FourCastNeXt: Improving FourCastNet Training with Limited Compute

Edison Guo Maruf Ahmed Rahul Mahendru Yue Sun Rui Yang

Harrison Cook Tennessee Leeuwenburg Ben Evans

1. Introduction

Recently, FourCastNet (Pathak et al., 2022) has shown impressive results on predicting various variables on ERA5. While FourCastNet enjoys quasi-linear time and memory complexity in sequence length compared to quadratic complexity in vallina transformers, training FourCastNet on ERA5 from scratch still requires large amount of compute resources which is expensive or even inaccessible to average researchers. In this work, we will show improved methods that can train FourCastNet using only 1% compute required by the baseline, while maintaining model performance on par or even better than the baseline. In this technical report, we will provide technical details of our methods along with experimental results and ablation study of different components of our methods. We term our improved model FourCastNeXt, in a similar spirit to ConvNeXt (Liu et al., 2022). To summarize, our main contributions are as follows:

- Large training set
- Deep-norm initialization
- Smaller embedding patch size
- Learning the temporal flow field
- multi-step fine-tuning

2. Methods

In this section, we discuss our main contributions to improve FourCastNet. We will show that these improvements better condition the model and the training process, which enable the training process to reach the same level of error faster than the baseline using a fraction of compute resources.

2.1 Large training set

The baseline FourCastNet was trained on a fixed set of 50k-ish training examples sampled from the raw ERA 5 dataset. This amount of training samples, however, is only a small fraction of the raw dataset. Instead of having a fixed training set, we expand the training set by employing the following data augmentation strategies to construct the training examples on the fly. In other words, we do not pre-compute the training set and store them on disk but generate the training examples on the fly by reading the raw ERA 5 files.

1) Randomly picking time steps

Our training set spans between 1959 and 2017. The time steps are evenly spaced for an interval of 6 hours. For each training example, we randomly pick a time step.

2) Random cropping over space

ERA 5 variables have a spatial size of 721 x 1440. We randomly crop a region with a size of 640 x 1280.

A theoretical estimation suggests that the above data augmentation strategies would result in (2017-1959 + 1) x 4 x 365 x (721-640+1) x (1440-1280+1) = 1,137,220,280 training examples, which is several orders of magnitude larger than the baseline training set. Several works (Hoffmann et al.,2022, Touvron et al., 2023) have shown that a large training set will help the model generalize better to unseen examples.

We also noticed that the filesystem is not fast enough to keep up with the GPUs when we construct the training examples on the fly. To improve GPU utilization, we implemented our data loader on a Ray cluster that scales the data workers out of the GPU nodes.

2.2 Deep-norm initialization

The baseline FourCastNet used pre-norm for its AFNO block. Pre-norm is known to stabilize the early training but would result in worse performance than post-norm (Wang et al.,2022). Fortunately, recent work such as deep-norm initialization (Wang et al.,2022) has developed methods that can stabilize the training for very deep post-norm transformers while retaining the performance of post-norm transformers. To take advantage of deep-norm initialization, we modify FourCastNet to use post-norm for the residual branches of its AFNO blocks and apply deep-norm initialization to the model weights. We will show in ablation study that deep-norm initialization indeed helps stabilize the early training of the post-norm version of FourCastNet.

2.3 Smaller embedding patch size

The baseline FourCastNet used an 8x8 patch size for input embedding. We suggest that using a smaller patch size such as 4x4 (Liu et al., 2021) should give superior performance. Our rationale is that a typical vision transformer trained for an image classification task such as ImageNet only predicts a single scalar for the entire input image where an 8x8 or even 16x16 patch size (Dosovitskiy et al.,2021) might give good result. In contrast, the prediction of FourCastNet has the same resolution as the input images, which requires the model to capture fine-grained detailed features where 8x8 patch size might not be suitable. Our ablation study showed that this is indeed the case.

