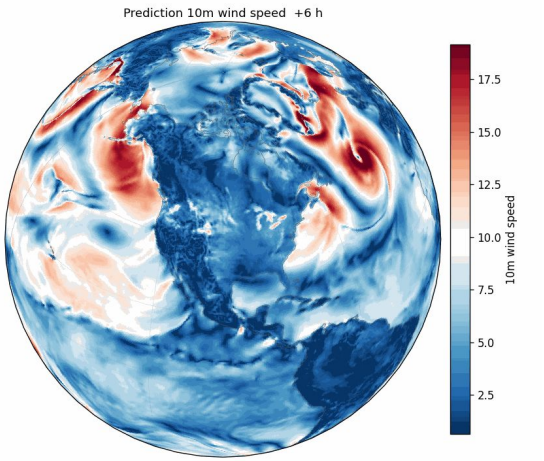
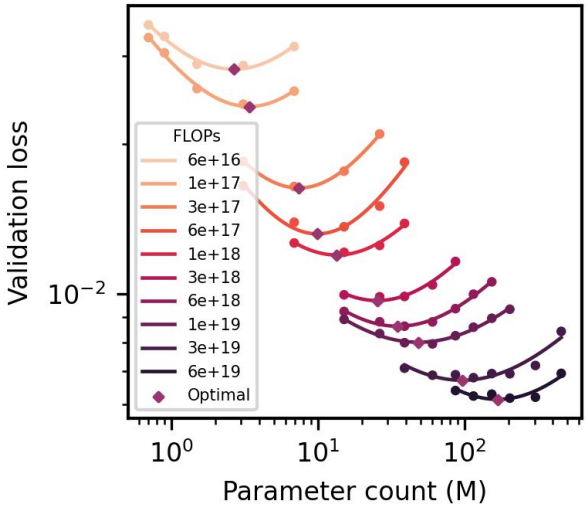




On Neural Scaling Laws for Weather Emulation Through Continual Training

SHASHANK SUBRAMANIAN, ALEXANDER KIEFER, ARNUR NIGMETOV,
AMIR GHOLAMI, DMITRIY MOROZOV, MICHAEL W. MAHONEY

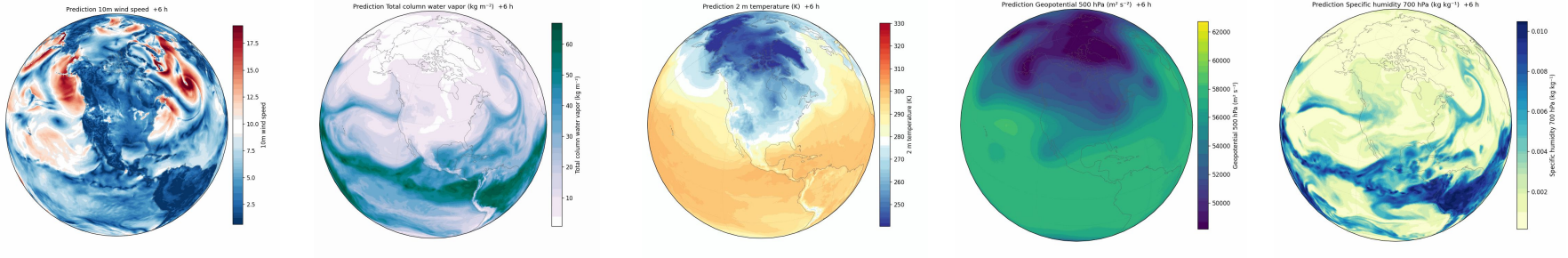
BERKELEY LAB

NERSC

OAK RIDGE National Laboratory

Why Neural Scaling for Weather?

- AI models can match the accuracy of numerical models
  - Orders of magnitude faster (usually a single GPU in inference)
- Large, but fragmented and expanding design space
  - More architectural (and engineering) complexity (neural operators, graphs, transformers), training / fine-tuning strategies, loss functions
  - Increasing cost: growing model scales (billions of parameters), spatiotemporal scales (example: decreasing patch size)



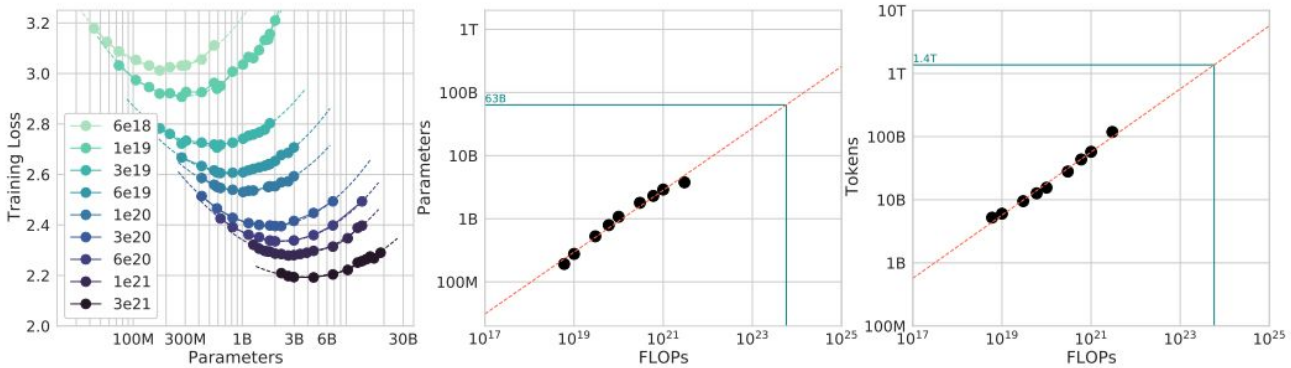
Why Neural Scaling for Weather?

