 4, by how each supervises the field. *BP-solver* = back-propagates cross-entropy through the ODE solver.

Arm	BP-solver?	Mechanism
NODE	yes	direct trajectory optimization via `dopri5`
RNODE	yes	NODE + kinetic-energy / Jacobian-Frobenius regularizers
STEER	yes	NODE + jittered integration end-time $T \sim U(1 \pm 0.2)$
FW	yes	NODE + randomized 3-node midpoint grid
SFNO	no	simulation-free regression onto a shared latent anchor
PM (ours)	no	on-policy bridge regression onto a per-sample target (Eq. 3)

Evaluation. We report test accuracy at best validation (seed 0) as the quality metric, and two efficiency metrics: *train-NFE*, the number of field evaluations per training step, and *ms/iter*, idle-GPU wall-clock per training step. To attribute accuracy between the encoder and the node we additionally report a *load-bearing probe* (Table 5): on the deployed model we set the node to identity and read the frozen head off the pre-node feature z_0 (head@z_0); the gap to deployed accuracy is what the node itself contributes. A complementary minimal-backbone probe (fresh linear head on z_0 plus a node-free shallow baseline) is reported in Appendix E.

Implementation details. We keep the non-NODE stack *minimal*—in particular a *linear* head—so that any accuracy above a linear probe on z_0 is attributable to the *node*, not the head. MNIST uses a minimal MLP stack (L=128, w=256, d_{node}=3, 100k steps); SVHN and CIFAR-10 share an exact-ResNet-20 conv template (L=64, w=256, d_{node}=4, 40k steps) with the standard crop augmentation (SVHN without horizontal flip). PM's frozen teacher is trained with SGD (0.1, momentum 0.9, weight decay 5×10^{-4}, Nesterov, cosine); the field is trained with AdamW. PM/FW use a 3-node midpoint solver; NODE/STEER use `dopri5` via the adjoint (constant memory). Full architectures, splits, and optimizer settings are in Appendices D–F.

3.2 MAIN RESULT: IMAGE CLASSIFICATION

Table 4 benchmarks PM against the NODE-training baselines across all three datasets.

Two regimes emerge. On MNIST and SVHN, where every trainable-node arm reaches the same accuracy, the win is purely *efficiency*: PM matches that accuracy while paying only a handful of field evaluations per step and no solver backprop, an order of magnitude cheaper in wall-clock than the adaptive end-to-end arms (NODE, RNODE, STEER) and several times cheaper than even FW's fixed-grid solver. On CIFAR-10 the efficiency win is joined by an *accuracy* win: because PM distils

Table 4: Image-classification benchmark. *train-NFE* = field evaluations per training step; *ms/iter* = wall-clock per step (adaptive-solver arms at their converged NFE). Cells are mean±std over 3 seeds for the cheap arms (PM, FW, SFNO, all datasets) and NODE/STEER on CIFAR-10; the expensive adaptive-solver cells are seed 0.

Dataset	Metric	NODE	RNODE	STEER	FW	SFNO	PM (ours)
MNIST	Test acc	**99.11**	98.98	99.01	98.98 ± 0.04	97.73 ± 0.06	99.03 ± 0.04
	train-NFE	91	95	95	6	**1**	**7**
	ms/iter	221.6	184.2	234.8	14.6	**6.9**	7.5
SVHN	Test acc	**96.25**	95.39	96.19	96.08 ± 0.12	92.16 ± 1.78	95.88 ± 0.41
	train-NFE	77	237	86	6	**1**	**7**
	ms/iter	213.7	1255.0	246.0	41.2	25.5	**11.9**
CIFAR-10	Test acc	88.43 ± 0.17	86.42	88.48 ± 0.25	88.33 ± 0.18	68.99 ± 6.21	$\mathbf{91.36 \pm 0.35}$
	train-NFE	60	154	74	6	**1**	**7**
	ms/iter	219.8	789.8	231.5	42.2	21.8	**10.9**

Table 5: Node load-bearing probe (deployed CIFAR-10, seed 0). *gap* = deployed − head@z_0 is the node's own contribution (**bold** = largest); RNODE alone is non-load-bearing, its regularizers pre-straightening z_0.

Arm	acc	head@z_0	gap
NODE	88.22	60.99	$+27.23$
RNODE	86.42	86.43	-0.01
STEER	88.22	56.90	$+31.32$
FW	88.16	59.78	$+28.38$
SFNO	75.68	30.03	$+45.65$
PM	**91.41**	37.51	$\mathbf{+53.90}$

Table 6: PhysioNet 2012 mortality (AUROC, mean over 3 splits). *late-std* = std of the last-six test AUROCs (stability, lower is steadier); *ms/iter* as in Table 4. PM (**bold**) matches NODE's AUROC at ≈29× lower per-step cost and is steadiest.

Arm	AUROC	late-std	ms/it
NODE	0.833	0.0016	160.0
RNODE	0.834	0.0010	90.9
STEER	0.829	**0.0001**	134.9
FW	0.825	0.0004	16.8
SFNO	0.817	0.0641	6.3
PM	**0.839**	0.0002	**5.5**

a strong SGD-trained teacher rather than backpropagating cross-entropy through the solver from scratch, it clears the from-scratch arms by ≈3 points—a margin that holds across seeds. The only comparably cheap competitor, SFNO's simulation-free recipe, is the weakest arm throughout and becomes both inaccurate and high-variance on the shared CIFAR backbone, a fragility we trace to its mechanism in Section 3.3.

Does the node actually do the transport? A cheap NODE trainer is only interesting if the node it trains is doing real work, rather than passing an already-separable feature through unchanged. We check this with a *load-bearing probe*: set the node to identity and read the frozen head directly off the pre-node feature z_0; the gap to deployed accuracy is what the node contributes (Table 5). Every trainable arm's node is load-bearing here, and PM's most of all, because we deliberately place the node where the work is—after an encoder (a ResNet-20 with its last two blocks removed) that leaves z_0 not yet linearly classifiable (Appendix F). The size of the gap is therefore a property of *where the node sits*, not a merit of PM—and indeed RNODE is the lone exception, its transport regularizers straightening z_0 until the head can read it off directly, the same over-regularization that makes it the most expensive arm. The point of the probe is not the magnitude but the qualitative fact it certifies: PM's efficiency is not bought by side-lining the node: PM trains a genuinely load-bearing node, and does so more cheaply than any solver-BP arm.

Why PM is faster, and the end-to-end picture. The per-step gap is not an implementation artifact but a *structural* one. Because PM detaches its rollout, it pays a single regression forward-and-backward on the harvested states, whereas the solver-BP arms backpropagate through every one of their function evaluations; a short cost model (Appendix H) shows this advantage survives in *both* hardware regimes—the latency-bound small field of our backbone and the compute-bound native field of SFNO—rather than being an artifact of one. The saving also compounds *end-to-end*: even after charging PM the one-time cost of pretraining its teacher, its total wall-clock to reach the reported CIFAR accuracy is several-fold below FW and an order of magnitude below the adaptive-solver arms (Appendix H, Fig. 9).

Beyond classification: a time-series check. The applications that motivate NODEs are often temporal, so we verify the efficiency story on a real, irregularly-sampled clinical benchmark—in-hospital mortality on PhysioNet 2012 (Silva et al., 2012; irregular 48h ICU series, class-imbalanced, scored by AUROC over three splits), the canonical Latent-ODE setting (protocol in Appendix I). Here a static baseline cannot shortcut the task, so we compare PM directly against end-to-end NODE training (Table 6): PM matches its predictive AUROC while training roughly an order of magnitude faster per step, and is far steadier late in training than the simulation-free SFNO recipe—the same instability it shows on the small-latent sweeps below. The mechanism PM is built for thus carries over from images to genuine time-series dynamics.

3.3 Architecture sensitivity and stability

Why does the simulation-free recipe fail on some architectures and not others? We vary the three architecture knobs—latent width L, field width w, and node depth d_{node}—one at a time about a shared MNIST center, averaging each point over 3 seeds (Fig. 2; full sweep, including a CIFAR run, in Appendix G). Two patterns hold across all three axes. **PM and FW are smooth and seed-stable**, tracking the teacher ceiling as capacity grows and degrading gracefully as it shrinks. **SFNO is instead both lower and seed-erratic once capacity is small**—its accuracy collapses and its variance explodes as L, w, or d_{node} is reduced, and it fails to train at all below d_{node}=3. PM's only weak point is at the smallest latent (L=8), and it is a *distillation-fidelity* effect (too thin a field to fit the frozen `mid`), not a crossing failure—the ceiling returns by L=32. The depth panel also marks the honest floor: with no node (d_{node}=0) all methods coincide, and a single layer is *worse* than none, so the node earns its keep only once deep enough to curve.

This is the crossing conflict surfacing at the architecture level. SFNO merges every sample of a class onto *one shared anchor* and fits the field simulation-free, so it cannot curve around a conflict; when capacity is small there is no room to arrange the straight $z_0 \to z_1$ segments without crossing, and the field averages conflicting velocities—the low accuracy and the seed instability being two faces of the same failure. FW curves through the solver and PM regresses a per-sample target with no forced merge, so neither hits this wall until capacity is tiny.

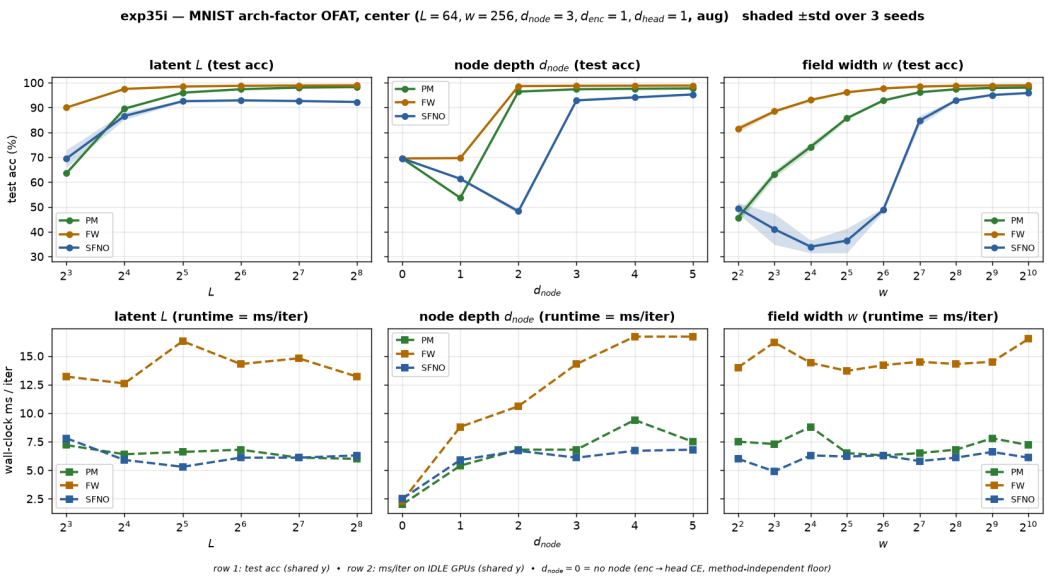


Figure 2: MNIST one-factor-at-a-time architecture sweep about the center L=64, w=256, d_{node}=3 (shaded ±std over 3 seeds). *Top:* test accuracy vs. latent L, node depth d_{node}, and field width w; *bottom:* wall-clock ms/iter for the same sweeps (idle-GPU timing). PM (blue) and FW (green) stay smooth and seed-stable across every axis; SFNO (red) turns both lower and *seed-erratic* at small capacity and fails to train below d_{node}=3. d_{node}=0 is the no-node floor.

