On Neural Scaling Laws for Weather Emulation through Continual Training

Shashank Subramanian
Lawrence Berkeley National Laboratory
shashanksubramanian@lbl.gov

Alexander Kiefer
Oak Ridge National Laboratory
kiefera@ornl.gov

Arnur Nigmetov
Lawrence Berkeley National Laboratory
anigmetov@lbl.gov

Amir Gholami
ICSI and UC Berkeley
amirgh@berkeley.edu

Dmitriy Morozov
Lawrence Berkeley National Laboratory
dmorozov@lbl.gov

Michael W. Mahoney
LBNL, ICSI, and UC Berkeley
mmahoney@stat.berkeley.edu

Abstract

Neural scaling laws, which in some domains can predict the performance of large neural networks as a function of model, data, and compute scale, are the cornerstone of building foundation models in Natural Language Processing and Computer Vision. We study neural scaling in Scientific Machine Learning, focusing on models for weather forecasting. To analyze scaling behavior in as simple a setting as possible, we adopt a minimal, scalable, general-purpose `Swin` Transformer architecture, and we use continual training with constant learning rates and periodic cooldowns as an efficient training strategy. We show that models trained in this minimalist way follow predictable scaling trends and can outperform standard cosine learning rate schedules. Cooldown phases can be repurposed to improve downstream performance, e.g., enabling accurate multi-step rollouts over longer forecast horizons or sharper predictions through spectral loss adjustments. We also systematically explore a wide range of model and dataset sizes under various compute budgets to construct IsoFLOP curves, and we identify compute-optimal training regimes. Extrapolating these trends to larger scales highlights potential performance limits, demonstrating that neural scaling can serve as an important diagnostic for efficient resource allocation. We open-source our code for reproducibility.

1 Introduction

Machine learning models, and deep learning models in particular, have demonstrated great success in scientific problems, including emulating atmospheric physics for weather forecasting. In recent years, several data-driven models [28, 11, 9, 7, 6, 35, 3] have matched or surpassed the accuracy of gold-standard Numerical Weather Prediction (NWP) systems that generate high-fidelity forecasts by solving complex, multiscale fluid-dynamics equations. These data-driven models can generate weather forecasts orders of magnitude faster than classical NWP models using substantially lower resources, often only a single GPU, in inference. The training costs of these deep learning weather emulators are rapidly increasing, as researchers explore a wide range of architectural choices, loss functions, and training methodologies in pursuit of state-of-the-art forecasting performance. These costs are further amplified by the growing scale of models, reaching $O(100)$ billion parameters, as well as increasing data resolutions, e.g., through smaller patch sizes in Transformer-based architectures

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

[17, 41, 42]. Similar scaling challenges arise in areas such as Natural Language Processing (NLP) and Computer Vision (CV). However, scientific data present additional unique challenges, perhaps most notably their spatiotemporal structure. The proliferation of increasingly-complex architectures, in particular in Scientific Machine Learning (SciML), raises important questions: Are we learning about science or merely about the idiosyncrasies of ever more elaborate (domain-motivated) neural architecture designs? What is the simplest possible architecture to obtain scaling that is good for downstream SciML applications?

These questions are timely since, in many areas of machine learning, progress has ultimately been driven not by increased architecture complexity, but by scale—more data, larger models, and more compute—a phenomenon sometimes referred to as the *bitter lesson*. Understanding how performance scales with these quantities therefore requires studying simple, general-purpose models under controlled scaling regimes. Without such a foundation, it becomes difficult to disentangle genuine scaling behavior from artifacts introduced by specialized architectural choices.

In domains such as NLP, this challenge has been addressed through the development of *neural scaling laws* [20, 23, 21], a set of empirical patterns that guide practitioners in scaling data quantity, model size, and the amount of compute, so that none of them saturate. When this happens, the model loss follows a power-law relationship as a function of the size of the model, of the dataset, and of the computational resources available for training. Within NLP, these trends can span more than seven orders of magnitude [21, 16]. The study of neural scaling laws is important because it builds confidence and trust that model performance will improve with increased scale. This predictive aspect is valuable because it makes neural scaling laws not only an analytical method but also actionable for planning efficient resource allocation.

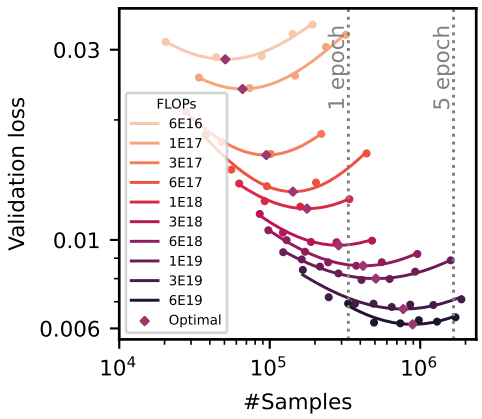


Figure 1: **Neural scaling for weather emulation.** We pre-train several models using continual training (constant learning rates with periodic cooldowns; see §3), and we identify compute-optimal regimes to train the neural emulator so that neither data nor model size saturate at different compute budgets. At each FLOP budget, several model sizes (up to 457M) are trained to different dataset sizes to form IsoFLOPs that demonstrate the tradeoff between model and data size. Unlike NLP, these models are trained for multiple epochs (indicated by vertical dotted lines), causing samples to be revisited after the first epoch and effectively be treated as pseudo-samples. We fit parabolas to each IsoFLOP, and we track the compute-optimal model at each budget.

In the existing literature, there has been limited effort in understanding the neural scaling behavior in SciML, in particular for deep learning models for weather forecasting. Existing studies do not explicitly investigate the joint relationship between model and data size, as a function of available computational budget, and/or their analyses remain incomplete. For example, Bodnar et al. [9] report predictable scaling with model size; but their study does not perform compute-optimal analyses, relies on scaling with GPU time rather than FLOPs, and incorporates data outside the traditional weather-forecasting domain, such as pollutants and sea state, which makes interpreting the scaling behavior harder. In contrast, Karlbauer et al. [24] suggest that performance stalls, for neural operators and related architectures, at even small model scales, although it should be noted that this conclusion is drawn from experiments at coarser spatial resolution than we consider. More broadly, prior works stop short of identifying compute-optimal combinations of model size and data volume at each scale, making it difficult to disentangle true scaling laws from artifacts of architectural, data, or compute choices.

In this work, we aim to systematically characterize the neural scaling relationships governing data-driven weather forecasting models. Our main contributions are:

1. **Minimalist transformer architecture for neural scaling.** Rather than designing specialized architectures for weather forecasting, we intentionally adopt a minimalist and widely

used backbone, the `Swin` Transformer [29], with minimal domain-specific architectural modifications or custom loss functions, during pre-training. This design choice follows the principle that understanding scaling behavior requires reducing architectural confounders. Our goal is therefore not to engineer the most specialized weather model, but to understand how far simple architectures can be pushed through scale. To enable such experiments across a wide range of scales, we further implement $2D$ spatial parallelism along with data parallelism, with the former being crucial for high-resolution inputs.

2. **Continual training for weather emulators.** Inspired by Hägele et al. [22], we apply learning rate (LR) cooldowns after a period of continual training with a constant LR. Here, "continual training" refers to the ability to continue optimizing a model across different compute budgets from a checkpoint without restarting from scratch, rather than the standard multi-task continual learning setting. We show that this fixed LR followed by a cooldown protocol can outperform the standard cosine schedule commonly used in pre-training weather emulators [9, 28] (see Fig. 3a). This approach enables efficient exploration of a wide range of compute budgets without retraining models from scratch, by periodically cooling down the LR to match the desired budget. We show that even a cooldown period as short as 5% of the total training iterations is sufficient to achieve consistent performance (see Fig. 3b).
3. **Repurposing cooldowns for better downstream performance.** We demonstrate that the short cooldown period can serve a dual purpose: completing the pre-training LR schedule and adapting the model to downstream objectives, similar to task-specific fine-tuning in weather modeling. For example, we can perform multi-step rollouts in training during cooldown to achieve higher accuracy over longer forecast horizons (see Fig. 3b, 4). We may spectrally adjust the loss function (Fig. 4b) to allow for sharper forecasts to capture critical high-resolution features. This allows us to disentangle pre-training scaling studies from downstream adaptation, reusing the same pre-trained checkpoints for different loss functions and rollout objectives without repeating expensive pre-training.
4. **Neural scaling to identify compute-optimal training regimes.** We construct scaling laws by pre-training several models (up to $457M$ parameters) using compute budgets from 6E16 to 6E19 FLOPs on the 0.25° ERA5 [19] dataset ($O(300,000)$ samples at an hourly temporal resolution). For each budget, different models are cooled down to different iterations that give rise to IsoFLOP curves (see Fig. 1, 5), a set of training configurations (model sizes and training iterations) that achieve a fixed total training FLOPs. Across these budgets, we observe clear compute-optimal scaling behavior, where each budget admits an optimal combination of model size and effective dataset size. To probe the limits of these trends, we extrapolate the scaling laws to 3E20 and 2.25E21 FLOPs and train 385M and 1.24B parameter models, respectively. Both runs show signs of saturation before reaching their projected losses. This suggests that scaling in this regime may become limited by data size and spatiotemporal resolution, highlighting the importance of neural scaling analyses for diagnosing progress.

2 Related Works

Neural weather emulators. Models such as FourCastNet [26] approached the accuracy of NWP in medium-range weather forecasting (10–14 days), while being orders of magnitude faster. Newer models [7, 28, 9, 11, 35] have surpassed the accuracy of NWP, while retaining the speedups. The architecture landscape of these models includes Neural Operators [26, 11], graph-based models [28, 35], and Transformer-based models [13, 32, 7, 43]. In addition to a variety of architectures, a variety of loss functions, including standard MSE, autoregressive rollout MSE [28, 12, 9], and spectral-based losses [38], have been employed, depending on the downstream objective. Despite this progress, with a focus on architectural design and accuracy, there has been relatively little effort in systematic analysis of how model design, data scale, and compute budget interact in weather emulators, which are central goals here.

Neural scaling laws in large ML models. Other domains, most notably NLP, have established the utility of neural scaling laws for guiding model development. Kaplan et al. [23] demonstrated that language model loss scales predictably as these axes increase, enabling researchers to estimate performance for larger models before training. Hoffmann et al. [21] then formalized compute-optimal training ("Chinchilla" scaling), showing how to better balance model parameters and training data. Recent efforts have refined these laws, extended them to new architectures, and explored

their theoretical foundations [16, 5, 8, 25, 2]. Finally, Hägele et al. [22] show that LR scheduler changes that allow for continual training make the cost and design of neural scaling experiments more manageable. Our methodology is inspired by this.

Neural scaling laws in large SciML models. There are also emerging efforts to apply scaling law concepts to SciML models, beyond NLP. For example, universal models of atomic systems have been developed with empirical scaling laws to guide capacity and data allocation across compute budgets, illustrating how scaling insights can transfer to scientific domains [44, 40]. Another example is Evo [31], a foundation model that predicts the function and structure of biological sequences and explores the design space through neural scaling laws. Comprehensive neural scaling studies remain scarce in weather forecasting, and existing analyses do not identify compute-optimal regimes and exhibit mixed findings [9, 24]. Today, large models (billions of parameters) [17, 41] have already been trained, emphasizing the need to understand this design space. To the best of our knowledge, this is the first work (alongside concurrent work by Yu et al. [45]) to systematically study neural scaling laws in weather forecasting. Yu et al. [45] scale up to 1E18 FLOPs for a few existing models in the literature with sub-epoch training, and they demonstrate how some models show flavors of compute optimal scaling, whilst others do not. The main advantage of our work is its emphasis on (i) continuous training with cooldowns for efficient neural scaling and downstream alignment, (ii) scaling to over an order of magnitude higher compute (6E19 FLOPs) via spatial parallelism to address memory constraints, (iii) demonstrating clearer compute-optimal regimes (even with multiple epochs) through a minimalist Transformer backbone that matches state-of-the-art performance under the WeatherBench metrics [36], and (iv) probing scaling limits by extending compute to O(2E21) FLOPs.

3 Overview of Methods

Neural scaling involves pre-training a series of models across a practical range of compute budgets. At each compute budget, several model sizes are trained to different numbers of data samples in order to remain on an IsoFLOP. The classical approach, based on the "Chinchilla" scaling laws [21], involves training models *from scratch* for each compute budget and model size. The need to train models from scratch is due to the reliance on the cosine LR scheduler. Hoffmann et al. [21] showed that the final performance of any pre-trained model is optimal when the cosine LR decay length matches the total training duration. This LR scheduler is also popular in other domains, including weather forecasting [26, 9]. Several frontier NLP models as well as other SciML models have explored neural scaling laws in this manner [16, 8].

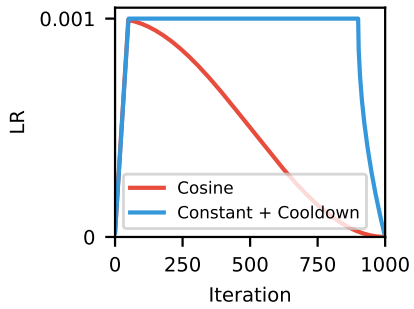


Figure 2: **LR schedules.** Cosine schedule follows a half-cosine decay after warmup; the constant+cooldown has a constant LR after warmup, but cools down rapidly to 0 at the end.

Scaling with periodic cooldowns. While the above approach is popular, it is expensive. It would be more beneficial to change the LR scheduler to a constant schedule, followed by a rapid cooldown phase to zero LR [22]. If one could show that the performance of this constant+cooldown LR scheduler matches that of the cosine decay scheduler, then this would allow for training the series of models *just once*, considerably reducing the computational expense of neural scaling. We record the loss at a given compute budget by applying a LR cooldown to zero in the final pre-training iterations. To reach larger budgets, we simply resume from the checkpoint prior to the cooldown, continue training with the original LR, and then cool down at the next target budget.

3.1 Weather forecasting problem formulation

Our goal is to model (emulate) global atmospheric dynamics using a learned neural network by predicting the temporal evolution of the state $\mathbf{u}(\mathbf{x}, t)$, with $\mathbf{x} \in \mathbb{R}^2$ and $t \in [0, \infty)$. Our training dataset consists of discrete snapshots $\mathbf{u}_n$ sampled at regular time steps with interval Δt with $\mathbf{u}_n \in$

$\mathbb{R}^{H\times W\times C}$. Here, $H \times W$ is the height and width of the discretized latitude-longitude projection $\mathbf{u}$; C is the number of state variables (as wind velocities, temperature, and others; see §A.2, with vertical dimension treated as channels). The temporal evolution is modeled as:

$$\mathbf{u}_{n+1} = \mathcal{F}_\theta(\mathbf{u}_n), \tag{1}$$

where $\mathcal{F}_\theta$ is a neural network parameterized with weights θ. We consider a standard mean-squared error (`MSE`) loss between the predicted $\mathbf{u}_{n+1}$ and target $\hat{\mathbf{u}}$ at the same time step to pre-train $\mathcal{F}_\theta$. Once trained, the model produces forecasts during inference for N steps, t_{n+i} with $i \in \{1, \cdots, N\}$, by autoregressively feeding each predicted state back as input to produce subsequent forecasts. Based on prior work, we consider two post-training strategies to better align the pre-trained model with the forecasting task:

1. **Adjusted** `MSE` **(**`AMSE`**)**: Weather datasets contain time steps (Δt) that are undersampled; they are not fine enough to sufficiently resolve fine-scale dynamics. With chaotic dynamics, this causes models trained with `MSE` to smooth high-frequency structure. To (partially) deal with this, we consider the adjusted `MSE` loss [38] (see §A.2), which disentangles decorrelation and spectral amplitude components of the error, enabling the model to retain high-resolution features.
2. **Autoregressive Rollouts (**`AR`**)**: Autoregressive inference introduces generalization error due to a distribution shift, as predicted states are used as inputs instead of samples from the training distribution [12]. To mitigate this, models are fine-tuned to autoregressively predict multiple time steps. While `AR` fine-tuning reduces error over longer horizons, it leads to smooth predictions due to the chaotic dynamics, causing forecasts to behave like ensemble-means [35].