- AI models can match the accuracy of numerical models
  - Orders of magnitude faster (usually a single GPU in inference)
- Large, but fragmented and expanding design space
  - More architectural (and engineering) complexity (neural operators, graphs, transformers), training / fine-tuning strategies, loss functions
  - Increasing cost: growing model scales (billions of parameters), spatiotemporal scales (example: decreasing patch size)

- ❑ Are we learning about weather (or science, more generally) or architecture idiosyncrasies?
- ❑ How far can simple architectures be pushed through scale?

Why Neural Scaling for Weather?

- Neural scaling: Builds trust by systematically scaling data, model, and compute, so that none of them saturate
- Frontier AI models are driven through neural scaling
- Actionable for efficient resource planning



Training Compute-Optimal Large Language Models. Hoffmann et. al, DeepMind, 2022

Challenges for Neural Scaling in Weather

- ➔ Architecture and training design space is complex and fragmented
  - ◆ Architecture confounders can interfere with scaling analyses
- ➔ Neural scaling is computationally expensive
  - ◆ Many models need to be trained from scratch across different budgets
- ➔ Downstream task differs from pre-training objective
  - ◆ May perform well on single step prediction but not translate downstream
- ➔ “Compute-optimal” behavior is unclear for weather models
  - ◆ Models may not show clean scaling behavior

Our Contributions

➔ Architecture and training design space is complex and fragmented

1 Minimalist and scalable Transformer that disentangles complexity and scale

➔ Neural scaling is computationally expensive

2 Continual training with periodic cooldowns that outperform cosine schedule

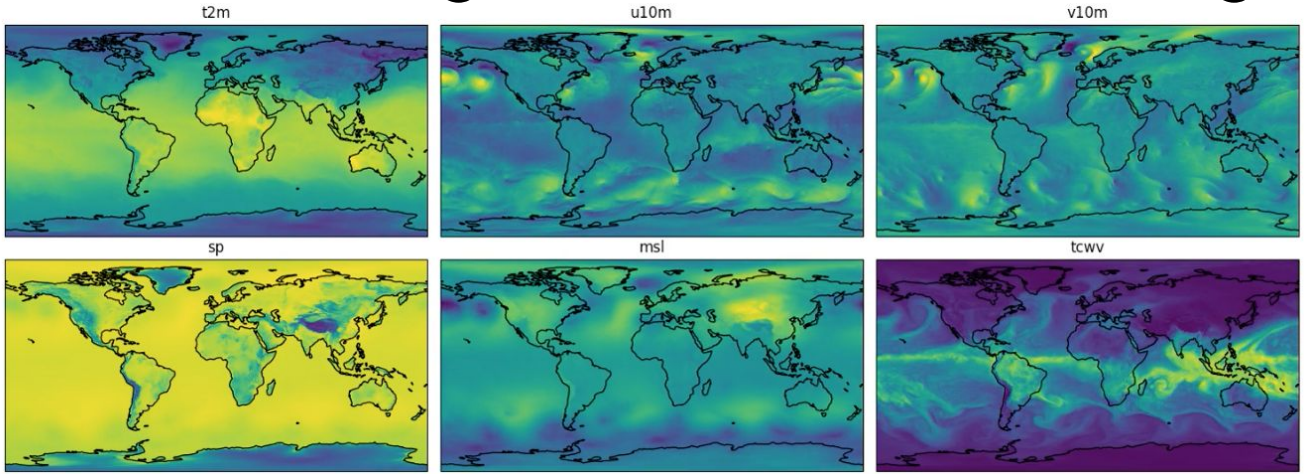
➔ Downstream task differs from pre-training objective

3 Re-purposing cooldowns for aligning downstream performance

➔ “Compute-optimal” behavior is unclear for weather models

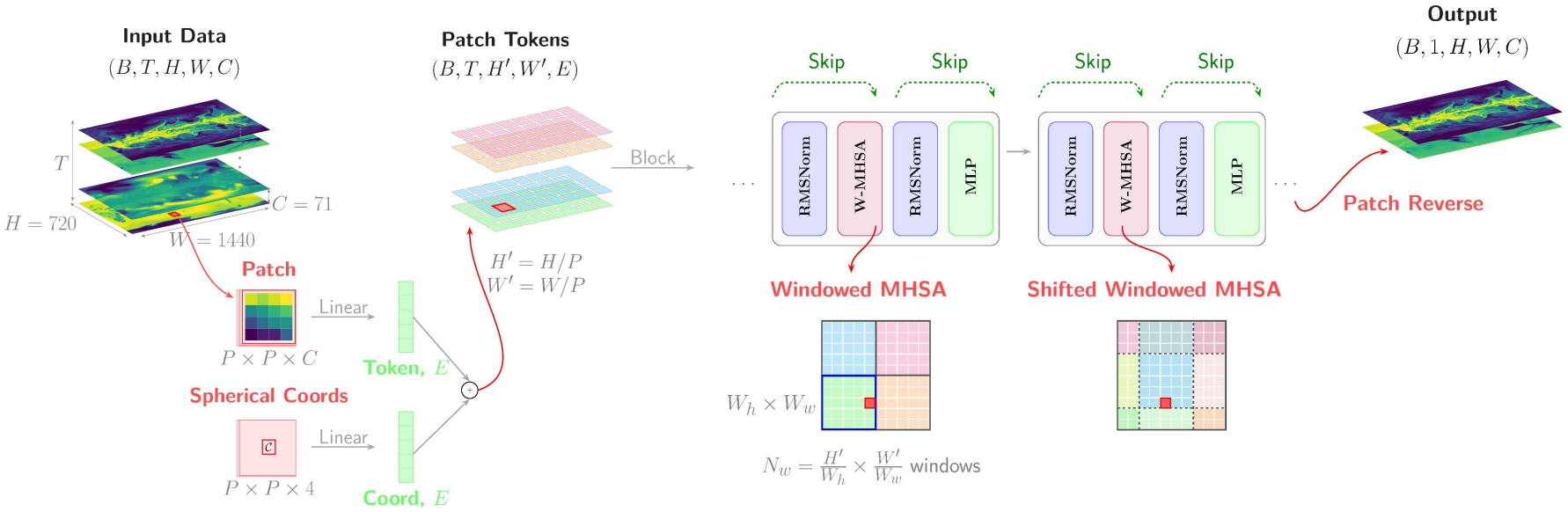
4 Identify compute-optimal training regimes and scaling trends