3.4 CONTROLLED PAIRED-TRANSPORT TASKS

Finally, we isolate the crossing conflict PM is designed for on controlled paired-transport tasks (D=3, N=20, ≈10k-parameter field, midpoint-64 solver), where PM matches or beats every baseline on all four tasks (Table 7) while being the *cheapest* to train. Its largest margin is on the split task (G_2_MoG)—the denoising regime the NODE is designed for—won at the lowest per-step cost, because the bridge target is regressed at the rollout state with an $O(1)$-memory loss and no long-horizon solver backprop. STEER edges ahead only on the two low-conflict tasks (identity, swap); on the split and merge tasks its end-time jitter diverges or fails to finish, exactly the one-to-many / many-to-one regimes where a single straight field must fan out or collapse—so its wins are confined to cases with no routing conflict.

Table 7: Inference MSE on 4 controlled tasks (± std of the training-loss curve over the last 25% of iters). Lower is better; **bold** = best.

Task	**FW**	**SFNO**	**PM (ours)**	**STEER**
G_2_G (identity)	0.0123 ± 0.0011	0.0281 ± 0.0069	0.0031 ± 0.0030	**0.0001** ± 0.0001
MoG_2_MoG (swap)	0.0030 ± 0.0010	0.0008 ± 0.0006	0.0009 ± 0.0010	**0.0001** ± 0.0006
G_2_MoG (split)	0.0032 ± 0.0009	0.0135 ± 0.0071	**0.0005** ± 0.0016	0.0867 ± 0.0000
MoG_2_G (merge)	0.0020 ± 0.0001	0.0021 ± 0.0010	**0.0004** ± 0.0017	—
runtime (ms/iter)	15.1	7.1	**5.6**	194.5

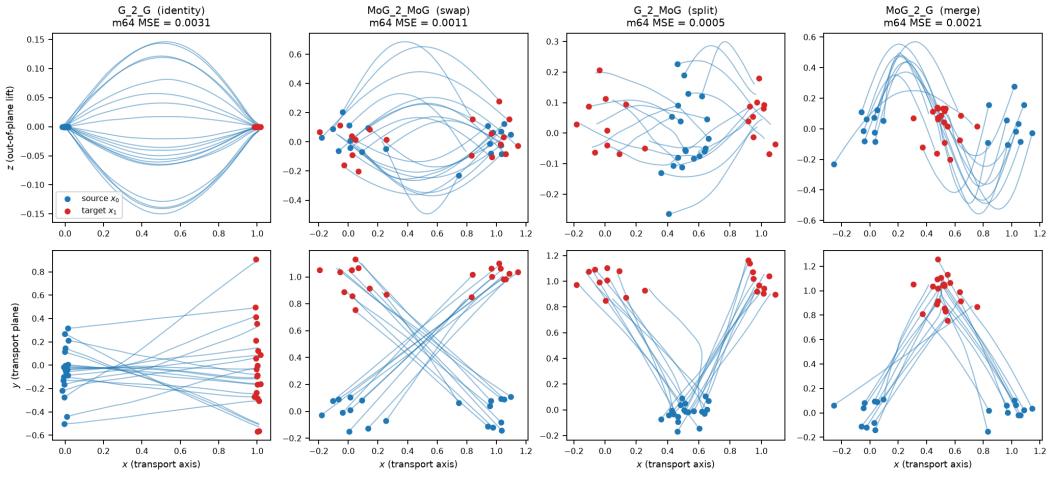


Figure 3: Learnt trajectories (blue) from source x_0 to target x_1 (red). **Top:** the x–z plane, where the z *lift* is automatically discovered; **bottom:** the x–y transport plane, showing the distribution of paired data.

Supplementary: geometry-constructed target paths and the path-stiffness measure. All results above use the default straight-line $w(t)$ (Section 2). Appendix B additionally constructs $w(t)$ from the data geometry and introduces a *path-stiffness* metric, σ (a velocity-disagreement score), that scores a candidate path's local stiffness *before training*: informally, it checks whether pairs of paths that pass close together in space also agree in velocity, and is large exactly where a field regressed on that path would be forced to average conflicting velocities. On the controlled tasks above, geometry-constructed paths lower σ relative to the straight-line default and give a modest further improvement on the harder split/merge tasks (Appendix B); we use σ only to *evaluate* candidate paths after the fact, not as a path-search objective during training (Section 5 explains why).

4 Related Work

We position on-policy Path Matching against four families it borders. Two attack the NODE cost by routes *orthogonal* to path crossing (efficient training; searching a better *pairing*); two share our vocabulary of "on-policy" or "self-distillation" but in a different sense (RL rewards; teacher-labelled imitation). In every case the gap PM fills is the same: supervising a native-space field on-policy, at the intermediate path, without a solver in the loop.

Efficient NODE training (the cost angle). Backpropagating through an adaptive solver's hundreds–thousands of function evaluations is the central NODE training cost (Chen et al., 2018; Grathwohl et al., 2019), and existing remedies attack it from angles orthogonal to path crossing. *Trajectory / temporal regularizers* shape the field for fewer solver steps—FW (Anonymous, 2025), a concurrent anonymous submission, randomizes the solver's intermediate times, RNODE (Finlay et al., 2020) adds optimal-transport/kinetic penalties, STEER (Ghosh et al., 2020) jitters the end-time—which we include as baselines in their faithful adaptive-solver (dopri5) form, the regime their regularizers are built for. *Multiple-shooting* (Turan & Jäschke, 2021) shortens the backprop horizon by co-optimizing segment states and field *jointly* in one constrained objective, whereas PM regresses the field on-policy against a fixed target path with no solver in the loop. *SFNO* (Kim et al., 2024)—the closest work—avoids long-horizon backprop by flow-matching *in a learned latent space*, absorbing crossings but sacrificing original-space interpretability (needed for time-series/physical applications) and, as our latent-dimension ablation shows (Section 3.3), breaking when the latent is too small for a non-crossing straight coupling. None supervises a native-space field on-policy or targets path crossing directly—the gap PM addresses.

Searching a better flow during training (re-pairing). A distinct family also *searches* the transport during training, but by changing the *pairing* of endpoints rather than the intermediate path. Rectified Flow / reflow (Liu et al., 2023) re-couples (x_0, x_1) from the current model's own generations and re-trains, iterating toward straight, few-step paths; minibatch-OT flow matching (Tong et al., 2024) picks the coupling by optimal transport within each batch. These are the works closest to "flow search," but they assume *free* endpoints and optimize the map. Our setting is *paired*—endpoints are fixed by the task—so re-pairing is unavailable; we instead search only the intermediate path *shape* given fixed endpoints, read from the on-policy rollout.

On-policy RL for flow models (a different sense of "on-policy"). A separate line post-trains a flow / CFM model with reinforcement learning: conditional FM as a robot policy class (π_0, Black et al., 2025) and Flow Policy Optimization (McAllister et al., 2025), which casts the FM loss as an advantage-weighted PPO objective. There "on-policy" is in the *RL* sense—the generated endpoint x_1 is rolled out and scored by a *reward*. We are on-policy in the *supervision* sense: the rollout state x_t supervises the field directly, with no reward and no policy gradient.

Learning from a model's own generations (bootstrap / self-distillation). On-policy imitation and self-distillation learn on states the current model visits—DAgger (Ross et al., 2011) for control, GKD (Agarwal et al., 2024) and Self-Distilled Reasoner (Zhao et al., 2026) for LLMs—but all depend on an *external teacher* to label those states. The drifting model (Deng et al., 2026) is closest in spirit: it *analytically* computes better targets and *adaptively selects* the learnable one, a curriculum-style self-improvement. PM as presented here uses a *fixed*, data-geometry target path; *re-deriving* that path from the field's own rollout—a self-distilled bootstrap that would place PM in this family—is a natural extension we outline as future work (Section 5), not part of the method we evaluate.

5 Discussion and Limitations

What we claim, and what we do not. PM's contribution is NODE training *efficiency* and interpretable in-space transport, *not* peak accuracy. We do not claim to beat the teacher, nor an MLP of matched capacity: no NODE-training method here surpasses the strong discriminative teacher it learns from (PM's CIFAR 91.4% sits below the 92.3% teacher). Training a NODE that *approaches* that teacher at a fraction of the solver-backprop cost is itself the contribution, and the $+3$ over the

from-scratch solver-BP arms on CIFAR is a *distillation by-product*, not a claim that our objective is intrinsically more accurate. For the same reason we report every baseline in its standard, strongest *end-to-end* form—the regime NODE, FW, STEER, and RNODE are designed for—rather than recasting them as distillers. A *jointly*-trained PM variant (encoder, field, and head optimized together) is also viable, but we do not develop it here, so as not to blur the two-phase recipe that carries our claims.

Where PM helps, and where it is inert. PM's advantage is specific to *conflict-heavy* transport, and it does no harm where that conflict is absent. On UCI regression (Appendix C) there is no designed crossing to resolve, so all methods tie on accuracy—the expected null; the informative axis there is *stability*, where the jointly-trained arms (SFNO, STEER) oscillate catastrophically late in training (up to $\approx 380\times$ their best RMSE across splits) while PM stays bounded—the endpoint-drift pathology that motivates PM's fixed-endpoint classification recipe. The contrast with the closest baselines sharpens the picture. SFNO reaches its endpoints through a *learned latent detour* whose simulation-free recipe is tied to its native architecture, so it weakens on our shared backbone and collapses when the latent is too small (Section 3.3); PM's recipe instead transfers unchanged across the MLP and conv stacks. FW is a strong, fast baseline, but buys its accuracy with backprop through the solver—which PM matches without.

The teacher-path dead end, and a self-distilled future. A natural extension is to make stiffness (σ) minimization the *mechanism*—searching an explicit non-crossing path per pair—rather than a diagnostic. We tried this and abandoned it for three reasons we believe generalize past our parametrizations (Appendix B): in low dimension the obstruction is *topological* (an ODE flow is injective, so an order-exchanging pairing has no crossing-free path); in high dimension the proximity signal *attenuates* with concentration of measure (consistent with the latent-width ablation, Section 3.3); and it does not *scale* (σ couples every pair, so there is no compact per-sample optimum). We therefore keep σ as a controlled-scale diagnostic (Section 3.4). A more promising route is to *re-derive* the target path on-policy from the field's own rollout, adopting a new per-particle target only when it lowers σ (teacher-free and monotone); at controlled scale this height bootstrap improves the harder transport tasks by 3–5$\times$ (Appendix B), but only along a fixed escape direction, and making it first-class and scalable is left to future work.

Scope of the evidence. The controlled-task spreads in Table 7 are within-run training-curve stds, and the UCI accuracy numbers are a single split (its stability claim is the one verified across three splits above); broader seed and split coverage strengthens rather than changes the efficiency conclusion, which is consistent across MNIST and CIFAR-10.

6 Conclusion

We introduced Path Matching (PM): an on-policy form of flow matching that supervises the field at the states the model actually visits and uses a contractive bridge target to pull a drifted rollout back onto a target path (a straight line between endpoints by default). PM trains NODEs without backpropagating through the solver, is the cheapest per iteration among the baselines we compare, and is the only method whose NODE genuinely carries the transport on real classification tasks. Our ablations trace SFNO's simulation-free failure to a concrete crossing mechanism that PM's curvable objective avoids.