Existing approaches typically rely on heuristic schedules with multiple tunable components, including the number of autoregressive steps, learning rate schedules, and fine-tuning duration, for both `AR` and `AMSE` strategies. Here, we repurpose the cooldown period to implement either strategy, aligning the model to the downstream task: maintaining high spectral resolution via `AMSE`, or reducing long-horizon forecast errors via `AR`. This approach allows downstream objectives to be explored through short cooldowns from a shared checkpoint trained with a constant learning rate, without needing to repeat pre-training for each choice.

3.2 Architecture and Training

We employ a standard architecture based on the Shifted Window (`Swin`) Transformer [29]. While many weather models introduce domain-specific modifications and customized components, we follow Willard et al. [43], who demonstrate that standard transformer backbones can be highly competitive when trained well. Our input $\mathbf{u}_n$ is projected to the latent space $\mathbf{X} \in \mathbb{R}^{H/p\times W/p\times E}$ using a standard patch embedding layer with patch size p and embedding dimensions E. We add a simple coordinate-based positional encoding using coordinates (x, y, z) (derived from spherical coordinates) and normalized time step $t \in \mathbb{R}$ for each pixel, projected to the latent space $\mathbf{PE} \in \mathbb{R}^{H/p\times W/p\times E}$ via an additional patch embedding layer. The positional encoding is added to the latent feature $\mathbf{X}$ before being passed to the `Swin` Transformer blocks. While alternative positional encoding strategies exist, we avoid learnable encodings, which would introduce a large number of parameters at high input resolutions (especially for small patch sizes p) and risk over-parameterizing this component relative to the rest of the model. We find this positional encoding to be sufficient in our experiments. For the `Swin` blocks, we remove components such as relative positional bias and other non-essential modifications, including the hierarchical `Swin` structure (e.g., patch merging and downsampling), and we retain a uniform Transformer block design consisting of windowed multi-head self-attention (W-MHSA) and a multi-layer perceptron (MLP) with pre-RMSNorm and residual connections:

$$\tilde{\mathbf{X}} = \text{RMSNorm}(\mathbf{X}), \quad \mathbf{Y} = \text{W-MHSA}(\tilde{\mathbf{X}}) + \mathbf{X},$$
$$\tilde{\mathbf{Y}} = \text{RMSNorm}(\mathbf{Y}), \quad \mathbf{O} = \text{MLP}(\tilde{\mathbf{Y}}) + \mathbf{Y},$$

where the W-MHSA operation partitions the latent patch grid into 2D windows and computes standard dot-product self-attention locally. We use QK-normalization for training stability [18]. Every alternate block shifts the windows to enable cross-window interactions, implemented as a rolling operation with attention masks. We modify the mask to preserve the left-right periodicity of the Earth system. The final layer is a reverse patch embedding layer.

Distributed training for neural scaling. As model, data, and input sizes increase, standard data parallelism becomes insufficient, with per-GPU batch sizes dropping to one. While NLP models address this with model parallelism, SciML domains such as weather face different challenges. High-resolution inputs and small patch sizes make these models heavily constrained by intermediate activation memory rather than weights, a limitation that is amplified during autoregressive rollout (`AR`) in training. Standard techniques such as Fully-Sharded Data Parallelism (`FSDP`) do little to alleviate this memory pressure. Moreover, I/O pressure increases due to high-resolution data. It is essential to build efficient distributed infrastructure to train across a wide range of scales. Towards this end, we implement spatial parallelism through domain decomposition, along with data parallelism. We employ a 3D mesh of GPUs $n_1 \times n_2 \times n_d$, where the last axis handles data parallelism and the first two partition the input $H \times W$. Since activation memory scales as $O(BHWE)$, with batch size B and embedding dimension E, this partitioning effectively reduces this memory to $O(BHWE/n_d n_1 n_2)$. The shifting windows in `Swin` require additional communication that we implement with a custom distributed roll in the forward and backward pass (for details, see §A.3).

3.3 Empirical Analysis Setup

Data. Our models are trained on the ERA5 [19] dataset that consists of around $350,000$ samples of the Earth's atmospheric state from 1979 to 2022 at hourly time steps and spatial resolution of $0.25°$. While ERA5 consists of hundreds of state (physical) variables, we use a 71 variable subset following Willard et al. [43] (see §A.2). When projected on a latitude-longitude grid, each sample is a $71 \times 721 \times 1440$ tensor with 721×1440 representing the spatial grid. We use the years 1979 to 2016 as training data, 2017 as validation, and 2020 for testing. We train our models to predict a time step of 6 hours.

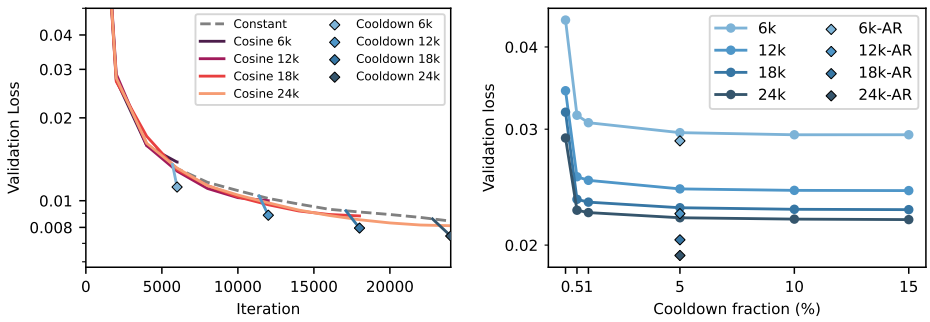
Loss and Metrics. Our pre-training loss is `MSE`. We avoid domain-specific weighting strategies that are variable dependent in the loss to maintain a simple objective for the neural scaling analysis. Our inputs are normalized with the per-variable mean and standard deviation derived from the training dataset. For validating our results, we use the area-weighted Root Mean-Squared Error (RMSE) and Power Spectral Density (PSD) prediction and target for each variable. We track each metric as a function of lead time up to 10 days, corresponding to the medium-range forecasting regime, and average the results over multiple initial conditions sampled across the evaluation period. We follow this evaluation protocol on initial conditions sampled regularly for the testing year 2020 (every 12 hours, 732 initial conditions) following the WeatherBench protocol [36]. We also retrieve the benchmark model predictions from Google Cloud Storage (using `gcsfs`), following Rasp et al. [36]. While RMSE provides a measure of average error over the globe, the PSD is commonly used to assess the effective resolution of the predictions and to quantify how well high-frequency, fine-scale features are captured. See §A.2 for the definitions.

Training. We train all our models with a batch size of 16. Through tuning experiments, we select a peak LR of 5E-4 for models larger than $100M$ parameters. Similar to NLP [23], we find smaller models can tolerate higher peak LRs, with the optimal LR approximately inversely proportional to the square root of the parameter count. We train our models with the AdamW optimizer with a small weight decay of 1E-4 and gradient norms clipped at 1. We find that this, with QK-normalization, is useful for training stability at the higher parameter counts. We use the constant+cooldown LR scheduler of the scaling experiments and cosine for initial comparisons. Both schedulers use 500 warmup iterations. We use mixed precision with `float16` for training. See Tab. A1 for all hyperparameter choices.

4 Main Results

We demonstrate the following results: **(R1)** Models can be continuously trained with periodic cooldowns using simple Transformer backbones; **(R2)** Cooldown periods can be repurposed with alternate losses to align to a weather-specific downstream task; and **(R3)** Compute-optimal regimes, through systematic neural scaling using cooldowns, can be identified across multiple compute budgets.

(R1) Cosine vs. Cooldowns. We first train a moderately sized $115M$ parameter `Swin` (see architecture hyperparameters in Tab. A1) from scratch multiple times using the cosine LR schedule, each run to a different number of iterations (batch size 16, corresponding to 20,819 iterations per epoch). We use the standard MSE loss with one-step prediction (predict the state $\Delta t = 6$ hours ahead). We



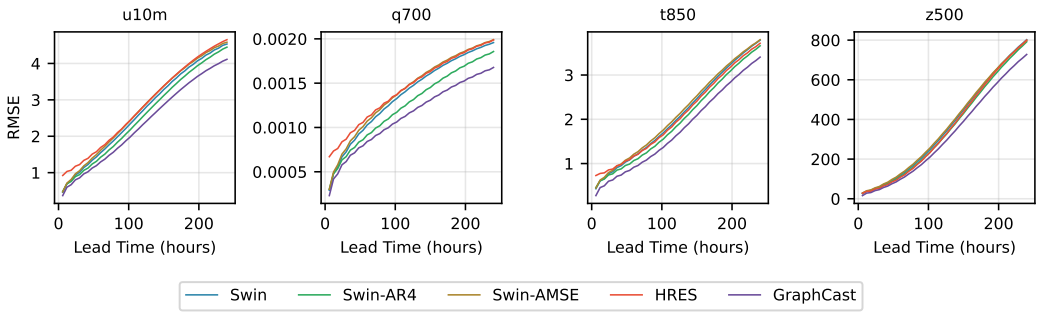
(a) Cosine vs. Cooldowns

(b) Cooldown fraction and alignment

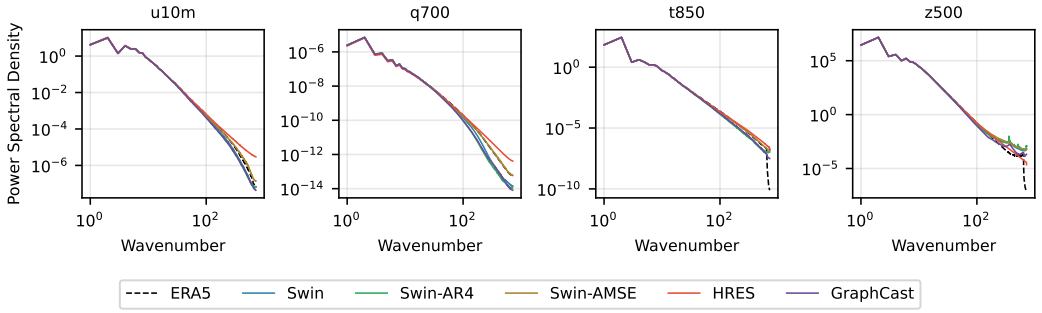
Figure 3: **(left) Cooldowns are more performant than cosine.** The validation loss of the model continuously trained with a constant LR and cooled down to different iteration counts (5% cooldown) shows better losses compared to the `Swin` trained from scratch with different cosine schedulers that match the total iteration count. **(right) Short cooldowns are sufficient and can help alignment.** We show the validation loss over 36 hours (6 `AR` steps) of prediction averaged over the validation data as a function of the fraction of total iterations used in cooldown. The loss decreases with longer cooldowns and this is true over multistep predictions. At $\sim$5%, the gains plateau. When the cooldown is repurposed with 4-step `AR`, we see lower errors across the longer horizon.

then replace this setup with the constant+cooldown LR schedule, cooling down to the same final iteration counts as the cosine runs using only 5% of the total iterations. We observe that the models that have been cooled down consistently show lower validation loss compared to the cosine LR (see Fig. 3a for the validation loss curves for each run). Independent LR sweeps for both schedules confirm lower cooldown losses across the tested LRs and budgets (Tab. A2), supporting its use for efficient scaling studies. We show in Fig. A3, A4 that other architectures also show similar behavior. In order to use the cooldowns for neural scaling, we investigate what the optimal fraction of the total iterations should be used for the cooldown period. We repeat the cooldown experiments using different cooldown fractions (from 0% to 15%) for each final iteration count. We note that while the one-step prediction is important for training, the true utility of a neural emulator lies in its ability to produce reliable forecasts over longer time horizons using autoregressive inference, as described in §3. Hence, we perform autoregressive rollouts during validation over multiple steps to track the time-averaged loss. Importantly, a cooldown that improves one-step loss but does not translate to longer-horizon accuracy may reduce its practical benefits. In Fig. 3b, we track the validation loss over six steps (36 hours), and we find that the gains from cooldown translate to the downstream forecasting task as well. We observe that even a small cooldown fraction (0.5%) is able to quickly reduce the loss and also see that a cooldown of 5% is sufficient, beyond which the gains plateau. We use 5% cooldowns in all experiments henceforth.

(R2) Alignment with Cooldown. We demonstrate that the cooldown can be used to also align the model better with the downstream task of high-resolution forecasting for several time steps. Similar to task-specific fine-tuning, this adaptation uses alternative training objectives, but is integrated into the short LR cooldown. This is important because neural scaling can be conducted using a simple `MSE` objective on large models and data, while the model's behavior can be adapted during the cooldown using alternative objectives. As a result, the same pre-training scaling study can support multiple downstream adaptation objectives. We consider both autoregressive training (`AR`) and the `AMSE` loss for this purpose (as described in §3). While `AR` aligns the model for long-term forecasting skill (through smoother forecasts like an ensemble), `AMSE` aligns the models to produce high-resolution features. In Fig. 3b, we show that if the model is cooled down with 4 steps of `AR`, the validation loss systematically improves at all the iteration counts. In Fig. 4a, we show the RMSE of the `Swin` predictions as a function of lead time for several variables, compared to the NWP gold standard (HRES) [14] as well as the deterministic state-of-the-art emulator GraphCast [28]. We see that the `Swin` is performant (outperforming HRES) and can be systematically pushed to the GraphCast accuracy through `AR` (note that GraphCast was fine-tuned with 2 through 12-step `AR`). We can also see from the PSD metric in Fig. 4b that the RMSE gains occur through smoothing that emulates an ensemble prediction. We show later in Fig. 7 that when optimally scaled, `Swin` can match GraphCast.



(a) RMSE of different variables (labeled at the top) vs. lead time (in hours) for different models.



(b) PSD of different variables compared to PSD of the truth (ERA5) for different models at 24 hours lead time.

Figure 4: **Cooldowns can be used for alignment.** When evaluated on testing year, the `Swin` cooled down at 24000 iterations is able to surpass the NWP (HRES). When `AR` is used in cooldown (`Swin-AR4`), the RMSE drops further, consistent with the use of this loss. When `AMSE` is used (`Swin-AMSE`), the PSD retains high wavenumbers. This is easily seen in $q700$ where the `AMSE` spectrum closely matches that of ERA5, but other models blur significantly. `AR` contributes to more blurring in favor of reduced RMSE (visible in dissipation of power in high wavenumbers). We note that HRES models weather at a 0.1° resolution and hence shows higher resolution. While this `Swin` does not beat the state-of-the-art GraphCast, a larger `Swin` obtained through optimal scaling matches its performance (as demonstrated later in Fig. 7).

Similarly, using `AMSE` during cooldown, we see that RMSE remains largely unaffected; however, the PSD plots show that the model has more power in the higher wavenumbers that this loss was designed to capture. We emphasize that the goal here is not to achieve best possible performance, but rather to demonstrate that phases such as cooldowns can be repurposed to align model behavior, highlighting their flexibility without entangling them with the core scaling analysis.

(R3) Neural Scaling. We train a range of model sizes from $0.7M$ to $457M$ parameters (see all hyperparameters in Tab. A1), varying more than two orders of magnitude. Our patch-size, embedding-width, and depth ablations (Tab. A4) motivate using patch size 4. Width and depth are largely interchangeable away from extreme aspect ratios, with a preference for deeper models, possibly due to our minimalist architecture and higher compute budgets. This differs from the preference for wider models in the smaller compute range studied by Yu et al. [45]. We select three orders of magnitude of compute budgets from 6E16 to 6E19. For each budget, we estimate (see §A.3 for details on FLOP estimates) a range of model sizes and number of iterations to train such that each configuration lies on an IsoFLOP. Each model is trained only once and cooled down to the required iteration at each IsoFLOP. We note that several of our models train for more than one epoch. In Fig. 1, we show the IsoFLOP curves on the loss versus number of samples scaling plot, while Fig. 5 presents the corresponding scaling behavior of loss versus model size. For both plots, we fit parabolas to each IsoFLOP to estimate the compute-optimal dataset size and model capacity for a given budget (see §A.4 for sensitivity analyses).