ERA5 is a Strong Testbed for Scaling



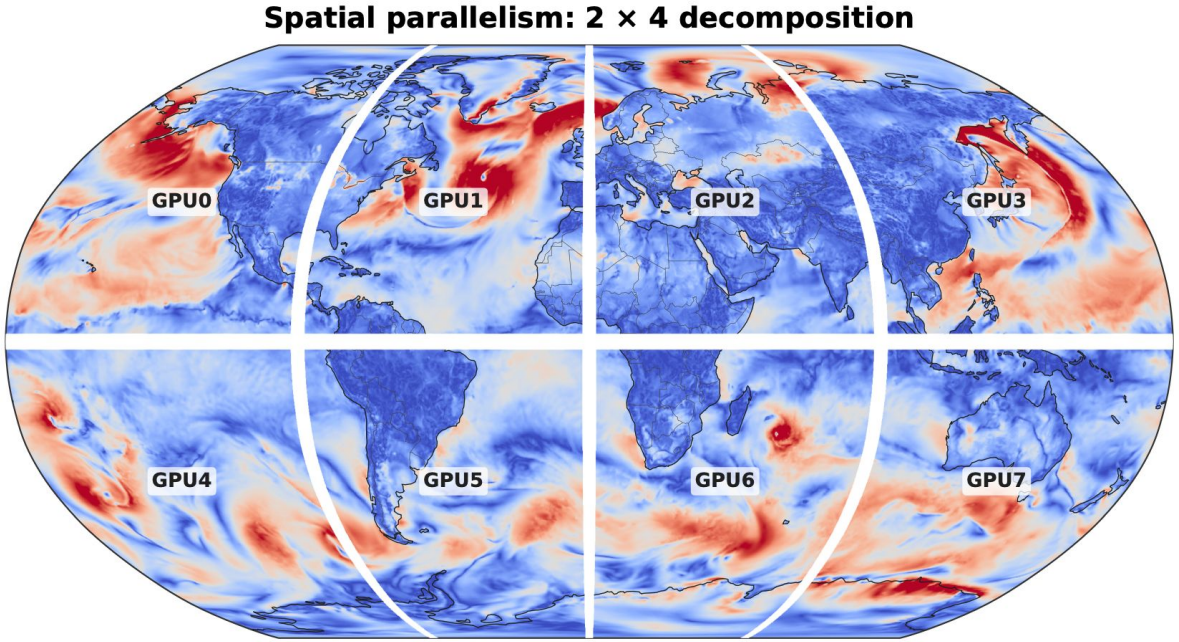
- Global [reanalysis product from ECMWF]
- Well-studied with trustworthy benchmarks through [WeatherBench2]
- Moderately large dataset (O(100)TB), complex physics + data assimilation; O(300,000) samples; O(100) variables; atmospheric state from 1940 - present at hourly resolution
- Gold Standard Numerical Model (IFS / HRES)

