LFNT: Forecasting training trajectories in the latent space of weights

Anonymous authors
Paper under double-blind review

Abstract

Forecasting neural network weights can accelerate training by skipping portions of the optimization trajectory, but existing approaches operate directly in weight space. Motivated by the success of learned latent representations in other domains, we propose forecasting optimization trajectories in a learned latent space of weights. We introduce **L**atent **F**orecasting of **N**eural **T**rajectories (**LFNT**, pronounced "elephant"), a three-stage approach that (i) learns a latent representation of model weights, (ii) aligns these representations using forecasting supervision, and (iii) trains a forecaster on the resulting latent trajectories. Across vision and language tasks, latent-space forecasting provides greater training speedups than weight space forecasting baselines, with speedups of up to 59%. Overall, the results show that the representation of optimization trajectories substantially affects forecasting performance, and that a learned latent space can provide a better alternative to forecasting directly in weight space.

1 Introduction

Representation learning aims to obtain features better suited for downstream predictors (Bengio et al., 2013). However, its role is also more fundamental, learned representations can change the geometry of the problem itself. Two stage generative models illustrate this clearly by mapping images from their pixel space to a learned latent space and effectively performing the generation there instead (Dai & Wipf, 2019; Van Den Oord et al., 2017; Esser et al., 2021; Rombach et al., 2022). The task remains the same but the space in which it is carried out changes. This illustrates the broader idea: the original representation of data is not always the most suitable space to operate in.

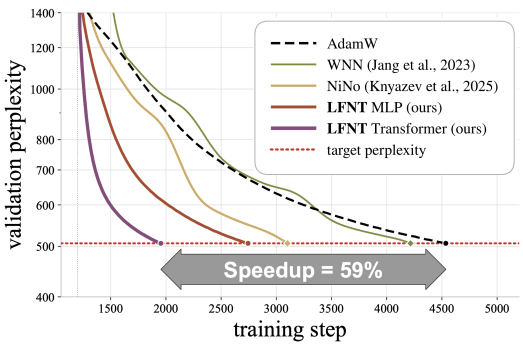


Figure 1: Training curves comparing latent space and weight space forecasting on a language task (LM1B).

A similar perspective has begun to emerge for neural network weights. Weight Space Learning (WSL) is a new ML discipline where neural network weights themselves are treated as data from which useful representations can be learned (Han et al., 2026). These representations support several downstream tasks such as the prediction of model properties from weights (Eilertsen et al., 2020; Unterthiner et al., 2020; Schürholt et al., 2021), and generation of model weights (Schürholt et al., 2024; Soro et al., 2025; Gupta et al., 2026).

Beyond these tasks, recent work suggests that learned weight representations can provide a more suitable space for directly manipulating model weights. Soro et al. (2026) showed that model merging in a learned latent space of weights outperforms analogous operations in the weight space, while Asefaw et al. (2026) demonstrate a similar advantage for gradient update steps.

Another promising application of WSL is accelerating neural network optimization. Optimization of neural networks produces a sequence of weight states $W_0, W_1, \ldots, W_T$, forming a trajectory through weight space. Weight forecasting methods such as WNN (Jang et al.,

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.

2023) and NiNo (Knyazev et al., 2025) exploit this structure to predict future model states from a history of checkpoints, thereby skipping portions of the trajectory. These methods, however, perform this forecasting directly in the original weight space, which may lack smoothness and robustness required for accurate prediction. This leads to the question of whether one can learn a latent space of weights which is more suitable for forecasting these optimization trajectories.

To study this, we introduce **L**atent **F**orecasting of **N**eural **T**rajectories (**LFNT**, pronounced "elephant"), a new method which forecasts future weights in a learned latent space of weights instead of directly in weight space. LFNT is trained in three stages: (i) learning a latent representation of model weights using a variational autoencoder; (ii) aligning this latent space using a forecasting objective to encourage forecastable trajectories; (iii) training a latent forecasting model that predicts latent representations of future weight updates. Across vision and language tasks, including out-of-distribution datasets and model depths, we find that forecasting in the learned latent space consistently provides greater training speedups than forecasting directly in weight space. Summarizing, we make the following contributions:

1. **Forecast-aligned weight representations.** We introduce a procedure for aligning a latent space of weights to optimization trajectories using a forecasting objective, improving forecasting performance while preserving the functionality of decoded weights.

2. **Forecasting training trajectories in the latent space.** Using this aligned latent space, we show that forecasting in the learned latent space consistently outperforms forecasting directly in weight space across CNN, ResNet, and GPT-2 models.

3. **Generalization of latent forecasting.** We show that the learned latent space and forecasters can transfer to unseen datasets and model depths without retraining, demonstrating that latent forecasting generalizes beyond the training data.

2 Related work

Weight forecasting. Weight forecasting methods exploit predictable structure in optimization trajectories to accelerate training. Introspection (Sinha et al., 2017) showed that future neural network weights can be predicted from previous checkpoints, while WNN (Jang et al., 2023) developed this idea into periodic weight forecasting from checkpoint histories. NiNo (Knyazev et al., 2025) extends this idea with a graph-based representation that models neuron interactions. Complementary work studying the directionality of optimization trajectories (Singh et al., 2025) provides further evidence that neural network training follows structured rather than arbitrary paths through parameter space. Chijiwa (2024) proposed an algorithm that transfers an existing optimization trajectory to a new task to accelerate training. Despite these advances, existing forecasting methods operate directly on the original model weights. Our work instead asks whether learning a representation of the weights can provide a space in which these optimization trajectories are easier to forecast.

Optimization and learned optimizers. A complementary line of work seeks to accelerate neural network training by improving the optimization procedure itself. Adaptive first-order methods such as Adam and AdamW improve parameter updates based on gradient moments (Kingma & Ba, 2014; Loshchilov & Hutter, 2019), while methods such as Muon (Jordan et al., 2024) exploit weight matrix structure in the optimization problem (Gupta et al., 2018; Vyas et al., 2025). Other approaches instead learn the optimization rule itself (Andrychowicz et al., 2016; Wichrowska et al., 2017; Metz et al., 2022; Moudgil et al., 2025). Optimizers and weight forecasting act at different levels: the former determines every step of the optimization trajectory, whereas the latter predicts future states along that trajectory. These approaches are complementary and could be combined. In this work, however, the central question is whether the space in which an optimization trajectory is represented affects forecasting performance.

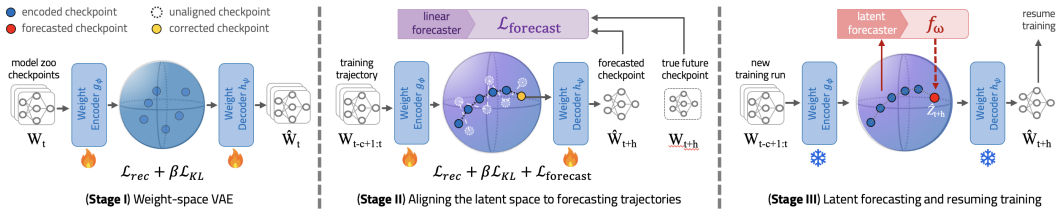


Figure 2: Overview of **LFNT**, a three-stage approach for accelerating neural network training by forecasting training trajectories in a learned latent space of weights. **Stage I:** We first learn a latent representation of neural network checkpoints using a weight-space variational autoencoder. **Stage II:** We then align this latent space with the optimization trajectories using a linear forecasting objective. **Stage III:** Finally, we freeze the learned representation and train forecasters on the resulting latent trajectories. During inference, checkpoints from a new training run are encoded as context, a future latent state is forecasted from the history, decoded back to model weights and used to resume training from the forecasted state.