We observe that the minimalist model *(i)* scales systematically to 6E19 FLOPs without saturating the loss and *(ii)* exhibits clear compute-optimal behavior where each budget affords an optimal

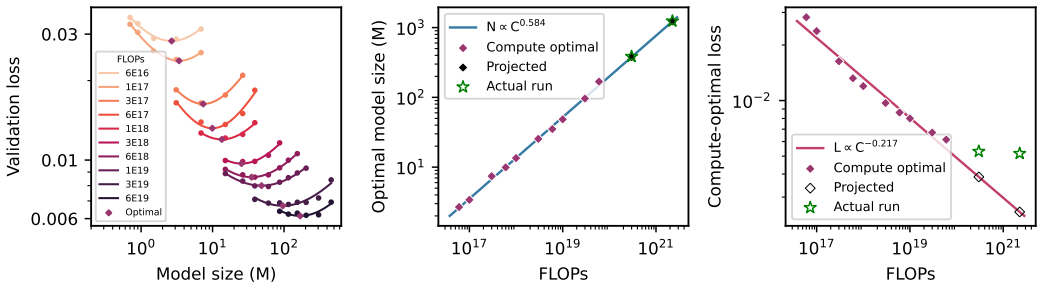


Figure 5: **Optimal model sizes as a function of compute.** (left) Similar to Fig. 1, we show the validation loss as a function of model size for different compute budgets. For each budget, the different model sizes are trained to a different number of iterations to create an IsoFLOP. We track the minima for each IsoFLOP (through a fitted parabola). (middle) We fit an empirical scaling law to find the optimal model size for any FLOP budget and project to two FLOP budgets: 3E20 and 2.25E21 FLOPs to find the optimal model. (right) We project the loss to the projected FLOP values—the measured loss starts to saturate.

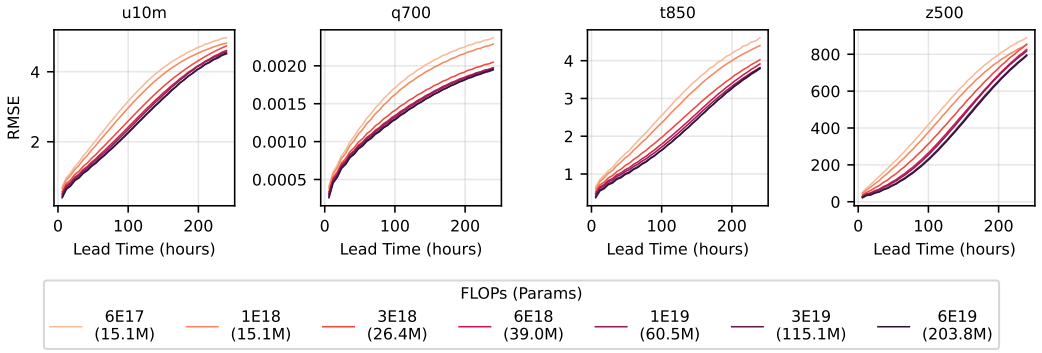


Figure 6: **RMSE performance of the closest compute-optimal `Swin` models at each compute budget.** We find that RMSE at extended forecast horizons, up to 240 hours, consistently improves with increased compute. However, at the highest compute levels, the rollout performance begins to exhibit signs of saturation.

model size and sample count (as opposed to other specialized architectures studied by Yu et al. [45]). Interestingly, this trend persists at several budgets where the model is trained for multiple epochs, demonstrating the value of data reuse. In Fig. 6 (and corresponding spectra in Fig. A7), we show that metrics maintain their gains throughout the rollout and across different compute budgets. In Fig. 7, we show that the model that is closest to compute-optimal at 6E19 FLOPs ($204M$) matches the state-of-the-art GraphCast (with other variables reported in Fig. A12). We estimate that GraphCast, while smaller ($37M$), trained longer and used close to O(1E21) FLOPs (see §A.3 for our estimation), with about 4 weeks of training [28] on 32 TPUv4 chips. In contrast, the $204M$ `Swin` trains with 32 NVIDIA A100s (comparable hardware generation) in about 6 hours (6E19 FLOPs) due to more efficient Transformer operations compared to (memory-bound) graph operations (and gradient checkpointing to save memory).

We then derive the following scaling: $S^\star(C) \propto C^{0.431}$ and $N^\star(C) \propto C^{0.584}$, where C is the budget, $S^\star$ is the optimal number of (pseudo) samples (with the understanding that multiple epochs are possible) and $N^\star$ is the optimal model size (see Fig. 5). We extrapolate to two FLOP budgets: 3E20 and 2.25E21 FLOPs ($5\times$ and $37.5\times$ the largest fitted budget of 6E19, respectively) to investigate how long the scaling holds. The empirical scaling suggests training the following two models: $379M$ and $1.23B$. We train a $385M$ model to $122K$ iterations (3E20 FLOPs) and $1.24B$ parameter model for $288K$ iterations (2.25E21 FLOPs). We observe in Fig. 5 that both these models (denoted by stars) saturate before hitting their projected loss values. The $385M$ and $1.24B$ runs correspond to approximately 6 and 14 epochs, respectively. The training and validation losses in Fig. A11

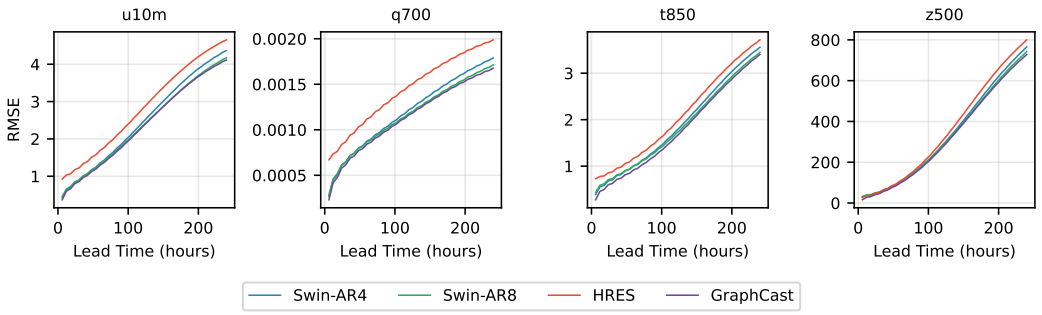


Figure 7: **RMSE performance of compute-optimal `Swin` vs baselines.** The 204M `Swin` (compute-optimal 6E19 FLOPs) outperforms NWP / HRES and is on par with GraphCast when evaluated over initial conditions in 2020. The model is cooled down with 4-step `AR`. With more steps (8-step `AR`), the performance gap reduces at longer lead times, as expected (GraphCast uses 2–12-step `AR`).

support overfitting from repeated exposure to finite data as the most likely explanation for this saturation. Stronger regularization reduces the train–validation gap of the $1.24B$ model, but it does not recover the extrapolated loss (Fig. A11), further supporting this interpretation. Thus, our scaling laws characterize a finite-data regime in which training repeatedly samples ERA5. Consistent with observations in data-constrained language models [30], data reuse remains useful over several epochs but offers diminishing returns. The weaker IsoFLOP fits at higher budgets and the deviations from the loss–compute trend in both extrapolation runs suggest limits to extending these empirical power laws beyond the fitted regime. Restricting the analysis to a single epoch avoids data repetition but leaves too few well-resolved IsoFLOP minima for reliable fitting (see the multi-epoch analysis in §A.4). This provides early evidence that scaling trends may begin to break down in this regime, strongly motivating this kind of neural scaling analysis before moving SciML models to frontier model scales, data scales, and resolutions.

5 Conclusions

We systematically studied neural scaling for data-driven weather forecasting SciML models. To do so, we used a minimalist Transformer backbone, without much domain-specific complexities, so that scaling behavior could be studied in its simplest and most interpretable form, without confounding architecture / loss function choices. We used a continual training approach with constant LRs and periodic cooldowns, an approach which offers several advantages. This outperforms the commonly-used cosine decay schedule in the tested settings; it enables rapid alignment of the model with downstream objectives during short cooldown phases; and, most importantly, it makes scaling law analysis practical by allowing IsoFLOP curves to be constructed without repeatedly training models from scratch. We demonstrated this by efficiently exploring a wide range of model sizes and compute budgets, identifying compute-optimal relationships between model size, dataset size, and training compute, with neural scaling trends emerging across three orders of magnitude in compute (even with multi-epoch training). Our extrapolation runs at larger compute budgets show signs of saturation. These results support limited data quantity and diversity as the most likely constraints in this multi-epoch regime. Even when following compute-optimal scaling, large models require multiple passes over the data, so simply increasing model size may not yield proportional gains. These observations suggest that very large SciML models should be carefully analyzed and planned before scaling further. A natural next step is to extend cooldown adaptation and scaling studies to probabilistic objectives such as CRPS, evaluating calibration and whether short cooldowns remain sufficient. Overall, our results emphasize that understanding neural scaling behavior for a given dataset is critical before training SciML models at frontier scale, and they provide a practical framework for doing so in spatiotemporal scientific domains.

Acknowledgments and Disclosure of Funding

This research used the Perlmutter supercomputing resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231. An award of computer time was provided by the ASCR Leadership Computing Challenge (ALCC) program at NERSC under ERCAP0038267. This work was also supported by the NERSC Science Acceleration Program (NESAP) at NERSC under ERCAP0038454, and supported by resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Advanced Scientific Computing Research programs in the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. SS would like to thank Ankur Mahesh for helpful discussions on weather modeling and downstream metrics. Our conclusions do not necessarily reflect the position or the policy of our sponsors, and no official endorsement should be inferred.

References

[1] Spherical harmonics: Power spectrum in signal processing. https://en.wikipedia.org/wiki/Spherical_harmonics#Power_spectrum_in_signal_processing. Wikipedia, The Free Encyclopedia.

[2] Samira Abnar, Harshay Shah, Dan Busbridge, Alaaeldin Mohamed Elnouby Ali, Josh Susskind, and Vimal Thilak. Parameters vs flops: Scaling laws for optimal sparsity for mixture-of-experts language models, 2025. URL https://doi.org/10.48550/arXiv.2501.12370.

[3] Ferran Alet, Ilan Price, Andrew El-Kadi, Dominic Masters, Stratis Markou, Tom R. Andersson, Jacklynn Stott, Remi Lam, Matthew Willson, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Skillful joint probabilistic weather forecasting from marginals, 2025. URL https://arxiv.org/abs/2506.10772.

[4] Mihai Alexe, Simon Lang, Mariana Clare, Martin Leutbecher, Christopher Roberts, Linus Magnusson, Matthew Chantry, Rilwan Adewoyin, Ana Prieto-Nemesio, Jesper Dramsch, Florian Pinault, and Baudouin Raoult. Data-driven ensemble forecasting with the aifs. *ECMWF Newsletter*, 181(Autumn 2024):32–37, 2024. doi: 10.21957/ma3p95hxe2. URL https://www.ecmwf.int/en/elibrary/81620-data-driven-ensemble-forecasting-aifs.

[5] Yasaman Bahri, Ethan Dyer, Jared Kaplan, Jaehoon Lee, and Utkarsh Sharma. Explaining neural scaling laws. *Proceedings of the National Academy of Sciences*, 121(27), June 2024. ISSN 1091-6490. doi: 10.1073/pnas.2311878121. URL http://dx.doi.org/10.1073/pnas.2311878121.

[6] Zied Ben-Bouallegue, Mariana CA Clare, Linus Magnusson, Estibaliz Gascon, Michael Maier-Gerber, Martin Janousek, Mark Rodwell, Florian Pinault, Jesper S Dramsch, Simon TK Lang, et al. The rise of data-driven weather forecasting. *arXiv preprint arXiv:2307.10128*, 2023. URL https://doi.org/10.48550/arXiv.2307.10128.

[7] Kaifeng Bi, Lingxi Xie, Hengheng Zhang, Xin Chen, Xiaotao Gu, and Qi Tian. Pangu-weather: A 3d high-resolution model for fast and accurate global weather forecast, 2022. URL https://doi.org/10.48550/arXiv.2211.02556.

[8] Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. Deepseek llm: Scaling open-source language models with longtermism, 2024. URL https://doi.org/10.48550/arXiv.2401.02954.

[9] Cristian Bodnar, Wessel P. Bruinsma, Ana Lucic, Megan Stanley, Anna Vaughan, Johannes Brandstetter, Patrick Garvan, Maik Riechert, Jonathan A. Weyn, Haiyu Dong, Jayesh K. Gupta, Kit Thambiratnam, Alexander T. Archibald, Chun-Chieh Wu, Elizabeth Heider, Max Welling, Richard E. Turner, and Paris Perdikaris. A foundation model for the earth system, 2024. URL https://doi.org/10.48550/arXiv.2405.13063.

[10] Boris Bonev, Thorsten Kurth, Christian Hundt, Jaideep Pathak, Maximilian Baust, Karthik Kashinath, and Anima Anandkumar. Spherical fourier neural operators: learning stable dynamics on the sphere, 2023. URL https://dl.acm.org/doi/10.5555/3618408.3618525.

[11] Boris Bonev, Thorsten Kurth, Ankur Mahesh, Mauro Bisson, Jean Kossaifi, Karthik Kashinath, Anima Anandkumar, William D. Collins, Michael S. Pritchard, and Alexander Keller. Fourcastnet 3: A geometric approach to probabilistic machine-learning weather forecasting at scale, 2025. URL https://doi.org/10.48550/arXiv.2507.12144.

[12] Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers, 2023. URL https://doi.org/10.48550/arXiv.2202.03376.

[13] Kang Chen, Tao Han, Junchao Gong, Lei Bai, Fenghua Ling, Jing-Jia Luo, Xi Chen, Leiming Ma, Tianning Zhang, Rui Su, Yuanzheng Ci, Bin Li, Xiaokang Yang, and Wanli Ouyang. Fengwu: Pushing the skillful global medium-range weather forecast beyond 10 days lead, 2023. URL https://doi.org/10.48550/arXiv.2304.02948.

[14] European Centre for Medium-Range Weather Forecasts. Atmospheric model high resolution 10-day forecast (set i - hres). https://www.ecmwf.int/, 2023. April 5, 2024.

[15] Facebook AI Research. fvcore: Collection of common code shared among fair computer vision projects. https://github.com/facebookresearch/fvcore, 2023. Accessed: 2026-01-28.

[16] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, et al. The llama 3 herd of models, 2024. URL https://doi.org/10.48550/arXiv.2407.21783.

[17] Väinö Hatanpää, Eugene Ku, Jason Stock, Murali Emani, Sam Foreman, Chunyong Jung, Sandeep Madireddy, Tung Nguyen, Varuni Sastry, Ray A. O. Sinurat, Sam Wheeler, Huihuo Zheng, Troy Arcomano, Venkatram Vishwanath, and Rao Kotamarthi. Aeris: Argonne earth systems model for reliable and skillful predictions, 2025. URL https://doi.org/10.48550/arXiv.2509.13523.

[18] Alex Henry, Prudhvi Raj Dachapally, Shubham Pawar, and Yuxuan Chen. Query-key normalization for transformers, 2020. URL https://doi.org/10.48550/arXiv.2010.04245.

[19] Hans Hersbach, Bill Bell, Paul Berrisford, Shoji Hirahara, András Horányi, Joaquín Muñoz-Sabater, Julien Nicolas, Carole Peubey, Raluca Radu, Dinand Schepers, et al. The ERA5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society*, 146(730):1999–2049, 2020. URL https://doi.org/10.1002/qj.3803.

[20] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically, 2017. URL https://arxiv.org/abs/1712.00409.

[21] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. Training compute-optimal large language models, 2022. URL https://doi.org/10.48550/arXiv.2203.15556.