Minimalist, Scalable Swin Transformer



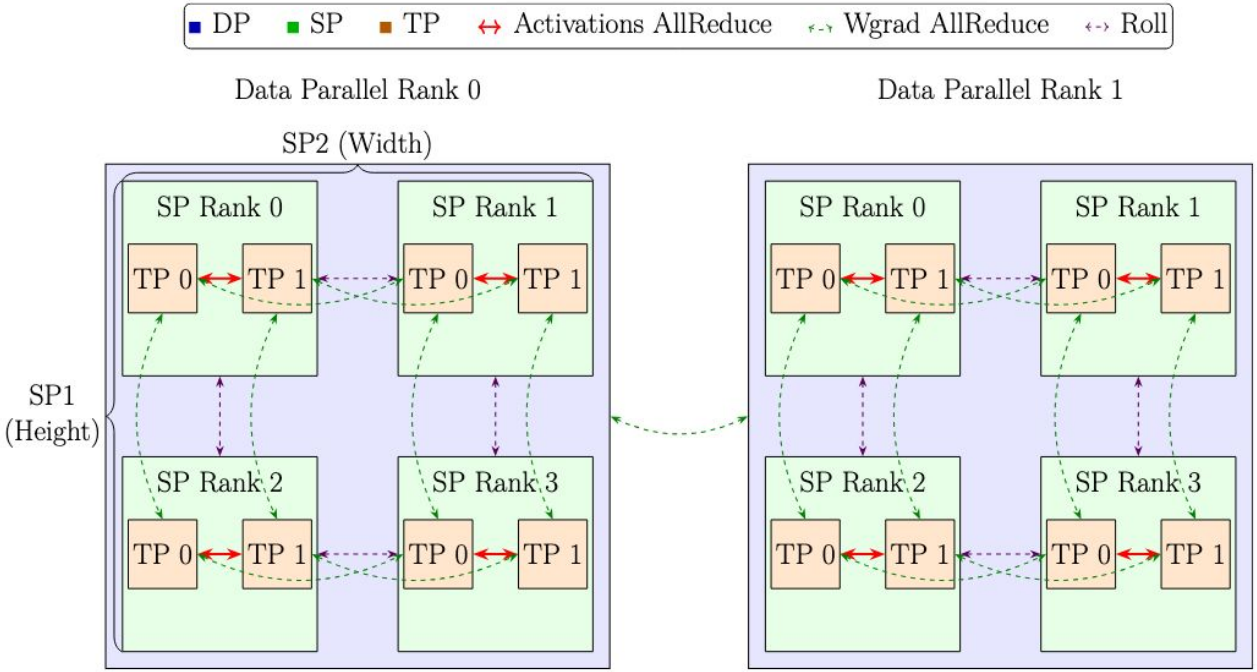
$$\tilde{\mathbf{X}} = \text{RMSNorm}(\mathbf{X}), \quad \mathbf{Y} = \text{W-MHSA}(\tilde{\mathbf{X}}) + \mathbf{X},$$
$$\tilde{\mathbf{Y}} = \text{RMSNorm}(\mathbf{Y}), \quad \mathbf{O} = \text{MLP}(\tilde{\mathbf{Y}}) + \mathbf{Y},$$

Distributed through Spatial Parallelism



- Scaling law analyses necessitate scalable architectures
- High resolution images = Activation memory bottleneck along with model parameters
- Standard LLM scaling through FSDP do little to alleviate this pressure
- 4D Parallelism: Data, Tensor, Spatial (2D)

Distributed through Spatial Parallelism



- Scaling law analyses necessitate scalable architectures
- High resolution images = Activation memory bottleneck along with model parameters
- Standard LLM scaling through FSDP do little to alleviate this pressure
- 4D Parallelism: Data, Tensor, Spatial (2D)

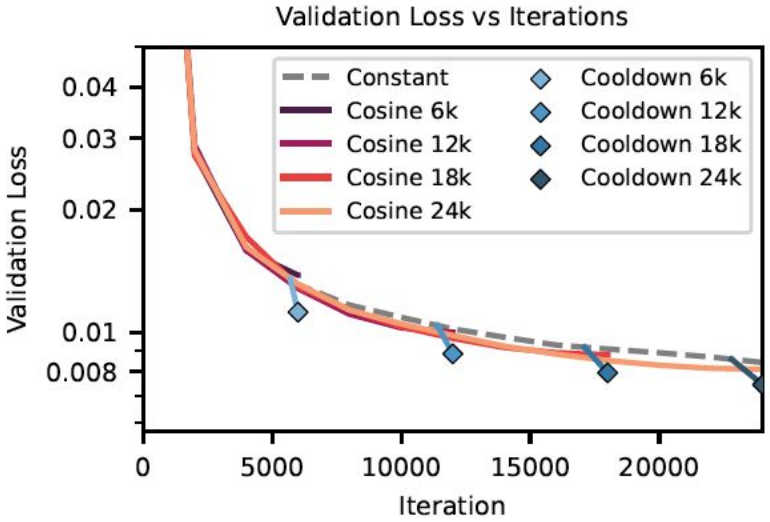
Distributed Roll for Forward / Backward

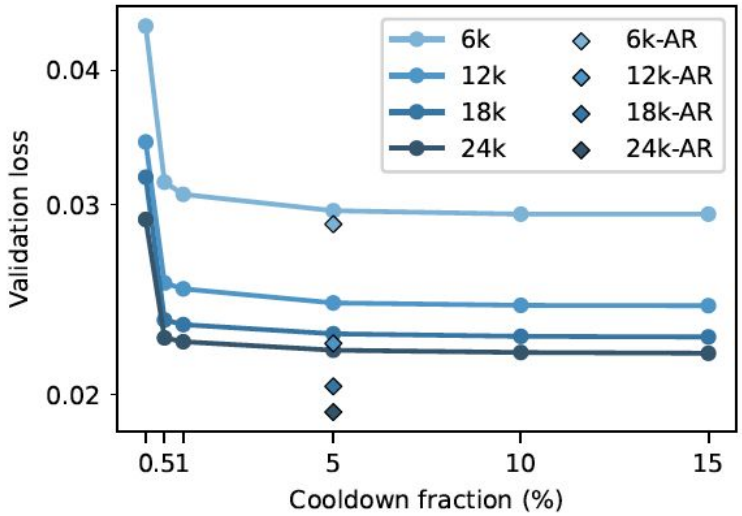
Algorithm 1 Distributed Roll for Shifting Windows in `Swin`