3 Method

We propose **LFNT**, which accelerates neural network training by forecasting training trajectories in a learned latent space of weights, across diverse datasets and architectures. To this end, the method consists of three stages as seen in Fig. 2. First, an autoencoder is trained to obtain a latent space of neural network weight representations. Second, this latent space is aligned to the optimization trajectories using a forecasting objective. Finally, the autoencoder is frozen and forecasters are trained directly on the resulting latent trajectories. At inference, the future latent is decoded back into model weights, which are used to resume standard training.

3.1 Problem Formulation

Consider the optimization trajectory of a neural network $\mathcal{T} = \{W_0, W_1, \ldots, W_T\}$, where $W_t \in \mathcal{W}$ denotes the model weights at optimization step t. We refer to a collection of independently trained trajectories as a model zoo, $\mathcal{M} = \{\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(M)}\}$, on which the autoencoder and forecasting models are learned. Following the literature (Jang et al., 2023; Knyazev et al., 2025; Shou et al., 2025), we consider the task of predicting a future model state W_{t+h} at horizon h, from a context of previous checkpoints $W_{t-c+1:t} = (W_{t-c+1}, \ldots, W_t)$. In this work a context of $c = 5$ checkpoints is used.

A weight-space forecaster f_ω directly maps the context to a prediction of future weights:

$$\hat{W}_{t+h} = f_\omega\left(W_{t-c+1:t}\right). \tag{1}$$

We represent each checkpoint in a learned latent space $\mathcal{Z}$. Let τ denote a tokenization (see Sec. 3.2) of weights, and let g_ϕ and h_ψ denote the encoder and decoder respectively. A checkpoint is represented as $Z_t = g_\phi(\tau(W_t))$, and a forecaster $f^{\mathcal{Z}}_\omega$ is applied to the latent trajectory:

$$\hat{W}_{t+h} = h_\psi\left(f^{\mathcal{Z}}_\omega\left(Z_{t-c+1:t}\right)\right), \qquad Z_t = g_\phi(\tau(W_t)). \tag{2}$$

Our objective is therefore not only to learn a representation space of model weights, but also to learn one whose structure is better for forecasting, whilst still being able to decode latent states back to weights. We study this setting under single-jump forecasting where a single future model state is predicted and decoded, after which standard optimization resumes from the forecasted weights.

3.2 Stage I: Learning a Latent Representation of Weights

We first learn a latent representation of weights from checkpoints sampled across the training model zoos, using a similar setup to Schürholt et al. (2024); Falk et al. (2025). Given a model checkpoint W_t, the tokenizer τ converts its weights into a sequence of tokens $X_t = [x_{t,1}, \ldots, x_{t,N}]$. Following (Falk et al., 2025), layer parameters are reshaped into

matrices and flattened into contiguous sequences, which are partitioned into fixed-size tokens and concatenated across layers. This densely packs the parameters into tokens, allowing models to be processed as token sequences.

A variational encoder g_ϕ maps the token sequence X_t to posterior parameters $\mu_t, \sigma_t \in \mathbb{R}^{N\times d}$, where N denotes the number of weight tokens and d the latent dimensionality of each token, from which we sample $Z_t = \mu_t + \sigma_t \odot \epsilon$, with $\epsilon \sim \mathcal{N}(0, I)$. A decoder h_ψ maps Z_t back to the original weight-space, yielding reconstructed model parameters $\hat{W}_t$.

In this first stage, checkpoints are treated independently of their position within an optimization trajectory. The encoder and decoder are optimized using the standard variational autoencoder objective, combining reconstruction and KL regularization,

$$\mathcal{L}_{\text{Stage I}} = \mathcal{L}_{\text{rec}} + \beta\mathcal{L}_{\text{KL}}, \tag{3}$$

where $\mathcal{L}_{\text{rec}}$ measures the mean-squared error between the original and reconstructed weights, and $\mathcal{L}_{\text{KL}}$ regularizes the approximate posterior towards the prior distribution.

At this stage, the latent space is optimized only to preserve information about individual models. No constraint is imposed on how the trajectory is arranged within the latent space. Stage II introduces such constraints by optimizing the latent space for forecasting.

3.3 Stage II: Shaping the Latent Space for Forecasting

In Stage II, we introduce a forecasting objective to explicitly encourage optimization trajectories to be predictable in the latent space. We use a linear forecaster whose parameters are obtained via ordinary least squares rather than optimized end-to-end with the entire model. This deliberately constrains the capacity and optimization of the forecaster and moves the burden of making trajectories predictable on the autoencoder.

Linear forecaster. Given a context of c latent checkpoints $Z_{t-c+1:t}$, with $Z_t \in \mathbb{R}^{N\times d}$, we estimate the local direction of the trajectory independently for each token and latent coordinate. Let $\delta_i = i - c \in \{-(c-1), \ldots, 0\}$ denote the position of checkpoint i relative to the most recent checkpoint in the context. For token n and latent coordinate j, the least-squares slope is:

$$V_{t,n,j} = \frac{\sum_{i=1}^{c}(\delta_i - \bar{\delta})Z_{t-c+i,n,j}}{\sum_{i=1}^{c}(\delta_i - \bar{\delta})^2}, \tag{4}$$

where $V_t \in \mathbb{R}^{N\times d}$ collects the resulting slopes. Given V_t, the latent state h steps into the future is obtained by linear extrapolation as $\widetilde{Z}_{t+h} = Z_t + hV_t$. This forecaster is fitted independently for each five-checkpoint context. Consequently, improvements in forecasting cannot be absorbed by an expressive forecaster but instead come from adapting the learned representations through the autoencoder.

Weight-displacement supervision. Rather than supervising the forecasted weights against the absolute future checkpoint, we compare the change induced by the latent forecasting to the actual change observed during optimization. The true displacement is defined as $\Delta W_{t,h} = W_{t+h} - W_t$, while the forecasted displacement is $\Delta\hat{W}_{t,h} = h_\psi(\widetilde{Z}_{t+h}) - h_\psi(Z_t)$. The mean-squared error is then minimized: $\mathcal{L}_{\text{forecast}} = \text{MSE}\left(\Delta\hat{W}_{t,h}, \Delta W_{t,h}\right)$.

Using displacements focuses the objective on the change induced by the forecast rather than on the absolute reconstruction of the future checkpoint. Suppose the decoder introduces a reconstruction error e, such that $h_\psi(Z_t) = W_t + e$. Even if the forecast induces exactly the correct change, its decoded forecast may be $h_\psi(\widetilde{Z}_{t+h}) = W_{t+h} + e$. An absolute objective comparing $h_\psi(\widetilde{Z}_{t+h})$ directly with W_{t+h} would still penalize the offset e, despite the forecast having captured the correct optimization dynamics. In contrast, the shared reconstruction offset cancels in the displacement:

$$h_\psi(\widetilde{Z}_{t+h}) - h_\psi(Z_t) = (W_{t+h} + e) - (W_t + e) = W_{t+h} - W_t. \tag{5}$$

The displacement objective therefore isolates the forecasted parameter change from the reconstruction errors handled by other separate losses.