[22] Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna Ben Allal, Leandro Von Werra, and Martin Jaggi. Scaling laws and compute-optimal training beyond fixed training durations, 2024. URL https://doi.org/10.48550/arXiv.2405.18392.

[23] 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, 2020. URL https://doi.org/10.48550/arXiv.2001.08361.

[24] Matthias Karlbauer, Danielle C. Maddix, Abdul Fatir Ansari, Boran Han, Gaurav Gupta, Yuyang Wang, Andrew Stuart, and Michael W. Mahoney. Comparing and contrasting deep learning weather prediction backbones on navier-stokes and atmospheric dynamics, 2024. URL https://doi.org/10.48550/arXiv.2407.14129.

[25] Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, Marek Cygan, and Sebastian Jaszczur. Scaling laws for fine-grained mixture of experts, 2024. URL https://doi.org/10.48550/arXiv.2402.07871.

[26] Thorsten Kurth, Shashank Subramanian, Peter Harrington, Jaideep Pathak, Morteza Mardani, David Hall, Andrea Miele, Karthik Kashinath, and Anima Anandkumar. Fourcastnet: Accelerating global high-resolution weather forecasting using adaptive fourier neural operators. In *Proceedings of the platform for advanced scientific computing conference*, pages 1–11, 2023. URL https://dl.acm.org/doi/10.1145/3592979.3593412.

[27] Thorsten Kurth, Max Rietmann, Mauro Bisson, Andrea Paris, Alberto Carpentieri, Jean Kossaifi, Anima Anandkumar, Christian Hundt, and Boris Bonev. A library for differentiable signal processing and machine learning on the sphere, 2026. URL https://arxiv.org/abs/2609.39737.

[28] Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Leonard Casanova, Geoffrey Tseng, et al. Learning skillful medium-range global weather forecasting. *Science*, 382(6677):1416–1421, 2023. URL https://doi.org/10.1126/science.adi2336.

[29] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021. URL https://doi.org/10.48550/arXiv.2103.14030.

[30] Niklas Muennighoff, Alexander Rush, Boaz Barak, Teven Le Scao, Nouamane Tazi, Aleksandra Piktus, Sampo Pyysalo, Thomas Wolf, and Colin A. Raffel. Scaling data-constrained language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/9d89448b63ce1e2e8dc7af72c984c196-Abstract-Conference.html.

[31] Eric Nguyen, Michael Poli, Matthew G. Durrant, Brian Kang, Dhruva Katrekar, David B. Li, Liam J. Bartie, Armin W. Thomas, Samuel H. King, Garyk Brixi, Jeremy Sullivan, Madelena Y. Ng, Ashley Lewis, Aaron Lou, Stefano Ermon, Stephen A. Baccus, Tina Hernandez-Boussard, Christopher Ré, Patrick D. Hsu, and Brian L. Hie. Sequence modeling and design from molecular to genome scale with evo. *Science*, 386 (6723):eado9336, 2024. doi: 10.1126/science.ado9336. URL https://www.science.org/doi/abs/10.1126/science.ado9336.

[32] Tung Nguyen, Rohan Shah, Hritik Bansal, Troy Arcomano, Romit Maulik, Veerabhadra Kotamarthi, Ian Foster, Sandeep Madireddy, and Aditya Grover. Scaling transformer neural networks for skillful and reliable medium-range weather forecasting, 2024. URL https://doi.org/10.48550/arXiv.2312.03876.

[33] NVIDIA. Transformer engine: A library for accelerating transformer models. GitHub repository. https://github.com/NVIDIA/TransformerEngine.

[34] NVIDIA. Nvidia data loading library (dali), 2026. URL https://github.com/NVIDIA/DALI.

[35] Ilan Price, Alvaro Sanchez-Gonzalez, Ferran Alet, Timo Ewalds, Andrew El-Kadi, Jacklynn Stott, Shakir Mohamed, Peter Battaglia, Remi Lam, and Matthew Willson. Gencast: Diffusion-based ensemble forecasting for medium-range weather. *arXiv preprint arXiv:2312.15796v2*, 2023. URL https://doi.org/10.48550/arXiv.2312.15796.

[36] Stephan Rasp, Stephan Hoyer, Alexander Merose, Ian Langmore, Peter Battaglia, Tyler Russell, Alvaro Sanchez-Gonzalez, Vivian Yang, Rob Carver, Shreya Agrawal, Matthew Chantry, Zied Ben Bouallegue, Peter Dueben, Carla Bromberg, Jared Sisk, Luke Barrington, Aaron Bell, and Fei Sha. Weatherbench 2: A benchmark for the next generation of data-driven global weather models. *Journal of Advances in Modeling Earth Systems*, 16(6):e2023MS004019, 2024. doi: https://doi.org/10.1029/2023MS004019. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2023MS004019. e2023MS004019 2023MS004019.

[37] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019. URL https://doi.org/10.48550/arXiv.1909.08053.

[38] Christopher Subich, Syed Zahid Husain, Leo Separovic, and Jing Yang. Fixing the double penalty in data-driven weather forecasting through a modified spherical harmonic loss function, 2025. URL https://doi.org/10.48550/arXiv.2501.19374.

[39] Paul A Ullrich, Colin M Zarzycki, Elizabeth E McClenny, Marielle C Pinheiro, Alyssa M Stansfield, and Kevin A Reed. Tempestextremes v2. 1: A community framework for feature detection, tracking, and analysis in large datasets. *Geoscientific Model Development*, 14(8):5023–5048, 2021.

[40] Alexius Wadell, Anoushka Bhutani, Victor Azumah, Austin R. Ellis-Mohr, Celia Kelly, Hancheng Zhao, Anuj K. Nayak, Kareem Hegazy, Alexander Brace, Hongyi Lin, Murali Emani, Venkatram Vishwanath, Kevin Gering, Melisa Alkan, Tom Gibbs, Jack Wells, Lav R. Varshney, Bharath Ramsundar, Karthik Duraisamy, Michael W. Mahoney, Arvind Ramanathan, and Venkatasubramanian Viswanathan. Foundation models for discovery and exploration in chemical space, 2025. URL https://doi.org/10.48550/arXiv.2510.18900.

[41] Xiao Wang, Siyan Liu, Aristeidis Tsaris, Jong-Youl Choi, Ashwin Aji, Ming Fan, Wei Zhang, Junqi Yin, Moetasim Ashfaq, Dan Lu, and Prasanna Balaprakash. Orbit: Oak ridge base foundation model for earth system predictability, 2024. URL https://doi.org/10.48550/arXiv.2404.14712.

[42] Xiao Wang, Jong-Youl Choi, Takuya Kurihaya, Isaac Lyngaas, Hong-Jun Yoon, Xi Xiao, David Pugmire, Ming Fan, Nasik Muhammad Nafi, Aristeidis Tsaris, Ashwin M. Aji, Maliha Hossain, Mohamed Wahib, Dali Wang, Peter Thornton, Prasanna Balaprakash, Moetasim Ashfaq, and Dan Lu. Orbit-2: Scaling exascale vision foundation models for weather and climate downscaling. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '25, pages 86–98, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714665. doi: 10.1145/3712285.3771989. URL https://doi.org/10.1145/3712285.3771989.

[43] Jared D. Willard, Peter Harrington, Shashank Subramanian, Ankur Mahesh, Travis A. O'Brien, and William D. Collins. Analyzing and exploring training recipes for large-scale transformer-based weather prediction, 2024. URL https://doi.org/10.48550/arXiv.2404.19630.

[44] Brandon M. Wood, Misko Dzamba, Xiang Fu, Meng Gao, Muhammed Shuaibi, Luis Barroso-Luque, Kareem Abdelmaqsoud, Vahe Gharakhanyan, John R. Kitchin, Daniel S. Levine, Kyle Michel, Anuroop Sriram, Taco Cohen, Abhishek Das, Ammar Rizvi, Sushree Jagriti Sahoo, Zachary W. Ulissi, and C. Lawrence Zitnick. Uma: A family of universal models for atoms, 2025. URL https://doi.org/10.48550/arXiv.2506.23971.

[45] Yuejiang Yu, Langwen Huang, Alexandru Calotoiu, and Torsten Hoefler. Scaling laws of global weather models. *arXiv preprint arXiv:2602.22962*, 2026.

A Appendix : Additional Details

A.1 Limitations

The main limitation of this paper is its focus on deterministic training of the weather model. Given the inherently chaotic nature of weather, probabilistic estimates are often more informative for forecasting at future time points, with metrics such as continuous ranked probability scores (CRPS) and spread–skill ratios (SSR) providing better diagnostics for a model's predictive capability. Addressing this would involve training (or cooling down) models using diffusion objectives [35, 4] or CRPS-based ensemble training [4, 11], which may exhibit different scaling behaviors; incorporating physics-informed penalties during cooldown and characterizing how upstream scaling translates to quantitative downstream metrics (such as cyclone-track error) are also left for future work. Our patch size is fixed for the scaling tests, but the internal Transformer resolution (sequence length) represents another dimension for systematic scaling studies. Another possible limitation is the focus on Transformer architectures. While this focus is intentional to demonstrate that simple architectures can scale and perform competitively, it remains important to explore other architecture types, such as FourCastNet [11], which incorporates geometric inductive biases through spherical transforms within the neural operator framework, or graph-based approaches like GraphCast [28]. Finally, to rigorously probe the implications of scaling laws and not be constrained by multi-epoch training, it is essential to extend beyond this single domain, incorporating diverse datasets from across the Earth Sciences (and beyond), and exploring cross-domain foundation model pre-training. This will allow us to evaluate scaling behavior in a broader, more complex data landscape.

A.2 ERA5 Dataset and Metrics

For the current weather state $u_n \in \mathbb{R}^{71\times721\times1440}$, we closely follow existing works [11, 43] and select the following variables: geopotential height (z), winds (u, v), temperature (t), and specific humidity (q) at 13 vertical pressure levels (50hPa, 100hPa, 150hPa, 200hPa, 250hPa, 300hPa, 400hPa, 500hPa, 600hPa, 700hPa, 850hPa, 925hPa, and 1000hPa). We also include the following surface variables: $u10m$ (u at 10m), $v10m$ (v at 10m), $t2m$ (temperature at 2m), surface pressure (sp), mean sea level pressure (msl), and total column water vapor ($tcwv$). Finally, we include land-sea mask, orography, and cosine of the latitude (to convey the curvature of the earth) as static (not dependent on time) masks (not predicted).

Our metrics are area-weighted RMSE and PSD for each channel C defined as follows. Given prediction u_n and ground truth $\hat{u}_n$ at time step n,

$$\mathrm{RMSE}(u_n, \hat{u}_n) = \sqrt{\frac{1}{H\,W}\sum_{h=1}^{H}\sum_{w=1}^{W} w_h \big(u_{c,h,w} - \hat{u}_{c,h,w}\big)^2},$$

with w_h being the area weights, proportional to the cosine of the latitude, normalized to one.

The PSD [1, 11] of u at spherical harmonic degree ℓ is defined as:

$$\mathrm{PSD}_\ell(u) = \frac{1}{2\ell+1}\sum_{m=-\ell}^{\ell} |u_{\ell m}|^2, \tag{2}$$

where $u_{\ell m}$ are the coefficients of u in the spherical harmonic decomposition:

$$u(\theta,\phi) = \sum_{\ell=0}^{\infty}\sum_{m=-\ell}^{\ell} u_{\ell m}\, Y_{\ell m}(\theta,\phi),$$

with $Y_{\ell m}$ denoting the spherical harmonic functions of degree ℓ and order m, and (θ, ϕ) the latitude and longitude coordinates. We use `torch-harmonics` [10, 27] to implement the Spherical Harmonic Transform (SHT). We apply all SHTs in FP64 to avoid numerical issues. We use the SHTs for the metrics as well as the `AMSE` loss functions, defined below.

For the `AMSE` loss [38], we use the total power at each degree,

$$P_\ell(u) = \sum_{m=-\ell}^{\ell} |u_{\ell m}|^2 = |u_{\ell 0}|^2 + 2\sum_{m=1}^{\ell} |u_{\ell m}|^2 = (2\ell+1)\mathrm{PSD}_\ell(u). \tag{3}$$

Table A1: Model configurations used for the neural scaling: We run each of these models once to different final iteration counts based on the required FLOP budgets with 5% cooldown periods at the end for the LR.

Param (M)	Embed	Depth	LR
0.7	96	4	1.5E-3
0.9	96	6	1.5E-3
1.5	128	6	1.5E-3
3.1	192	6	1.5E-3
6.9	256	8	1.5E-3
15.1	384	8	1.5E-3
26.4	512	8	1.2E-3
39.0	512	12	1E-3
60.5	640	12	8E-4
86.8	768	12	6E-4
115.1	768	16	5E-4
153.5	1024	12	5E-4
203.8	1024	16	5E-4
304.5	1024	24	5E-4
456.7	1536	16	5E-4

The second equality holds for real-valued fields and accounts for the negative orders omitted by the real SHT. The loss between u and $\hat{u}$ is

$$\mathrm{AMSE}(u, \hat{u}) = \sum_{\ell} \left[\left(\sqrt{P_\ell(u)} - \sqrt{P_\ell(\hat{u})} \right)^2 + 2 \, \max\left(P_\ell(u), P_\ell(\hat{u}) \right) \left(1 - \mathrm{Coh}_\ell(u, \hat{u}) \right) \right], \quad (4)$$

with the coherence between u and $\hat{u}$ at degree ℓ as:

$$\mathrm{Coh}_\ell(u, \hat{u}) = \frac{\sum_{m=-\ell}^{\ell} \mathrm{Re}\left(u_{\ell m} \, \hat{u}^*_{\ell m} \right)}{\sqrt{P_\ell(u) \, P_\ell(\hat{u})}}. \quad (5)$$

A.3 Model hyperparameters and training details

We train the `Swin` models (each with global batch size 16) summarized in Tab. A1. We fix our patch size to 4 based on patch size ablations (see Tab. A4) We also fix the head dimension to 64 through minimal hyperparameter tuning experiments and the window size to 9×18 based on Willard et al. [43]. We observe minimal accuracy gains with larger windows and smaller ones begin to degrade the accuracy. For the feedforward (MLP) layers, we use a hidden size to embedding dimension ratio of 4 for all models. For the $1.24B$ `Swin`, we use an embedding dimension of 1792 and depth 32. For the cooldowns, we use a 1-sqrt() cooldown function based on Hägele et al. [22].

LR tuning. For cosine vs. cooldowns, we independently sweep the peak LR over $\{$1E-4, 5E-4, 1E-3$\}$ for the $115M$ model, with global batch size 16 and the same one-step MSE training objective. We compare schedules at matching total iteration budgets of 12K, 18K, and 24K, including the final 5% cooldown. We report the validation losses in Table A2. Constant+cooldown achieves lower loss at each tested LR and budget. Hence, this is a valid (and better performing) strategy for continual training to carry out cheaper neural scaling experiments in our setting. For the scaling experiments, we select a peak LR of 5E-4 for models larger than $100M$ parameters. We find that smaller models tolerate higher LRs, up to 1.5E-3, similar to observations in NLP [23], whereas the larger models eventually diverge. In our tuning experiments with moderately sized models, the selected LR is approximately inversely proportional to the square root of the parameter count. The model-specific LRs are listed in Tab. A1. For the $1.24B$ model, we did not test a higher LR because larger LRs had eventually diverged for the earlier model runs and tuning at this scale was prohibitively expensive.

Compute details. Given the high resolution ($64800 = 180 \times 360$ sequence length from patch 4), we can only train most models with a local batch size of one. All our models are trained on NVIDIA

Table A2: Independent LR sweeps for the $115M$ `Swin`. Validation loss at matched total iteration budgets (including cooldown); lower is better. Both schedules use global batch size 16. Constant+cooldown uses the final 5% of iterations for cooldown.