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

Constant LR + Cooldowns are Effective

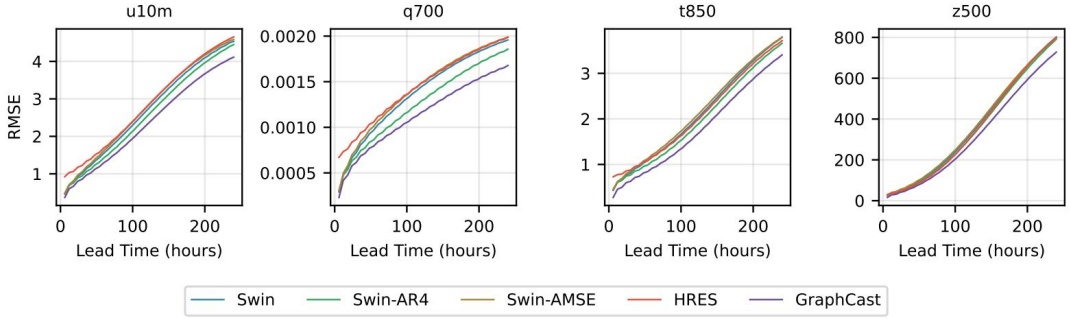




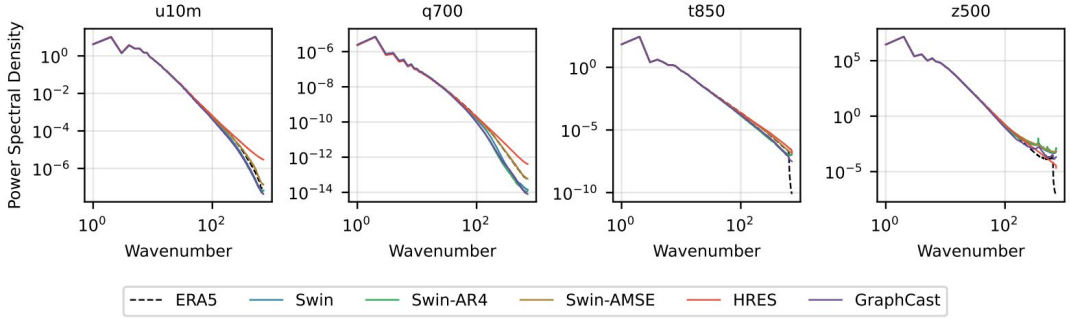
- Cooldowns at the last 5% (sqrt decay of LR) allow continual training
  - More performant than cosine schedules from scratch
- 5% is sufficient; Cooldowns can be repurposed with, for example, autoregressive rollout (AR) that shows lower loss when averaged over longer periods

Cooldowns Help Downstream Alignment

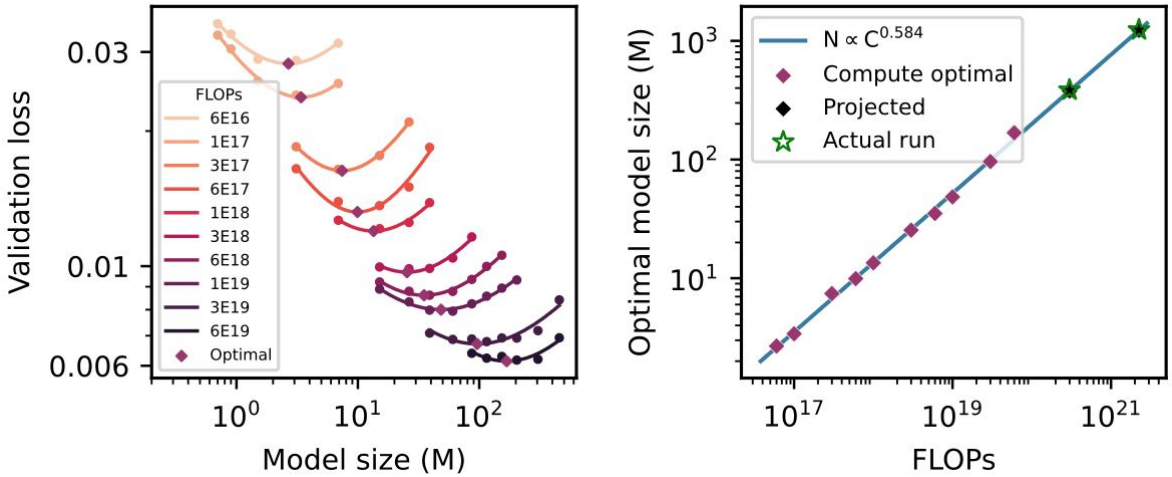
- Repurpose the short (5%) cooldown period with a different loss
- Autoregressive Rollout (AR)
  - Reduces RMSE in longer rollout through smooth forecasts
- AMSE (Adjusted MSE)
  - Spectral adjustment
  - Retains sharp spectra that matches ERA5
- Can possibly repurpose with probabilistic losses (CRPS), or other domain-specific metrics