Why supervise in weight space? A natural alternative is to directly regress the predicted latent state towards the encoded future checkpoint, $\|\widetilde{Z}_{t+h} - Z_{t+h}\|$, as commonly done when predicting learned representations. However, both sides of such an objective depend on the representations being learned. This introduces degenerate solutions in which the encoder can simplify the prediction problem by, for example, shrinking temporal differences between checkpoints causing a representational collapse, rather than by making the trajectory easier to forecast. Related representation-prediction methods (Grill et al., 2020; Assran et al., 2023; Chen & He, 2021) address this issue through mechanisms such as stop-gradients, asymmetric branches, or target encoders. We instead opt for the simpler approach of supervising the decoded forecast directly in weight space; latent forecasts are decoded back to model parameters and compared against a target fixed by the observed optimization trajectory. Thus the target, $\Delta W_{t,h}$, does not move together with the learned representation and provides an external reference for shaping the latent space.

To prevent the forecasting objective from destroying the reconstructive property of our latent space, we retain the objectives from Stage I. The complete Stage II objective is

$$\mathcal{L}_{\text{Stage II}} = \mathcal{L}_{\text{rec}} + \beta\mathcal{L}_{\text{KL}} + \lambda\mathcal{L}_{\text{forecast}}. \tag{6}$$

3.4 Stage III: Forecasting in the learned space

In Stage III, we freeze the encoder g_ϕ and decoder h_ψ, but replace the simple linear forecaster from Stage II with a new forecasting model. This way, Stage III reuses the structure imposed in Stage II, but trains a stronger forecaster of future model states.

Given a context of c latent checkpoints $Z_{t-c+1:t} = (Z_{t-c+1}, \ldots, Z_t)$, a forecaster $f_\omega^{\mathcal{Z}}$ predicts the future latent state: $\hat{Z}_{t+h} = f_\omega^{\mathcal{Z}}(Z_{t-c+1:t})$. We optimize the forecaster with an ℓ_1 loss between the predicted and target future latent states, $\mathcal{L}_{\text{l1}} = \|\hat{Z}_{t+h} - Z_{t+h}\|_1$. Unlike in Stage II, direct supervision in latent space no longer introduces a moving target, since the encoder is frozen and Z_{t+h} therefore remains fixed throughout training.

Forecasters. We consider two forecasters on the learned representations. Firstly, **Latent MLP**, which adapts WNN (Jang et al., 2023) to operate on latent trajectories rather than on the original model weights. The same context of five checkpoints is provided to the forecaster, but each checkpoint is represented by its latent weight representation Z_t. This provides a direct comparison between forecasting in the original weight space and forecasting after learning a representation of the weights. We additionally consider a Transformer forecaster, **Latent Transformer**, designed to model interactions across the dimensions of the latent weight representation. Given the sequence of c previous latent states, the Transformer jointly processes their temporal evolution and predicts the latent representation at the target horizon. This provides a complementary forecaster that is not tied to the architecture of WNN. Further details can be found in App. C.

Forecasting new training trajectories. Once the autoencoder and forecaster have been trained, all components remain fixed at inference. Given a new training run, standard optimization is first performed until c checkpoints have been collected. These checkpoints are tokenized and encoded using the frozen encoder to obtain $Z_{t-c+1:t}$, from which the forecaster predicts the future latent state $\hat{Z}_{t+h}$. The prediction is then decoded back to model weights, $\hat{W}_{t+h} = h_\psi(\hat{Z}_{t+h})$, and inserted back into the training run. Optimization resumes from these forecasted weights while retaining the optimizer state from step t. Importantly, no adaptation of autoencoder or forecaster is performed for the target dataset. An unseen dataset therefore only changes the optimization trajectory supplied to the frozen forecasting setup. We also evaluate transfer to previously unseen model depths within the same architecture family, applying the same autoencoder and forecaster without retraining. Our method performs a single forecast, after which standard optimization continues.

4 Experiments

4.1 Experimental Setup

Model zoos, datasets, and architectures. We evaluate our method individually across three architectures: CNN, ResNet (He et al., 2016) and GPT-2 (Radford et al., 2019). A model zoo consists of checkpoint trajectories trained on the same dataset, architecture, and optimization configuration, using a different random seed per trajectory. For CNN and ResNet18, autoencoder and forecaster training use 6 and 4 model zoos, respectively, each corresponding to a different image dataset. CNN trajectories are trained for 10k optimization steps with checkpoints stored every 8 steps, whereas ResNet and GPT-2 trajectories are trained for 25k steps with checkpoints every 200 steps. In-distribution (ID) evaluation is performed on unseen optimization trajectories for the same dataset and architecture, differing only in the seed. Out-of-distribution (OOD) dataset evaluation uses optimization trajectories from datasets unseen during autoencoder and forecaster training. For GPT-2 we use one model zoo for training and ID evaluation, with separate unseen trajectories for the latter, and 3 additional unseen datasets for OOD evaluation. The CNN training zoos contain 500 independently trained trajectories each, compared with 200 trajectories for the ResNet and GPT-2 zoos. To evaluate generalization across architecture depths, we additionally apply the ResNet18 autoencoder and forecasters to ResNet14, ResNet26, and ResNet34 trajectories, and the GPT-2 setup to trajectories for models with 2, 4, and 6 Transformer blocks, without retraining the autoencoder or forecaster.

Baselines. We compare against two weight-space forecasting methods, WNN (Jang et al., 2023) and NiNo (Knyazev et al., 2025). WNN predicts future weights directly in weight space from a history of checkpoints using a simple learned MLP forecaster, and serves as our primary baseline because its forecasting setup can easily be transferred directly to the latent space. Our Latent MLP forecaster uses the same architecture as WNN whilst the input is changed from weights to latent vectors. This provides a controlled comparison on the effect of just changing the representation space whilst NiNo provides a stronger recent weight-space baseline based on a graph representation of neural networks.

Forecasting protocol and metrics. Because our central question concerns the effect of representation space on forecasting performance, we hold the optimization setting fixed in our main comparisons to isolate the effect of changing from weight space to latent space. In App. B.2, we show that our approach remains effective when applied to trajectories produced under different settings. For the main comparisons, we evaluate all methods using a single forecast starting at the optimization step $T_{\text{start}} = 1k$. For CNN and ResNet, we forecast $h = 9$k steps ahead, and for GPT-2 $h = 4$k steps. Since WNN and NiNo were designed for periodic forecasting, we additionally evaluate them under their periodic forecasting setting and compare them against our single-jump approach. We report optimizer-step speedup as the number of optimization steps required to reach a fixed target performance. The target is defined by the performance of the reference Adam trajectory at a predefined reference step T_{ref} (10k steps for the vision experiments and 5k for GPT-2). If S_{base} and S_{forecast} denote the resulting steps-to-target, we report optimizer-step speedup as $(1 - S_{\text{forecast}}/S_{\text{base}})$, expressed as a percentage. All reported results are averaged over 20 new optimization runs. We additionally report wall-clock speedup under the same criteria, including the cost of encoding, forecasting, and decoding. More details on the setup are provided in App. C.

4.2 Properties of the Learned Weight Space

We first examine whether the learned representation has properties that make forecasting easier. We study two properties: directional consistency and robustness. First, motivated by the directional structure observed in neural-network optimization trajectories (Singh et al., 2025), we ask if the latent space makes optimization trajectories straighter and therefore easier to extrapolate. Second, the space should also be robust to forecast errors, such that imperfect forecasts cause only limited degradation in model performance after being decoded.