Iterations	Schedule	1E-4	5E-4	1E-3
12K	Cosine	0.026	0.010	0.009
	Constant+cooldown	0.019	0.0089	0.0082
18K	Cosine	0.020	0.0088	0.0079
	Constant+cooldown	0.014	0.0080	0.0074
24K	Cosine	0.017	0.0081	0.0074
	Constant+cooldown	0.012	0.0075	0.0071

A100 GPUs (40G memory capacity, unless specified). We find all models use most of the memory capacity, necessitating model parallelism at the higher scales. For models up to $100M$, we use only data parallelism (16 GPUs, local batch size 1, global batch size 16). For models up to 304M, we use a 2×1 spatial parallelism (32 GPUs in total). For the large models, we observe training times of 4–6 hours. For example, the $204M$ model trains to 6E19 FLOPs in about 6 hours on 32 GPUs. For the largest model ($456.7M$ parameters) in neural scaling, we use 2×2 spatial parallel (4 GPUs) for a total of 64 GPUs. This model trains to 6E19 FLOPs in about 3 hours. For AR cooldowns with 4 steps, the memory complexity increases linearly and hence this model would then require 16 GPUs for spatial parallel and a total of 256 GPUs. We estimate from the cluster summary usage that the neural scaling experiments (15 models, 57 different iteration counts for cooldowns in total) took about 4000 GPU hours, excluding the extrapolation runs, (several models were trained longer than they needed to be trained) with an additional O(2000) GPU hours for different hyperparameter experiments. The biggest compute expense was the $1.24B$ model. For only the $1.24B$ parameter model, we use 80G A100s and train it on 64 GPUs with spatial parallel 4. We do this since this training run is quite expensive. Our cluster contains 4-way NVLINK between GPUs and hence keeping spatial parallelism to less than 4 is ideal to avoid communication bottlenecks. This training run takes about 3.3 days to finish (to 2.25E21 FLOPs; about 5000 GPU hours) on the 64 GPUs and hence is an expensive run. This is about a 40% model FLOPs utilization (MFU) (compared to peak A100 Tensor Core FP16 FLOPs) which is reasonable for Transformer models.

Hybrid spatial (domain) + data parallelism. The main ingredient in spatial parallelism is implementing a distributed roll operation. Instead of shifting the windows directly, the original `Swin` implementation [29] simply rolls (cyclically shifts) the image instead and applies an attention mask to limit the attention to each sub-window. When the image is partitioned across GPUs, this roll operation must happen across the GPUs that shard the image. Hence, these spatial parallel GPUs must send and receive data from neighbors for an accurate roll. This boils down to a *collective permute* operation and we implement it using PyTorch's `torch.distributed.all_to_all_single` collective. Following frontier NLP research codes from `Megatron-LM` [37], we implement the forward and backward pass of this distributed operation using PyTorch's custom `Autograd` functions, where the backward pass is simply the conjugate forward operation: a reverse distributed roll. We show the distributed roll algorithm in Alg. 1. We note that this is not a full replacement for distributed `torch.roll`, as we assume we only need to send one slice to one neighboring GPU and receive one slice from another neighboring GPU. This is true for `Swin`, because we shift by a half of the window size, and windows are not sharded across GPUs in our setting (there is at least one window on every GPU). For brevity, we also omit the `permute` operations to bring the shift dimension to position 0 and back (`all_to_all_single` only acts on the first dimension).

Finally, spatial parallel introduces weights (and bias) tensors that are *shared* across the spatial parallel GPU groups (in addition to the data parallel GPUs that, by design, share the model weights). Hence, the weight gradients (`wgrads`) must be additionally reduced in this group (along with the data parallel group). Our simplistic `Swin` model design (with coordinate position embeddings) leads to all the weights (and bias) tensors shared across the spatial parallel group and hence we implement this as a joint `AllReduce` operation across the combined data and spatial parallel GPU group using a communication hook in PyTorch's `DistributedDataParallel`. We verify with Unit Tests that the distributed forward and backward passes (including weight gradients and adjoints) match the non-distributed version ensuring accuracy of our custom model parallelism. Finally, we also use `NVIDIA`

Algorithm 1 Distributed Roll for Shifting Windows in `Swin`

Require: Roll dimension d, local tensor x (spatially sharded in d across GPUs), shift size s, spatial parallel GPU group $\mathcal{G}$
Ensure: Tensor x rolled along dimension d
$(r, P) \leftarrow \text{rank_and_size}(\mathcal{G})$ {GPU rank and world size}
if $P = 1$ **or** $s = 0$ **then**
  return $\texttt{roll}(x, s, d)$ {purely local case}
right $\leftarrow (r + 1) \bmod P$
left $\leftarrow (r - 1) \bmod P$ {neighboring GPUs in the ring}
if $s > 0$ **then**
  send $\leftarrow x[\ldots, -s{:}, \ldots]$ {right boundary slice}
else
  send $\leftarrow x[\ldots, :{-s}, \ldots]$ {left boundary slice}
recv $\leftarrow \texttt{zeros_like}(\text{send})$ {receive buffer}
$\texttt{distributed_all_to_all_single}(\text{recv}, \text{send}, \mathcal{G})$ {$s > 0$ sends right / receives left} {$s < 0$ vice versa}
$x \leftarrow \texttt{roll}(x, s, d)$ {local roll (boundary will be corrected)}
if $s > 0$ **then**
  $x \leftarrow \texttt{concat}(\text{recv}, x[\ldots, s{:}, \ldots], d)$
else
  $x \leftarrow \texttt{concat}(x[\ldots, :s, \ldots], \text{recv}, d)$
return x

`TransformerEngine` [33] for easy access to fused operations and support for 4D parallelism with tensor parallelism. While we do not use tensor parallelism in any run, we note that the code supports this parallelism as well for scaling up. Here, an additional orthogonal array of GPUs may be employed to further shard the model weights (similar to NLP). `TransformerEngine` provides a drop-in replacement for `Linear` layers in our model to use tensor parallelism. We note that some `bias` tensors may be shared across tensor and spatial parallel GPU groups with this additional parallelism and we take care of this through custom communication hooks in `DistributedDataParallel`. Unit tests confirm that combined spatial, tensor, and data parallelism yield the correct distributed forward pass outputs and backward pass `wgrads` and `dgrads`.

Efficient and stateful dataloaders. Our data is stored in HDF5 files, with one file per year of ERA5, to efficiently leverage the Lustre filesystem in our experiments for high-throughput data access. We use `NVIDIA DALI` [34] dataloaders, which automatically overlap data loading with computation during the forward and backward passes and perform pre-processing operations, such as data normalization, directly on the GPU. This ensures that data I/O does not become a performance bottleneck. Finally, we train our models to fixed iteration counts (to support periodic cooldowns under a predefined FLOP budget), which requires checkpoint–restart functionality to operate correctly mid-epoch. To enable this, we implement stateful dataloaders within DALI that track both the shuffled sample indices and the number of completed iterations, allowing training to resume seamlessly from arbitrary mid-epoch checkpoints.

Code and FLOPs. We open-source our code at `https://github.com/ShashankSubramanian/neural-scaling-weather`. For all our FLOP computations, due to the homogeneous design of the `Swin`, we use standard analytical attention and MLP FLOP estimates, taking into account the windowed attention, and verify with `fvcore` [15] that our analytical values are close to the correct estimates. We include the patch, position, and reverse-patch embedding FLOPs as well in our estimation. The FLOP expressions can be found in `utils/flops_utils.py` in our code.

For comparison to GraphCast [28], we use their open-sourced code base `https://github.com/google-deepmind/graphcast/blob/main/graphcast_demo.ipynb` to get a preliminary estimate on the FLOPs. We use the `compiled.cost_analysis()` from `JAX` to compute the FLOPs for a forward pass of the saved GraphCast checkpoint on the 0.25° ERA5 data. The saved checkpoint has $36.4M$ parameters (with latent dimension 512) and the FLOP estimate for a single forward pass (batch size 1) is 2.97E13 FLOPs. We then multiply with the following training details reported by Lam et al. [28]: batch size 32 with $300,000$ iterations for pre-training. This results in 2.85E20 FLOPs for the full forward pass. Assuming backward is $2\times$, the pre-training cost is 8.55E20

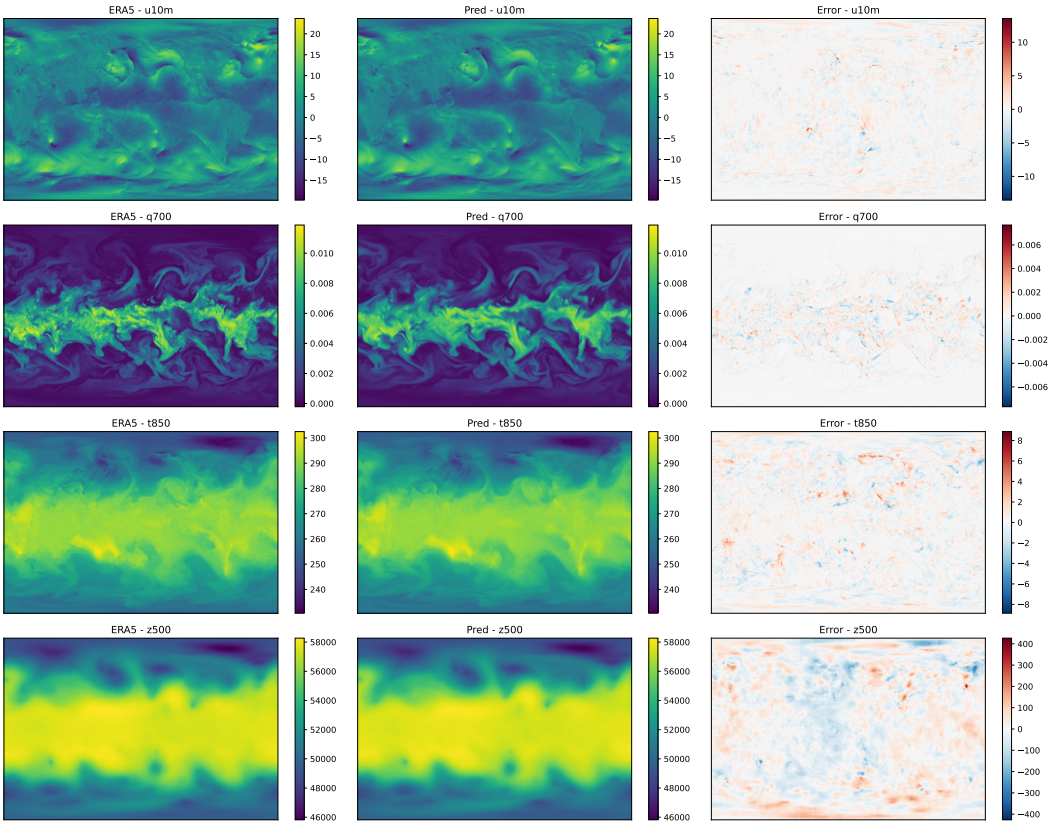


Figure A1: **Predictions from the 204M parameter model:** We show the ground truth ERA5, the prediction, and the error in physical units for four variables. The predicted forecasts also qualitatively match the ground truth.

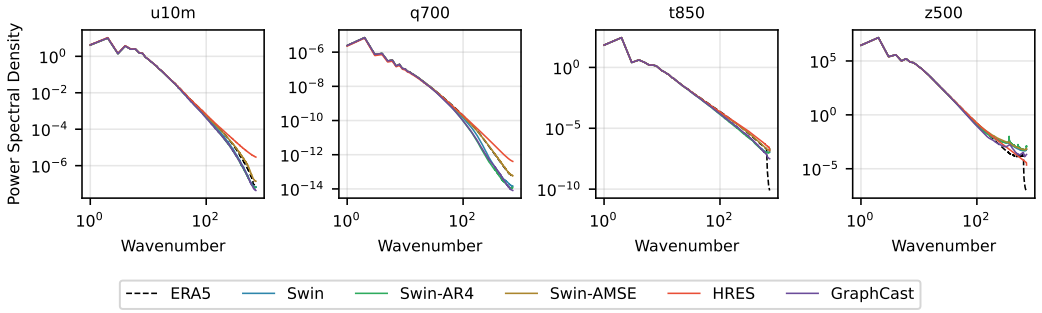
FLOPs. In fine-tuning, GraphCast performs 2–12-step AR over $11,000$ iterations (AR increased by 1 every 1000 iterations) . Assuming that AR-N amplifies the FLOP count by N, this is equivalent to $(2+3+\cdots+12)\times 1000 = 77\times 1000 = 77,000$ 1-step forward passes in fine-tuning, or 2.19E20 FLOPs. Roughly 20% of the total compute was used for fine-tuning, with a total of around 1E21 FLOPs. For the $204M$ Swin, we use approximately 6E19 FLOPs in pre-training and with 8-step AR in cooldowns (5%), the total amounts to 8.4E19 FLOPs.

A.4 Additional results and analyses

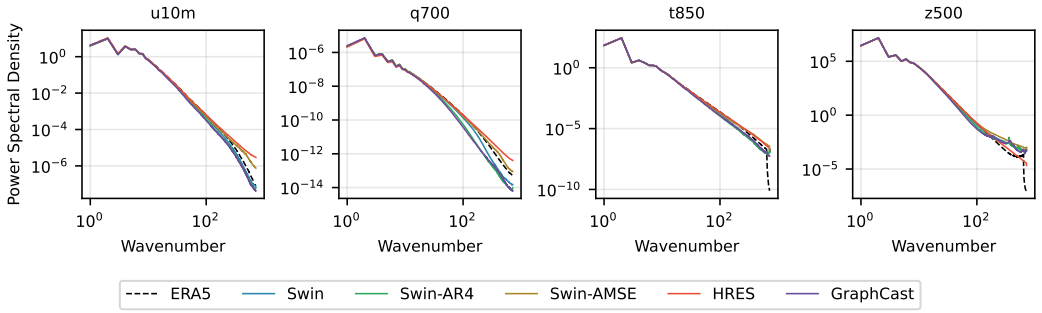
Prediction visualization. We visualize the prediction from the compute-optimal model ($203.8M$) at the large budget 6E19 in Fig. A1, along with the bias (error) between the predictions and the ground truth (ERA5). The predictions are obtained on the testing year 2020, with initial conditions every 12 hours (for a total of 732 initial conditions). Beyond the quantitative metrics defined earlier, we observe that the model produces realistic predictions of the physical fields, capturing the main spatial structures and patterns present in the reference data.

Inference speed. All the Swin models fit on a single GPU (A100 40G) during inference. Most models generate 10-day forecasts in less than a minute (including metric computations). For example, the $204M$ Swin takes 32 seconds, while the largest $1.24B$ Swin takes around 96 seconds to generate the same 10-day forecast. These models preserve the emulator's substantial computational advantage over the NWP, achieving a speedup of more than $100\times$ [4].

Cooldowns and Alignment. In Fig. A2, we show the PSD of different variables for two lead times (1 day and 5 days). The Swin model cooled down with AR4 loses a significant amount of energy at small scales, resulting in overly dissipated forecasts and an effective resolution much coarser than what the 0.25° grid can support. In contrast, the Swin model cooled down with AMSE matches the PSD of



(a) PSD of different variables compared to PSD of the truth (ERA5) for different models at 24 hours lead time.



(b) PSD of different variables compared to PSD of the truth (ERA5) for different models at 120 hours lead time.

Figure A2: **PSD of predictions at 1 day and 5 days:** While `AR` produces blurrier forecasts (though with lower RMSE), `AMSE` helps retain power at high wavenumbers. At longer lead times, such as 120 hours, `Swin-AMSE` continues to show significant power at these small scales. This allows practitioners to define an effective resolution based on this noise-level as outlined by Subich et al. [38].

the ground truth almost exactly at short lead times. At longer lead times, it exhibits higher energy than the ground truth at the smallest scales suggesting a noise-driven effective resolution that exceeds that of the dissipated `AR4` forecasts, as noted by Subich et al. [38]. Overall, the two cooling strategies guide the model toward different objectives: one prioritizes lower overall RMSE (and skill) over long lead times, akin to ensemble forecasting, while the other emphasizes high realism, preserving small-scale features in the forecast.