AI use statement

In this work, we used generative AI tools as a coding and writing assistant: for drafting and refactoring experiment code, for producing figures and tables from result files, and for language editing of this manuscript. All AI-assisted code was executed, verified, and its numerical outputs checked by the authors; all reported numbers come from experiments we ran and inspected. We did not use generative AI to originate the research ideas, and we take responsibility for the final content of this work, including all text, claims, and artifacts.

Reproducibility statement

The method is fully specified in Section 2 (bridge loss, Eq. 3) and Appendices A–B (the $\kappa(t)$ family and the path-stiffness (σ) teacher construction, including every tunable axis). Experimental protocols—architectures, splits, optimizer settings, and solver configurations—are given per experiment in Appendices C–I (UCI regression, MNIST, CIFAR-10, the latent-dimension ablation, the runtime scaling law, and the PhysioNet time-series benchmark), including a faithful reproduction of the SFNO baseline in its native architecture.

References

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations (ICLR)*, 2024.

Anonymous. Latent flow transformer. 2025. Under review; author names withheld to preserve double-blind anonymity.

Kevin Black, Noah Brown, Danny Driess, Sergey Levine, et al. π_0: A vision-language-action flow model for general robot control. In *Robotics: Science and Systems (RSS)*, 2025.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Mingjian Deng, Hao Li, Tianhong Li, Yilun Du, and Kaiming He. Generative modeling via drifting. *arXiv preprint arXiv:2602.04770*, 2026.

Chris Finlay, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M. Oberman. How to train your neural ODE: the world of jacobian and kinetic regularization. In *International Conference on Machine Learning (ICML)*, 2020.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning (ICML)*, 2016.

Arnab Ghosh, Harkirat S. Behl, Emilien Dupont, Philip H. S. Torr, and Vinay Namboodiri. STEER: Simple temporal regularization for neural ODEs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. FFJORD: Free-form continuous dynamics for scalable reversible generative models. In *International Conference on Learning Representations (ICLR)*, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Semin Kim, Jaehoon Yoo, Jinwoo Kim, Yeongmin Cha, Saehoon Kim, and Seunghoon Hong. Simulation-free training of neural ODEs on paired data. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

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

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 (ICLR)*, 2023.

David McAllister, Songwei Ge, Brent Yi, Chung Min Kim, Ethan Weber, Hongsuk Choi, Haiwen Feng, and Angjoo Kanazawa. Flow matching policy gradients. *arXiv preprint arXiv:2507.21053*, 2025.

Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

Yulia Rubanova, Ricky T. Q. Chen, and David Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Ikaro Silva, George Moody, Daniel J. Scott, Leo A. Celi, and Roger G. Mark. Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. In *Computing in Cardiology (CinC)*, 2012.

Alexander Tong, Kilian Fatras, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *Transactions on Machine Learning Research (TMLR)*, 2024.

Evren Mert Turan and Johannes Jäschke. Multiple shooting for training neural differential equations on time series. *IEEE Control Systems Letters*, 2021.

Shen Zhao, Zhen Xie, Ming Liu, Jie Huang, Guansong Pang, Feng Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. In *International Conference on Machine Learning (ICML)*, 2026.

A THE $\kappa(t)$ TARGET FAMILY

The bridge loss of Section 2 is one member of a one-parameter family of targets that all share the form

$$\text{target}(x_t, t) = \underbrace{\dot{w}(t)}_{\text{velocity alignment}} + \underbrace{\kappa(t)\,(w(t) - x_t)}_{\text{trajectory alignment}}, \tag{4}$$

differing only in the *gain schedule* $\kappa(t) \geq 0$ (x_t = detached rollout state). The bridge we use sets $\kappa(t) = 1/(1-t)$. This appendix records (i) *why the trajectory-alignment term stabilizes*, (ii) a *constant-gain* member of the family, and (iii) the endpoint-aligned $V^+ + V^-$ target it replaces and the blow-up that aligning to $w(t)$ rather than x_1 avoids.

Why the trajectory-alignment term stabilizes. Let $e(t) = x_t - w(t)$ be the rollout's drift off the target path. At convergence the field reproduces its target, $\dot{x}_t = \dot{w}(t) + \kappa(t)(w(t) - x_t)$, so

$$\dot{e} = \dot{x}_t - \dot{w} = -\kappa(t)\, e. \tag{5}$$

Any drift *decays* at rate $\kappa(t)$—the closed loop is contractive, so a perturbed anchor is pulled back onto the path. With $\kappa \equiv 0$ only velocity alignment remains; the target is x_t-independent, the error obeys $\dot{e} = (\partial v_\theta / \partial x)\, e$ with an *unconstrained* Jacobian, and drift is at best *neutrally* stable—it accumulates and random-walks.

gain $\kappa(t)$	**drift rate** $\dot{e}/e$	**behavior**
$1/(1-t)$ (bridge, *used*)	$-1/(1-t)$	contractive, endpoint-exact
0 (velocity-only)	0	*neutral* → drift accumulates
const κ	$-\kappa$	contractive, bounded, no endpoint pull

Table 8: The gain schedule and the resulting drift dynamics.

The $1/(1-t)$ growth of the bridge gain is *benign here* because it multiplies the drift toward the *path point* $w(t)$, which is small and well-directed for a non-crossing w—not the drift toward the endpoint x_1, which is where the blow-up below comes from.

A constant-gain member. Setting $\kappa(t) \equiv \kappa$ constant removes the $1/(1-t)$ factor entirely and matches the path derivative $\dot{w}(t)$ up to a bounded trajectory-alignment term. It has two appealing properties—a *stable, bounded target* for all t, and *no V^- to design* (separation is baked into the geometry of a non-crossing w, whose derivative is single-valued by construction). The trade-off is that a constant gain gives *no endpoint pull* as $t \to 1$: exact-endpoint behavior then relies entirely on velocity alignment integrating cleanly. The bridge $\kappa(t) = 1/(1-t)$ is the default we use throughout; the constant-gain member is a stable alternative when endpoint-exactness is not required.

Empirical robustness (CIFAR-10). The drift analysis says the trajectory-alignment term *stabilizes*, not that its exact shape is delicate: for a non-crossing w the correction $\kappa(t)(w(t) - x_t)$ is small and well-directed, so PM should be largely insensitive to how the gain is scheduled or capped. Table 9 confirms this on CIFAR-10 at the benchmark configuration (L=64, w=256, d_{node}=4, 40k steps, seed 0; the row *clipped* $\kappa_{\max}$=20*, all* 3 *nodes* is the Table 4 PM cell). Every schedule—the clipped bridge at caps $8/20/64$, a constant gain, a linear ramp $\kappa_{\max} t$, and even the pure velocity-only target ($\kappa \equiv 0$, which the drift table calls neutrally stable)—lands within 0.5% of the default. Endpoint-exactness of the gain is thus a convenience, not a requirement, at this step budget: the non-crossing geometry of w, not the precise late-t pull, is what makes the target learnable.

Node-collection ablation. PM harvests the on-policy rollout at its M=3 midpoint nodes and regresses the field at each. We can instead keep only a random K of the 3 harvested nodes in the gradient step each iteration (the rollout still visits all 3 sequentially; only the number of supervised (x_t, t) points is reduced, re-drawn each step so every node is still supervised over training). Table 9 shows this costs almost nothing on accuracy—K=2 holds 91.15% and K=1 holds 91.00% vs. 91.50% for all three—while cutting the per-step cost (16.2 ms/iter at K=1 vs. 19.4 at K=3), a further efficiency knob orthogonal to the solver-free rollout.

Table 9: PM ablations on CIFAR-10 (benchmark config L=64, w=256, d_{node}=4, 40k steps, seed 0). *Top:* gain schedule $\kappa(t)$ and cap $\kappa_{\max}$. *Bottom:* number K of the M=3 harvested rollout nodes kept in the gradient regression. The bolded row (clipped $\kappa_{\max}$=20, K=3) is the Table 4 PM cell. Test accuracy at best validation; *LB-gap* = deployed − linear-probe(z_0).

Ablation	Variant	Test acc	LB-gap	ms/iter
gain $\kappa(t)$	**clipped, $\kappa_{\max}$=20 (default)**	**91.50**	+54.0	19.4
	clipped, $\kappa_{\max}$=8	91.18	+53.7	19.4
	clipped, $\kappa_{\max}$=64	91.06	+53.6	19.5
	const, κ=20	91.26	+53.8	19.2
	linear, $\kappa_{\max} t$	91.28	+53.8	19.4
	velocity-only, $\kappa \equiv 0$	91.23	+53.7	19.4
nodes kept K	**K=3 (all, default)**	**91.50**	+54.0	19.4
	K=2	91.15	+53.6	18.1
	K=1	91.00	+53.5	16.2

B THE TARGET PATH w: CONSTRUCTION AND BOOTSTRAP

Section 2 builds the target path w from data geometry. This appendix records exactly *how* the current implementation constructs w, defines the *path-stiffness* metric σ (Eq. 7) we use to *evaluate* a candidate path's learnability, and describes the optional on-policy height bootstrap (a self-distilled refinement we treat as future work, Section 5): the escape *direction* is fixed up front from data geometry and the bootstrap, if used, adjusts only the per-particle *lift height*. All quantities are computed from the paired data (x_0, x_1) and the field's own rollout—*never* from labels, cluster identities, or an assignment oracle.

The path family. Every particle i follows a *sin-arch* between its fixed endpoints,

$$w_i(t) = (1-t)\, x_{0,i} + t\, x_{1,i} + \sin(\pi t)\, e_i, \qquad \dot{w}_i(t) = (x_{1,i} - x_{0,i}) + \pi \cos(\pi t)\, e_i, \tag{6}$$

where $e_i \in \mathbb{R}^D$ is the *escape vector*—a lift orthogonal to the straight-line velocity $v_i = x_{1,i} - x_{0,i}$. Its norm $\|e_i\|$ is the *arch height* and $e_i/\|e_i\|$ the *escape direction*. Because $\sin(\pi t)$ vanishes at $t \in \{0, 1\}$, the endpoints stay exact for *any* e_i—the escape is unbiased by construction, and the whole design problem is choosing $\{e_i\}$.

The escape field: minimizing path stiffness σ (within-neighborhood velocity disagreement). The direction and an initial height come from one small optimization over $\{e_i\}$, run *once* at initialization (a few $\times 10^3$ Adam steps on the $N \times (D-1)$ coefficients that parameterize each e_i in the null space of v_i; $\sim$10s, not per training iteration). We parameterize $e_i = \sum_c a_i^c \, n_i^c$ with $\{n_i^c\}$ an orthonormal basis of $v_i^\perp$, and minimize the *conflict-weighted within-neighborhood velocity disagreement*

$$\sigma(\{w\}) = \frac{\sum_t \omega(t) \sum_{i<j} k_{ij}(t) \, \|\dot{w}_i(t) - \dot{w}_j(t)\|}{\sum_t \omega(t) \sum_{i<j} k_{ij}(t)}, \tag{7}$$

a proximity-weighted average of pairwise velocity mismatch, with a time weight $\omega(t) = 1 - 0.75t$ (early trajectory matters more) and a *soft* neighborhood kernel $k_{ij}(t) = \exp(-\|w_i(t) - w_j(t)\|^2/r^2)$, where r is the median source nearest-neighbor gap. The stiffness σ measures a *local Lipschitz* property of the induced field: when two paths pass close in space (large k_{ij}) their velocities should agree, or the field is multivalued there and hard to learn. Two properties matter:

- **It subsumes non-crossing.** A crossing is the limit $\|\Delta w\| \to 0$ with $\|\Delta \dot{w}\| > 0$, i.e. infinite disagreement—σ drives it away without a separate crossing penalty.
- **Non-crossing is not sufficient; velocity coherence is.** Merely un-crossing the paths (minimizing $\sum 1/d^2$) leaves the field stiff where paths run close but disagree in velocity; σ targets that stiffness directly. This is the criterion that best separated learnable from unlearnable teachers on the crossing stress-tests (`exp23g`).