Directional consistency. We first investigate whether optimization trajectories evolve more smoothly in the learned latent space than in weight

space by measuring consecutive-step cosine similarity. For consecutive checkpoints, we define $s_t = \bar{Z}_{t+1} - \bar{Z}_t$, where $\bar{Z}_t$ denotes the token-mean representation at checkpoint t, and compute the cosine similarity $\cos(s_t, s_{t+1})$. The same quantity is measured for the corresponding tokenized weights. Higher values indicate greater directional consistency and thus locally straighter trajectories. As shown in Fig. 3, directional consistency decreases as training approaches convergence, but latent trajectories remain generally smoother at matched training positions, with the largest differences occurring during early and intermediate training.

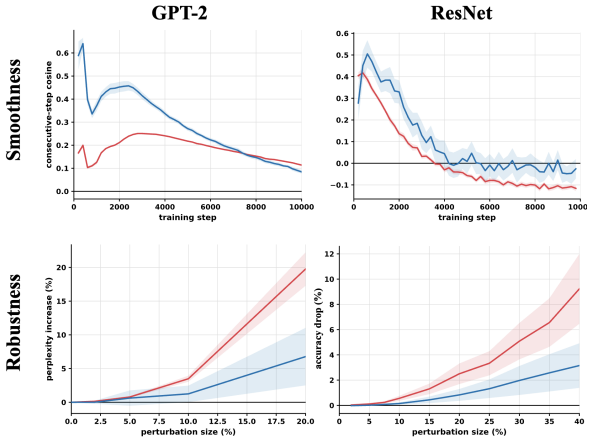


Figure 3: Top row: Trajectory smoothness across steps. Bottom row: Robustness to perturbations. Blue denotes latent space and red weight space. Results are shown for GPT-2 transformer blocks and for the ResNet18 models.

Robustness to perturbations. Smooth trajectories alone are not sufficient if small forecasting errors translate into large changes in model behavior. Therefore we next test whether the learned latent space is more tolerant to such errors than weight space. At held-out checkpoints, we perturb each representation by the same fraction f of its T_{start}-to-T_{ref} training displacement and measure the drop in validation performance relative to the unperturbed model. As shown in Fig. 3, latent perturbations consistently cause smaller performance degradation. Thus, in addition to producing straighter trajectories, the learned representation is more tolerant to deviations from the true trajectory.

Together these results provide evidence that the learned representation exhibits properties favorable for forecasting. We next evaluate whether this translates into improved training acceleration.

4.3 Forecasting in Latent Space and Weight Space

In this section, we compare latent- and weight space forecasting under matched protocols, first on new in-distribution trajectories, then under dataset and model-depth shifts, and finally in a periodic forecasting setting.

In-distribution forecasting. We first evaluate forecasting on in-distribution trajectories. As shown in Table 1, forecasting in the learned latent space consistently improves optimizer-step speedup across all architectures. The effect is stronger for ResNet and GPT-2, where forecasting in weight space provides small speedup (7-16%), but jumps to 27-42% after moving to the latent representation, indicating that the choice of representation space is important for forecasting performance. Latent Transformer provides a further improvement over Latent MLP to 29 and 59% respectively.

Table 1: Optimizer-step speedup (%) on in-distr. optimization runs, averaged over 20 runs per dataset; wall-clock speedup in parentheses.

Method	CNN	ResNet	GPT-2
WNN	13.2 (13.1)	−0.9 (−1.0)	−1.8 (−1.9)
NiNo	22.5 (22.2)	7.2 (7.0)	15.7 (15.0)
Latent MLP	28.0 (27.7)	26.5 (25.5)	41.9 (41.1)
Latent Transf.	**30.5** (29.8)	**28.9** (27.7)	**59.2** (58.2)

Although latent-space forecasting introduces additional encoding, decoding, and forecasting cost, the optimizer-step gains translate closely to wall-clock savings (Table 1). WNN has the lowest forecasting overhead, followed by NiNo's graph-based predictor and the latent methods, with the Transformer being the most expensive. Despite this additional cost, the larger speedups in latent space compensate for the overhead, which remains small relative to the saved optimization time.

Table 2: Optimizer-step speedup (%) on out-of-distribution datasets, averaged over 20 runs per dataset. Our latent-space methods are highlighted in gray.

Modality	Arch.	Dataset	Weight Space		Latent Space	
			WNN	NiNo	Latent MLP	Latent Transf.
Vision	CNN	BreakHis	6.2	18.4	**24.8**	23.4
		LEGO	13.1	25.0	47.3	**50.5**
		Simpsons	30.5	37.2	37.6	**39.4**
		Mean	16.6	26.9	36.5	**37.8**
	ResNet	BreakHis	11.1	18.5	26.6	**33.3**
		LEGO	7.5	15.8	17.6	**18.2**
		Simpsons	5.8	16.5	37.9	**38.6**
		Mean	8.2	17.0	27.4	**30.0**
Language	GPT-2	WikiText	3.1	15.2	30.7	**43.9**
		Shakespeare	1.8	-5.8	38.4	**42.6**
		OpenWebText	1.7	12.0	24.6	**42.8**
		Mean	2.2	7.1	31.2	**43.1**

Out-of-distribution forecasting. We first consider a dataset shift, where the model architecture is kept fixed but the trajectories come from datasets not seen during autoencoder or forecaster training. As shown in Table 2, the advantage of latent-space forecasting persists across the different architectures, demonstrating that the learned representation and forecaster transfer across datasets rather than being restricted to the training distributions. One interesting case is GPT-2 on Shakespeare, where NiNo degrades substantially more than on the other corpora. A possible explanation is that as NiNo learns a separate embedding for each vocabulary token, it might overfit to corpus-specific token statistics, which transfer reasonably to WikiText and OpenWebText, whose token statistics are more similar to the LM1B corpus, but not Shakespeare. Interestingly, the absolute speedups are, in several cases, larger than those for the ID experiments, which happens because the OOD baselines reach their target earlier. Thus, the denominator of the speedup metric is lower, and even fewer absolute steps saved correspond to a larger speedup. The results should therefore not necessarily be interpreted as the OOD trajectories being easier to forecast, but rather that the speedup persists under distribution shifts from the datasets used during training.

We next consider an architecture shift in Table 4, where the same learned representation and forecasters are applied to previously unseen model depths whilst using an ID dataset. An interesting finding is that the speedup increases monotonically with model depth for the GPT-2 transformers, suggesting that the forecaster generalizes well to Transformer blocks and benefits from the fact that deeper GPT-2 models contain more of them. The drop of the weight-space methods, particularly on unseen ResNet depths, shows the difficulty of forecasting weights directly under such changes in parameterization, whilst the latent methods remain effective. Thus, latent forecasting generalizes better than the weight space baselines across the evaluated model-depth shifts.

Periodic forecasting. Our main experiments use a matched single-jump protocol, although WNN and NiNo were originally designed for periodic forecasting. We therefore additionally evaluate periodic NiNo on LM1B-6/24 using its original k-decay schedule. As shown in Table 3, our single latent-space jump gives a much larger early speedup, but the gap narrows as NiNo accumulates gains through repeated forecasts.