Cooldowns vs. cosine. With some preliminary experiments, we show that the performance improvement of cooldowns w.r.t cosine can be true for other architectures as well. In Fig. A3, we demonstrate this for a simple Vision Transformer and in Fig. A4, we show this for a SOTA model, FourCastNetv3.

Channel, area weighting, and AMSE. Other domain-specific modifications during cooldown can also be done. For example, models commonly weight the loss to emphasize certain variables or regions through channel and area weighting. We employ the channel weighting in GraphCast [28] which down-weights the loss as the pressure level decreases and also introduced surface variable weights and the standard cosine latitude area weighting (similar to the RMSE) for area weights. We train a `Swin` model with weighted losses in pre-training and compare with MSE pre-training and cooling down with the weighted losses. We report different ablations in Tab. A3. We observe that the cooldown can be repurposed with channel or area weighting with similar results as pre-training from scratch, demonstrating that the neural scaling can be disentangled from channel weighting as well. We also see that channel weighting generally reduces the loss and area weighting increases it. Similarly, AMSE-only cooldowns yield similar losses as training AMSE from scratch; both variants preserve high-resolution features in the PSD. From-scratch AMSE training sometimes produced noisy pixels which we were able to mitigate by adding a residual connection from the input to the output; we did not observe the instability in the AMSE cooled down models. This supports using AMSE for adaptation, consistent with Subich et al. [38], while reusing MSE-pre-trained checkpoints.

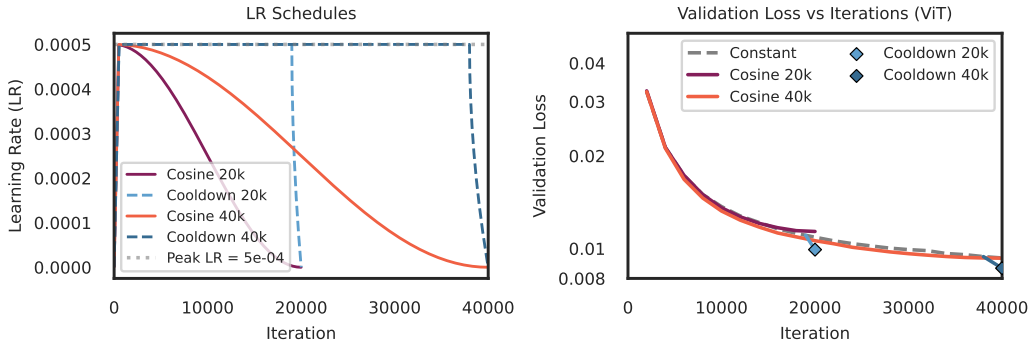


Figure A3: **Cosine vs cooldowns for a Vision Transformer (ViT):** We train a simple ViT (patch size 8, embedding 768, depth 12, LR 5E-4, learnable position embeddings, batch size 16) using both cosine schedules (to 20,000 and 40,000 iterations) and a constant + cooldown schedule. The continuous training shows that the cooldowns are superior to the cosine schedule for this ViT as well. This preliminary evidence points to the possibility that the cooldown methodology is agnostic to the architecture.

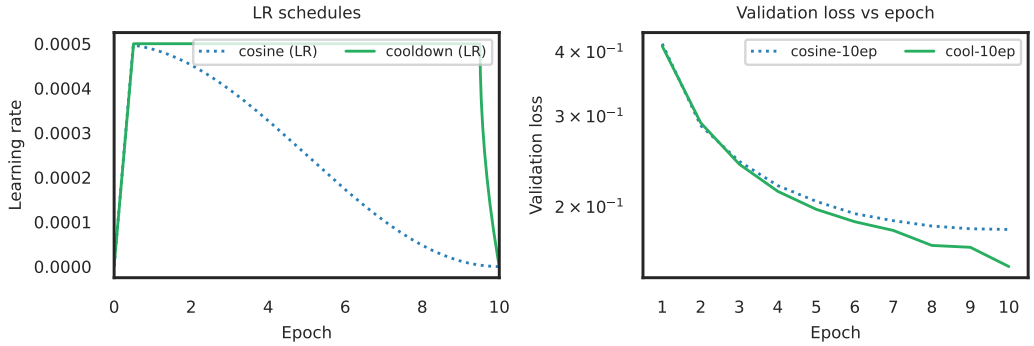


Figure A4: **Cosine vs cooldowns for FourCastNet3:** We train a small FourCastNet3 model using the deterministic training framework from the Makani repository. Each pressure, surface, and auxiliary variable is embedded with a dimension of 12; and we use the default 360 x 720 internal resolution (scale factor 2). Since the repository supports training and logging at the epoch level, we track validation loss after each epoch. To keep training manageable, we use 16,000 samples per epoch with a batch size of 16. We compare two learning rate schedules: cosine decay and a constant+cooldown schedule. In this preliminary experiment, the constant+cooldown schedule proves beneficial for Neural Operator-based models as well such as FourCastNet3 (Spherical Fourier Neural Operator). These initial results suggest that the cooldown methodology may be largely architecture-agnostic.

Table A3: **Pre-training versus cooldown objectives.** All runs use the 115M model trained for 40,435 iterations (closest compute-optimal for 3E19 FLOPs). CHWT and AWT denote channel and area weighting of MSE. AMSE loss values are not directly comparable to the MSE-based losses.

Pre-training objective	Cooldown objective	Validation loss
MSE	MSE	0.0068
MSE	MSE (CHWT)	0.0062
MSE (CHWT)	MSE (CHWT)	0.0061
MSE	MSE (CHWT + AWT)	0.0075
MSE (CHWT + AWT)	MSE (CHWT + AWT)	0.0072
AMSE	AMSE	0.109
MSE	AMSE	0.119

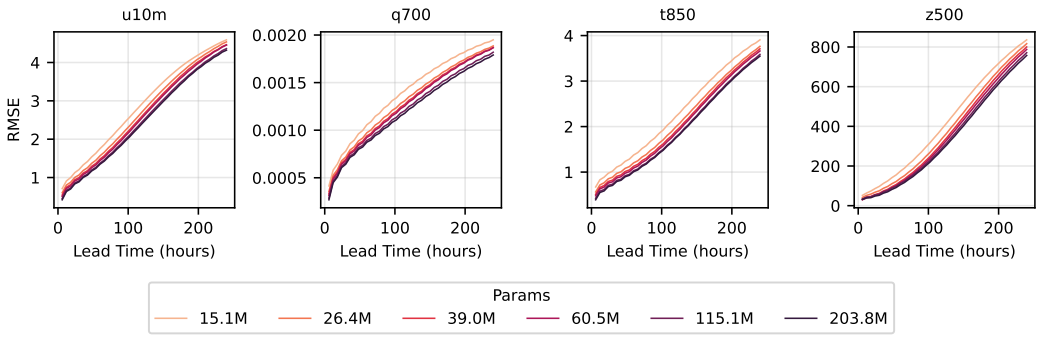


Figure A5: **Cooldown with AR4 is valid for different model sizes.** AR4 gains consistently improve with model size (and compute), similar to Fig. 6 that shows the same improvements in the AR1.

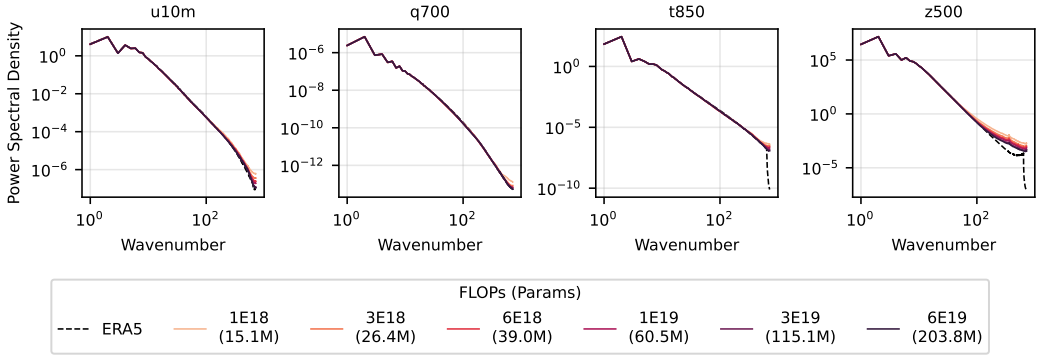


Figure A6: **Cooldown with AMSE is valid for different model sizes.** AMSE allows all models to match the ERA5 PSD more closely. PSD is computed at 24h lead time.

Downstream performance and compute. In Fig. 6, we present the RMSE over the time horizon using the closest compute-optimal model at each budget. As the compute increases, accuracy improves for both one-step predictions (as shown in Fig. 5) and over longer time horizons. At the highest budgets, however, we start to see a saturation. Fig. A5 and Fig. A6 show that repurposing cooldowns with either AR or AMSE is valid across different model sizes (and compute). This demonstrates the generality of the approach, allowing models to be aligned with different downstream objectives during the cooldown period. In Fig. A7 and Fig. A8, we show the PSD for the closest compute-optimal model at each compute budget for a 24-hour and 120-hour lead time. While all models faithfully capture the PSD at the lower wavenumbers, the higher compute models show better PSD at the higher wavenumbers as well, better capturing the sharper features. However, improvements with increasing compute are gradual and show signs of saturation. Additionally, lower-compute models exhibit artificial power at high wavenumbers, which becomes more pronounced at longer lead times (120 hours). In contrast, higher-compute models maintain more consistent performance across lead times.

These spectral improvements are more visible in a specific physical case study in Fig. A9. We reproduce the study that analyzes the Tropical Cyclone (TC) patterns of Typhoon Hagibis by Price et al. [35]. We use the same TC tools as the study—TempestExtremes [39], in order to visualize TC tracks. Here, we track the TC structure across the compute-optimal models initialized at 1, 3, and 5 days from the TC landfall time on October 12, 2019, 06 UTC. We visualize the specific humidity at 700hPa for all the forecasts as well as the PSD (compared to ERA5). For short forecasts (1 day), all models are faithful with increased compute resulting in sharper predictions (also quantified in the PSD). With longer forecasts (3 and 5 days earlier), the smaller models exhibit artificial features (patch artifacts, discontinuities in the prediction) that are also seen in the spectral power at the high wavenumbers. With more compute, the models begin to produce realistic TCs at these lead times as well. Although different hyperparameters might improve the smaller models, the artifacts are more

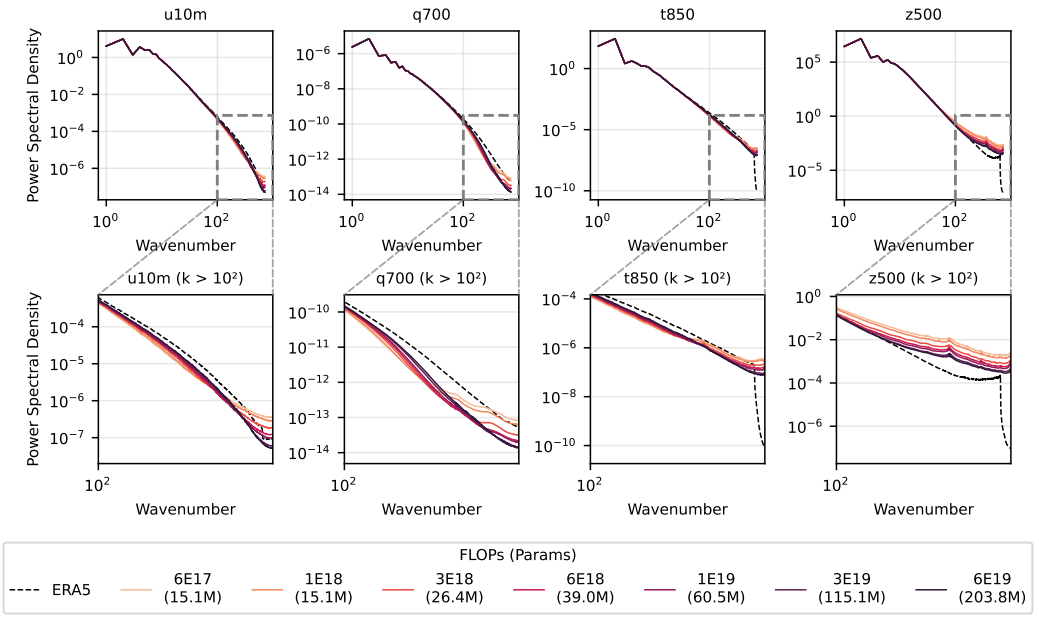


Figure A7: **PSD performance at lead time 24 hours of the closest compute-optimal `Swin` models at each compute budget:** As in Fig. 6 that shows consistent improvement of rollout RMSE with increased compute, the PSD also exhibits improvements, though they are smaller in magnitude and progress more gradually. Here, the PSD is at 24-hour lead time (averaged over the test year). In the bottom row, we zoom into the high wavenumbers and can observe that the bigger (more compute) models show sharper features and resemble ERA5 closer.

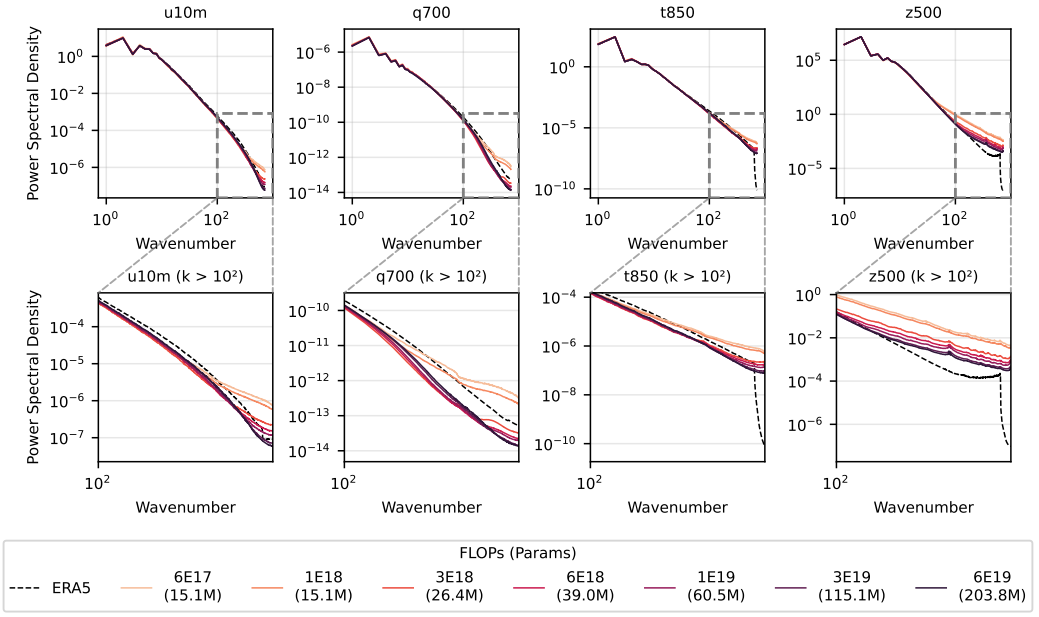


Figure A8: **PSD performance at lead time 120 hours of the closest compute-optimal `Swin` models at each compute budget:** The PSD is evaluated at a 120-hour lead time and averaged over the test year. All models accurately capture the low wavenumbers, while higher wavenumbers show gradual improvement as compute increases. The bottom row zooms in on the high-wavenumber range, where larger models exhibit spectra that more closely match ERA5, whereas smaller models appear more blurred and display some artificial high-wavenumber features. These features are more dominant at this longer lead time compared to 24 hours in Fig. A7.