The ratio is taken *globally*—one weighted mean over all (t, ij) samples—rather than per-time. The two differ because the per-time neighbor mass $\sum_{ij} k_{ij}(t)$ is not flat: paths are far apart at take-off and landing but bunch up mid-flight, where $\sum k$ peaks ($\approx 5\times$ its minimum on our toys). The global ratio therefore weights disagreement *by where close encounters actually occur*, whereas per-time normalization would give an empty slice the same say as a crowded one. After the solve the field is rescaled so $\max_i \|e_i\| = h_{\text{data}}$, where $h_{\text{data}} = \text{mean}_i \|x_{0,i} - \bar{x}_0\|$ is the mean in-plane source radius—a pure data statistic.

Stiffness (σ) design choices (the tunable axes). σ is not a single fixed formula; five choices define it, and they—not the solver settings—are the actual design surface.

1. **Time weight** $\omega(t)$ (used: $1 - 0.75t$). Selects *which part of the path* σ scores. Up-weighting $t \approx 0$ forces source-neighbors to agree in initial velocity; we tried this ($\tau = 0.15$) and it over-constrains splits, so smoothness is better controlled *spatially* (via r) than *temporally*.
2. **Neighborhood radius** r (used: median source nearest-neighbor gap). The non-crossing↔smoothness dial. The useful r appears to be set by network capacity, not data geometry alone: r fixes the spatial frequency of the escape field, and a field varying faster than the network can represent is unlearnable regardless of how coherent σ judges it.
3. **Kernel profile** k_{ij} (used: Gaussian). Sets *how sharply* "neighbor" cuts off, independently of the scale r.
4. **Disagreement norm** (used: $\|\Delta \dot{w}\|$, an L_2 velocity gap). L_2 vs. $\|\Delta \dot{w}\|^2$ changes how hard the single worst pair is weighted; the scale-free ratio $\|\Delta \dot{w}\|/\|\Delta w\|$ is the stiffness / local-Lipschitz variant (`exp23g`).
5. **Static vs. dynamic neighbors** (used: dynamic). Freezing k_{ij} to the source geometry would decouple "who is a neighbor" from "how high the arch is."

A sixth, discussed above, is *global vs. per-time normalization*; we use global.

The on-policy height bootstrap. The escape *direction* $\hat{e}_i = e_i/\|e_i\|$ is then *frozen*. Every `refresh_iter` steps the bootstrap proposes a new *scalar height per particle* read from the field's own rollout: it integrates the current field to the midpoint x_i^{mid}, forms the lift $\ell_i = (x_i^{\text{mid}} - \frac{1}{2}(x_{0,i} + x_{1,i}))$, and *projects it onto the frozen direction*, $a_i = \hat{e}_i^\top \ell_i$. The candidate path uses $e_i^{\text{cand}} = a_i\,\hat{e}_i$. *The gate (no oscillation):* a candidate is adopted only if it *lowers* stiffness: $\sigma(w^{\text{cand}}) < \sigma(w^{\text{best}})$; otherwise the previous best is kept. Training always supervises against w^{best}, so the target is monotone in σ and cannot thrash.

What the bootstrap can and cannot do. The bootstrap moves *height along a fixed direction only*; it never rotates the escape. Empirically the height bootstrap improves the harder tasks by 3–$5\times$ but *cannot rescue a stiff direction* the teacher bakes in—the teacher's velocity coherence sets the achievable floor and on-policy refinement is a bounded lever on top of it. *Known limitation (identity-like tasks):* the stiffness σ objective still prefers a small arch even when the optimal teacher is a straight line; on the near-identity `G_2_G` task $\sigma(e{=}0) \approx 0.45$ but the solve settles at mean $\|e_i\| \approx 0.06$, a slight curvature the field then wastes capacity fitting. Replacing the $\max \|e\| = h_{\text{data}}$ rescale with a small $\lambda\|e\|^2$ spring, so coherent geometries relax to $e \approx 0$, removes this residual curvature and changes only the escape solve—not the bootstrap or the loss.

C Regression performance (UCI)

Beyond the controlled toys, we test on real *UCI regression* tasks, following the protocol of SFNO (Kim et al., 2024): input x and scalar target y are lifted into a shared 64-d latent ($z_0 = \text{enc}(x)$, $z_1 = \text{labelproj}(y)$), a NODE transports $z_0 \to z_1$, and a linear head decodes the endpoint. Data follows the standard splits of Gal & Ghahramani (2016) (the official held-out test partition, with the train partition further split 6:4 into train/validation). We select the checkpoint by *validation* RMSE and report *test* RMSE at that checkpoint (label-unnormalized, dopri5 solver). All methods share the *same* backbone and parameter count. This is a *single-split proof-of-concept*; the $\pm$ shown is the spread of the last-10 test evaluations (a within-run stability proxy), *not* a cross-split std.

Table 10: UCI test RMSE at the best-validation checkpoint (single split, 40k iters, dopri5). Lower is better; **bold** = best per task. On these low-conflict regression tasks *all methods are within noise of each other*—there is no designed path crossing to resolve, so PM's on-policy machinery neither helps nor hurts. This confirms PM does no harm on easy problems.

Task	SFNO	FW	STEER	RNODE	PM	PM-frozen
`boston`	2.45	2.55	2.45	2.39	2.53	**2.33**
`concrete`	5.33	5.19	**4.88**	5.34	5.33	5.01
`energy`	0.47	**0.39**	0.42	0.44	0.41	0.50
`yacht`	0.80	**0.67**	0.92	0.91	0.73	0.83
`wine`	0.65	0.65	**0.63**	0.63	0.64	0.66

Late-training oscillation, replicated across splits. The *jointly-trained* NODE objectives (SFNO, STEER) show catastrophic late-training RMSE oscillation on the smallest sets, whereas the checkpoint *selected by validation* stays stable. To check this is not a single-run artifact, we re-run the two spiking tasks (`wine`, `energy`) with all three joint arms over *three independent data splits* and measure, per run, the spread of the last six test evaluations (the within-run late-training oscillation, not a cross-split std). The instability replicates and is *arm*- and *task*-specific: on `wine`, SFNO spikes late on *all* $3/3$ splits (worst last-6 evaluation reaching $\approx 381\times$ its best-validation RMSE, mean last-6 std 53.3), STEER on $1/3$; on `energy`, STEER is the unstable one (worst $\approx 54\times$, mean std 3.6, $2/3$ splits). Crucially, PM's worst case is *orders of magnitude milder*—its last-6 evaluations never exceed $1.6\times$ its best RMSE on any split of either task (mean std 0.017 on `wine`, 0/3 catastrophic spikes)—so PM does not *eliminate* late oscillation so much as keep it bounded where the joint arms diverge. A companion study isolates the cause to *joint* training moving the latent endpoints while the NODE chases them: *freezing* the endpoints removes the oscillation but *hurts* generalization, so the endpoints must keep co-adapting. This motivates PM's classification recipe, which fixes the

endpoints *before* committing the field—pretraining and freezing the endpoint-defining block, then regressing the field on-policy against that frozen, per-sample target.

Reproduction check. Our SFNO re-implementation matches the published SFNO (Kim et al., 2024) numbers within their reported cross-split std on every task (e.g. boston 2.45 vs. paper 3.50 ± 0.80; concrete 5.33 vs. 5.67 ± 0.55). The released SFNO code contains no RNODE/STEER implementation, so those are our own re-implementations; the RNODE regularizers use $\lambda = 0.01$ (Finlay et al., 2020) default. Input dimensions: boston 13, wine 11, concrete/energy 8, yacht 6.

D MNIST AND SVHN CLASSIFICATION BENCHMARK

We benchmark PM against the NODE-training baselines on *MNIST classification*, following the SFNO (Kim et al., 2024) latent-flow setup but with a deliberately *minimal* non-NODE stack so that any accuracy above a linear probe on the encoder output is attributable to the *node*, not the head. SVHN uses the same protocol on the 32×32 RGB digit images but shares the CIFAR-10 conv template (Appendix F, the exact-ResNet-20 carve at $L{=}64$) rather than the MNIST MLP stack, with SVHN normalisation statistics and random-crop (no horizontal flip) augmentation.

Shared model (all arms identical). Every arm trains the *same* architecture and parameter count; they differ *only* in how the field is trained.

- **Encoder** $\mathrm{enc} : 784 \to 256 \to 64$—`Flatten` → `Linear(784,256)` → `BatchNorm1d` → `ReLU` → `Linear(256,64)`, then an *affine-free* `BatchNorm1d(64)` standardizing z_0.
- **Field (the NODE)**—an MLP velocity $v(z,t) : 65 \to 256 \to 256 \to 64$ (time t concatenated, ELU), depth 3, width 256. **99,136 trainable parameters**—the only trained module in the PM arm.
- **Head**—a single `Linear(64,10)` (the attribution property above).
- **Total** 317,706 params (baselines train all of it; PM trains only the 99,136-param field).

Figure 4(a) draws this MNIST model; SFNO's own native MNIST architecture (described below) instead uses a 784-d latent and a two-layer MLP head, and so does *not* isolate the node the way our linear-head design does.

Arms (the training-method axis).

- **PM (ours)**—two-phase *distillation*. Phase 1: pretrain $\mathrm{enc} + \mathrm{mid} + \mathrm{head}$ end-to-end with cross-entropy, where `mid` is a 2-hidden-layer MLP $64 \to 256 \to 256 \to 64$ (the block the node replaces). Phase 2: *freeze* $\mathrm{enc} + \mathrm{mid} + \mathrm{head}$ and train the field to regress the frozen `mid`'s per-sample output $z_1 = \mathrm{mid}(z_0)$ via the bounded PM loss on *detached* endpoints—*no backprop through the solver*, no stiffness search ($\kappa_{\max} = 20$, trust radius 0.3, midpoint-3 jittered rollout). The distillation target is a *smooth learned* per-sample map, hence fittable by the ODE (unlike a free per-sample z_1).
- **NODE / FW / STEER (baselines)**—the whole $\mathrm{enc} \to \mathrm{field} \to \mathrm{head}$ stack trained *end-to-end with cross-entropy through the solver*, differing only in the integration grid: NODE = DTO through adaptive dopri5; FW = randomized 3-node midpoint grid (Anonymous, 2025); STEER = jittered end-time $T \sim U(1 \pm 0.2)$, dopri5 (Ghosh et al., 2020).
- **SFNO (baseline)**—its own *joint, simulation-free* recipe: regress the field onto $z_1 - z_0$ on the straight interpolant $z_t = (1-t)z_0 + tz_1$ (no solver), plus a noised label-AE $\mathrm{CE}(\mathrm{head}(z_1 + \sigma n), y)$ with $z_1 = \mathrm{labelproj}(\mathrm{onehot}(y))$, $\sigma = 3$ (Kim et al., 2024).