Table 3: Optimizer-step speedup (%) on LM1B. Column headers report the Adam step and corresponding perplexity targets.

Method	**5k** (501)	**10k**(399)	**15k**(369)
NiNo periodic	36.0	46.2	47.7
Ours, 1-jump ($h = 4k$)	58.0	49.7	45.0
Ours, 2-jumps ($h = 4k$)	58.1	56.6	51.5
Ours, 3-jumps ($h = 4k$)	57.8	54.4	50.8
Ours, 1-jump ($h = 16k$)	**65.7**	59.7	53.1
Hybrid (Ours+NiNo)	**65.7**	**64.5**	**66.8**

Table 4: Optimizer-step speedup (%) when generalizing to architecture depths unseen during training. Presented here using CIFAR-100 for ResNet and LM1B for GPT-2.

Family	Arch.	Weight Space		Latent Space	
		WNN	NiNo	Latent MLP	Latent Transf.
ResNet	R14	-0.7	5.6	34.3	**36.3**
	R26	-5.1	1.0	30.4	**32.4**
	R34	-4.6	0.4	27.9	**30.1**
	Mean	-3.5	2.3	30.8	**33.0**
GPT-2	2/24	-4.2	13.6	36.1	**55.0**
	4/24	0.6	16.8	43.5	**57.8**
	6/24	2.9	15.7	46.2	**58.1**
	Mean	-0.2	15.4	41.9	**57.0**

Nevertheless, our single-jump method remains close through 15k steps, and ahead when increasing the forecasting horizon h further. Repeating latent forecasts provides modest additional benefit and more importantly still shows the decaying speedup as training progresses, likely because reconstruction errors accumulate across repeated steps. This motivates a hybrid strategy that uses our latent forecast for the initial jump and periodic NiNo thereafter, achieving the highest speedup at all evaluated targets. Overall, latent forecasting is particularly effective for large early jumps whilst periodic weight-space forecasting is complementary for maintaining the resulting advantage.

4.4 Limitations

One limitation of our approach is the upfront requirement for a collection of optimization trajectories to learn the representations and train forecasters. This cost is not included in the reported speedups and must be amortized over subsequent runs, making the approach most attractive when reused across many tasks. Recent work has begun to reduce this dependency by learning representations from existing heterogeneous model collections (Falk et al., 2025). Secondly, although we demonstrate transfer to unseen datasets and model depths, we still train a separate representation and forecaster for each architecture family. Recent WSL methods suggest that shared representations across heterogeneous architectures are possible (Falk et al., 2025; Hanna et al., 2026), making this a promising direction for future work. Latent forecasting also requires decoding predictions back to weight space, introducing reconstruction error that may accumulate across repeated interventions, which currently favors large single-jump forecasts over frequent forecasting. Finally, due to a fundamentally challenging problem of learning on weights, our experiments are limited to CNNs, ResNets, and relatively small GPT-2 models, leaving scalability to substantially larger models open.

5 Conclusion

This work studies whether neural network optimization trajectories are easier to forecast in a learned latent space of weights than in the original weight space. We introduce LFNT, which aligns latent representations of weights with optimization trajectories using a forecasting objective and trains a forecaster on the resulting latent trajectories. Across different tasks, latent-space forecasting yields larger optimizer-step and wall-clock savings than weight-space forecasting, with gains extending to unseen datasets and model depths. Our analysis further shows that the learned space exhibits greater directional consistency and is more robust to perturbations, providing insight into why latent-space forecasting is effective. More broadly, our results suggest that, much like pixel space is not always the most suitable space for image modeling, the original weight space need not be the most suitable space for neural network modeling. Learning a latent representation of weights can improve the forecasting of optimization dynamics compared to operating directly in weight space.

AI use statement

In this work, we used generative AI tools to provide feedback on research methodology and implement methods. But we have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, propose or refine hypotheses, assist with translation, clean and reformat datasets, support qualitative and thematic data analysis, or to interpret results. And the rest of the required disclosure tasks such as formulating mathematical claims, providing critical ingredients for proving mathematical claims and assisting in the writing of proofs, are not applicable to this work. Additionally, we used generative AI tools to create or edit software code and edit a research paper to improve readability. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

Reproducibility statement

We provide detailed descriptions of the model architectures, model-zoo construction, datasets, preprocessing, forecasting baselines, training procedures, hyperparameters, and evaluation protocols in the main paper and Appendix. The Appendix further documents architecture-specific implementation details, depth-transfer procedures, and additional ablations required to reproduce the reported results. Following the review process, we will release the full training and evaluation code together with the model zoos, autoencoder and forecaster checkpoints, and configuration files used in this work.

References

Samiul Alam, Tahsin Reasat, Rashed Mohammad Doha, and Ahmed Imtiaz Humayun. NumtaDB: Assembled bengali handwritten digits. *arXiv preprint arXiv:1806.02452*, 2018. 17

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, volume 29, pp. 3981–3989, 2016. 2

Aron Asefaw, Konstantinos Tzevelekakis, Damian Falk, Léo Meynent, and Damian Borth. WeightCLIP: Aligning datasets and models for weight space learning. In *Forty-third International Conference on Machine Learning*, 2026. 1, 16

Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15619–15629. IEEE, 2023. 5

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013. 1

Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014. 17

Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *2021 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 15745–15753. IEEE, 2021. 5

Daiki Chijiwa. Transferring learning trajectories of neural networks. In *International Conference on Learning Representations*, volume 2024, pp. 50228–50253, 2024. 2

Bin Dai and David Wipf. Diagnosing and enhancing vae models. *arXiv preprint arXiv:1903.05789*, 2019. 1

Gabriel Eilertsen, Daniel Jönsson, Timo Ropinski, Jonas Unger, and Anders Ynnerman. Classifying the classifier: dissecting the weight space of neural networks. In *European Conference on Artificial Intelligence*, 2020. 1, 14

Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12873–12883, 2021. 1

Damian Falk, Konstantin Schürholt, Konstantinos Tzevelekakis, Léo Meynent, and Damian Borth. Learning Model Representations Using Publicly Available Model Hubs. *arXiv preprint arXiv:2510.02096*, 2025. 3, 9, 14

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020. 5

Saumya Gupta, Scott Biggs, Moritz Laber, Zohair Shafi, Robin Walters, and Ayan Paul. Deepweightflow: Re-basined flow matching for generating neural network weights. In *The Fourteenth International Conference on Learning Representations*, 2026. URL `https://openreview.net/forum?id=fOwsr1VTi8`. 1

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International Conference on Machine Learning*, pp. 1842–1850. PMLR, 2018. 2

Xiaolong Han, Zehong Wang, Bo Zhao, Binchi Zhang, Jundong Li, Damian Borth, Rose Yu, Haggai Maron, Yanfang Ye, Lu Yin, et al. A survey of weight space learning: Understanding, representation, and generation. *arXiv preprint arXiv:2603.10090*, 2026. 1

Joëlle Hanna, Damian Falk, Stella X. Yu, and Damian Borth. Geosane: Learning geospatial representations from models, not data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 27804–27814, June 2026. 9

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016. 6

Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. EuroSAT: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7): 2217–2226, 2019. doi: 10.1109/JSTARS.2019.2918242. 17

Jinhyeok Jang, Woo-han Yun, Won Hwa Kim, Youngwoo Yoon, Jaehong Kim, Jaeyeon Lee, and ByungOk Han. Learning to boost training by periodic nowcasting near future weights. In *International Conference on Machine Learning*, pp. 14730–14757. PMLR, 2023. 1, 2, 3, 5, 6, 18