2.4 Learning the temporal flow field

The baseline FourCastNet directly predicts the outcome of the next time step from the current time step as input. This implies that the model is responsible for learning the temporal dynamics between the time steps which might not be trivial in the early training phase. Often, the temporal dynamics might only result

in slight changes between the neighbor time steps. The slight changes, for example, might be a shift of a few pixels in the wind velocity fields between the neighbor time steps. If we can capture those slight temporal deformations as much as possible, the main model will only be responsible for learning the temporal residuals after the deformations, which lowers the learning curve especially at the early training phase. To realize this insight, we proposed the following method inspired from optical flow:

value, flow = FourCastNet(input)

output = value + temporal_warp(input, flow)

Value and flow are computed via separate heads. We also initialize the flow head to near-zero weights which results in near identity warping at the start of training. In other words, the model initially learns from the pixel-wise difference between the neighbor temporal steps. Our ablation study showed that this method dramatically speeds up the training convergence and achieves better final error.

2.5 multi-step fine-tuning

The baseline FourCastNet was trained on single time steps. When predicting multiple time steps into the future, the prediction errors tend to accumulate over time. We propose to fine-tune the model on multiple time steps after the training on the single time steps. Concretely speaking, we employ a curriculum learning strategy that sequentially fine-tunes for more time steps. However, we noticed that naively fine-tuning the model for more time steps has a catastrophic forgetting problem that the error of early time steps increases quickly after the fine-tuning. To prevent the catastrophic forgetting problem, we designed the following fine-tuning algorithm that incorporates a prior preservation loss (Ruiz et al., 2023) via a frozen teacher model:

Initialize the teacher model to the one pre-trained on single time steps.

Initialize the student model to the one pre-trained on single time steps.

Freeze the weights of the teacher model

for curr_step = 1 .. max_time_steps

    while student_model not converged:

        teacher_step = randomly sample between 1 and curr_step

        loading training example obs1 at t(1), obs2 at t(teacher_step-1) and target at t(teacher_step)

        teacher_output = auto-regressive run teacher_model(obs1) to teacher_step – 1

        multi_step_loss = loss(student_model(teacher_output), target)

        single_step_loss = loss(student_model(obs2), target)

        loss = multi_step_loss + single_step_loss

    teacher_model = student_model

freeze the weights of the teacher model

As can be seen from the above algorithm, the *single_step_loss* preserves the prior knowledge on the single steps. The *multi_step_loss* preserves the prior knowledge on the previous time steps because we randomly sample a step between 1 and *curr_step*.

3. Experiments and Results

3.1 Training Details

We train our model in two stages. The first stage is to pre-train the model on single time steps for a maximum of 40,000 steps. Once the first stage is complete, we fine-tune the model for 2000 steps for each increment of prediction steps until a maximum step of 4. We use LAMB optimizer (You et al., 2020) with cosine learning rate schedule for all our training stages. The initial learning rate for pre-training is 3e-3 and the final learning rate is 3e-4. The initial learning rate for fine-tuning is 1e-4 and the final learning rate is 1e-5. The total batch size follows a schedule (Chowdhery et al., 2022) that starts with 4 and doubles from step 12000. The model is trained with Pytorch Lightning using Distributed Data Parallel (DDP) strategy with fp16 mixed precision. Our training cluster consists of four V100 GPUs with 32GB memory each. The total training walltime including both single-step and multi-step is about 35 hours.

In comparison, the baseline FourCastNet was trained on a cluster of 32 A100 GPUs with 80GB memory each. An A100 has a total theoretical performance of 624TFLOPS (NVIDIA, 2020) which is 5 times V100 with a total theoretical performance of 125TFLOPS (NVIDIA, 2017). Therefore, training 48,000 steps on four V100 only consumes about 1% of the compute resources required to train the baseline FourCastNet.