Protocol. 50k/10k/10k train/val/test split (seed-0), 25k iterations, batch 256. Baselines/SFNO: AdamW $\mathrm{lr} = 10^{-3}$; PM field: AdamW $\mathrm{lr} = 2 \times 10^{-3}$, cosine schedule, 2k straight-line FM warm-up steps. Model selection by best *validation* accuracy; we report *test* accuracy at that checkpoint. *Inference* uses the adaptive dopri5 solver for every arm.

The MNIST rows of Table 4 report the result: all trainable-node arms tie on accuracy ($\approx 98.4\%$), and PM reaches this with train-NFE 7 and no solver backprop—a fraction of the per-step cost of

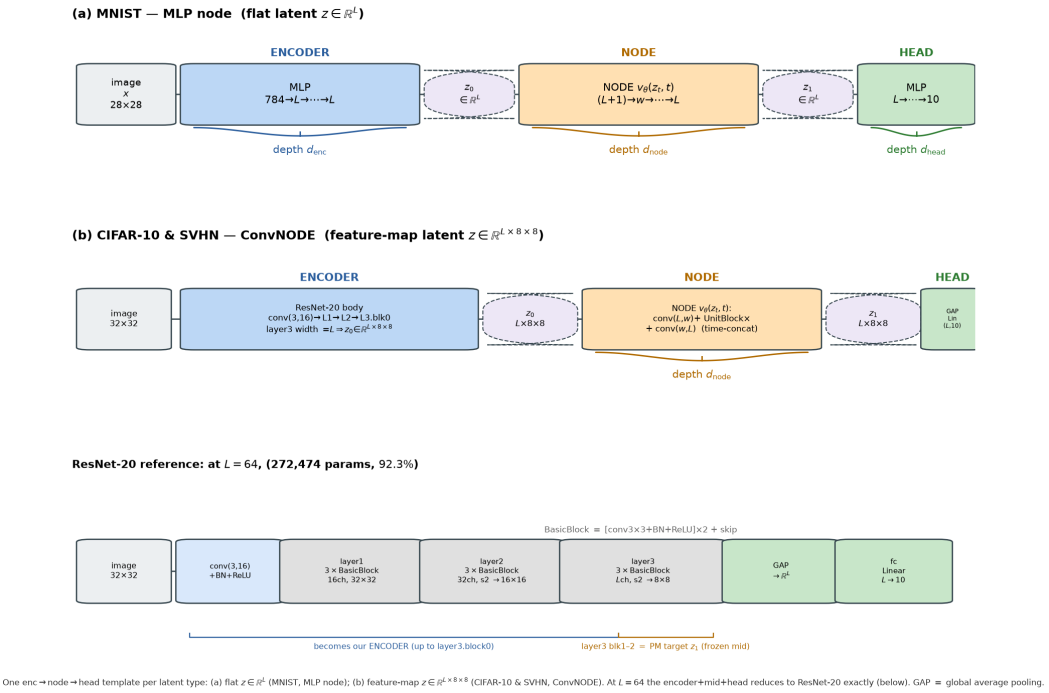


Figure 4: Our shared enc → node → head template, one instantiation per latent type. **(a)** *MNIST*—flat latent $z \in \mathbb{R}^L$: a linear/MLP encoder $784 \to L$, an `MLPODEFunc` node $v_\theta(z_t, t) : (L{+}1) \to w \to \cdots \to L$, and a *linear* head. **(b)** *CIFAR-10 & SVHN*—feature-map latent $z \in \mathbb{R}^{L\times 8\times 8}$: a ResNet-20 body (stem+`layer1`+`layer2`+`layer3.block0`, with `layer3` width set to L) as encoder, a `ConvODEfunc` node, and a GAP+`Linear`$(L, 10)$ head; a single template serves both datasets (they differ only in label-AE noise σ and augmentation). *Bottom:* the standard ResNet-20 (He et al., 2016) the (b) encoder is carved from—at $L{=}64$ the enc+mid+head is *bit-exact* ResNet-20 (272,474 params, 92.3%), the blue brace marking the span that becomes our encoder and the orange brace the two `layer3` blocks the node replaces (and that PM distills). Our *linear* heads mean any accuracy above a linear probe of z_0 is the node's doing. GAP = global average pooling.

the other arms. This is an *efficiency* win, not an accuracy win. Here the MNIST encoder is already close to linearly separable at z_0, so the node's net contribution over a node-free shallow baseline is modest (+6; the minimal-backbone workload probe of Appendix E quantifies this passenger regime and contrasts it with the load-bearing deployed CIFAR carve of Table 5).

SFNO's own MNIST architecture (for reference). Our SFNO *row above* runs SFNO's training recipe on *our* minimal shared model for an apples-to-apples comparison. For completeness, the architecture SFNO (Kim et al., 2024) uses in *its own* released MNIST config is the *same three-component template*, only larger, and drops our linear head. Writing L for the latent dim, w for the field's hidden width, and d for its depth:

- **Encoder** (`in_projection` = enc, the whole encoder, not just the flatten): `Flatten` → two `Linear`(L, L) +BN+ReLU blocks, $L = 784$.
- **Field** an `MLPODEFunc` $v_\theta(z, t)$—*structurally the same as our node*: a time-conditioned MLP $(L{+}1) \to w \to \cdots \to w \to L$ with t concatenated at every layer, $w = 2048$, $d = 3$ (ours is the identical functional form with $L{=}64$, $w{=}256$, $d{=}3$).
- **Head** (`out_projection`) a *2-layer MLP* `Linear`(L, L) →BN→ReLU→`Linear`$(L, 10)$ — i.e. $784 \to 784 \to 10$—not a single linear map.
- **Anchor branch** `label_projection` → z_1 a `Linear`$(10, L)$, noised label-AE $\sigma = 3$.

Training: linear dynamics, CE, AdamW lr 10^{-4}, batch 1024, 100k steps. Their reported MNIST accuracy is ≈98.3% (our own 784-latent reproduction matches: 98.33% test). The field is the *same*

MLPODEFunc as ours, only wider (w=2048 vs 256) on a bigger latent (L=784 vs 64). The one substantive difference is the head: because SFNO's head is a *2-layer MLP* it can itself perform nonlinear class separation, so full-system accuracy no longer *attributes cleanly to the node*—one cannot tell how much of the accuracy came from the node's transport versus the nonlinear head. Our *linear* `Linear(`$L, 10$`)` head cannot separate classes on its own, so any accuracy above a linear probe of z_0 is necessarily the node's contribution; that is the attribution property ("isolating the node") our minimal design is chosen for.

E THE NODE-AS-PASSENGER REGIME: A MINIMAL-BACKBONE PROBE

The main-text load-bearing probe (Table 5) places the node mid-depth, where z_0 is not yet linearly usable and the node carries real work. The opposite regime is equally informative and we report it here for honesty: when the node is placed *after* an already-strong encoder, it degrades to a near-identity passenger. We measure this on a deliberately *minimal shared backbone* (L=64, seed 0) in which the encoder is trained end-to-end and is already close to linearly separable at z_0. For each arm we train a *fresh* linear head on frozen z_0 ($P(z_0)$, a linear-separability probe) and compare against (a) the deployed model and (b) a *shallow* baseline—encoder+head trained end-to-end with *no* node, under the full model's optimizer and augmentation.

Table 11: Node-as-passenger regime: per-method workload probe on a minimal backbone (seed 0, L=64) where the encoder is already near-separable. $P(z_0)$ = fresh linear head on frozen z_0; *shallow* = encoder+head, no node; *node vs* z_0 = model$-P(z_0)$; *net vs shallow* = model$-$shallow. For PM/FW the encoder already does the work, so the node's net contribution is small; SFNO's node *degrades* its own z_0 (negative). Contrast with the deployed carve configuration (Table 5), where every node is load-bearing.

Dataset	Arm	test acc	$P(z_0)$	node vs z_0	net vs shallow
MNIST	shallow (enc+head)	92.29	–	–	–
	PM	98.01	91.18	+6.83	+5.72
	FW	98.37	91.53	+6.84	+6.08
	SFNO	77.51	89.69	**−12.18**	−14.78
CIFAR-10	shallow (enc+head)	89.57	–	–	–
	PM	90.21	90.14	+0.07	+0.64
	FW	89.92	89.91	+0.01	+0.35
	SFNO	31.82	46.02	**−14.20**	−57.75

Two readings follow. (i) For the end-to-end-CE arms (PM, FW) a trained encoder already makes z_0 largely linearly separable (≈91% MNIST, ≈90% CIFAR), so a node bolted on afterwards is a near-identity passenger—its net gain over a node-free shallow baseline is +5.7 on MNIST (where the *linear* head bottlenecks the encoder, so the added transport depth helps) and a negligible +0.6 on CIFAR. This is exactly why the deployed benchmark places the node mid-depth instead (Table 5). (ii) SFNO behaves qualitatively differently: its deployed accuracy is *below* a linear probe of its own z_0 (77.5 vs 89.7 MNIST; 31.8 vs 46.0 CIFAR)—its simulation-free straight-line flow toward shared class anchors *destroys* separability the encoder had already found, which is why we also run SFNO on its *native* architecture (Appendix F), where its arch-specific inductive bias makes the flow productive.

F CIFAR-10 ARCHITECTURE AND THE SFNO-RECIPE CAVEAT

We port the same benchmark to *CIFAR-10*. Because CIFAR features are not linearly separable at the cut point, the *distillation target must be per-sample and rich*, so PM distills a *frozen ResNet-20's own intermediate blocks* rather than a small MLP `mid`.

Our shared CIFAR model. Here the latent is a *feature map*, so L denotes its channel count and w the field's hidden channel width.

- **Encoder** $\to z_0$—a ResNet-20 stem + `layer1` (3 blocks, 16ch) + `layer2` (3 blocks, 32ch, stride 2) + `layer3.block0` (L=64ch, stride 2), giving an $L \times 8 \times 8$ feature, then affine-free `BatchNorm2d(`L`)`. This is a ResNet-20 *missing its last two layer-3 blocks*—a deliberately weakened classifier, so the block the node replaces carries real work *by construction*.
- **Field (ConvNODE)**—$v_\theta(z,t)$ = `conv3x3(`$L{+}1, w$`)` $\to$ `BN` $\to$ `ReLU` $\to$ `conv3x3(`w, L`)` $\to$ `BN` (time t appended as a channel), width w=256 and depth d_{node}=4 for the benchmark of Table 4. (This is the *conv* analogue of the MNIST MLPODEFunc: same time-conditioned residual velocity, convolutions instead of dense layers.)
- **Head**—`AdaptiveAvgPool2d(1)` $\to$ `Linear(`$L, 10$`)`, again a single linear map for clean attribution.
- **PM target** z_1 = the frozen ResNet-20's *real* `layer3` blocks 1–2 output (per-sample). Ceiling = the intact frozen net ($\approx 92\%$).

Baselines (NODE/FW/STEER) train this same $\text{enc} \to \text{ConvNODE} \to \text{head}$ stack end-to-end with CE through the solver (SGD lr 0.05, momentum 0.9, wd 5×10^{-4}, cosine, 60 epochs). Figure 4(b) draws this model; its bottom row shows the intact ResNet-20 our encoder and distillation target are carved from, and SFNO's native conv-latent architecture is described below.