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

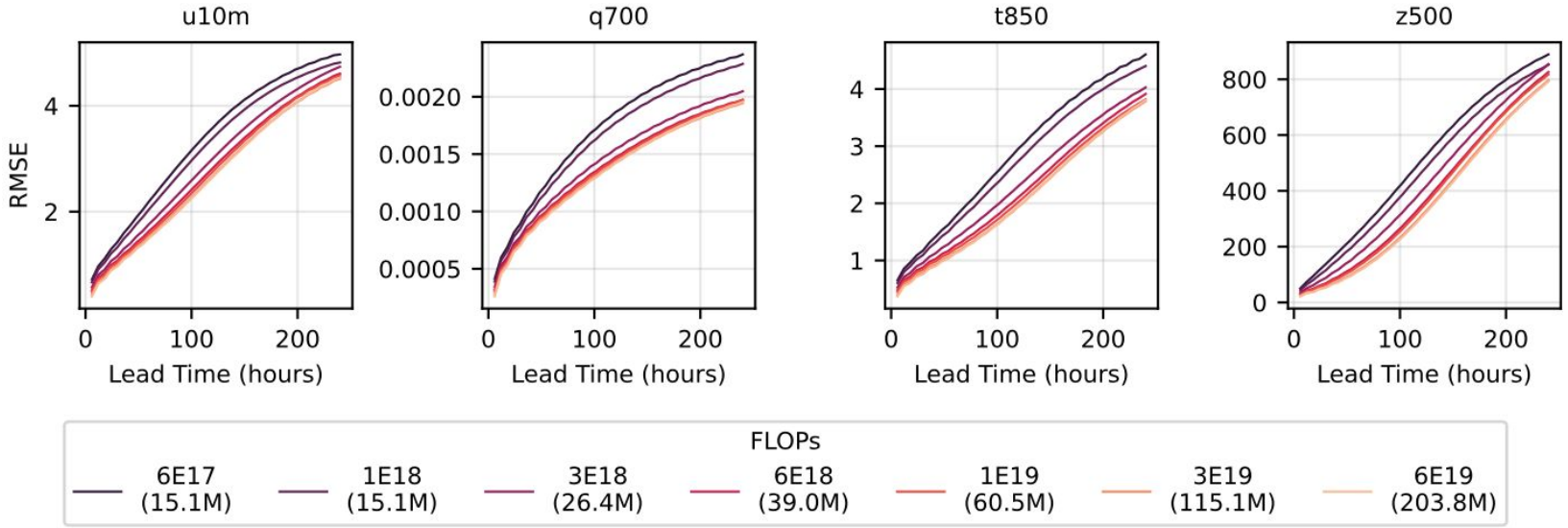


Neural Scaling Shows Clear Compute-Optimal Regimes

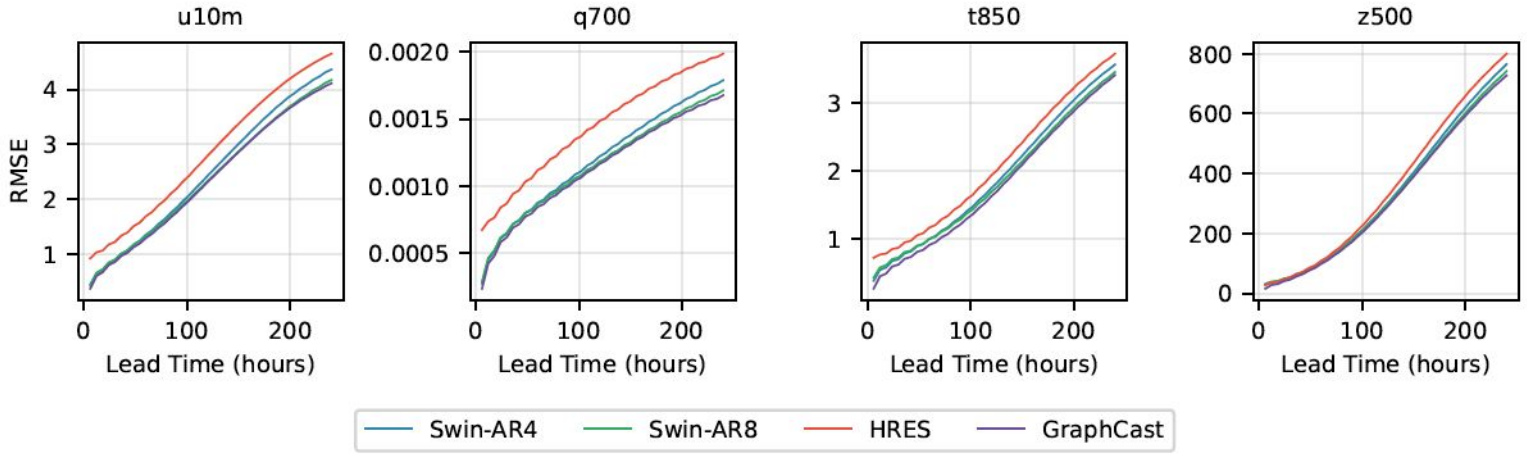


- All models are trained once and cooled down to different iterations to construct **IsoFLOPs**
  - Compute-optimal trends over three orders of magnitude (6E16 - 6E19 FLOPs, 0.7M to 456M)