3.2 Results

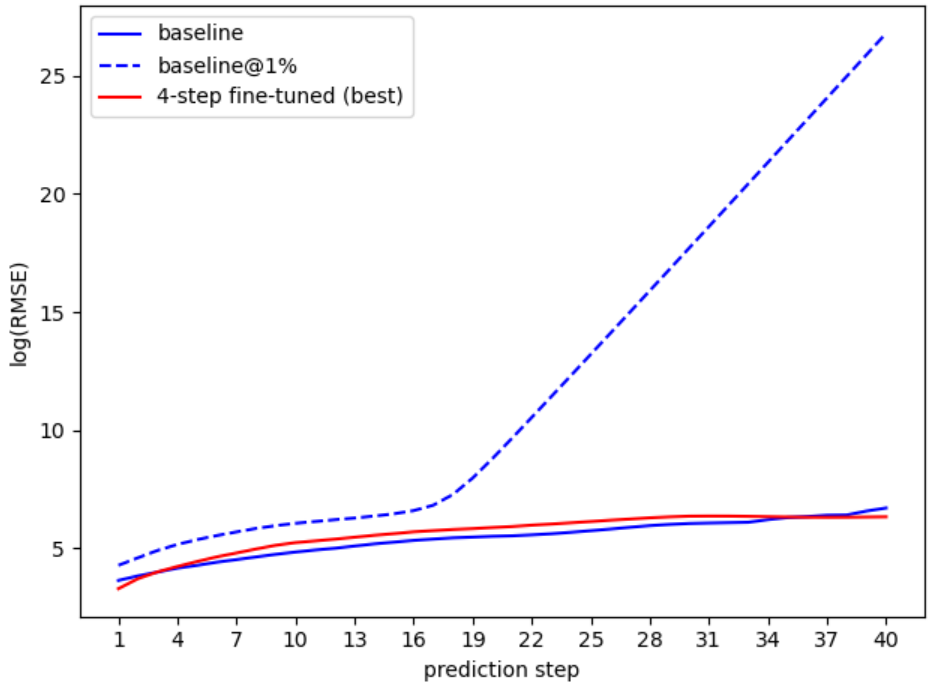
We report all our results from the checkpoint of the final training step. For validation, we compute root mean squared error (RMSE) for both baseline and our model for year 2018 as the metric for model performance. To compute the validation RMSE, we firstly un-standardize the network predictions by multiplying the standard deviations followed by adding the means. Then, we use *torch.sqrt(F.mse_loss(predictions, ground_truth))* to compute the RMSE. The means, the standard deviations and the pretrained weights of the baseline model are provided by NVLab.

Let:

- *baseline* denote the model pretrained by NVLab.
- *baseline@1%* denote the baseline model trained using 1% compute.
- *single-step pretrained* denote the model trained on single steps using our improved methods and 1% compute.
- *4-step fine-tuned (best)* denote our model fine-tuned 4 steps after the single-step pretraining, which is currently our best model.

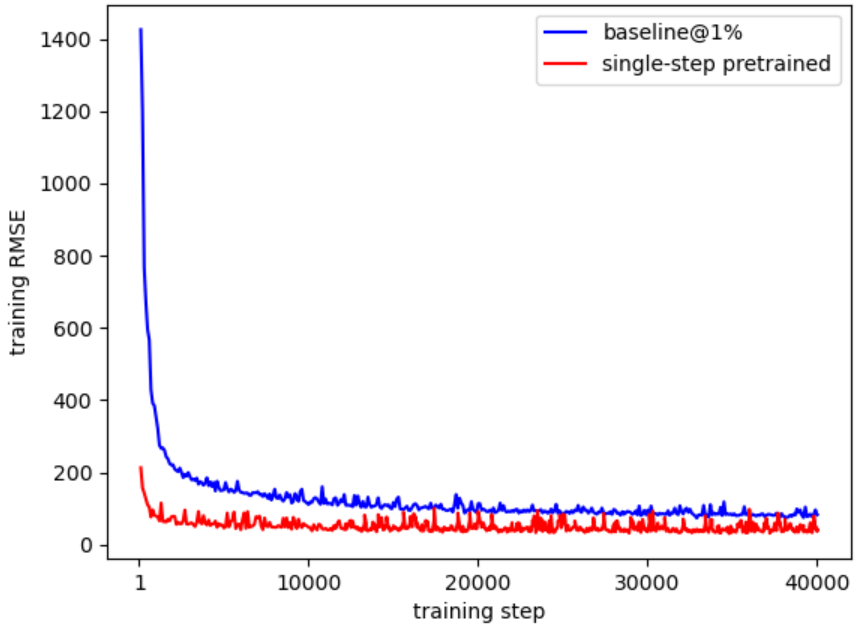
3.2.1 Training the baseline model with 1% compute (baseline@1%)

For a quick sanity check, we trained the baseline model with 1% compute. We observed that the predictions of baseline@1% diverged from step 16. The following log-plot of RMSE shows the divergence. We have to plot on the logarithmic scale because the RMSE of baseline@1% grows exponentially.