The CIFAR-10 rows of Table 4 report the result. Here PM *leads on accuracy* at 91.4%, $+3.2$ over the from-scratch solver-BP arms (NODE/FW/STEER all $\approx 88.2\%$), and reaches it with train-NFE 7 and no solver backprop. The gain is a distillation effect: PM regresses the frozen SGD-trained teacher (ceiling 92.3%), whereas backpropagating cross-entropy through the solver from scratch plateaus lower at the same 40k-step budget. This is a stronger claim than the earlier minimal-backbone probe (Table 5), where an encoder+head-only shallow baseline nearly matched the full model and the node's *net* contribution was $\approx +0.6$; on the exact-ResNet-20 carve with an SGD teacher, distilling that teacher into the node is what buys the accuracy margin.

The SFNO-recipe caveat (a genuine finding). SFNO's joint simulation-free recipe, which *works on our MNIST model* (97.0%), is *markedly weaker on our CIFAR conv model* (75.7% vs PM's 91.4%; on SVHN, 91.0% vs 95.4%): under joint training the sim-free flow-matching target and the conv field are poorly matched—the field stiffens (inference NFE climbs, up to $300+$ in the earlier free-latent variant) and accuracy lags well below the trainable-node arms, under both SGD and AdamW and across label-noise settings. This is consistent with the hypothesis that *SFNO's method is coupled to its specific architecture*, which is why we also report its native-arch result ($\approx 89\%$) below.

SFNO's native CIFAR-10 architecture (Kim et al., 2024). For a faithful comparison we transcribe the released model exactly (`data_dim` $= 3$, `in_latent_dim` $= 64$, `latent_dim` $=$ `hidden_dim` $= 256$, `emb_res` $= (7,7)$, `f_add_blocks` $=$ `h_add_blocks` $= 4$, `num_classes` $= 10$, linear dynamics). The `in_projection` maps the image to a $256 \times 7 \times 7$ latent via a stem, 4 residual conv blocks, and two stride-2 convs; the `odeblock` is a `ConvODEfunc(256, hidden=256, add_blocks=4)` conditioning on time by appending a constant t-channel; `out_projection` is `ReLU` $\to$ `AdaptiveAvgPool2d(1)` $\to$ `Linear(256,10)`; `label_projection` is `Linear(10,256)` broadcast to $256 \times 7 \times 7$. The large label-AE noise $\sigma = 10$ grows a wide correctly-decoding basin around each prototype. AdamW lr 3×10^{-4}, batch 1024, 100k steps; augmentation `RandomCrop(32, pad 4)` + HFlip.

We reproduce this native model in a plain PyTorch loop (no Lightning/wandb) and recover $\approx 89\%$ test accuracy—matching the paper—confirming SFNO's method *does* work, but only *in its own architecture*. We therefore treat SFNO on our shared CIFAR backbone as *out-of-distribution for its own method*, while PM's distillation recipe is *architecture-robust* (it works unchanged on both the MNIST MLP stack and the CIFAR ResNet stack).

PM and FW on SFNO's home turf (exp35g). As a stress test we also run our two objectives on *this exact native architecture*, changing only the field-training loss. FW trains the full ConvModel end-to-end with cross-entropy through a fixed 3-step midpoint solver (no distillation); PM uses the two-phase recipe (a ConvODEfunc-shaped mid-block teacher, pure CE, then the odefunc distills the

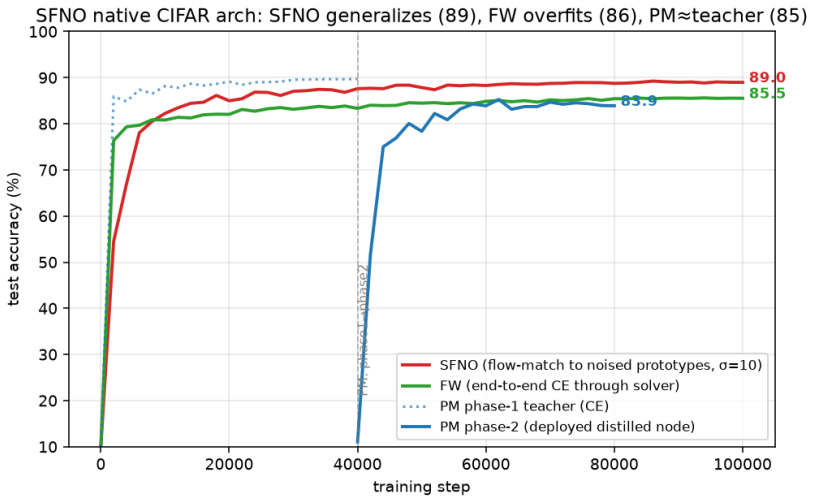


Figure 5: Test-accuracy training curves on SFNO's *native* CIFAR-10 ConvModel (identical 4.04M-parameter architecture and augmentation; only the field-training objective differs), illustrating the mechanism analysed in the text: FW (green) overfits and plateaus at 85.6%, SFNO's noise-regularised generative loss (red) generalises to 89.0%, and PM's distilled node (solid blue, from step 40k) tracks 83.9–85.3% below its pure-CE phase-1 teacher (dotted, 89.7%). Logs: `results/exp35g/cifar_{FW,PM}.log`, `results/exp35d/full_100k.log`; plot `experiments/35g_plot_curves.py`.

frozen per-sample $z_1{=}\mathrm{mid}(z_0)$ on detached endpoints, $D{=}12{,}544$). At 100k/80k steps FW reaches 85.6% and PM 85.3%—a tie with each other, both below SFNO's own 89.0% in its home architecture. The training curves (Fig. 5) reveal *why*, and it is a property of the *objective*, not the architecture (identical 4.04M-parameter model and augmentation for all three). FW trains the whole ConvModel as a discriminative classifier—cross-entropy backpropagated through the ODE rollout—and *overfits*: its train CE collapses to 0.0000 and validation to 100% by ≈40k, after which test accuracy is frozen at 85.6%. SFNO instead optimises a *generative* flow-matching loss onto class prototypes corrupted with large label-AE noise ($\sigma{=}10$); that noise carves a wide correctly-decoding basin around each prototype and acts as a strong margin regulariser, so SFNO never drives a discriminative loss to zero and generalises to 89.0%. Two honest readings follow. (i) SFNO's simulation-free straight-line-to-anchor loss is genuinely well-matched to this hand-designed convolutional flow and its heavy prototype-noise regularisation, and neither our solver-backprop (FW) nor our distillation (PM) objective improves on it there; PM in particular is bounded by its phase-1 teacher (89.7%, itself a pure-CE model that—unlike FW—trains *without* solver-backprop and generalises fine, so the deployed PM gap is distillation fidelity, not the teacher). (ii) The per-step speed advantage compresses on this architecture (1.27× vs. 3.1× on the shared backbone) because the native ConvODEfunc is a large *compute-bound* field; Appendix H derives and reconciles this regime shift. PM stays faster, but the decisive launch-limited margin is a property of the *small* shared-backbone field, which is why our headline benchmark (Table 4) uses it and this native-arch comparison is reported here for completeness.

F.1 Architecture-search protocol and the enc→field→head template

All of our classification experiments—the shared-backbone benchmark (Table 4), the latent-dimension sweep (Appendix G), and the native-arch study above—instantiate a *single three-component template*, and our architecture search moves along its three structural knobs. We spell out the template here because the same three roles recur under two different latent representations. Figure 6 draws both.

The three components and the search knobs. The three roles—encoder $\mathrm{enc} : x \mapsto z_0$, field (node) $v_\theta(z,t) : \mathbb{R}^L \to \mathbb{R}^L$ (the *only* module that changes between training methods), and *linear*

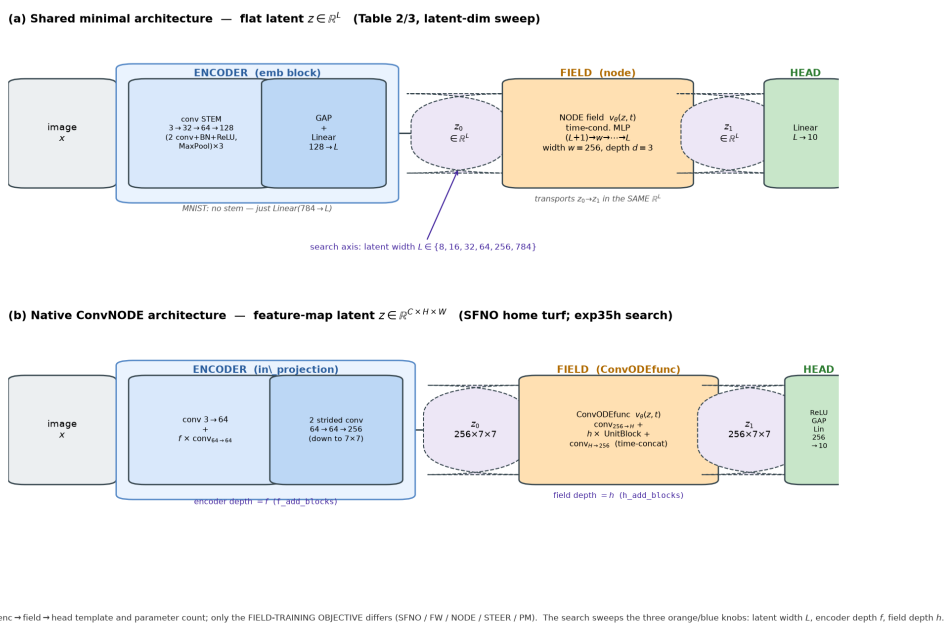


Figure 6: The shared enc→field→head template under two latent representations, and the axes of our architecture search. **(a)** Shared minimal architecture (Table 4/5 and the latent-dim sweep): the latent is a *flat* vector $z_0 \in \mathbb{R}^L$; the field is a time-conditioned MLP $(L{+}1) \to w \to \cdots \to L$ (width $w{=}256$, depth $d{=}3$); the head is $\texttt{Linear}(L \to 10)$. **(b)** SFNO's native ConvNODE architecture: the latent is a *feature map* $z \in \mathbb{R}^{C \times H \times W}$ ($256 \times 7 \times 7$); the field is a `ConvODEfunc` (time-concatenated convolutions with h residual unit-blocks); the head is $\mathrm{ReLU} \to \mathrm{GAP} \to \texttt{Linear}(256 \to 10)$. *Every arm shares the template and parameter count; only the field-training objective differs* (SFNO / FW / NODE / STEER / PM). The search sweeps three knobs (highlighted): latent width L, encoder depth f (`f_add_blocks`), and field depth h (`h_add_blocks`). Script: `experiments/35h_arch_fig.py`; search: `experiments/35h_arch_search.py`.

head $z_1 \mapsto$ logits—are instantiated concretely per dataset in Appendices D (MNIST MLP stack) and F (CIFAR/SVHN ResNet carve and SFNO's native $256 \times 7 \times 7$ ConvNODE). Two of the three carry search knobs used below: the native encoder's residual-block count f (`f_add_blocks`), which sets how linearly separable z_0 already is, and the field's residual-block count h (`h_add_blocks`), which sets its transport capacity. (In the flat MLP field, $w{=}256$ is the *hidden* width, not the latent dim L.)

What "stem" means. The *stem* is the front convolutional feature extractor *inside* the encoder—it is a sub-component of the encoder, not a fourth stage. In the shared minimal CIFAR encoder the stem is three VGG-style blocks (channels $3 \to 32 \to 64 \to 128$, each block $= 2 \times [\text{conv}3{\times}3{+}\text{BN}{+}\text{ReLU}]$ then 2×2 MaxPool) that turn the 32×32 image into a 128-channel feature map; GAP+`Linear` then collapse it to the L-vector z_0. MNIST has *no* stem—its encoder is a single linear map. So "encoder = stem + pool + linear-to-L" on images, and "encoder = linear" on MNIST. (The 32 and 64 that appear in the CIFAR stem are intermediate channel counts, *not* the latent dimension, which is $L{=}64$.)