Compute Helps Improve Systematically to SOTA

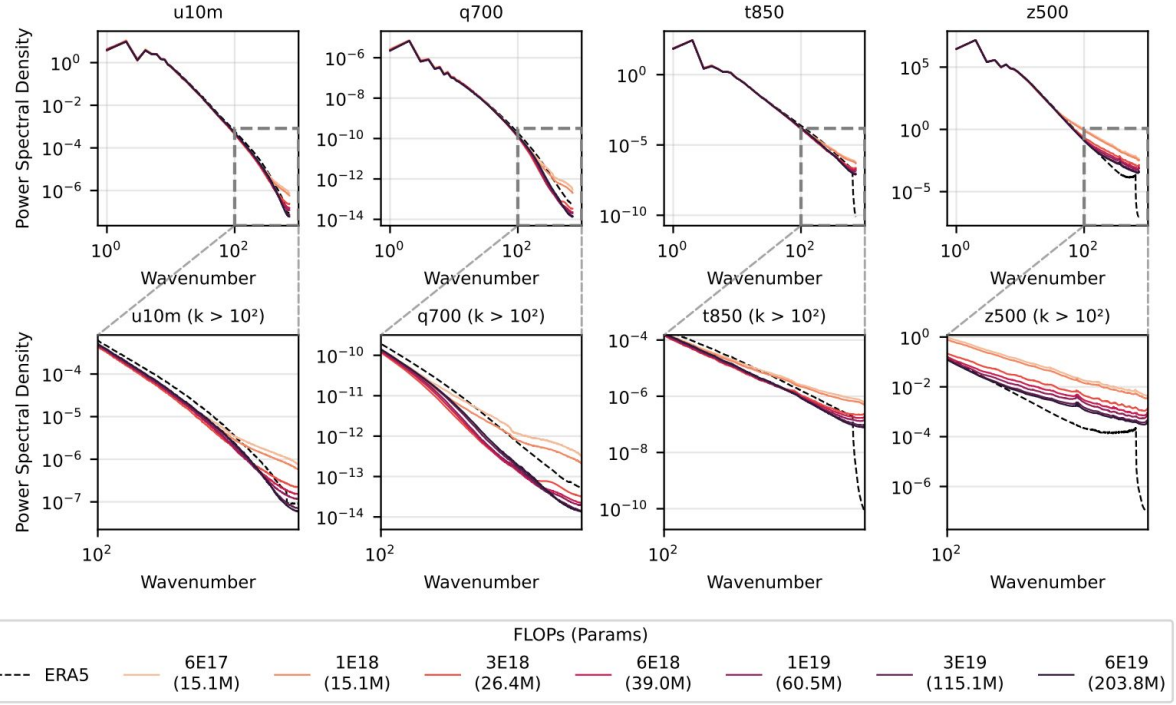


Compute Helps Improve Systematically to SOTA

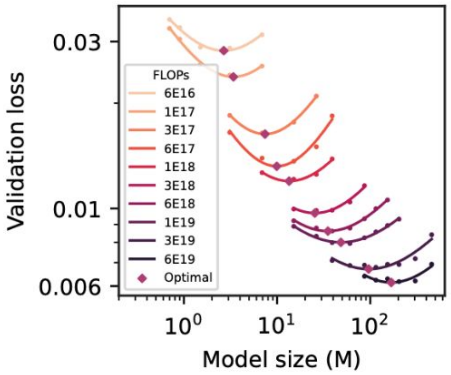


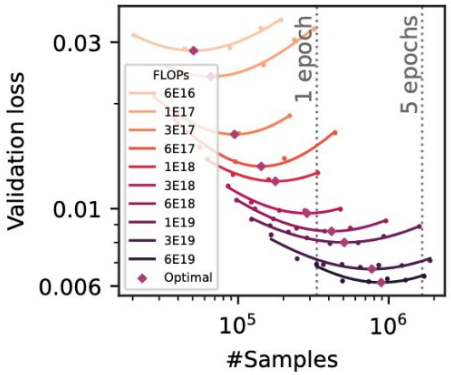
- Swin-AR8 (8-step rollout in cooldown period) matches GraphCast
  - **GraphCast: ~1E21 FLOPs, 4 weeks on 32 TPUv4, ~1% MFU**
  - **Swin-AR8: ~8E19 FLOPs, 6 hours on 32 A100, ~30% MFU:** lesser FLOPs, more efficient due to dense Transformers

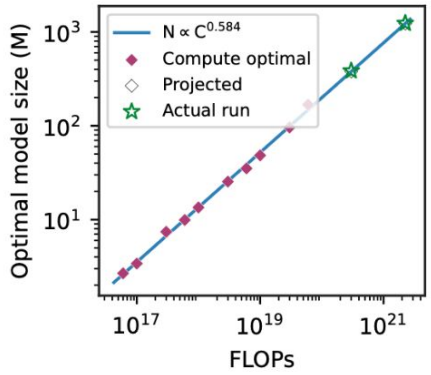
Bigger Models Preserve Spectra Better

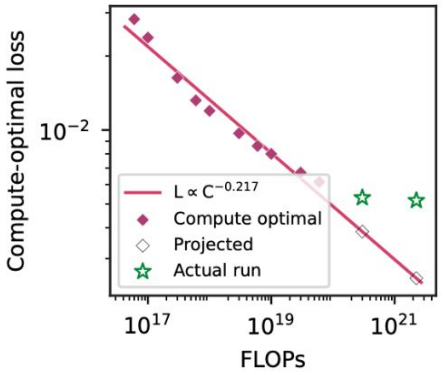


Limited Data Prevents Extrapolation









- Compute-optimal behavior is seen up to 5 epochs
- Projecting to 1.24B uses 14 epochs = overfitting, performance plateaus, and scaling stops
- Neural scaling is a helpful diagnostic before frontier-scale pre-training runs