In contrast, *4-step fine-tuned (best)* closely follows the baseline RMSE and shows lower RMSE at the beginning and the end of the prediction steps.

We also plot the training curves for *baseline@1%* and *single-step pretrained* as follows:



As can be seen from the above training curve plot, the training RMSE for *single-step pretrained* is consistently lower than *baseline@1%* throughout the entire training process, which demonstrates the effectiveness of our improved methods.

3.2.2 Comparisons between *baseline* and *4-step fine-tuned (best)*

Single-step comparison

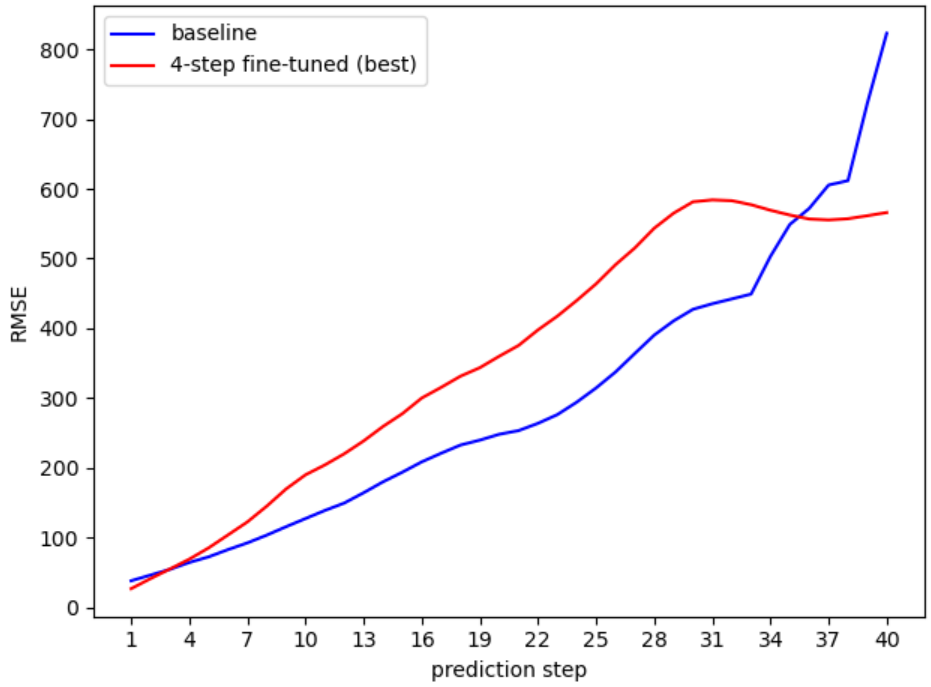
ERA5 Variable	Baseline (RMSE)	4-step fine-tuned (best)
10u	0.531046	0.564317
10v	0.526364	0.562536
2t	0.755716	0.759327
sp	137.290480	**43.658535**
msl	38.836962	46.771553
t850	0.512118	0.525746
u1000	0.582749	0.616646
v1000	0.578866	0.614807
z1000	29.882532	36.547699
u850	0.792813	0.820308
v850	0.775644	0.805753
z850	26.972977	34.094856
u500	1.070337	1.150244
v500	1.083850	1.157692
z500	33.929530	43.161873

t500	0.399981	0.427304
z50	79.489610	**78.297844**
r500	6.551108	6.946574
r850	5.525181	5.582663
tcwv	0.821443	0.825033
Average RMSE	38.423431	**27.084917**