Wonyong Jeong, Hayeon Lee, Geon Park, Eunyoung Hyung, Jinheon Baek, and Sung Ju Hwang. Task-adaptive neural network search with meta-contrastive learning. *Advances in Neural Information Processing Systems*, 34:21310–21324, 2021. 17

Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL `https://kellerjordan.github.io/posts/muon/`. 2

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2

Boris Knyazev, Abhinav Moudgil, Guillaume Lajoie, Eugene Belilovsky, and Simon Lacoste-Julien. Accelerating training with neuron interaction and nowcasting networks. In *International Conference on Learning Representations*, volume 2025, pp. 29569–29588, 2025. 2, 3, 6, 18, 19

Miltiadis Miltos Kofinas, Boris Knyazev, Yan Zhang, Yunlu Chen, Gertjan J Burghouts, Efstratios Gavves, Cees G Snoek, and David Zhang. Graph neural networks for learning equivariant representations of neural networks. In *International Conference on Learning Representations*, 2024. 14

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 17

Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. 2015. URL `https://api.semanticscholar.org/CorpusID:16664790`. 17

Derek Lim, Haggai Maron, Marc T. Law, Jonathan Lorraine, and James Lucas. Graph metanetworks for processing diverse neural architectures. In *International Conference on Learning Representations*, 2024. 14

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=Bkg6RiCqY7`. 2

Luke Metz, James Harrison, C Daniel Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, and Jascha Sohl-Dickstein. Velo: Training versatile learned optimizers by scaling up. *arXiv preprint arXiv:2211.09760*, 2022. 2

Abhinav Moudgil, Boris Knyazev, Guillaume Lajoie, and Eugene Belilovsky. Celo: Training versatile learned optimizers on a compute diet. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL `https://openreview.net/forum?id=SLqJbt4emY`. 2

Aviv Navon, Aviv Shamsian, Idan Achituve, Ethan Fetaya, Gal Chechik, and Haggai Maron. Equivariant architectures for learning in deep weight spaces. In *International Conference on Machine Learning*, pp. 25790–25816. PMLR, 2023. 14

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, pp. 4. Granada, 2011. 17

William Peebles, Ilija Radosavovic, Tim Brooks, Alexei A Efros, and Jitendra Malik. Learning to learn with generative models of neural network checkpoints. *arXiv preprint arXiv:2209.12892*, 2022. 14

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 6

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *2022 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pp. 10674–10685. ieee, 2022. 1

Konstantin Schürholt, Dimche Kostadinov, and Damian Borth. Self-supervised representation learning on neural network weights for model characteristic prediction. In *Advances in Neural Information Processing Systems*, volume 34, pp. 16481–16493, 2021. 1, 14

Konstantin Schürholt, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Hyper-representations as generative models: Sampling unseen neural network weights. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27906–27920, 2022. 14

Konstantin Schürholt, Michael W Mahoney, and Damian Borth. Towards scalable and versatile weight space learning. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *PMLR*, pp. 43947–43966, 2024. 1, 3, 14, 18

Xiao Shou, Debarun Bhattacharjya, Yanna Ding, Chen Zhao, Rui Li, and Jianxi Gao. Less is more: Efficient weight "farcasting" with 1-layer neural network. In *International Conference on Database Systems for Advanced Applications*, pp. 3–18. Springer, 2025. 3

Sidak Pal Singh, Bobby He, Thomas Hofmann, and Bernhard Schölkopf. The directionality of optimization trajectories in neural networks. In *International Conference on Learning Representations*, volume 2025, pp. 21802–21862, 2025. 2, 6

Abhishek Sinha, Mausoom Sarkar, Aahitagni Mukherjee, and Balaji Krishnamurthy. Introspection: Accelerating neural network training by learning weight evolution. *arXiv preprint arXiv:1704.04959*, 2017. 2

Bedionita Soro, Bruno Andreis, Hayeon Lee, Wonyong Jeong, Song Chong, Frank Hutter, and Sung Ju Hwang. Diffusion-based neural network weights generation. In *International Conference on Learning Representations*, 2025. 1, 14

Bedionita Soro, Aoxuan Zhang, Bruno Andreis, Jaehyeong Jo, Song Chong, and Sung Ju Hwang. Ls-merge: Merging language models in latent space. In *International Conference on Learning Representations*, volume 2026, pp. 147812–147833, 2026. 1, 14

Fabio A. Spanhol, Luiz S. Oliveira, Caroline Petitjean, and Laurent Heutte. A dataset for breast cancer histopathological image classification. *IEEE Transactions on Biomedical Engineering*, 63(7):1455–1462, 2016. doi: 10.1109/TBME.2015.2496264. 17

Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya Tolstikhin. Predicting neural network accuracy from weights. *arXiv preprint arXiv:2002.11448*, 2020. 1, 14

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017. 1

Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. Soap: Improving and stabilizing shampoo using adam for language modeling. In *International Conference on Learning Representations*, volume 2025, pp. 93423–93444, 2025. 2

Kai Wang, Dongwen Tang, Boya Zeng, Yida Yin, Zhaopan Xu, Yukun Zhou, Zelin Zang, Trevor Darrell, Zhuang Liu, and Yang You. Neural network diffusion. *arXiv preprint arXiv:2402.13144*, 2024. 14

Kai Wang, Dongwen Tang, Wangbo Zhao, Konstantin Schürholt, Zhangyang Wang, and Yang You. Scaling up parameter generation: A recurrent diffusion approach. In *Advances in Neural Information Processing Systems*, volume 38, pp. 103214–103246, 2025. 14

Olga Wichrowska, Naitram Dahl, Sean Huval, Justin Gilmer, Samuel S Schoenholz, Surya Raghu, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 3730–3739. JMLR.org, 2017. 2

Allan Zhou, Kaien Yang, Kaylee Burns, Adriano Cardace, Yiding Jiang, Samuel Sokota, J Zico Kolter, and Chelsea Finn. Permutation equivariant neural functionals. *Advances in neural information processing systems*, 36:24966–24992, 2023. 14

A Additional related works

Weight-space learning. Weight-space learning treats neural network weights as a data modality from which representations can be learned. Early work focused primarily on predicting model properties from weights (Unterthiner et al., 2020; Eilertsen et al., 2020; Schürholt et al., 2021; Lim et al., 2024; Kofinas et al., 2024), whilst later approaches explored the permutation symmetries of weights and developed permutation-equivariant architectures that explicitly account for them (Zhou et al., 2023; Navon et al., 2023). Schürholt et al. (2022) learned latent representations of neural network weights using autoencoders, supporting both discriminative and generative tasks, with Schürholt et al. (2024); Falk et al. (2025) scaling these representations to larger models. More recent work has used these representations for model generation (Wang et al., 2024; Soro et al., 2025; Peebles et al., 2022; Wang et al., 2025) and for performing operations such as model merging directly in latent space (Soro et al., 2026). Our work builds on this line of research but focuses on the temporal structure of training, explicitly shaping a latent space of weights to improve forecasting performance of optimization trajectories.

B Additional Results

B.1 Full ID per dataset breakdown

In Table 5 and Table 6, we present the per dataset speedups for the averaged results in Table 1, along with the wall clock speedups in parentheses. The latent forecasters outperform both weight space baselines on almost all datasets for the CNN, with SVHN being the only exception, and indeed all of them for the ResNet datasets.