Conclusions

➔ Architecture and training design space is complex and fragmented

1. Minimalist and scalable Transformer that disentangles complexity and scale

➔ Neural scaling is computationally expensive

2. Continual training with periodic cooldowns that outperform cosine schedule

➔ Downstream task differs from pre-training objective

3. Re-purposing cooldowns for aligning downstream performance

➔ “Compute-optimal” behavior is unclear for weather models

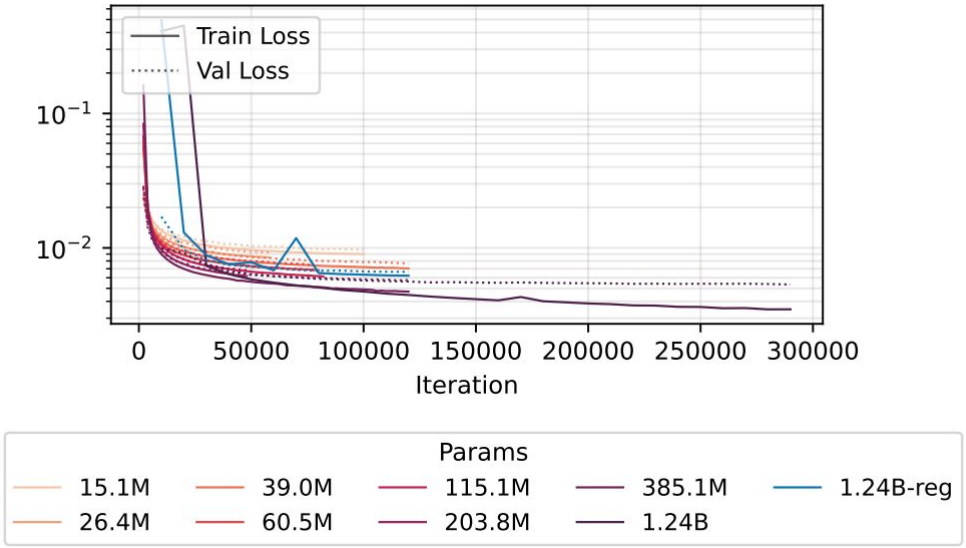
4. Identify compute-optimal training regimes and scaling trends

Paper Code

S Subramanian, A Kiefer, A Nigmetov, A Gholami, D Morozov, M Mahoney, ``On Neural Scaling Laws for Weather Emulation through Continual Training'', NeurIPS (2026), Code

Extra

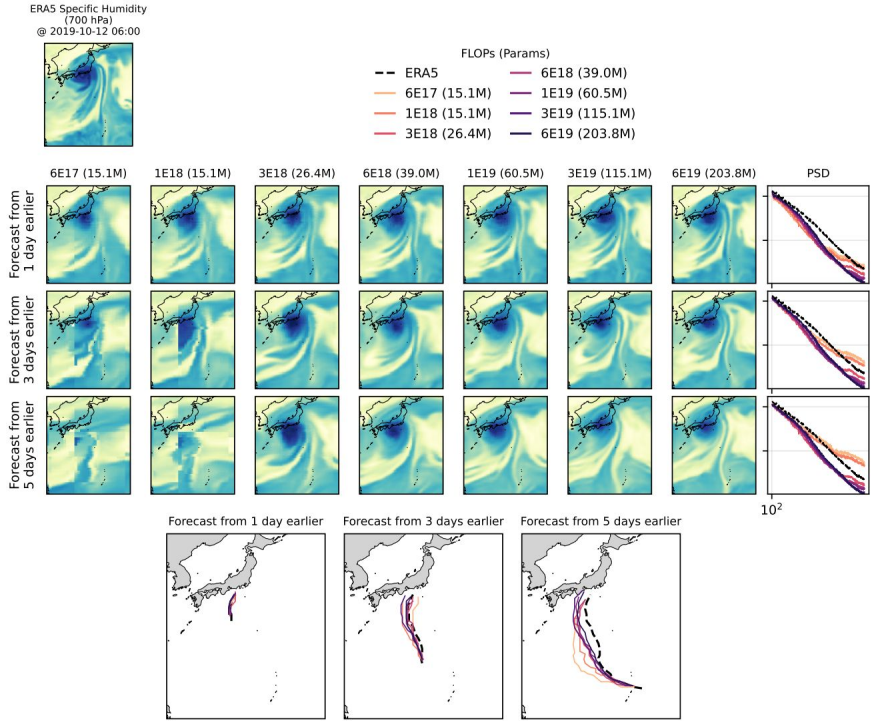
Training data is limited for ERA5



- Training / Validation loss track closely within the scaling range 6E16 - 6E19 FLOPs
- At O(1E21), the model size is 1.24B ~ overfits significantly
  - Regularization may help but does not recover extrapolation
  - Limited (diversity / complexity) training data may remain a bottleneck towards clean scaling

Bigger Models Preserve Spectra Better

- Track Typhoon Hagibis landfall from 1, 3, 5 days earlier forecasts with the different compute-optimal models
- Bigger models show sharper spectra
- The small models show unrealistic, “patchy” features when initialized from earlier times
- Tropical Cyclone tracks get better with compute, but deterministic prediction limits our conclusion



Hyperparameter Choices

- We use patch size 4, head dimension 64 for all models
  - Smaller models can handle larger LRs - HPO
  - First order HPO shows patch size 4 gives lower losses when measured on an IsoFLOP (smaller patches with smaller models or larger patches with larger models); but not too much sensitivity
  - Very shallow or very deep models show higher losses: Aspect ratio needs to be reasonable; Deeper models are slightly preferred
- Models are trained on 40G A100s with hybrid data-spatial parallelism
- DALI dataloader for performant I/O
  - Stateful to ensure mid-epoch checkpointing

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

Patch Size	Embedding	Depth	Window Size	TFLOPs/Iter	#Params	LR	Validation 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