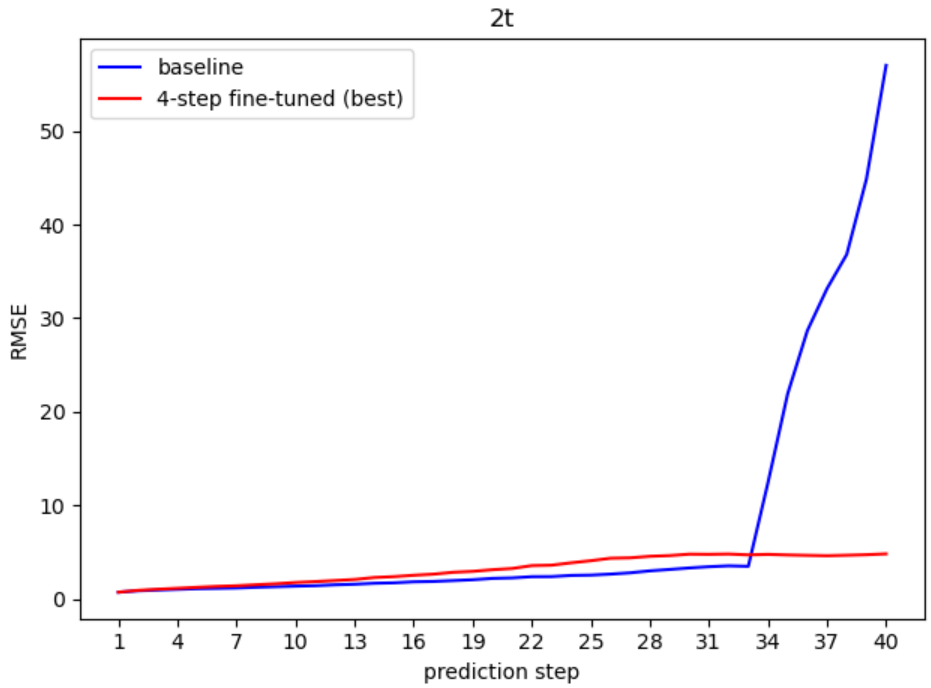
The single-step comparison suggests that the performance of our model is mostly on par with the baseline for most of the variables. Variable *sp* and *z50* outperform the baseline which leads to lower average RMSE compared to the baseline.

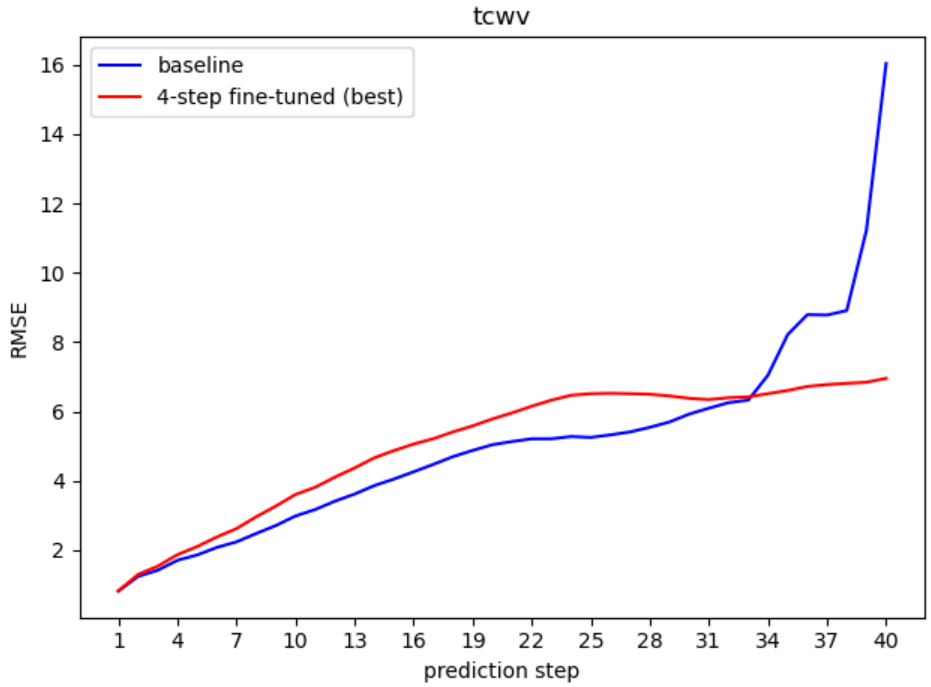
Multi-step comparison

The following plot is the average RMSE over all the predicted ERA5 variables versus the prediction steps.