Table 5: Optimizer-step speedup (%) on held-out CNN trajectories for image datasets used during training. Wall-clock speedup is shown in parentheses.

Dataset	WNN	NiNo	Latent MLP	Latent Transf.
CIFAR-10	20.2 (20.1)	28.5 (28.2)	37.2 (36.8)	**38.6** (38.2)
SVHN	17.0 (17.1)	**29.2** (29.0)	21.8 (21.6)	22.6 (22.1)
notMNIST	11.6 (11.7)	20.4 (20.3)	29.2 (29.0)	**29.7** (29.2)
ASL Alphabet	7.7 (8.0)	15.6 (15.7)	23.0 (22.2)	**26.4** (26.1)
EuroSAT	11.1 (11.2)	18.2 (17.6)	25.2 (25.1)	**29.7** (29.0)
NumtaDB	11.4 (11.9)	22.9 (22.7)	31.7 (31.2)	**35.6** (34.9)
Mean	13.2 (13.1)	22.5 (22.2)	28.0 (27.6)	**30.5** (29.9)

Table 6: Optimizer-step speedup (%) on held-out ResNet trajectories. Wall-clock speedup is shown in parentheses.

Dataset	WNN	NiNo	Latent MLP	Latent Transf.
CIFAR-10	3.3 (3.3)	13.6 (13.4)	14.6 (13.5)	**15.6** (14.3)
CIFAR-100	−5.6 (−5.6)	4.7 (4.5)	34.9 (33.9)	**38.9** (37.6)
Food-101	−1.5 (−1.5)	5.7 (5.6)	34.9 (33.9)	**36.2** (35.0)
Tiny-ImageNet	−0.1 (−0.1)	4.8 (4.6)	21.6 (20.7)	**24.9** (23.9)
Mean	−0.9 (−1.0)	7.2 (7.0)	26.5 (25.5)	**28.9** (27.7)

B.2 Generalization Across Optimization Settings

Our main experiments keep the optimization setting fixed to isolate the effect of the representation space on forecasting. Here, we instead vary the optimizer and learning rate to test whether the resulting speedup is specific to the optimization setting seen during training.

Table 7: Step speedup (%) across different optimization settings. Target perplexity is shown in parentheses.

Optimizer	Learning Rate					
	2×10^{-4}	4×10^{-4}	6×10^{-4}	8×10^{-4}	2×10^{-3}	6×10^{-3}
AdamW	**59.2** (504.8)	51.3 (408.7)	41.3 (381.8)	33.0 (368.9)	9.3 (350.3)	7.0 (348.3)
Muon	**47.3** (576.8)	34.3 (460.3)	34.0 (422.5)	33.3 (403.5)	27.0 (356.8)	11.7 (351.2)

Table 7 reports forecasting speedup for GPT-2 trajectories produced by AdamW and Muon across several learning rates. For each setting, the target performance is defined by its corresponding standard training run at 5k steps and is reported as perplexity in parentheses. For Muon, we vary only the learning rate of the auxiliary Adam optimizer, while keeping the Muon-specific hyperparameters fixed to the configuration of the first result on Track 3 Optimization run from the modded-nanoGPT benchmark[1]. Forecasting provides positive speedup across all evaluated settings, demonstrating that the approach is not restricted to a particular optimizer or learning-rate configuration and can be applied across different training trajectories.

B.3 Forecast-loss weight

Table 8 studies the effect of our Stage II forecasting objective. We present the average speedup for the vision tasks on the OOD datasets and ID on the language task. Starting from the Stage I autoencoder, which is only supervised with the $\mathcal{L}_{\text{Stage I}} = \mathcal{L}_{\text{rec}} + \beta\mathcal{L}_{\text{KL}}$ and thus designated as $\lambda = 0$ in the table, we train multiple Stage II autoencoders with increasing λ and record the speedup for each when using the Latent MLP forecaster. We can see that already at Stage I the representations support forecasting, however, in a limited capacity similar to the weight space approaches seen in Table 1. A major increase in performance is seen when introducing the forecasting supervision already at $\lambda = 0.01$, and increasing the weight generally leads to larger speedups as well, clearly demonstrating that our forecasting supervision has the intended effect on shaping our latent space for forecasting. As the forecasting supervision trades off against the reconstruction loss in Stage II, we can expect there to be an optimal point where increasing the λ further does not yield higher speedups.

Table 8: Effect of the forecast-loss weight (λ) on speedup (%).

λ	Speedup (%)		
	ResNet	**CNN**	**GPT-2**
0	8.3	21.5	24.5
0.01	15.2	27.4	–
0.03	–	31.6	41.9
0.04	23.7	31.5	–
0.06	**27.3**	34.6	45.4
0.07	–	**36.5**	–

[1] https://github.com/KellerJordan/modded-nanogpt/tree/master/records/track_3_optimization#notable-results-history

B.4 Pretrained encoder

Our main experiments train the weight-space autoencoder from scratch in Stage I before shaping the resulting representation for forecasting in Stage II. Here, we additionally test whether the same procedure can build on a pretrained weight autoencoder that has not been trained on our dense trajectories, but has been exposed to a more diverse collection of models. Specifically, we initialize the autoencoder from an extended WeightCLIP (Asefaw et al., 2026) pretraining setup for CNNs covering 86 datasets, compared with the 6 datasets used in this work and the 20 datasets used in the original WeightCLIP setup. We then compare this pretrained representation against the weight autoencoder learned from scratch in Stage I.

The pretrained autoencoder already improves forecasting performance over the Stage-I representation learned from scratch, suggesting that a representation learned from a broader and more diverse model collection can transfer to the forecasting task even without having been trained on densely sampled trajectories. Applying Stage II forecast shaping on top of this pretrained representation yields a further improvement, showing that generic weight-space pretraining and forecasting supervision capture complementary structure. These results indicate that our approach does not require the representation to be learned exclusively from the forecasting trajectories and can instead refine an existing pretrained weight representation towards one better suited for forecasting.

Table 9: Effect of initializing from a pretrained WeightCLIP autoencoder and subsequently applying forecast shaping on optimizer-step speedup (%).

Representation	**Speedup**
Stage-I from scratch	23.5
WeightCLIP pretrained	25.5
WeightCLIP + forecast shaping	29.4

C Implementation Details

C.1 Datasets

We summarize the datasets used to construct the model zoos described in Section 4.1 in Tab. 10. The datasets are drawn from the TANS Jeong et al. (2021) meta-dataset collection, but restricted to 6 training datasets for the CNN zoo and 4 for the ResNet zoo. As in TANS, we cap datasets to at most 20 classes during zoo construction. When available, we cite the original dataset papers for the Kaggle sources, including BreakHis (Spanhol et al., 2016), NumtaDB (Alam et al., 2018), EuroSAT (Helber et al., 2019), SVHN (Netzer et al., 2011), Food-101 (Bossard et al., 2014), Tiny-ImageNet (Le & Yang, 2015), CIFAR-100 (Krizhevsky et al., 2009) and CIFAR-10 (Krizhevsky et al., 2009).

Table 10: Datasets used for model-zoo construction and OOD evaluation. For vision datasets, "Instances" denotes train/validation counts.

(a) Training datasets used to construct the CNN model zoos.