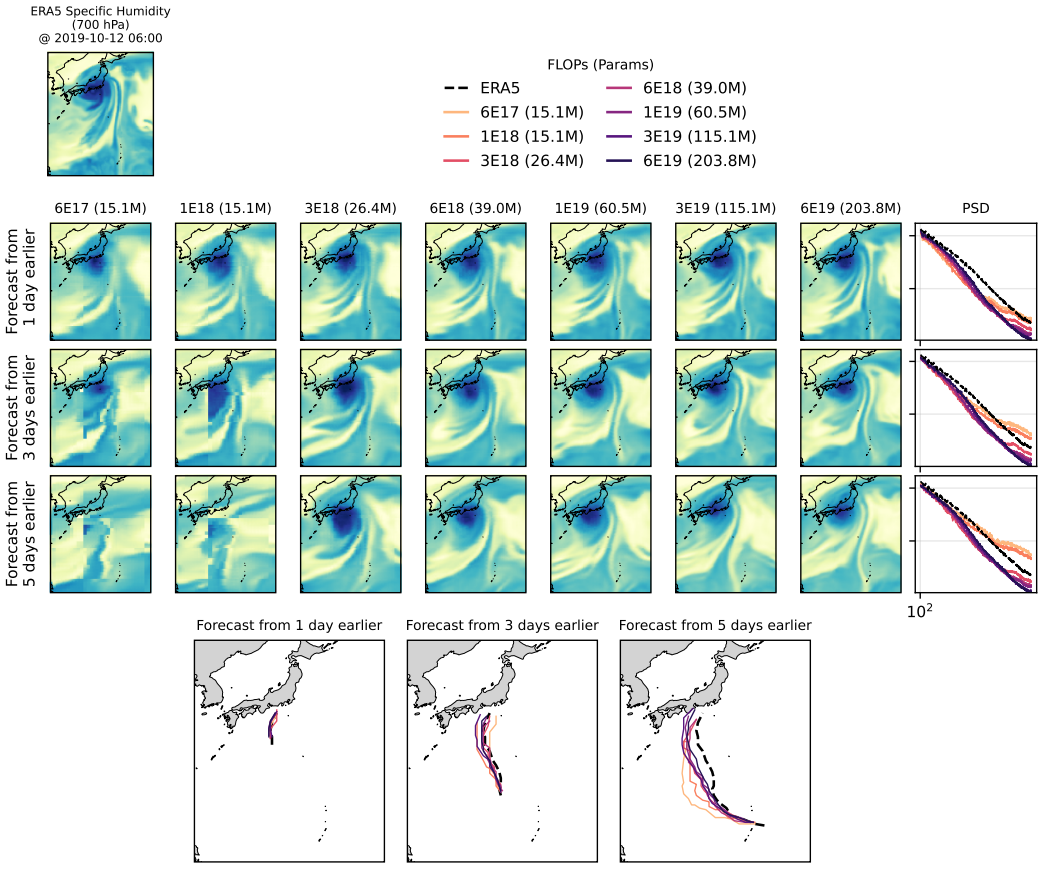


Figure A9: **Tropical Cyclone (TC) visualization and tracks before Typhoon Hagibis made landfall in Japan**: We reproduce the TC analysis of Price et al. [35]. We show the ERA5 ground truth for specific humidity at 700hPa on October 12, 2019, 06 UTC in the top left. The following three rows show the `Swin` predictions at the same time from all the closest compute-optimal models (between 6E17 and 6E19 FLOPs), initialized 1, 3, and 5 days earlier, respectively. The last column shows the PSD for the predictions (zoomed into the high wavenumbers). Forecasts initialized 1 day earlier show that all models capture the TC structure, although the larger models produce systematically sharper features, which is also reflected in the PSD. For initializations 3 and 5 days earlier, the smaller models fail to capture the TC structure, exhibiting patching artifacts and generally inconsistent blending of physical features. This behavior is likewise evident in the PSD as artificial high-wavenumber energy. In contrast, the larger models consistently capture the TC across all lead times. The last row shows the TC track from the different initializations. Although a deterministic model is insufficient for accurately assessing track performance, some variability among the different models is seen.

Table A4: **Patch size, embedding, depth ablations**. We vary patch size, embedding, and depth to keep models approximately on an IsoFLOP budget. All models are trained and cooled down to 20,000 iterations. Results show that (i) patch size 4 achieves the lowest loss, but the variance amongst patch sizes is not too high, and (ii) embedding and depth are largely interchangeable unless taken to extremes (very shallow or very deep), which increases loss.

Patch Size	Embedding	Depth	Window Size	TFLOPs/Iter	#Params	LR	Val. Loss
2	512	10	(18, 36)	20.0	31.79	1.0E-3	0.0085
2	384	18	(18, 36)	21.4	32.11	1.0E-3	0.0083
2	256	40	(18, 36)	23.5	31.65	1.0E-3	0.0086
2	768	5	(18, 36)	21.2	35.86	1.0E-3	0.0091
4	1024	12	(9, 18)	20.4	153.49	5.0E-4	0.0075
4	768	22	(9, 18)	21.2	157.62	5.0E-4	0.0074
4	1536	5	(9, 18)	19.2	145.25	5.0E-4	0.0086
4	512	48	(9, 18)	20.8	152.31	5.0E-4	0.0077
8	2048	12	(5, 10)	20.3	623.62	2.5E-4	0.0084
8	1536	22	(5, 10)	20.8	637.64	2.5E-4	0.0081
8	1024	44	(5, 10)	20.1	613.94	2.5E-4	0.0085
8	768	88	(5, 10)	20.6	630.45	2.5E-4	0.0089

consistent with undertraining or limited capacity. The two runs with window artifacts in Fig. A9 were cooled down before 10,000 iterations, while validation loss was still decreasing significantly; the same models trained longer do not show these artifacts. We also visualize the TC tracks. While all models follow the general track of ERA5, the smaller models show more variability at longer lead times due to artifacts developing. We note that, due to the deterministic modeling framework, these results should be interpreted as illustrative. Accurately tracking such phenomena, particularly at longer lead times, requires probabilistic (ensemble) forecasting, as the chaotic nature of atmospheric processes naturally introduces more spread (variance) in the predictions, particularly at longer lead times.

Patch size and model aspect ratio ablation. To determine the appropriate patch size for the neural scaling experiments, we conduct an ablation study in Tab. A4 in which we vary patch size, embedding, and depth to keep models approximately on an IsoFLOP budget. Smaller patches use fewer parameters due to increased compute from longer sequences, while larger patches use more parameters; embedding and depth are also balanced to consume similar compute where we can train wider or deeper models. The parameter count includes the patch embedders and head. We also adjust the window size to maintain constant physical window area, so larger patches use proportionally smaller windows—this makes the IsoFLOPs slightly approximate as seen in the TFLOP per iteration values. All models are trained and cooled down to 20,000 iterations. Our results show that (i) patch size 4 achieves the lowest loss, but the variance amongst patch sizes is not too high, and (ii) embedding and depth are largely interchangeable unless taken to extremes (very shallow or very deep), which increases loss. We find a slight preference for deeper models, differing from the preference for wider models in the smaller compute range studied by Yu et al. [45]. Our minimalist architecture and higher compute budgets are plausible explanations: narrow embeddings may impose a stronger information bottleneck at smaller scales, whereas additional depth may be more useful once that bottleneck is reduced. We do not isolate these effects here. Hence, we select patch size 4 for our scaling experiments. We note that resolution scaling remains an important future direction where this ablation is repeated across data, model sizes, and compute.

Neural scaling sensitivity. For the neural scaling law analysis, we fit a quadratic curve to the loss for each IsoFLOP setting as a function of log-parameters or log-samples (similar to standard neural scaling analyses [21]):

$$L = p(\log N)^2 + q(\log N) + r,$$

using a second-order polynomial fit (e.g., `polyfit` with degree 2). We include all loss values in the fit (no outlier removal) and use the resulting expression to estimate the local minimum. The quadratic-fit design matrices have condition numbers ranging from 13 to 798, indicating no severe numerical ill-conditioning. The coefficient of determination (R^2) ranges from 0.951 to 0.998 across the first eight IsoFLOP settings; the two highest budgets, 3E19 and 6E19, yield $R^2 = 0.883$ and 0.836, respectively. Although these values still indicate strong fits, the decrease suggests increasingly similar behavior among larger models and longer training runs. This trend is also reflected in the

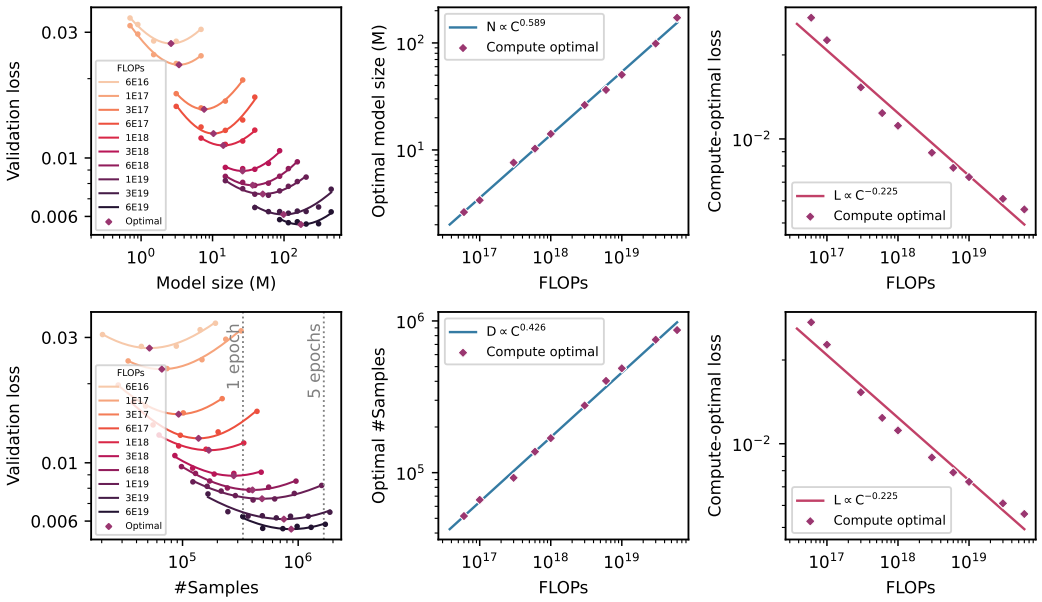


Figure A10: **Neural scaling with channel weighting loss function.** We cool down all models using the GraphCast channel weights in the loss function that includes both weights based on pressure level as well as surface variable weighting. We find that the overall exponents do not change too much (see legends for each scaling law). The compute-optimal models are all approximately the same. Loss values have slightly improved consistently due to the channel weights (also visible in the y-range).

fitted coefficients: the quadratic term p decreases with increasing compute, corresponding to flatter parabolas; a phenomenon also observed in LLMs [16]. The power-law fits to the compute-optimal points yield $R^2 = 0.9974$ for model size and $R^2 = 0.9668$ for loss, evaluated in log–log space.

Neural scaling across multiple epochs. We note that multiple epochs are necessary for a robust scaling analysis on this dataset. From Fig. 1, we observe that for many IsoFLOP settings, the characteristic U-shape of the parabola becomes visible only when including data points beyond one epoch. Restricting the analysis to less than one epoch limits the number of IsoFLOP regimes where a clear U-shaped curve can be identified. In such cases, only the left portion of the parabola is observed, requiring *extrapolation* to estimate the optimal point. This significantly reduces the stability and reliability of the analysis. Therefore, we emphasize the use of multiple epochs to ensure that the parabolic trend is captured through interpolation across observed data points. Another challenge arises when reducing compute budgets below 6E17 in order to obtain more points within the sub-epoch regime. This necessitates training smaller ($<$ 1M parameter) models. However, such small models require many iterations to reach a given compute budget, often exceeding one epoch and thereby forcing the use of even smaller models. To capture both sides of the parabola under these constraints, larger models must be trained for very short durations (on the order of $\sim$1000 iterations), which approaches the warmup period (500 iterations). As a result, the models operate in an *undertrained* or *small-model* regime, where training dynamics and architectural choices (e.g., very shallow depths) have a disproportionate impact. This makes it challenging to reliably probe smaller compute budgets. For these reasons, we focus on a compute range spanning three orders of magnitude, from 6E16 to 6E19, where stable and consistent scaling trends can be observed. With larger and more diverse datasets (e.g., spanning multiple Earth science domains or cross-domain corpora), it may be possible to extend this range further, similar to LLMs.

Variable-specific scaling. All 71 channels exhibit clear compute scaling, with strong loss–compute fits (median $R^2 = 0.969$, minimum 0.924). For $L \propto C^{\gamma}$, wind and temperature have exponents ($\gamma \approx -0.24$ to -0.25) relatively close to the aggregate loss exponent of -0.217. Pressure and geopotential scale fastest (-0.39 to -0.43), while moisture generally scales more slowly (-0.19 to -0.21). The smoother spatial structure of geopotential and the fine-scale structure of moisture are plausible explanations, but we do not establish their causes here. Vertical variation in scaling

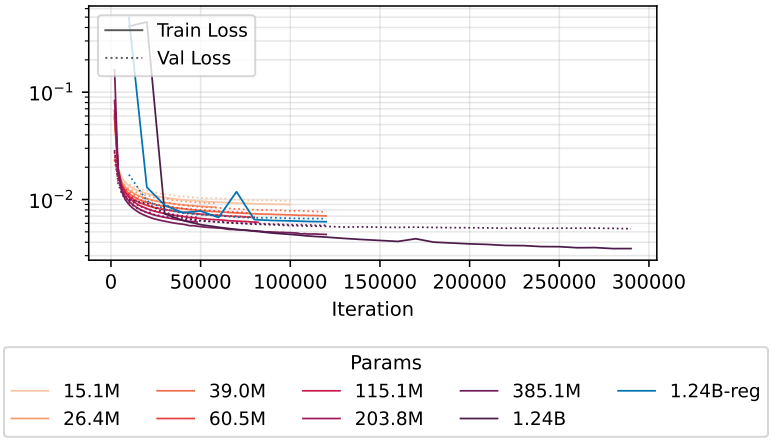


Figure A11: **Training and validation loss as a function of iterations for different models:** We show the training curves for the closest compute-optimal models, including the 385M intermediate-budget run and the 1.24B extrapolation run. Solid and dotted lines indicate training and validation loss, respectively. The extrapolation runs develop train–validation gaps as validation improvements diminish. The blue curve (1.24B-reg) uses stronger regularization (dropout 0.1 and weight decay 0.1), reducing the train–validation gap but plateauing at a higher validation loss; training was stopped after approximately 120K iterations.

rates is modest for wind and geopotential but more pronounced for temperature and humidity; across model sizes, the distribution of error across pressure levels remains qualitatively similar. These exploratory findings suggest differences in variable-dependent predictability and could motivate channel weighting during cooldown based on scaling rates. Establishing the roles of spatial or temporal resolution, physical processes, and observational uncertainty requires a dedicated analysis, which we leave to future work.

Neural scaling sensitivity to channel weighting. We select channel weighting as a popular training strategy to reduce the loss (as observed in Tab. A3). We demonstrate that neural scaling can be repeated with channel weighting in the cooldown period. We repeat the full IsoFLOP curves with channel weighting in Fig. A10 where each model is cooled down with channel weights. We observe that while the validation loss has improved uniformly across all the models (due to channel weighting), the compute-optimal trends are very similar. The exponents for optimal model size vs. compute have changed from 0.584 to 0.589 with slightly different scaling constant, and for loss vs. compute has changed from -0.217 to -0.225. Hence, the exponents can be a little sensitive to the choice of loss, but the larger point remains: the models still show compute-optimal scaling behavior over three orders of magnitude of compute.

Extrapolation analysis. The $385M$ and $1.24B$ extrapolation runs fall short of their predicted losses (Fig. 5) after approximately 6 and 14 epochs, respectively, with train–validation gaps consistent with overfitting in this data-reuse regime. We show this in Fig. A11 where for all the models in the scaling regime (6E17 to 6E19) the training and validation loss track closely. However, for the models extrapolated to 3E20 FLOPs and 2.25E21 FLOPs, while the training loss reaches close to the extrapolated loss values, the validation loss plateaus at around 0.005, pointing to significant overfitting. We also trained a 1.24B model with stronger regularization (dropout 0.1 and weight decay 0.1). This reduced the train–validation gap, but validation loss plateaued near 0.006, slightly above the original run, so we stopped training after approximately 120K iterations. Stronger regularization therefore reduced overfitting without recovering the extrapolated scaling trend. Together with the intermediate-budget result, this supports finite data as the most likely limitation in our experiments, although gains from further hyperparameter tuning cannot be ruled out.