As can be seen from the above plot, the average RMSE of the baseline has tendency to grow quickly in the last a few steps. We also checked the RMSEs for a few individual variables and they show the same pattern. The following are a couple of examples that demonstrate this pattern. Note that we do not cherry-pick these examples. The pattern is generally applicable to other variables too.

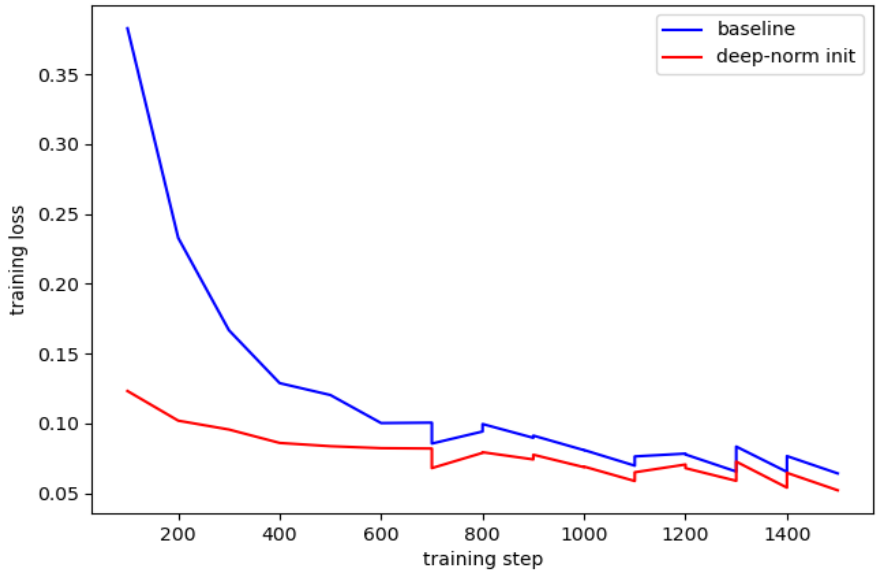




4. Ablation Study

4.1 Deep-norm initialization versus baseline

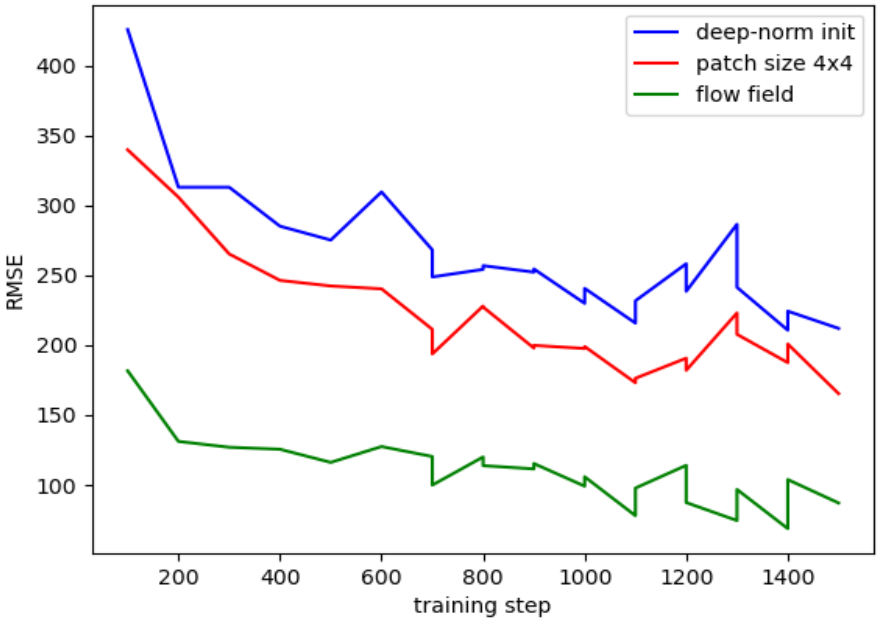
The following plot is the training loss of deep norm initialization versus the baseline.