Search axes. The search varies exactly the three highlighted knobs while holding the template and parameter budget fixed: (i) *latent width* $L \in \{8, 16, 32, 64, 256, 784\}$ (Appendix G), which sets whether a non-crossing straight coupling can even exist; (ii) *encoder depth* f, which controls how linearly separable z_0 already is before the node; and (iii) *field depth* h, which controls how much transport capacity the node has. Sweeping f and h on the native arch (exp35h) is how we test whether a method's win is due to its *objective* or merely to a favourable depth—e.g. whether SFNO's advantage survives when the encoder is shallow enough that z_0 is *not* already separable.

G LATENT-DIMENSION ABLATION: WHEN THE STRAIGHT PATH HAS NOWHERE TO GO

The SFNO-recipe caveat (Appendix F) raises a mechanistic question: *why* does the simulation-free joint recipe fail on some architectures and not others? We isolate the responsible variable—*latent dimension* L—with a controlled sweep on both MNIST and CIFAR-10.

Controlled design. Every arm shares one architecture and differs *only* in the flow objective. The encoder is deliberately minimal so that L is the sole knob:

- **MNIST**: enc $=$ Linear$(784 \to L)$—a *single* learnable layer—followed by an affine-free `BatchNorm1d(L)` that merely standardizes z_0 (zero parameters).
- **CIFAR-10**: a shared conv stem (VGG-ish, three stride-2 blocks $\to 128 \times 4 \times 4$) $\to$ global-average-pool $\to$ Linear$(128 \to L)$ + affine-free `BatchNorm1d(L)`. The L-projection is the analogue of MNIST's single encoder layer.

On top of $z_0 \in \mathbb{R}^L$: a NODE field $v(z, t)$ (width 256, depth 3, $h = L$) and a *linear* head $L \to 10$. We sweep $L \in \{8, 16, 32, 64, 256\,(, 784)\}$ and compare *MLP* (the pure-CE teacher; also PM's distillation source), *PM* (distill the frozen `mid`, no solver-BP), *FW* (end-to-end CE through a randomized midpoint grid, solver-BP), and *SFNO* (joint simulation-free FM with shared per-class anchors $z_1 = \text{labelproj}(\text{onehot}(y))$). MNIST arms run 25k iterations; CIFAR arms run 60 epochs; seed 0.

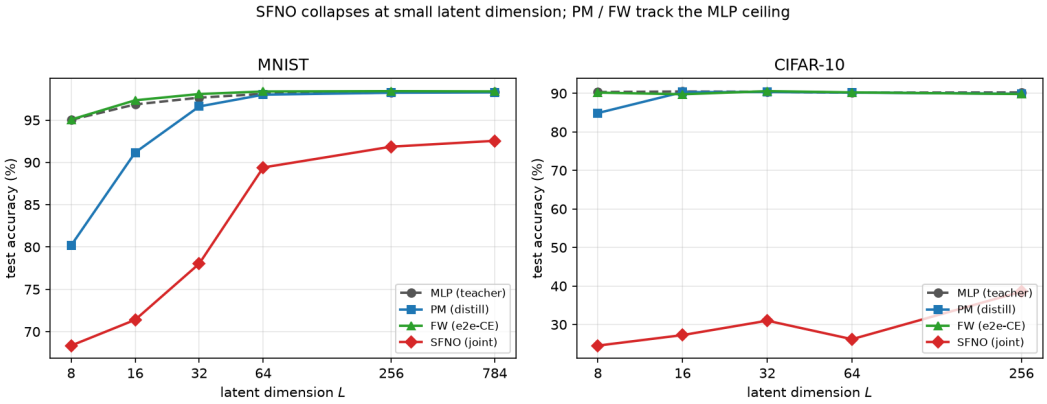


Figure 7: Test accuracy vs. latent dimension L (log-scale x-axis). **SFNO (red) collapses as L shrinks**—on MNIST from 92.6% at $L = 784$ to 68.4% at $L = 8$, and on CIFAR it never leaves the 24–39% band at *any* L under our shared conv stem—while **PM (blue)** and **FW (green)** track the *MLP teacher ceiling* (grey dashed, ≈98% MNIST / ≈90% CIFAR) down to small L. PM's only real degradation is at MNIST $L = 8$ (80%), and that is a *distillation-fidelity* effect (an 8-d field cannot fit the frozen MLP `mid` closely), *not* a crossing failure—it recovers the ceiling by $L = 32$.

Mechanism. The mechanism is the crossing conflict at the architecture level, analysed in the main text (Section 3.3): SFNO's simulation-free straight-line-to-shared-anchor fit *cannot curve around* a conflict, so once L is too small to place the 10 anchors and all encoded points without the straight $z_0 \to z_1$ segments crossing, the field averages conflicting velocities and samples land in the wrong basin. The sweep here pins the responsible variable (L) quantitatively and thus supplies the concrete cause behind the Appendix F caveat: SFNO needs a latent generous enough for a non-crossing straight coupling to exist, whereas FW (which curves through the solver) and PM (which regresses a per-sample smooth target, no forced merge) do not.

H RUNTIME SCALING LAW OF NODE TRAINING

The main-text efficiency claim (Table 4) is that PM trains at a fraction of the per-step cost of the solver-backpropagating baselines. Here we make that claim quantitative with (i) a *cost model*

Table 12: Test accuracy (%) at best-validation. On MNIST, PM and FW stay within ≈1% of the MLP teacher for $L \geq 32$; SFNO trails by 6–27% and its *ceiling* over the whole sweep (92.6%) never reaches the other arms' ≈98%. On CIFAR the gap is severe: SFNO stays at chance-plus (24–39%) for *every* L, whereas PM matches the ≈90% teacher from $L \geq 16$. (CIFAR MLP/FW ≈90% across L; omitted for width.)

	MNIST				**CIFAR-10**	
L	MLP	PM	FW	SFNO	PM	SFNO
8	95.0	80.2	95.1	68.4	84.9	24.5
16	96.9	91.2	97.3	71.4	90.3	27.3
32	97.7	96.6	98.1	78.0	90.3	31.0
64	98.1	98.0	98.4	89.4	90.2	26.2
256	98.3	98.2	98.4	91.9	89.9	38.6
784	98.4	98.3	98.4	92.6	—	—

for a NODE training step, and (ii) a controlled microbenchmark that isolates the field from the data pipeline (synthetic latent batches; all timings CUDA-synchronised, median of 100 reps after warmup, on an NVIDIA RTX A6000, latent $h = 64$).

Field-evaluation cost: $O(d\,w^2)$ in FLOPs. One evaluation of the velocity field $v_\theta(z, t)$ is a forward pass through the width-w, depth-d MLP $(h{+}1) \to w \to \cdots \to w \to h$. Writing N for the field parameter count (the analogue of the model size in the Kaplan et al. (2020) LLM law $C = 6ND$), the forward FLOPs per sample are $\approx 2N$, with

$$N \;=\; (h{+}1)w + (d{-}2)w^2 + wh \;\approx\; (d{-}2)\,w^2 \quad (w \gg h), \qquad F \equiv 2N, \tag{8}$$

so cost is *linear in depth* d and *quadratic in width* w. (Kaplan's $C = 6ND$ is the same $2N$ forward $+\ 4N$ backward accounting summed over D tokens; below we do *not* assume its implicit backward/forward ratio of 2, because our tiny field is latency- rather than compute-bound.) Wall-clock, however, follows FLOPs only in the compute-bound regime. Table 13 shows the two regimes: at the training batch size (256) the field is *latency-bound*—forward time is essentially flat in width (0.12 → 0.14 ms as w grows 16× and params 94×) because each $w \times w$ matmul is dominated by kernel-launch and memory latency, not arithmetic—whereas at a large batch (16,384) it becomes *compute-bound* and time tracks w^2 (0.13 → 1.33 ms). *Depth*, by contrast, costs in *both* regimes: layers form a sequential dependency chain the GPU cannot overlap, so wall-clock is linear in d regardless of batch ($R^2 = 0.99$; each added layer ≈ 117 μs at batch 256). This is precisely why PM's per-step time was flat across the latent-dimension sweep (Appendix G): its field sits in the latency-bound corner, so growing L (hence input/output width) is nearly free until the batch is large.

Table 13: Forward-evaluation time (ms) of the field vs. size, in the two hardware regimes. *Width* (depth fixed at 3) is flat at the training batch but tracks w^2 at large batch; *depth* (width fixed at 256) is linear in both. This is the empirical content of Eq. equation 8: cost is $O(d\,w^2)$ in FLOPs, but the *deployed* cost is governed by depth (sequential) far more than width (parallelisable) until the batch is large.

	vary width (d=3)			**vary depth** (w=256)		
	w=64	w=256	w=1024	d=2	d=4	d=8
batch 256 (latency-bound)	0.123	0.134	0.135	0.094	0.172	0.326
batch 16384 (compute-bound)	0.134	0.403	1.327	0.294	0.444	0.978

The PM-vs-FW ratio: isolating "no backprop through the solver." Let β = (backward cost)/(forward cost) per field evaluation. To compare the two *fairly* we hold the integrator fixed: both roll out the same S-step grid (S=3) with k field evaluations per step (k=2 for midpoint), so the rollout makes $M \equiv kS$ evals (M=6). The only difference is *where the gradient*

flows. FW retains the graph through all M evals and backpropagates through each. PM instead rolls out under `no_grad` and then does a *single* regression forward+backward on the S harvested target nodes (one per step, on the teacher path)—a batch of $S{\cdot}N$ points. Writing the wall-clock of that one harvested pass as $c\,(1{+}\beta)F$, where $c \in [1, S]$ measures its cost in units of a single N-batch evaluation,

$$T_{\mathrm{PM}} \approx \underbrace{MF}_{\substack{\text{no-grad rollout}\\(M \text{ sequential evals})}} + \underbrace{c\,(1{+}\beta)F}_{\substack{\text{one harvested}\\\text{fwd+bwd on } S \text{ nodes}}} = \big(M + c(1{+}\beta)\big)\,F, \qquad T_{\mathrm{FW}} \approx \underbrace{M(1{+}\beta)F}_{\substack{\text{fwd+bwd chained}\\\text{through all } M \text{ evals}}}, \quad (9)$$

so the field-size factor F cancels and

$$\frac{T_{\mathrm{FW}}}{T_{\mathrm{PM}}} \approx \frac{M(1{+}\beta)}{M + c(1{+}\beta)}. \quad (10)$$

The effective cost c of PM's harvested pass *depends on the regime*, and—this is the key point—so does β, along the *same* axis (batch×width):

- **Latency-bound** (small field, our benchmark backbone): a batch of $S{\cdot}N$ harvested points costs the same wall-clock as a single N-batch, because the eval is dominated by fixed kernel-launch/reduction overhead rather than arithmetic—so $c \approx 1$. In the *same* regime the backward, launching $\sim 2\times$ the kernels of an already overhead-dominated forward, gives a large $\beta \approx 9$. Then Eq. equation 10 predicts $M(1{+}\beta)/(M{+}1{+}\beta) \approx 3.8$ at $M{=}6$.
- **Compute-bound** (large field, e.g. SFNO's native 256-channel $\mathrm{ConvODEfunc}$): the harvested $S{\cdot}N$ batch is $S\times$ the FLOPs of a single N-batch, so $c \approx S$; simultaneously β relaxes to the textbook ≈ 2 ($2N$ fwd, $4N$ bwd, Kaplan's $C{=}6ND$). Then $M(1{+}\beta)/(M{+}S(1{+}\beta)) \approx 1.2$ (note c is capped at $S{=}3$, not $M{=}6$: PM harvests one target per *step*, whereas FW backprops through all k evals per step—this factor-of-k gap between M and S is exactly why two counts are needed).