No.	Dataset	Instances	Cls.
1	ASL Alphabet	36,000 / 4,500	15
2	notMNIST	381,052 / 47,630	10
3	Bengali Digits	57,620 / 7,203	10
4	EuroSAT	21,600 / 2,700	10
5	CIFAR-10	40,000 / 10,000	10
6	SVHN	58,605 / 14,651	10

(b) Training datasets used to construct the ResNet model zoos.

No.	Dataset	Instances	Cls.
1	CIFAR-10	40,000 / 10,000	10
2	CIFAR-100	40,000 / 10,000	100
3	Food-101	60,600 / 15,150	101
4	Tiny-ImageNet	100,000 / 10,000	200

(c) Training dataset used to construct the GPT-2 model zoo. Token counts are reported as train/validation using the GPT-2 tokenizer.

No.	Dataset	Tokens
1	LM1B	804.7M / 42.4M

(d) Vision datasets used for OOD evaluation.

No.	Dataset	Instances	Cls.
1	Lego Bricks	7,319 / 915	20
2	Breast Histology	6,323 / 789	8
3	Simpsons	9,989 / 1,250	20

(e) Language datasets used for OOD evaluation. Token counts are reported as train/validation using the GPT-2 tokenizer.

No.	Dataset	Tokens
1	OpenWebText	223.5M / 55.8M
2	Shakespeare	269.3k / 67.6k
3	WikiText-103	111.2M / 5.8M

C.2 Model zoos

Each training dataset contributes 500 or 200 independently trained models (different random seeds), yielding 3,000 CNN trajectories, 800 ResNet18 trajectories and 200 GPT-2 trajectories in total. Tab. 11 summarizes the training configuration for the zoo models.

Table 11: Model zoo training hyperparameters. Every model in a zoo is trained with the same recipe; indvidual zoos differ only in the random seed.

Hyperparameter	CNN	ResNet18	GPT-2
Optimizer	Adam	Adam	AdamW
Initialization	Kaiming uniform	Kaiming uniform	Normal (σ=0.02)
Act. Function	GELU	ReLU	GELU
Learning Rate	sweep	sweep	2×10^{-4}
Scheduler	constant	constant	constant
Weight Decay	0	0	10^{-2}
Dropout	0	0	0.1
Batch Size	128	128	32×1024 tokens
Training Steps	10,000	25,000	25,000
Checkpoint Interval	8 steps	200 steps	200 steps
Models / Dataset	500	200	200

C.3 Architectures

The CNN is designed for 32×32 RGB images. It consists of three convolutional blocks followed by two fully connected layers. GELU activations are used throughout to improve robustness across heterogeneous datasets. In total, the network has approximately 11K parameters.

The ResNet-18 models are a modified version using a channel width multiplier of 0.5. Thus, channel dimensions are reduced to $\{32, 64, 128, 256\}$, resulting in 2.8M parameters. This choice was made because of compute constraints.

The GPT-2 models are a Transformer with 3 Transformer blocks and a hidden dimension of 24. Each block follows the standard GPT-2 architecture with causal self-attention, an MLP, residual connections, and layer normalization. The vocabulary contains 50,257 tokens, resulting in approximately 1.25M parameters, the majority of which are contained in the token-embedding matrix.

C.4 Weight space autoencoder

Following SANE (Schürholt et al., 2024), we represent each network as a sequence of tokens obtained from the flattened parameter vector: every tensor is split row-wise into fixed-width tokens and zero-padded to the token size. Tokens are grouped into windows and processed by a transformer encoder–decoder. For the CNN and ResNet18 experiment, the autoencoder is variational. For GPT-2, we use a deterministic autoencoder with a group-balanced MSE reconstruction loss, which weights each parameter group (attention projections, MLP, layer norms, embeddings) so that the token embedding does not dominate the objective. Tab. 12 reports the exact configuration and training length for all three. Additional training parameters are provided in the released code.

C.5 WNN

We implement WNN (Jang et al., 2023) with the per-parameter MLP of NiNo's WNN+ implementation (Knyazev et al., 2025). Each parameter is forecast independently from its last five values, at checkpoints 200 steps apart; a linear layer maps this history to 128 dimensions and a four-layer MLP of width 128 with SiLU activations outputs the normalised residual to the forecast horizon. To stay close to the original WNN, we remove NiNo's direct

Table 12: Autoencoder configurations and training epochs.

Parameter	CNN	ResNet18	GPT-2
Token Size	289	576	576
Latent Dimension	182	384	447
Window size	41	512	2242
d_{model}	1024	1024	1024
Layers	4	8	4
Attention Heads	8	8	8
Dropout	0.10	0.10	0.10
Objective	recon + KL	recon + KL	group-balanced MSE
Training Epochs (stage 1 + 2)	70 + 90	80 + 50	55 + 50

multi-step (DMS) component and standardize inputs using a single mean and standard deviation over all parameters of the context window, rather than layer-wise.

The forecaster is trained by sampling random trajectories from the corresponding training zoo and minimizing an ℓ_1 loss on the normalized residual of randomly sampled parameters. We use AdamW with a learning rate of 3×10^{-3}, weight decay of 10^{-2}, cosine learning-rate decay, and gradient clipping at 1.0, and retain the checkpoint with the lowest validation loss. This is also the recipe and budget of the Latent MLP, which is the same forecaster but applied to latent coordinates.

C.6 NiNo

We use NiNo's graph neural network and training code (Knyazev et al., 2025), retrained separately on our CNN, ResNet, and GPT-2 model zoos. NiNo represents each neural network as a graph derived from its computational structure and pairs the parameter histories from the previous five checkpoints with the corresponding graph elements. Parameters are normalized using NiNo's layer-wise scaling, and the model jointly predicts the set of forecast horizons used in the original implementation. For GPT-2, we additionally use NiNo's learned identifiers for rows of the token-embedding matrix and 8 Laplacian eigenvectors as positional encodings.

Across the architecture families, we train it on the corresponding training trajectories with an ℓ_1 loss between the predicted and target future parameter features over the supervised graph edges. We train it for 20,000 steps with AdamW (learning rate of 3×10^{-3}, weight decay of 10^{-2}), cosine decay, batch size 4 and gradient clipping at 5. In the main experiments, NiNo is evaluated under the same single-jump protocol as the other forecasting methods. For periodic forecasting we follow NiNo's schedule: a jump every 1,000 steps, with k decaying from 40 to 1 as $k_i = \text{round}(\ell_i^2)$, ℓ_i linearly spaced from $\sqrt{40}$ to 1, and the optimiser state kept across jumps.

C.7 Latent transformer

The Latent Transformer uses the same inputs, normalization, forecast horizons, objective, and training procedure as the Latent MLP, but allows the coordinates within each latent token to interact. While the MLP forecasts each of a token's d coordinates independently, the Transformer represents these coordinates as a sequence of d sub-tokens and applies self-attention across them. Each sub-token is formed by projecting its five-step history to an embedding of dimension $d = 128$, to which we add a learned coordinate-position embedding and the shared type embedding. The resulting sequence is processed by a two-layer pre-norm Transformer encoder with four attention heads, a feed-forward dimension of 256, GELU activations, and no dropout. A linear output head maps each sub-token to its 40 forecast-horizon residuals, producing an output with the same structure as the Latent MLP. Attention is restricted to coordinates within a single latent token, while different tokens are forecast independently.