As can be seen from the above plot, deep-norm initialization stabilizes the training at early phase and speeds up the training process by reaching the same level of training loss faster than the baseline.

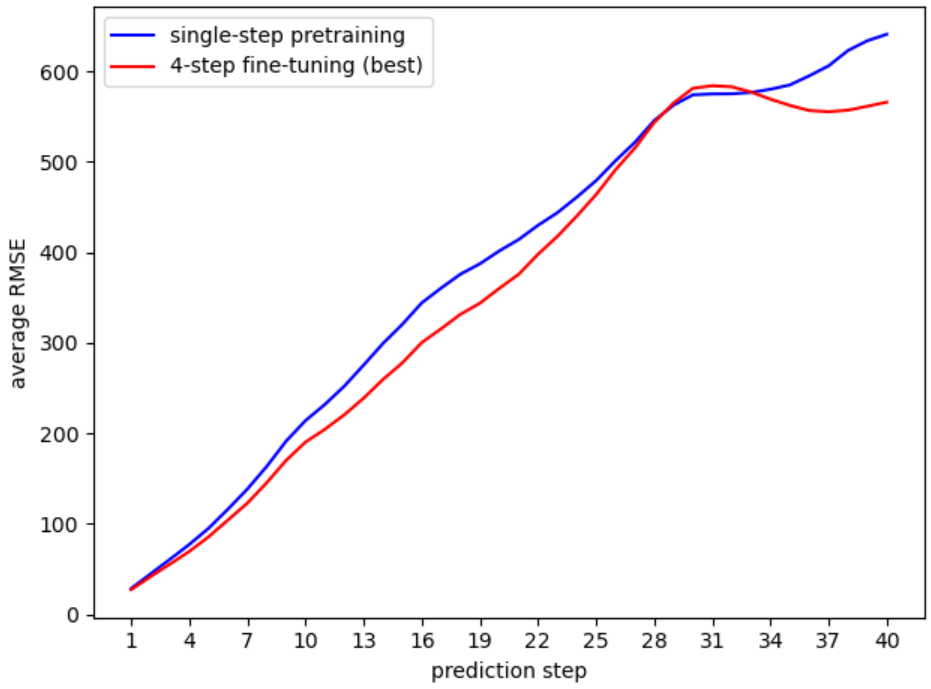
4.2 Patch size 4x4 and flow field

In this section, we will show that a smaller patch size of 4x4 and flow field further speed up the training in addition to the deep-norm initialization.



As can be seen from the above plot, small patch size and flow field consistently outperform deep-norm initialization. In addition, Flow field yields the largest reduction in RMSE, which confirms our hypothesis discussed in section 2.4.