Both predictions are confirmed by direct timing (`experiments/36_runtime_scaling.py`, `36c_native_field_ratio.py`). *Latency-bound backbone:* the ratio is flat at 3.6–4.0 across $w \in \{64, 128, 256, 512\}$ and depths 2–8 (as F cancels), and pins β *two independent ways that agree*—inverting Eq. equation 10 at the observed 3.76 ($M{=}6$, $c{=}1$) gives $\beta \approx 9$, while directly timing forward vs. forward+backward of one eval gives $\beta = (t_{\mathrm{fb}} {-} t_{\mathrm{f}})/t_{\mathrm{f}} \approx 8$. A ratio of 3.8 is moreover *impossible* for $\beta \le 2$ with $c{=}1$ (which caps at $1{+}\beta = 3$), independently proving $\beta > 2$ in this regime. *Native compute-bound field:* the isolated $\mathrm{ConvODEfunc}$ (batch 1024) measures $\beta = 1.75$ (i.e. $\approx$2, as the compute-bound limit predicts) and a field-only PM/FW ratio of 1.19, against the model's 1.16 at $\beta{=}1.75$, $c{=}S{=}3$—essentially exact, and reconciling the full-step 1.27 measured end-to-end on the native arch (exp35g). Figure 8 traces the β half of this transition directly. The upshot: PM is faster than FW in *both* regimes (the ratio exceeds 1 whenever $\beta > 1$), but the margin is largest for small latency-bound fields—the $\approx 3.8\times$ per-step speedup over FW seen on the MNIST/CIFAR shared backbone (Table 4)—and shrinks toward $\approx 1.2\times$ as the field grows compute-bound. This is the concrete, quantitative content of "no backprop through the solver": PM pays one flat backward where FW pays M chained ones, a gap that is dramatic exactly when a training step is launch- rather than FLOP-limited. The reference implementations are `experiments/36_runtime_scaling.py` (forward/β/step timing), `36b_beta_sweep.py` (Fig. 8), and `36c_native_field_ratio.py` (native compute-bound field).

β is a regime, not a constant. Figure 8 sweeps $\beta = (t_{\mathrm{fb}} - t_{\mathrm{f}})/t_{\mathrm{f}}$ over field width and depth at three batch sizes, and confirms the claim above directly. At our training batch (256, red) the field is latency-bound at every size and β stays pinned at $\approx$9–11: the forward eval of a small MLP is dominated by fixed per-kernel launch and reduction overhead, the backward launches roughly twice as many kernels, and the *ratio* of two overhead-dominated numbers is large. As the batch grows to 16,384 (blue) each eval becomes compute-bound once the width is large enough that the w^2 matmul dwarfs the fixed overhead, and β collapses onto the textbook value: from $\approx$11 at $w{=}64$ down to $\beta \approx 2$ at $w \ge 512$, sitting exactly on the Kaplan $C{=}6ND$ line ($2N$ forward, $4N$ backward). Crucially the collapse is driven by *width×batch* (matmul size), not by depth: at fixed width 256 (right panel) adding layers keeps the same per-layer latency floor, so β barely moves within a batch. This is why the $\beta \approx 9$ we use in Eq. equation 10 is the correct value *for the regime the models are actually trained in*, and—because the harvested-pass cost c moves from 1 toward S along the same

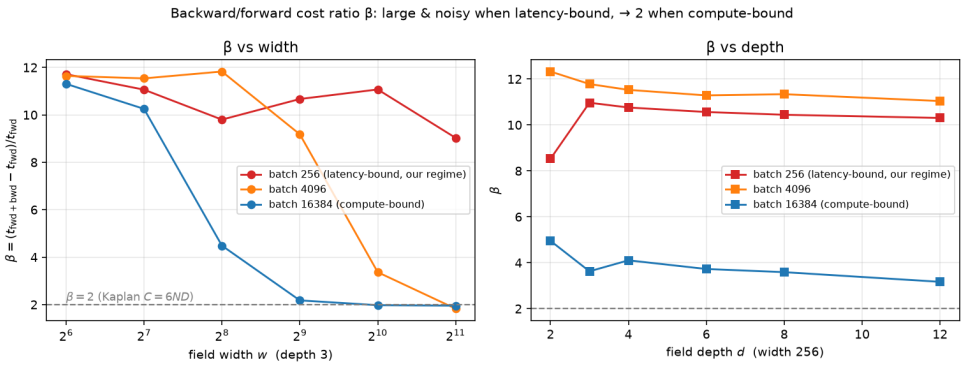


Figure 8: Backward/forward per-eval cost ratio β versus field width (left, depth 3) and depth (right, width 256), at three batch sizes; the transition analysed in the text. Dashed line: the compute-bound value β=2 implicit in Kaplan's C=6ND. RTX A6000, median of 200 CUDA-synced reps. Source: `experiments/36b_beta_sweep.py`.

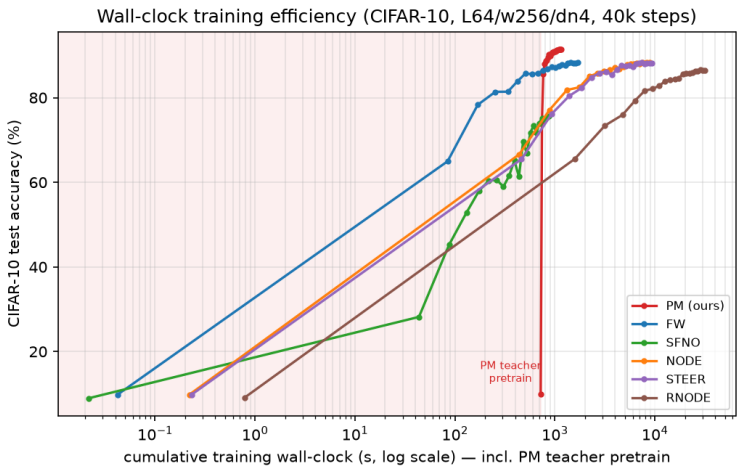


Figure 9: Wall-clock training efficiency on CIFAR-10 (L=64, w=256, d_{node}=4, 40k steps). x-axis is cumulative training wall-clock (log scale) using the Table 4 per-step cost; PM's curve is shifted right by its full teacher-pretraining time (shaded, ≈720 s). PM reaches the highest accuracy and finishes left of every solver-BP arm even with the teacher cost included.

axis—why a production-scale field trained at large batch sees the PM/FW ratio relax toward ≈1.2 (both β and c pushing the same way), not toward $1+\beta$. PM stays faster throughout, but the decisive margin is a small-field, launch-limited phenomenon.

End-to-end wall-clock, teacher cost included. The per-step ratio understates PM's total advantage because two effects cut in opposite directions: PM converges in the same number of steps as the solver-BP arms (so its per-step win compounds), but it also pays a one-time *teacher pretraining* cost the from-scratch arms do not. Figure 9 charts CIFAR-10 test accuracy against *cumulative wall-clock*—every arm's step k placed at $k \times$ (ms/iter), and PM's curve shifted right by its full teacher-pretraining time so the comparison is honest. The teacher is the SGD ResNet-20-exact classifier (272k params, 92.31%); pretraining it for 40k steps at ≈18 ms/iter costs ≈720 s. Even carrying that offset, PM reaches its ceiling first *in absolute time*: total wall-clock ≈1156 s (720 teacher + 436 field) to 91.4%, versus 1688 s for FW (88.3%), 8792 s for NODE, 9260 s for STEER, and 31592 s for RNODE. The one caveat visible in the figure is that FW crosses the 85% line slightly earlier (506 vs. 764 s) because PM must first amortize the teacher before its field training begins.

I PHYSIONET TIME-SERIES BENCHMARK

This appendix backs the time-series check of Section 3.2. The applications that motivate NODEs—latent dynamics for irregularly-sampled series (Rubanova et al., 2019)—are temporal, yet our main benchmarks are image classification; here we confirm the efficiency story transfers to a real clinical time-series task rather than a synthetic one.

Task and featurization. PhysioNet/CinC Challenge 2012 (Silva et al., 2012) asks for in-hospital-mortality prediction from the first 48h of an ICU stay: each patient is an irregularly-sampled multivariate series of 37 vitals/labs plus a few static descriptors, and $\approx 14\%$ of patients die (class-imbalanced, so we score *AUROC*, not accuracy). To keep a single shared backbone across every arm—exactly as in the UCI and image benchmarks—we featurize each patient's irregular series into a fixed 226-d vector of per-parameter summary statistics (mean/min/max/first/last/count over the 48h, plus statics). Every method sees the identical input z_0; this is a standard non-recurrent PhysioNet featurization and the point is not SOTA interpolation but a fair, arm-agnostic test of NODE-training efficiency on genuine dynamics. We pool set-a+set-b (8000 patients) and report over three independent $70/10/20$ train/val/test splits.

Backbone and arms. The shared model is $\mathrm{enc}(226{\to}64{\to}64) \to \mathrm{field}\,(\text{NODE}, w{=}64, d{=}3) \to \mathrm{head}(64{\to}1)$, class-imbalanced BCE at the endpoint. All arms match those of the image benchmark: NODE/STEER/RNODE backprop cross-entropy (here BCE) through an adaptive `dopri5` solver, FW through the randomized 3-node grid, SFNO uses its simulation-free latent recipe with the noised label auto-encoder, and PM is the same two-phase distillation as in the main text (freeze an end-to-end BCE teacher, then regress the field onto its per-sample $z_1 = \mathrm{mid}(z_0)$ with the bridge loss, no solver backprop). All arms train 40k steps.

Result. Table 14 gives the full breakdown behind the headline in Table 6—per-split std, the late-training stability proxy (std of the last-six test AUROCs), train-NFE, and ms/iter. Because the featurized task is not linearly separable, a static readout cannot shortcut it, so unlike the image benchmark we compare PM *directly* against end-to-end NODE without a load-bearing probe. The efficiency and stability story is the one from image classification, here on real temporal data: PM matches NODE's AUROC at a fraction of the per-step cost and with far less late-training oscillation than the only other solver-free arm, SFNO.

Table 14: PhysioNet 2012 in-hospital-mortality, full version of Table 6 with per-split std and train-NFE (AUROC, higher is better; mean$\pm$std over three independent splits). *late-std* = std of the last six test-AUROC evaluations (late-training stability proxy, lower is steadier); *NFE* = field evaluations per training step; *ms/iter* = idle-GPU wall-clock per step.

Arm	AUROC	late-std	NFE	ms/iter
NODE	0.833 ± 0.011	0.0016	102	160.0
SFNO	0.817 ± 0.008	0.0641	52	**6.3**
FW	0.825 ± 0.007	0.0004	46	16.8
STEER	0.829 ± 0.004	0.0001	82	134.9
RNODE	0.834 ± 0.008	0.0010	14	90.9
PM (ours)	$\mathbf{0.839 \pm 0.008}$	**0.0002**	44	**5.5**