Results for more variables. We show the comparison with GraphCast and HRES for more variables in Fig. A12, Fig. A13, and Fig. A14, where we see that the 8-step `AR Swin` is similar in performance to GraphCast. We also show the performance with compute for all the additional variables in Fig. A15 and Fig. A16, with similar behavior as the variable subset.

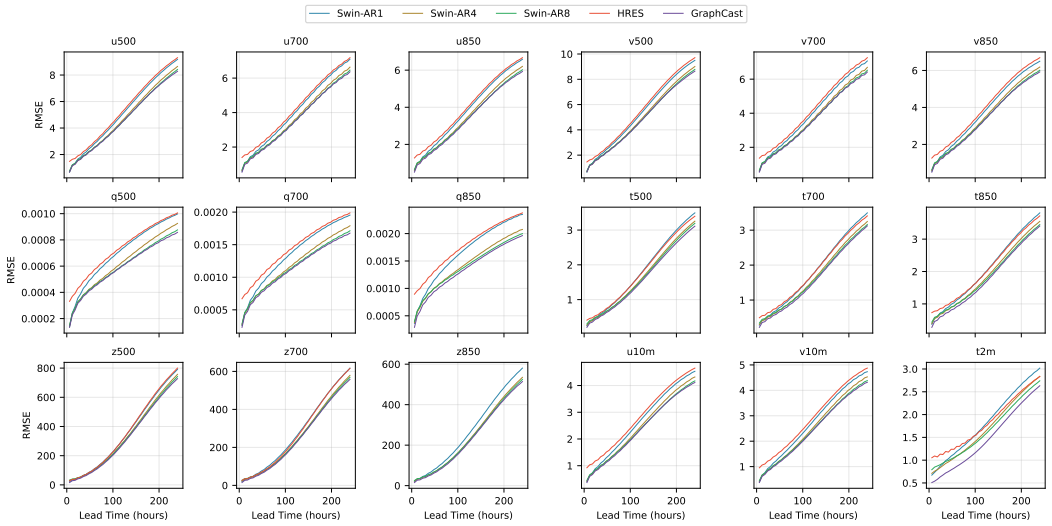


Figure A12: **RMSE vs time for a larger subset of variables.** Reproducing Fig. 7 with more variables. Generally, we observe that the single step 204M `Swin` (compute-optimal 6E19 FLOPs) outperforms HRES. With 4-step `AR`, the `Swin` starts to approach GraphCast. With 8-step `AR` cooldowns instead, the performance is similar to GraphCast. We note that (i) there was no data for z850 on HRES from gcsfs and (ii) the `Swin`'s performance on only 2m temperature is sub-optimal, although it shows similar behavior of approaching GraphCast performance with more `AR`. This motivates custom channel weighting (different from the tuned weights for GraphCast) for specific variables.

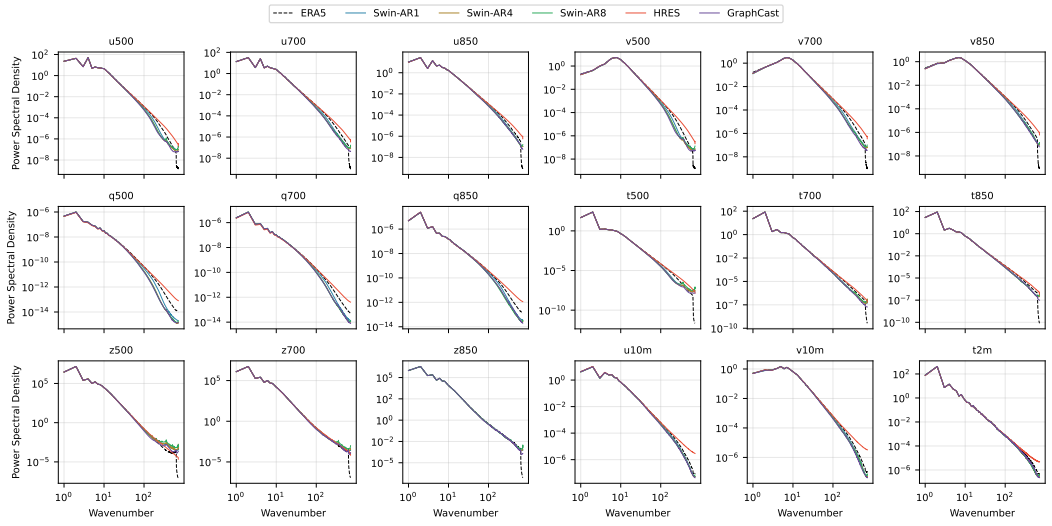


Figure A13: **PSD for a larger subset of variables at 24-hour lead time.** As with the main paper figures, we observe that with more `AR` the `Swin` also achieves the lower RMSEs observed in the above Fig. A12 through a smoother forecast. This is more visible at longer lead times (see next figure).

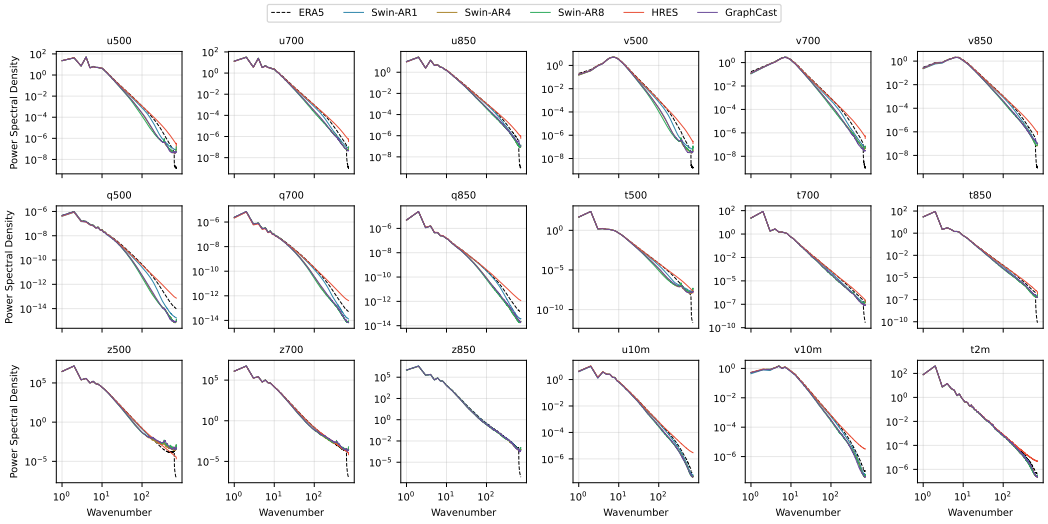


Figure A14: **PSD for a larger subset of variables at 120 hour lead time.** At 120h, we can clearly see how both GraphCast and `Swin-AR8` over smooth to achieve the low forecast errors, thereby emulating an ensemble forecast. This is particularly visible in the specific humidity variables such as q700 (second row, second column). The one-step `Swin` shows lesser smoothing.

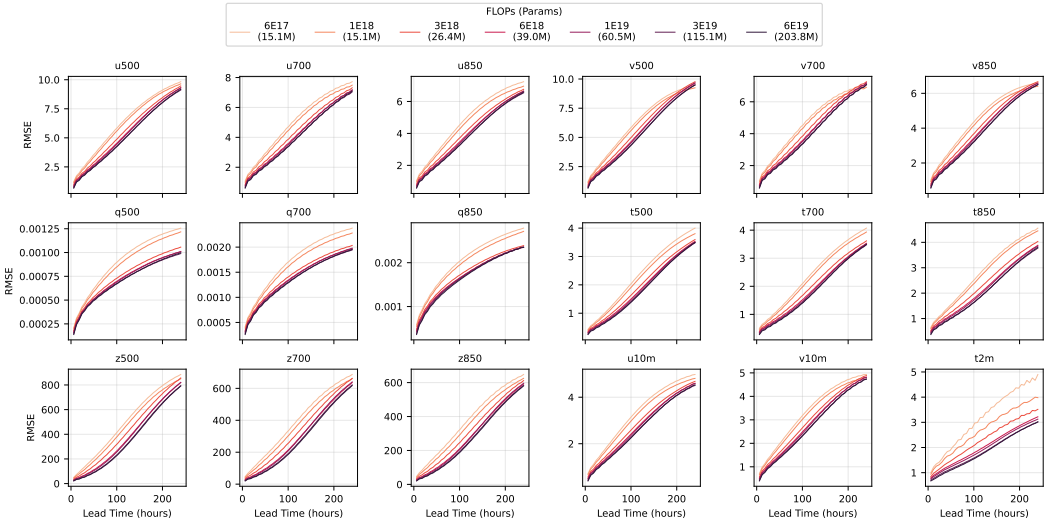


Figure A15: **RMSE performance of the closest compute-optimal `Swin` models at each compute budget for a larger subset of variables.** We show Fig. 6 for more variables.

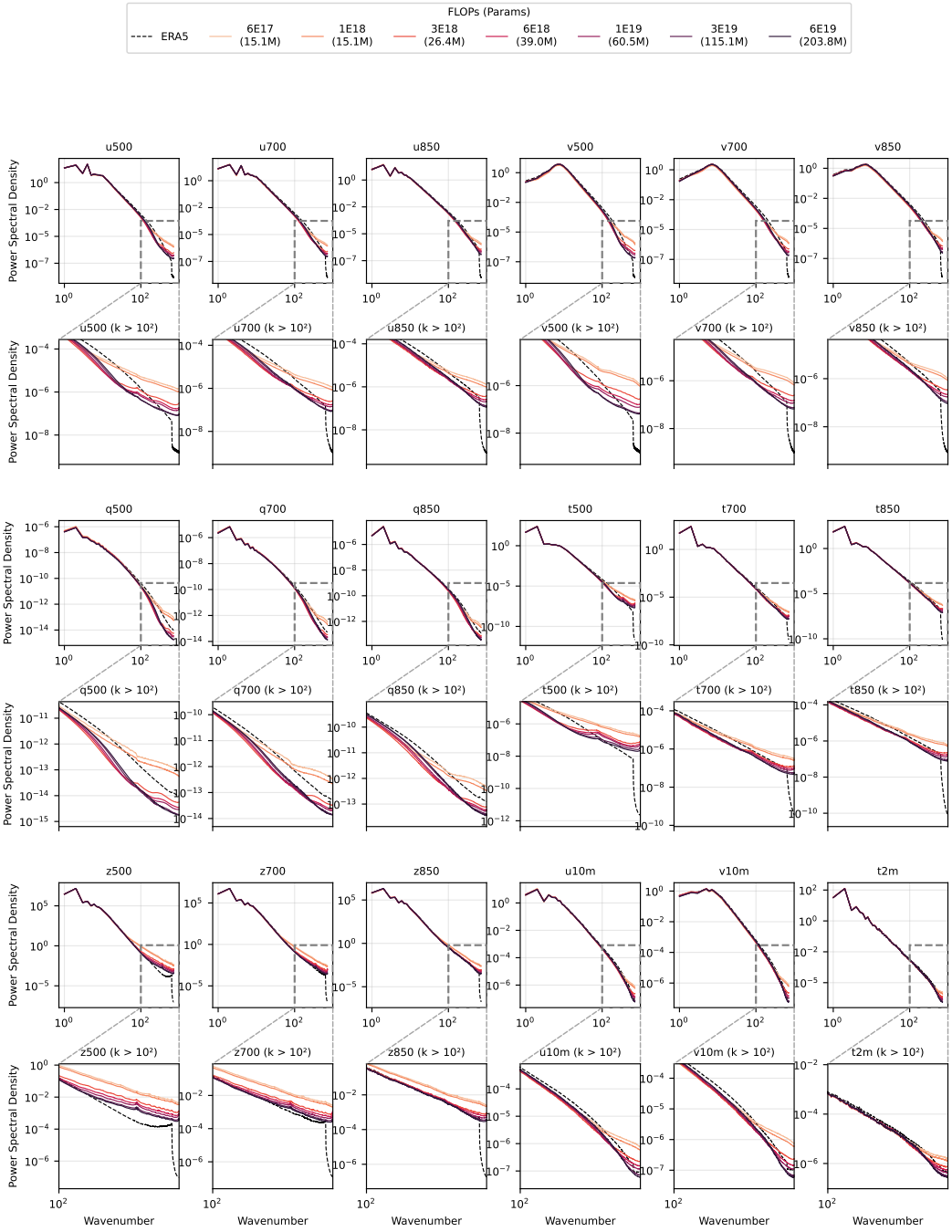


Figure A16: **PSD performance at lead time 120 hours of the closest compute-optimal `Swin` models at each compute budget for a larger subset of variables**. We show Fig. A8 for more variables.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: We demonstrate in the paper's main text (and appendix) that the minimalist Transformer backbone (see §3) shows clear compute-optimal neural scaling behavior (see §4) across three orders of magnitude of compute with scaling laws derived. We demonstrate that continual training with cooldowns are beneficial for efficient scaling as well as downstream alignment (see §4). We also extrapolate this model to high compute budgets to demonstrate the limits of scaling due to data exhaustion (see §4). Our code is open-sourced as well.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Limitations are discussed in the limitations section §A.1. Computational details are discussed in §3, with more details on the algorithms, compute, and scalability in §A.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

   Justification: The paper is experimental; no theory / proofs.

   Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: All model and training details are discussed in §3 with hyperparameters and list of models run in Tab. A1 and computational details in §A. The code with a detailed README is open-sourced at `https://github.com/ShashankSubramanian/neural-scaling-weather`. ERA5 data is open.

   Guidelines:

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

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

   Answer: [Yes]

   Justification: All model and training details are discussed in §3 with hyperparameters and list of models run in Tab. A1 and computational details in §A. The code with a detailed README is open-sourced at `https://github.com/ShashankSubramanian/neural-scaling-weather`. ERA5 data is open. Benchmark data are pulled from Google Cloud and details are discussed in §3.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: All model and training details are discussed in §3 with hyperparameters and list of models run in Tab. A1 and computational details in §A. The code with a detailed README is open-sourced at `https://github.com/ShashankSubramanian/neural-scaling-weather`. ERA5 data is open. Benchmark data are pulled from Google Cloud and details are discussed in §3.

   Guidelines:

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

7. **Experiment statistical significance**

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

Answer: [Yes]

Justification: Neural scaling goodness of fits, conditioning of the fits with R^2 values are presented in §A under "Neural scaling sensitivity". We also evaluated sensitivity of exponents w.r.t choice of loss function in the same section. Error bars around the full neural scaling is computationally expensive and would require repeating the whole analysis multiple times—this is not practical and is not common for neural scaling analysis, in general.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: The compute resources, parallelism used, memory, and time are detailed in §A.3.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: Our work does not involve any potential harm, societal impact, or violation.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.

- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

    Answer: [N/A]

    Justification: Our primary contributions are methodological. We use public data and benchmarks. Our contributions are in furthering the understanding of neural weather models. Improvements in neural weather models may have broader societal implications, such as informing early warning systems and weather risk assessments. These are appropriately discussed in §1.

    Guidelines:

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

11. **Safeguards**

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

    Answer: [N/A]

    Justification: This paper uses public data and benchmarks. Comparisons with existing models are done by clearly citing the original research.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: The paper used public data and benchmarks and they are clearly cited.

    Guidelines:

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

13. **New assets**

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

    Answer: [Yes]

    Justification: Our code is open-sourced with a well-documented README at `https://github.com/ShashankSubramanian/neural-scaling-weather`.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: No crowdsourcing or human subjects.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: No potential risk for research with human subjects.

    Guidelines:

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

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: LLMs were used only for minor editing purposes of this paper.

    Guidelines:

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