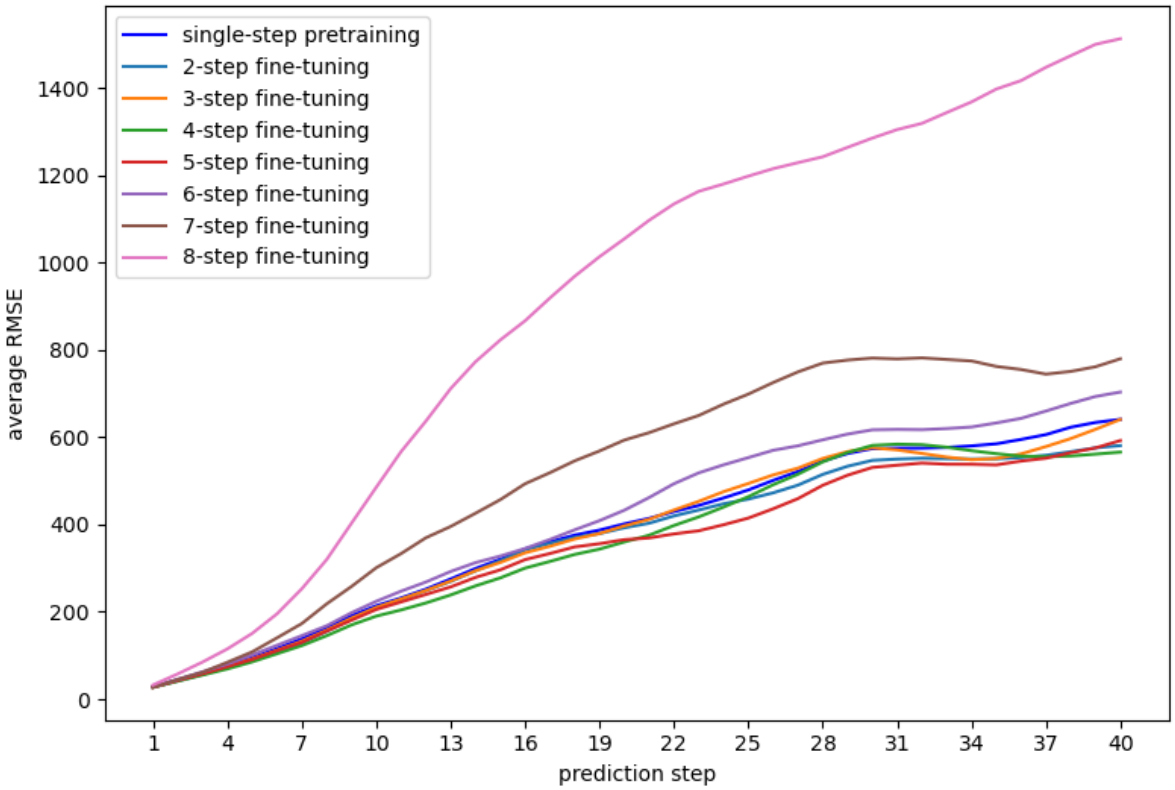
4.3 Multi-step fine-tuning versus single-step pretraining



As can be seen from the above plot, multi-step fine-tuning consistently improves the average RMSE for predictions over a long horizon.

4.4 Fine-tuning for different number of steps

We attempted to fine-tune the model for more than 4 steps reported in section 3.1. A plot of average RMSE versus prediction steps for fine-tuning different number of steps is as follows:



Fine-tuning more steps does not yield better results. We attribute this phenomenon to the fact that the model architecture lacks the ability to capture long-range temporal dependencies.

4. References

Jaideep Pathak, Shashank Subramanian, Peter Harrington, Sanjeev Raja, Ashesh Chattopadhyay, Morteza Mardani, Thorsten Kurth, David Hall, Zongyi Li, Kamyar Azizzadenesheli, Pedram Hassanzadeh, Karthik Kashinath, Animashree Anandkumar. FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators. *arXiv:2202.11214*. 2022

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell and Saining Xie. A ConvNet for the 2020s. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022

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, Laurent Sifre. Training Compute-Optimal Large Language Models. *arXiv:2203.15556*. 2022

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard

Grave, Guillaume Lample. LLaMA: Open and Efficient Foundation Language Models. *arXiv:2302.13971*. 2023

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, Furu Wei. DeepNet: Scaling Transformers to 1,000 Layers. arXiv:2203.00555. 2022

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv:2010.11929*. 2021

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, Baining Guo. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. *arXiv:2103.14030*. 2021

Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, Kfir Aberman. DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation. *arXiv:2208.12242*. 2023

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, Cho-Jui Hsieh. Large Batch Optimization for Deep Learning: Training BERT in 76 minutes. *arXiv:1904.00962*. 2020

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, Noah Fiedel. PaLM: Scaling Language Modeling with Pathways. *arXiv:2204.02311*. 2022

NVIDIA. V100 datasheet. *https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf*. 2017

NVIDIA. A100 datasheet. *https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf*. 2020